Proceedings of META'2023

9th International Conference on Metaheuristics and Nature Inspired computing

Sponsors

A comparative analysis of simheuristics to address the berth allocation problem in bulk ports

Daniel Mendoza-Casseres*[1]

[1]Universidad del Atlántico – Colombia

Abstract

This study deals with the Berth Allocation Problem (BAP) in bulk ports which are NP-hard problems. Bulk cargo is one that is transported in large quantities without packaging. The bulk terminals carry out the loading through devices called shiploaders. Metaheuristics such as Genetic Algorithms, Tabu Search, Greedy Randomized Adaptive Search Procedure (GRASP) and Simulated Annealing (SA) have been implemented for BAP in bulk ports. Other studies have used simulation for the BAP; however, there is little research that uses simulation-optimization approaches, mainly using the Simheuristic method. Simheuristics extend metaheuristics by adding a simulation layer that allows the optimization component to deal with scenarios under uncertainty.This short paper proposes the scheduling of vessels with the variation of shiploaders in bulk ports using GRAPS or SA metaheuristics in a simulated environment to make a comparison. Initially, the SA and GRASP metaheuristics have been implemented for the BAP in bulk ports without simulation. This extended abstract of work-in-progress proposes a simheuristic approach for addressing the berth allocation problem in bulk ports. Stochastic loading times and stochastic set up times will be considered in a bulk terminal to show the use of the resources used by the port. The simulation model discrete BAP with metaheuristics will be built using FlexSim Software 2023. An optimization cycle will be scheduled. This cycle consists of configuring an optimization loop using C++ and FlexSim. The purpose is to obtain results from stochastic instances with the Simheuristics to minimize the total penalty cost. **Keywords**: Simheuristics, Scheduling, GRASP, SA, BAP.

*Speaker

A systematic review and ranking of operators in adaptive large neighborhood search for vehicle routing problems

Stefan Voigt[1]

Catholic University of Eichstätt-Ingolstadt
Ingolstadt School of Management
Auf der Schanz 49, 85049 Ingolstadt, Germany
stefan.voigt@ku.de

1 Introduction

The adaptive large neighborhood search (ALNS) is a metaheuristic that extends the large neighborhood search by selecting removal and insertion operators adaptively [4]. The performance of the ALNS heavily depends on the operators chosen.

The goals of this review are to classify operators using consistent terminology, analyze their performance and establish a common basis for future research. To achieve these goals, we conduct a network meta-analysis of 211 articles that meet our criteria, and we employ incomplete pairwise comparison matrices, similar to rankings used in sports, to rank the operators.

Our review makes the following contributions: (1) We propose a consistent nomenclature and classification scheme for the existing operators. (2) We conduct a rigorous analysis of the performance of different operators, helping researchers make informed decisions when selecting operators for their specific problem. (3) We identify key design principles, which can serve as guidelines for designing future ALNS implementations. (4) We discuss best practices and future research directions for not only ALNS but also for metaheuristics in general.

2 Research methodology: Network meta-analysis

In terms of methodology, we employ a network meta-analysis to systematically review remvoval and insertion operators. A network meta-analysis is a statistical approach that summarizes research and compares multiple treatments used in various studies. Our study is inspired by the works of [3] and [5]. In contrast to their work, our study goes further by comparing not just two treatments (i.e., ALNS versus non-adaptive LNS) but several treatments (i.e., several operators).

2.1 Identification and review of studies

We identify relevant studies by searching for publications with *adaptive large neighborhood search* in the title on Google Scholar following the procedure of [5]. In addition, we search for articles with *adaptive large neighborhood search* or *ALNS* as keywords in SCOPUS following the procedure of [3]. We identified a total of 211 relevant studies. We reviewed these studies in chronological order and classified the operators used within them.

2.2 Ranking

We create a ranking of operators based on their performance in identified studies. The two most common methods to assess the performance are the *frequency-based* and *ablation-based* performance analysis. The *frequency-based* performance analysis shows the number of times the operators have been applied when solving a set of test instances. The *ablation-based* performance analysis assesses the performance of the operator by comparing the results when the operator is present compared to when the operator is excluded from the ALNS. We conduct pairwise comparisons for each operator in every study, use these to construct a pairwise comparison matrix and ultimately derive a ranking of the performance of operators. The ranking of operators based on an incomplete comparison matrix is not trivial and is similar to ranking teams or players in sports [1]. The incomplete pairwise comparison matrix A is used to determine a weight vector $w = (w_1, w_1, \ldots, w_n)$, where the elements $a_{i,j}$ are estimated by $\frac{w_i}{w_j}$ and $\sum_{i=1}^{n} w_i = 1$. We use the logarithmic least squares method to determine w. Missing values can be determined by solving linear equations [2], [1].

3 Sample results

3.1 Removal operators

Classification We classify removal operators according to the amount of information they use to determine the set of removed customers. Simple removal operators do not use information on current or previous solutions. More advanced operators use information derived from the current solution. The most advanced operators use information from the current and previous solutions, taking the history of the search into account.

Naming convention We suggest using a consistent and informative convention, as follows.

(1) Removal: (2) criteria - (3) restricted set of removal candidates - (4) seed

The naming convention indicates that it is a removal operator (1). The components (2-4) provide clarity on three key aspects of the removal operator:

(2) What is the criteria used to determine which customers are removed?
(3) Is the set of customers that can be removed restricted in any way?
(4) Does the removal operator use a seed customer/route?

Ranking Table 1 shows the results when deriving the weight vector w (see Section 2.2). The higher the weight, the better the operator. We also show the number of comparisons n. The most frequently used removal operators (*standard set*) occupy ranks three to five: *R1* ranks fifth, *R16* ranks fourth, and *R13* ranks third. The two sequence-based removal operators *R9* and *R10* occupy the first and second rank but are not even among the top 10 most frequently used operators. This suggests that adding one of these two operators to the *standard* set may improve its performance. Despite this, the ranking confirms that the *standard* set remains a solid choice for ALNSs.

Table 1. Top 10 ranked removal operators

#	Removal operator	Weight	n
1	R9: All customers - from randomly selected sequence within concatenated routes	0.1248	15
2	R10: All customers - from one of two Kruskal clusters from randomly selected route	0.0878	21
3	R13: A-posteriori score related customers - to seed customer	0.0641	112
4	R16: Worst cost customers	0.0592	164
5	R1: Random customers	0.0535	205
6	R19: All customers - from random route - multiple	0.0528	40
7	R5: A-priori distance related customers - to static seed customer	0.0509	31
8	R18: All customers - from random route - single	0.0495	80
9	R12: A-posteriori distance related customers - to seed customer	0.0382	8
10	R4: Random customers and their nearest neighbor	0.0371	17

3.2 Insertion operators

Classification We classify insertion operators based on how they determine in which position the previously removed customers are inserted. Random insertion operators insert customers at a random position. Best cost insertion operators select the position that results in the least increase in objective value. Best cost insertion operators with constrained options do not evaluate each and every insertion position. Best timing insertion operators use time criteria instead of the objective value. Finally, look-ahead insertion operators overcome the myopic behavior of best insertion operators by incorporating information on the effect of inserting the current customer on the insertion of following customers. The second criteria used to classify insertion operators is the order in which the list of insertion candidates is sorted.

Naming convention We suggest using a naming convention that reflects firstly that it is an insertion operator and secondly how customers are inserted.

(1) Insertion: (2) order - (3) position - (4) noise - (5) restricted set of insertion positions

The components (2-5) describe four key aspects of the insertion operator:

(2) Does the operator sort the list of removed customers?
(3) Into which position is the selected customer inserted?
(4) Is the chosen position deterministic or does it vary by applying noise?
(5) Is the set of insertion positions restricted?

Ranking Table 2 shows the results of deriving the weight vector w. *I13* ranks first, being superior to other look-ahead insertion operators, and in particular dominating *I11* which achieves rank 6. Note that the difference in weights is significant with a sufficiently large number of comparisons (53 and 26). Regarding insertion order, regret-based ordering methods appear to be more efficient than random and removal ordering methods, with the exception of *I5*, which ranks third. Regarding insertion position, *at best position* is the method of choice, other orders are either infrequently used or show inferior performance, such as *at random position* (rank 10). To summarize, we suggest that the *standard* set of insertion operators should include *I13* instead of *I11*, and *I5* because of its simplicity, instead of *I7*.

Table 2. Top 10 ranked insertion operators

#	Insertion operator	Weight	n
1	I13: Customer with highest position regret - at best position	0.1847	53
2	I12: Customer with highest route regret - at best position - with noise	0.1179	8
3	I5: In random order - at best position	0.0927	22
4	I14: Customer with highest position regret - at best position - with noise	0.0912	26
5	I7: Customers with lowest cost first - at best position	0.0862	72
6	I11: Customer with highest route regret - at best position	0.0807	26
7	I8: Customers with lowest cost first - at best position - with noise	0.066	29
8	I3: In removal order - at best position	0.0631	29
9	I10: In random order - at best position - restricted to random route	0.053	7
10	I1: In removal order - at random position	0.0463	7

4 Conclusion

We conducted a thorough analysis of the removal and insertion operators used in ALNS for VRPs by reviewing 211 relevant articles. Our analysis revealed a total of 1266 removal operators (an average of 6 per ALNS) and 788 insertion operators (an average of 3.73 per ALNS). We identified 57 distinct removal operators and 42 insertion operators and established a consistent nomenclature. Furthermore, we ranked the operators using an incomplete pairwise comparison matrix and found a slight discrepancy between the most frequently used and the most effective operators. These findings provide valuable insights into the effectiveness of different operators and can assist researchers in selecting appropriate operators for their specific VRP.

References

1. Sándor Bozóki, László Csató, and József Temesi. An application of incomplete pairwise comparison matrices for ranking top tennis players. *European Journal of Operational Research*, 248(1):211–218, jan 2016.
2. Sándor Bozóki, János Fülöp, and Lajos Rónyai. On optimal completion of incomplete pairwise comparison matrices. *Mathematical and Computer Modelling*, 52(1-2):318–333, jul 2010.
3. Setyo Tri Windras Mara, Rachmadi Norcahyo, Panca Jodiawan, Luluk Lusiantoro, and Achmad Pratama Rifai. A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, 146:105903, oct 2022.
4. Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, nov 2006.
5. Renata Turkeš, Kenneth Sörensen, and Lars Magnus Hvattum. Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search. *European Journal of Operational Research*, 292(2):423–442, jul 2021.

Impact of Structural Bias on the Sine Cosine Algorithm: A Theoretical Investigation Using the Signature Test

Kanchan Rajwar[1], Kusum Deep[1], and Muthu Mathirajan[2]

[1] Department of Mathematics, Indian Institute of Technology Roorkee, Uttrakhand - 247667, India
[2] Department of Management Studies, Indian Institute of Science, Bangalore 560012, India
krajwar@ma.iitr.ac.in, kusum.deep@ma.iitr.ac.in, msdmathi@iisc.ac.in,

Abstract: Metaheuristic algorithms have been recognized for their effectiveness in solving non-convex and non-linear complex optimization problems. These algorithms are influenced by landscape bias, guided by objective function values, and algorithmic operator bias directed by the operator used in the algorithms. The presence of algorithmic operator bias, also known as structural bias, forces the population to revisit a particular region, badly affecting the algorithm's exploration capacity. Also, since the population revisits the same place without gaining new information, it increases computational costs and slows the convergence rate. Therefore, it is crucial to identify and address structural bias to enhance algorithm performance and reduce computational time. To the best of our knowledge, no previous study has focused on investigating the structural bias of the Sine Cosine Algorithm (SCA) in the existing literature. Therefore, the main objective of this study is to examine the structural bias present in the SCA, a widely used metaheuristic algorithm. To investigate structural bias signature test is employed. Additionally, average Euclidean distances of the population is calculated to assess spatial relationships and overall distribution. Our analysis uncovers a prominent bias in the SCA towards the axes and the origin, suggesting a strong tendency to converge towards specific regions within the search space. By understanding and characterizing this bias, we provide valuable insights into the behavior of the SCA, which can contribute to the research community's understanding and guide future improvements in algorithm design.

keywords: Metaheuristic algorithm, Sine Cosine Algorithm, Structural bias, Signature test, Theoretical analysis.

1 Introduction

Optimization techniques play a crucial role in solving real-life optimization problems. As problem complexity increases, newer and more efficient optimization algorithms are needed. Metaheuristic algorithms have been proposed and have become popular for solving non-convex optimization problems. The survey paper [1] highlighted the top 10 metaheuristic algorithms in terms of citation as follows: Particle Swarm Optimization (PSO) [2], Genetic Algorithm (GA) [3], Simulating Annealing [4], Differential Evolution [5], Ant Colony Optimization [6], Tabu search [7], Grey Wolf Optimization [8], Artificial Bee Colony [9], Cuckoo Search [10], and Harmony Search [11]. As of the current date, there exist more than 540 metaheuristic algorithms in the literature. An exhaustive survey of the state-of-the-art can be found in the work by [1].

However, only a limited amount of research such as [12–14] have been dedicated to understanding the behavior of these algorithms [15]. Most of the proposed algorithms rely solely on simulation results, indicating a pressing need for in-depth analysis to comprehensively understand the behavior of these algorithms. Recent studies [1, 16] strongly emphasize the necessity for a thorough examination of these algorithms to gain deeper insights into their characteristics and performance. Motivated by this gap in knowledge, the primary focus of this study is to gain a deeper understanding of the population dynamics of the algorithm under specific circumstances.

Common to all population-based optimization algorithms are two essential components: (*i*) a means of evaluating and discriminating solutions, and (*ii*) a sequential collection of mechanisms that modify solutions through various operators [15]. Upon closer examination of these algorithms' mechanisms, we observe the following pattern: an initial set of feasible solutions is generated randomly, and their objective function values are evaluated. This set is then iteratively updated using operators that balance exploration and exploitation, two contrasting yet crucial aspects. The ability to explore new regions of the search domain and exploit promising areas are fundamental strategies in these algorithms [19]. Surprisingly, despite decades of research on population-based optimization algorithms, a precise definition of exploration and exploitation is still lacking. Only a handful of

studies have been conducted to understand the algorithms' exploration-exploitation trade-off, with a comprehensive review available in [20]. Current consensus defines exploration as the process of venturing into entirely new regions of the search space, while exploitation entails visiting areas within the vicinity of previously explored solutions. Achieving an appropriate balance between exploration and exploitation remains a critical research question in Computational Intelligence (CI) [21].

Metaheuristic algorithms are impacted by two key factors: landscape bias and algorithmic operator bias. Landscape bias directs the population towards improved objective function values, while algorithmic operator bias steers the population towards specific locations within the search domain. Within the context of these algorithms, the latter factor is referred to as "structural bias" in [22]. It compels the population to revisit a particular area of the search domain, leading to increased computing costs without yielding any new information. Hence, detecting structural bias is essential for any algorithm. In Figure 1, the illustration showcases the impact of structural bias on population movement. The figure demonstrates that the subsequent position (X_i^{t+1}) of the candidate is influenced by the combined effect of the landscape and the structural bias exerted on the current position (X_i^t). Unfortunately, there is no study or technique exist in literature that measure these two force quantitatively.

In an ideal algorithm, the structural force or bias should be non-existent, allowing candidates to move freely towards optimal solutions. However, when a structural bias is present, it compels the population to revisit a specific region without acquiring any new information. This not only increases the computational cost but also delays the convergence rate. Consequently, it becomes crucial to detect and eliminate structural bias in order to enhance the algorithm's performance. The focus of this study is solely on identifying the presence of structural bias.

The signature test, introduced by Clerc [23], is a methodology utilized to identify structural bias in stochastic algorithms. While some theoretical studies have examined different type of bias of algorithms such as GA, PSO, ABC, ACO, and DE, there has been limited analysis conducted on other algorithms. Notably, no research has been conducted to analyze the structural bias of Sine Cosine Algorithm (SCA) [24]. Therefore, In this study, we focus on detecting the structural bias of one of such popular but less theoretical analysed algorithm SCA using signature tests.

The remainder of this paper is presented as follows: Section 2 provide a brief description of Sine Cosine Algorithm. Section 3 covers structural bias and the signature test. Section 4 presents the results of our simulation experiments. Finally, Section 5 summarizes our findings and proposes directions for future research.

2 Sine Cosine Algorithm

In the realm of metaheuristic optimization, various algorithms like GA, PSO, DE, ABC, and ACO have been extensively studied in diverse contexts. However, there exist relatively new algorithms that have not received as much theoretical attention. This paper focuses on one such algorithm, namely the Sine Cosine Algorithm (SCA).

Introduced by Mirjalili in 2015 [24], SCA is a population-based metaheuristic algorithm inspired by the mathematical properties of sine and cosine functions. It is specifically designed to tackle optimization problems. Like other metaheuristic algorithms, SCA starts with a randomly initialized set of solutions. The position update of each solution is determined by Equations (1) and (2):

$$X_i^{t+1} = x_i^t + r_1 \times \sin(r_2) \times |r_3 P_i^t - X_i^t| \tag{1}$$

$$X_i^{t+1} = X_i^t + r_1 \times \cos(r_2) \times |r_3 P_i^t - X_i^t| \tag{2}$$

Here, X_i^t represents the current solution's position in the i-th dimension at the t-th iteration, while r_1, r_2, and r_3 denote random numbers. The parameter P_i represents the position of the destination point in the i-th dimension, and —— denotes the absolute value. These equations are utilized in SCA as shown in Equation (3):

$$x_i^{t+1} = \begin{cases} X_i^t + r_1 \times \sin(r_2) \times |r_3 P_i^t - X_i^t| & \text{if } \ r_4 < 0.5 \\ X_i^t + r_1 \times \cos(r_2) \times |r_3 P_i^t - X_i^t| & \text{if } \ r_4 \geq 0.5 \end{cases} \tag{3}$$

r_4 is a uniformly distributed random number in the interval [0, 1].

The SCA algorithm encompasses four main parameters: r_1, r_2, r_3, and r_4. r_1 determines the movement direction for the next position, either towards or away from the solution-destination space. r_2 governs the extent of movement towards or away from the destination. r_3 introduces a random weight for the destination, emphasizing ($r_3 > 1$) or de-emphasizing ($r_3 < 1$) its effect on defining the distance. Finally, r_4 equally switches between the sine and cosine components in Equation (3).

During the initial half of the iterations, r_1 contributes to exploration, while in the latter half, it prioritizes exploitation. Mathematically, r_1 is defined as:

$$r_1 = a - a\frac{t}{T} \tag{4}$$

Here, t represents the current iteration, T is the maximum number of iterations defined as a termination criterion for SCA, and the constant 'a' is typically set to 2.

Figure **??** visually illustrates the effects of sine and cosine functions in Equations (1) and (2) on the next position of a solution. For more details on SCA and its source code, refer to [24].

3 Signature Test

In order to identify structural bias in an iterative optimization algorithm, it is necessary to separate the effects of the algorithm's operators from the effects of the optimization landscape. To accomplish this, Clerc proposed the signature test [23] in 2015.

To create the signature of the stochastic algorithm, a $2D$ plot of $10,000$ feasible solutions is generated within the search region $[0,1]^2 \in R^2$ using the constant single objective function f_0 defined by Eq. (5). The test algorithm will be performed ten times, generating 1000 feasible solutions each time. These points are then superimposed on a single graph for analysis.

$$\text{Min. } f_0 : [0,1]^2 \to [0,1], \quad \text{where } \forall x, f_0(x) = 1 \tag{5}$$

If the algorithm uses a greedy selection operator, it can be replaced with a random selection operator to completely eliminate the operator's effect. Clearly, the optima of such function also follow a uniform distribution. By utilizing f_0 as the objective function to minimize with a reasonable computational budget, the final positions obtained should also be uniformly distributed throughout the search space. Therefore, any non-uniformity observed in the distribution of the final positions could suggest the presence of structural bias. One primary method to identify such deviations is through visual analysis of the positions of the feasible solutions. When comparing two or more algorithms, visualization may not be effective, requiring the adoption of statistical measures. However, since this study solely focuses on a single algorithm, visual analysis is sufficient to identify any existing bias.

4 Simulation Results

To generate signatures, we performed independent simulations of SSA for 10 independent runs, with a population size of 1000 per run. Each run consisted of 50 iterations. We analyzed the population movement during the first five iterations, as well as the population after the 10^{th}, 15^{th}, 20^{th}, 25^{th}, 30^{th}, 35^{th}, 40^{th}, 45^{th}, and 50^{th} iterations. By overlaying the corresponding iterations from all 10 runs using the signature test, we obtained a total of $10,000$ feasible solutions plotted for each iteration. The simulations were implemented in Python and executed on a system equipped with an Intel(R) Core(TM) i7-3770 processor and 8.00 GB of RAM.

The simulation results are illustrated in Figures 1-16. In Figure 1, the initial randomly generated population is shown to be well distributed throughout the region $[0,1]^2$. Each color represents a different independent simulation (run), and the legend can be found in Figure 16. Figure 2 demonstrates that the populations overlap with each other, yet they remain relatively well distributed across the region, making it difficult to discern any bias in the SSA. However, Figure 3 reveals that the populations are moving towards both axes and the origin. The populations become progressively closer to the axes and the origin in the consecutive iterations 3, 4, and 5, as depicted in Figures 4, 5, and 6 respectively. This population dynamics exemplifies the inherent bias of the SSA towards the axes and the origin.

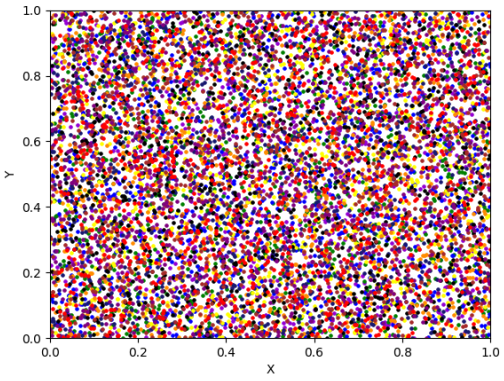


Fig. 1. Randomly generated initial population

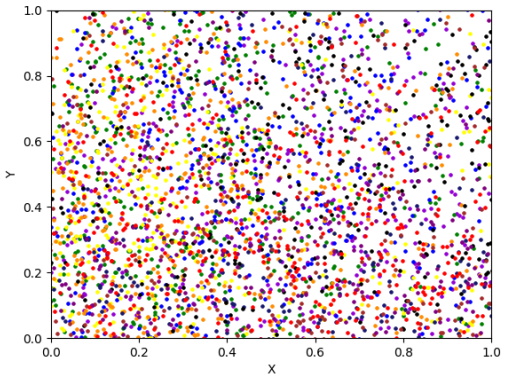


Fig. 2. Population after the 1^{st} iteration

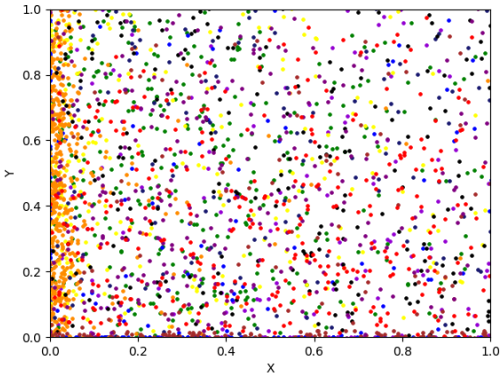


Fig. 3. Population after the 2^{nd} iteration

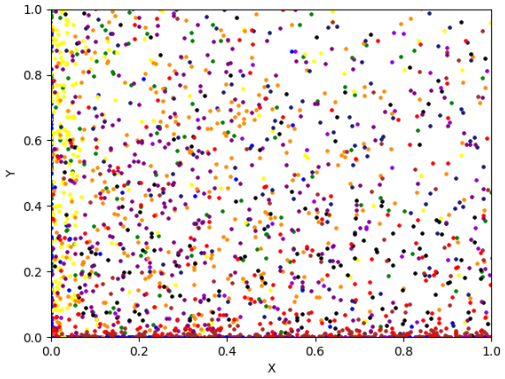


Fig. 4. Population after the 3^{rd} iteration

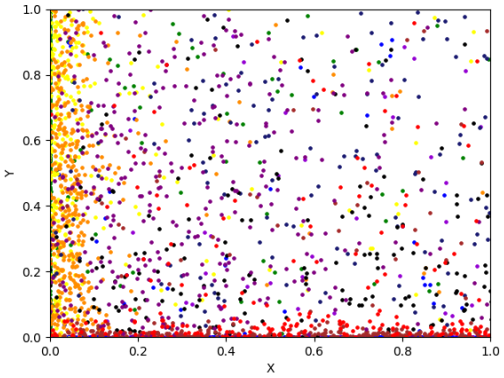


Fig. 5. Population after the 4^{th} iteration

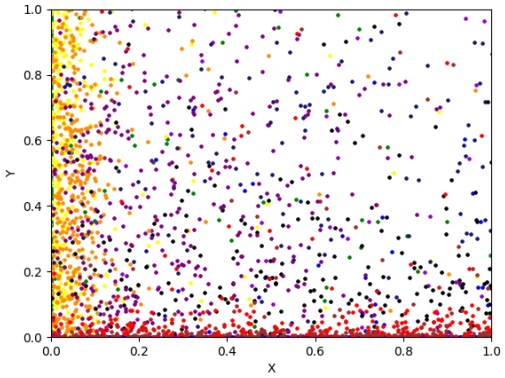


Fig. 6. Population after the 5^{th} iteration

The population dynamics of the SSA for the 10^{th}, 15^{th}, 20^{th}, 25^{th}, 30^{th}, 35^{th}, 40^{th}, 45^{th}, and 50^{th} iterations are also shown in Figures 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16 respectively. From these figures, it is evident that as the number of iterations increases, the populations tend to cluster closer to the axes and the origin. Based on these results, it is clear that the SSA exhibits a significant bias towards both the origin and the axes.

Furthermore, a numerical test was conducted to estimate the diversity of the population. In order to assess spatial relationships and overall distribution, the average Euclidean distances (AED) between pairs of individuals in the population were calculated. The average distance was determined by summing all the calculated distances from each point to every other point in the population,

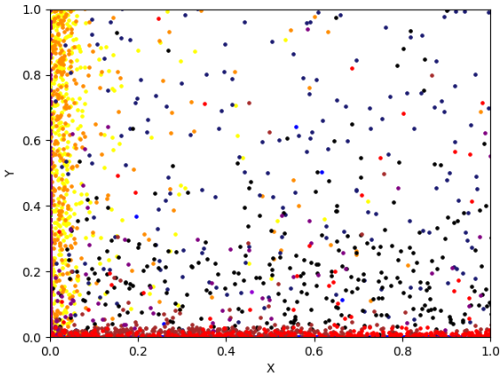


Fig. 7. Population after the 10^{th} iteration

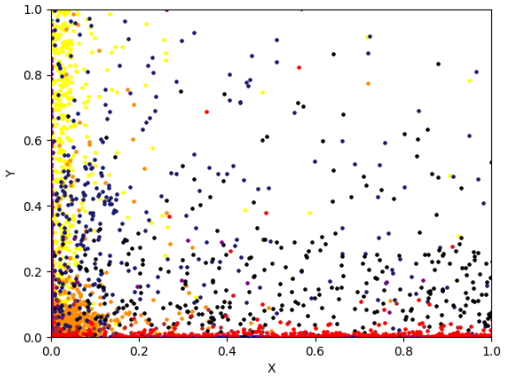


Fig. 8. Population after the 15^{th} iteration

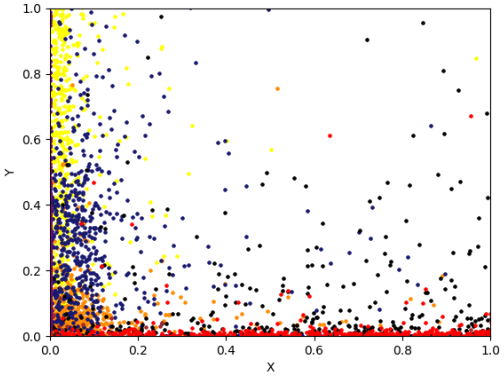


Fig. 9. Population after the 20^{th} iteration

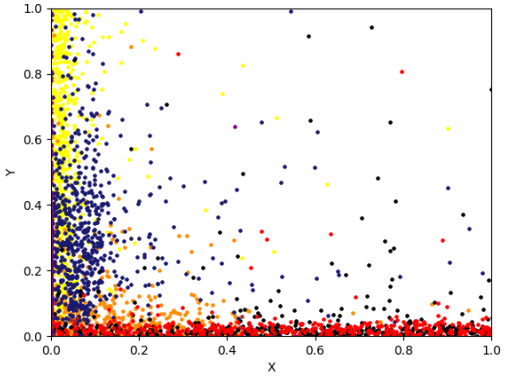


Fig. 10. Population after the 25^{th} iteration

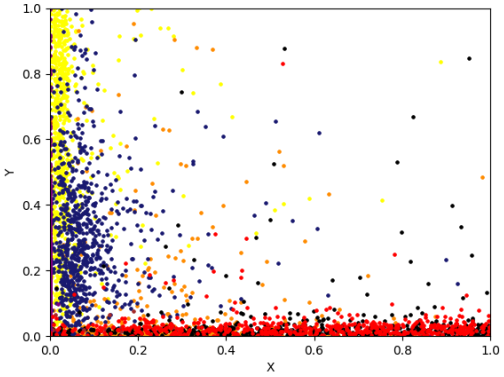


Fig. 11. Population after the 30^{th} iteration

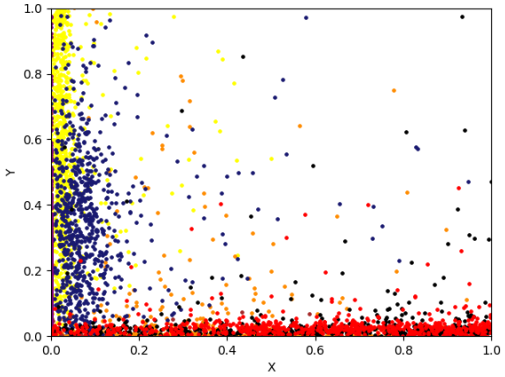


Fig. 12. Population after the 35^{th} iteration

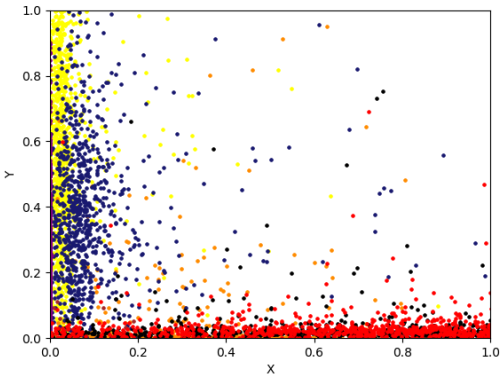


Fig. 13. Population after the 40^{th} iteration

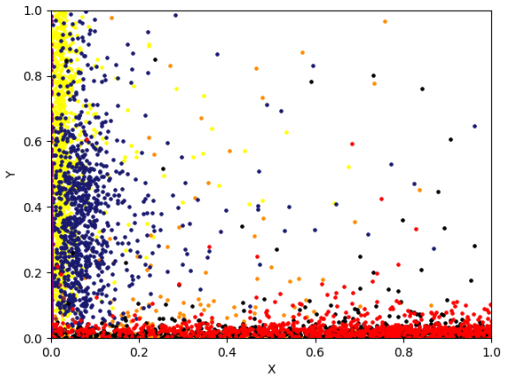


Fig. 14. Population after the 45^{th} iteration

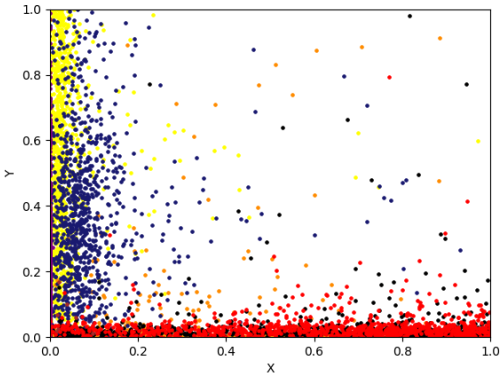


Fig. 15. Population after the 50^{th} iteration

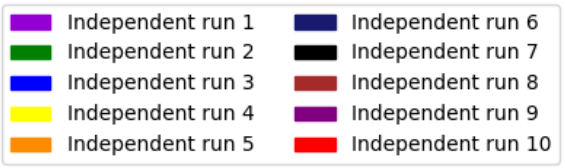


Fig. 16. Signatures of SSA for different independent run

and then dividing the sum by the total number of pairs. This average distance serves as a measure of the average separation between individuals, with a larger value indicating greater dispersion or diversity.

During the experiment, each iteration of the independent simulations involved a population of size 1000. Therefore, the total number of distinct pairs (taking into account that (x, y) and (y, x) are equivalent) was $(1000 \times 999)/2$. The distances between each pair were calculated using the Euclidean distance formula shown in Equation 6. The distances were then summed up, and the resulting sum was divided by $(1000 \times 999)/2$ to obtain the AED.

$$\text{distance}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{6}$$

Table 1 displays the AED for specific iterations in each run. The results demonstrate that, for each run, as the iteration increases, the AED decrease significantly. This indicates that population diversity decreases and becomes synchronized with the iterations, particularly towards the axes and origin, as revealed by the signature test.

5 Conclusion

In this study, we conducted an analysis of the theoretical behavior of population movement in the Sine Cosine Algorithm (SCA) with regard to structural bias. Using the signature test as a simple technique, we observed that the SCA demonstrates a significant bias towards the origin and both axes, indicating that it is an origin-axes biased algorithm. To validate these observations, Additionally, we calculated the average Euclidean distances of the population to assess spatial relationships and overall distribution. This characteristic of the SCA makes it particularly well-suited for optimization problems where the optimal solution is located near the origin or axes.

Table 1. Average Euclidean distance for different simulation (independent run)

Iteration	Average Euclidean Distance									
	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5	Simulation 6	Simulation 7	Simulation 8	Simulation 9	Simulation 10
1	1.45e+0	1.62e+0	1.66e+0	1.20e+0	1.33e+0	1.65e+0	1.37e+0	1.17e+0	1.16e+0	1.54e+0
5	1.19e+0	1.09e+0	1.23e+0	1.12e+0	1.13e+0	1.04e+0	1.33e+0	1.12e+0	1.12e+0	1.03e+0
10	5.15e-1	3.90e-1	1.19e+0	8.91e-1	1.33e+0	2.36e-1	1.23e+0	8.79e-1	1.19e+0	3.74e-1
15	1.03e-1	6.31e-2	1.01e+0	7.93e-1	3.37e-1	5.97e-2	1.01e+0	6.38e-1	2.33e-1	7.31e-2
20	8.64e-3	8.64e-3	3.48e-1	6.89e-1	2.12e-1	7.87e-3	2.73e-1	6.19e-1	1.92e-1	6.54e-3
25	7.39e-4	9.63e-3	8.94e-2	6.06e-1	2.82e-1	8.76e-3	7.89e-2	6.01e-1	1.83e-1	8.73e-3
30	1.95e-4	1.48e-4	3.69e-2	5.37e-1	3.16e-1	1.79e-4	3.24e-2	5.15e-1	1.32e-2	1.38e-4
35	7.02e-5	3.00e-5	2.78e-2	4.51e-2	3.20e-3	4.53e-5	2.13e-2	4.45e-1	9.32e-3	3.10e-5
40	7.50e-5	2.35e-5	1.15e-2	3.90e-1	3.01e-3	3.45e-5	1.12e-2	2.34e-1	4.30e-3	2.23e-5
45	5.84e-5	2.03e-5	9.41e-3	3.52e-3	2.78e-4	2.14e-5	9.39e-3	2.14e-1	3.23e-3	1.53e-5
50	6.08e-5	1.80e-5	9.68e-3	3.34e-1	2.68e-4	2.18e-6	9.57e-3	1.11e-4	2.68e-3	1.40e-5

However, it is essential to note that for black box optimization problems, we cannot determine the optimal solution in advance.

Further research is needed to in dept understand the population dynamics of SCA, opening opportunities for future investigations. A significant direction for future work lies in developing operators that can effectively prevent bias in algorithms. However, it should be noted that the signature test utilized in this study is limited to two dimensions, thereby constraining its ability to detect bias in higher dimensions. This underscores the importance of conducting additional research to explore alternative techniques. We anticipate that this analysis will provide valuable insights to the research community and make contributions to the advancement of metaheuristics.

Acknowledgements This research is carried out during during the internship 'Prof. Ravi Ravindran and Operational Reserach Socity of India - Karnataka Internship Award 2022'. The authors would like to express their gratitude to the anonymous reviewers for their invaluable feedback and insightful comments, which significantly contributed to improving the overall quality of the paper.

References

1. Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. Artificial Intelligence Review, 1-71.
2. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of ICNN'95-International Conference on Neural Networks, 4, 1942-1948). IEEE.
3. Holland, J. H. (1992). Genetic algorithms. Scientific American, 267(1), 66-73.
4. Van Laarhoven, P. J., Aarts, E. H., van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing (pp. 7-15). Springer Netherlands.
5. Storn, R., & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 11(4), 341.
6. Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. IEEE Computational Intelligence Magazine, 1(4), 28-39.
7. Glover, F., & Laguna, M. (1998). Tabu search (pp. 2093-2229). Springer US.
8. Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. Advances in Engineering Software, 69, 46-61.
9. Karaboga, D. (2010). Artificial bee colony algorithm. Scholarpedia, 5(3), 6915.
10. Yang, X. S., & Deb, S. (2010). Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation, 1(4), 330-343.
11. Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. Simulation, 76(2), 60-68.
12. Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1), 67-82.
13. Schmitt, L. M. (2001). Theory of genetic algorithms. Theoretical Computer Science, 259(1-2), 1-61.
14. Van den Bergh, F., & Engelbrecht, A. P. (2006). A study of particle swarm optimization particle trajectories. Information Sciences, 176(8), 937-971.

15. Yang, X. S. (2010). Nature-inspired metaheuristic algorithms. Luniver Press.
16. Yang, X. S. (2011). Metaheuristic optimization: algorithm analysis and open problems. In Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings 10 (pp. 21-32). Springer Berlin Heidelberg.
17. Yang, X. S., & Hossein Gandomi, A. (2012). Bat algorithm: a novel approach for global engineering optimization. Engineering Computations, 29(5), 464-483.
18. Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. Knowledge-Based Systems, 89, 228-249.
19. Eiben, A. E., & Smith, J. E. (2015). Introduction to evolutionary computing. Springer-Verlag Berlin Heidelberg.
20. Črepinšek, M., Liu, S. H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys (CSUR), 45(3), 1-33.
21. Morales-Castañeda, B., Zaldivar, D., Cuevas, E., Fausto, F., & Rodríguez, A. (2020). A better balance in metaheuristic algorithms: Does it exist?. Swarm and Evolutionary Computation, 54, 100671.
22. Kononova, A. V., Corne, D. W., De Wilde, P., Shneer, V., & Caraffini, F. (2015). Structural bias in population-based algorithms. Information Sciences, 298, 468-490.
23. Clerc, M. (2015). Biases and Signatures. Guided Randomness in Optimization, 139-145.
24. Mirjalili, S. (2016). SCA: a sine cosine algorithm for solving optimization problems. Knowledge-Based Systems, 96, 120-133.

Network flow models for days off scheduling

G. Tlig[1] and F. Jarray[2]

[1] Ecole Supérieure d'Electronique de l'Ouest, Paris, France
ghassen.tlig@eseo.fr
[2] LIMTIC Laboratory, UTM University, Tunis, Tunisia
Higher institute of computer science of Medenine, Gabes University, Tunisia
fjarray@gmail.com

Abstract. This paper studies the days off scheduling problem when the demand for staffing may differ from day to another and when the total load is fixed in advance for each employee. The scheduling problem is then to assign on-days and days-off to employees with different objectives: (1) exactly met the demand and the offer requirement (2) satisfy as best as possible the requirements. For each one, we propose a polynomial time algorithm based on network flow to construct a feasible scheduling.
Keyword: Workforce Scheduling, Maximum Flow, Polynomial Time Algorithm.

1 Introduction

The days-off scheduling problem basically involves determining the days-off and the on-days for each staff member over the planning horizon under various constraints. It is a large area of research with both theoretical and practical aspects, especially for organizations operating seven days a week or 24 hours a day such as hospitals. Days off scheduling can serve as a framework for addressing various real-life tasks [12, 13].

Different models and approaches are proposed in literature to schedule days-off. The earliest approaches are concerned with minimizing the size of full-time staff to cover the staffing demand of a company under several operating constraints [3], [4], [17]. Zolfaghari et al.[20] develop a genetic algorithm for the retail staff scheduling problem. Shahnazari et al. [15, 16] proposed a fuzzy multi-objective mathematical model for scheduling multi-skilled manpower with vagueness on target values of employers' objectives and employees' preferences.

Recently, has been a particular focus on scheduling including consecutive days off have been studied too. Several special cases or general problems can be handled through heuristics based on network and integer programming. [7, 9] proposed a mixed integer programming formulation for compressed work scheduling problem where each employee works 4 days and takes 3 days off per week. Costa et al.[6], Jarray[11, 10] present decomposition approaches for workforce scheduling while meeting the labor demands and a prespecified number of workdays per employee over the planning horizon. Veldhoven et al. [19] propose a two-phase integer programming for the personnel scheduling problem. Maenhout and Vanhoucke [14] propose a column generation approach to integrate the scheduling of project and personnel staffing. The reader is referred to [2, 18] for surveys days- and on-days scheduling.

Our main contribution to solve the days-off scheduling problem is the consideration of acyclic demand. Moreover, we take into account the number of worked days of each employee over the planning horizon as well as the daily staffing demand. The problem is to allocate days off to employees in order to cover the staffing demand on each day and the total workload of each employee under various evaluation criteria. As in many studies, we suppose that all the employees have the same professional qualifications (see [1], [8]) such as call center scheduling.

Let a company with daily demand of staffing to be satisfied by m employees $e_i,\ i = 1, \ldots, m$, over a planning horizon of n days $d_j,\ j = 1, \ldots, n$. For a weekly planning, we have $n = 7$. Each day can be either a working day (on-day) or a rest day (day-off).

We denote h_i the load (the number of working days) over the planning horizon for employee $e_i, i = 1, \ldots, m$, and v_j the demand (number of employees required) of day $d_j,\ j = 1, \ldots, n$. We suppose that the load of an employee is fixed in advance by an employment contract. We call an additional day or extra day, a day worked beyond the normal load. The vectors $H = (h_1, \ldots, h_m)$ and $V = (v_1, \ldots, v_n)$ are respectively called the labor demand and the load offer vectors.

The problems considered in this paper seek to assign on-days to employees while satisfying staffing requirement, i.e. v_j employees on work day d_j and insuring that each employee e_i works h_i days over the planning horizon.

The paper is organized as follows. In the next section, we examine the basic problem policy where we exactly satisfy the requirements d_j and h_i. In section 3, we consider the flexible offer problem where we exactly satisfy the daily demand and satisfy as best as possible the employees requirement. In section 4, we consider the flexible demand problem were we exactly satisfy the load offer and satisfy as best as possible the daily demand. In section 5, we consider the general flexible problem where we satisfy as best as possible both requirements.

2 The basic problem: $P1$

We are going to develop an algorithm to solve the basic problem. For a solution to exist we must have $\sum_{i=1}^{m} h_i = \sum_{j=1}^{n} v_j$.

We will model the basic problem by a max-flow problem in a bipartite graph $G(R, C, E)$ where $R = \{r_i,\ i = 1, \ldots, m\}$ represents the employees and $C = \{c_j,\ j = 1, \ldots, n\}$ represents the days of the planning horizon (see Figure 2). We add two nodes to $G(R, C, E)$: a source s and a sink t. The source s has an edge of capacity h_i to every vertex r_i with which is the load offer of employee i. By symmetry, the destination t has an edge of capacity v_j from every vertex c_j which is the demand of day j. There is a unit capacity edge (r_i, c_j) between every pair of nodes r_i and c_j which corresponds to the schedule to reconstruct. So the basic problem is equivalent to the max-flow problem in G. In particular, the basic problem has a solution if and only if the maximum flow from the source to the sink is of value $\sum_{i=1}^{m} h_i = \sum_{j=1}^{n} v_j$. Since the capacities are integers, there exists an optimal integer flow. A flow has a value 1 on an edge (r_i, c_j) means that employee e_i works on day d_j.

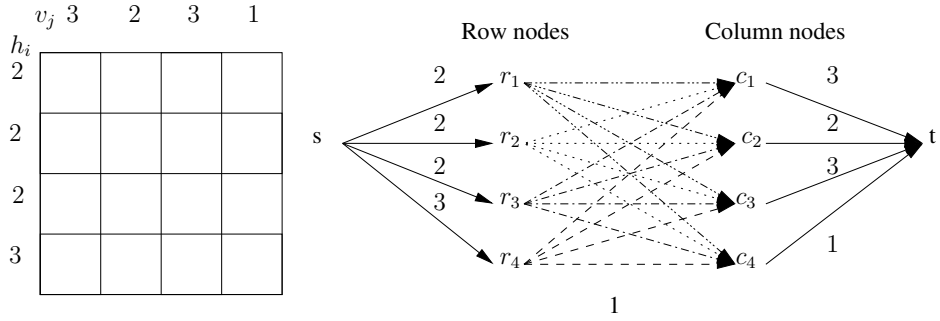


Fig. 1. Modelling the basic problem through a max flow problem. The load offer is $H = (2, 2, 2, 3)$ and the labour demand is $V = (3, 2, 3, 1)$

2.1 An example

Consider a company with four employees and a planning horizon of seven days ($m = 4$ and $n = 3$). The number of worked days of each employee is given by $H = (5, 5, 5, 5)$ and the daily request of staffing is given by $V = (4, 4, 3, 3, 3, 2, 1)$.

The max-flow associated problem is depicted in Figure 2. On each arc of this graph, we write also the value of the maximal flow. So we deduce a solution to the scheduling problem (see Figure 3).

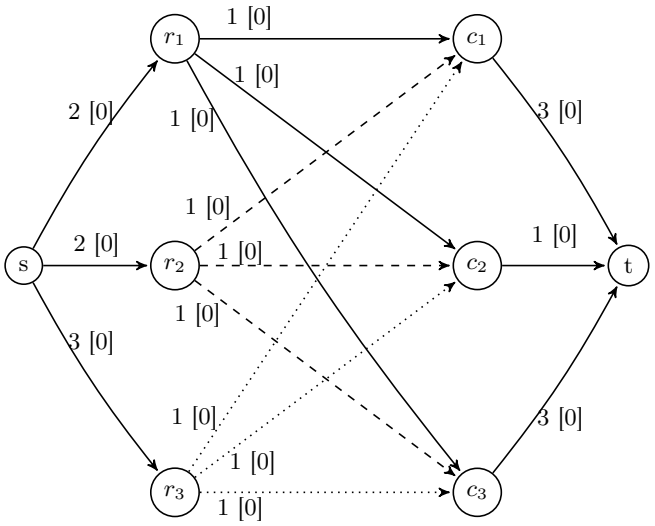


Fig. 2. Modelling the basic problem through a max flow problem. The load offer is $(2, 2, 3)$ and the labour demand is $(3, 1, 3)$

Let S be a schedule. We denote by h_i^s the effective number of days worked by employee i and by v_j^s the effective number of employees working during day j. The vector H^s and V^s are called the vector of the effective demand and load. We note that $\|V^s\| = \|H^s\|$. The surplus of employees on day j is $max(0, v_j^s - v_j)$. The uncovered demand on day j is $max(0, v_j - v_j^s)$.

3 The flexible offer problem: *P*2

If the basic problem admits a solution, we are done, otherwise, we relax the load offer constraint. We seek to exactly satisfy the daily demand while minimizing the cost of the addition on-days or extra days. In this problem overtime may occurs to satisfy the demand.

We suppose that the overtime cost is an increasing function. Here we limit ourselves to maximum two extra days per employee. For each employee e_i, we denote c_i^1 and c_i^2 the cost of respectively the first and the second extra day with $c_i^2 > c_i^1$. We will propose a min-cost max-flow model in a bipartite network $G2$ (see Figure 3) to solve this problem. We build a new graph $G2$ in the following way. We take a copy of G and we set to zero the cost of each edge.

We replace every edge (s, r_i) G by three parallel edges (see Figure 3). The first edge has a capacity of h_i and a zero cost. The second edge has a unit capacity and a cost equal to c_i^1 witch denotes the cost of the first extra working day. The third edge has a uni capacity and a cost equal to c_i^2 witch denotes the cost of the second extra working day.

Within the framework of our model it is easy to very that in the min cost max flow optimal solution, the flow value on the second arc (s, r_i) is non null only if the first arc (s, r_i) is saturated and that the flow value on the third arc (s, r_i) is non null only if the second arc (s, r_i) is saturated.

We note that our model can be generalized to consider an arbitrary maximum number of extra days for each employee.

Definition 1 *Given an n-dimensional vector X. The 1-norm of X (or the norm for short), $\|X\|$, is defined by $\|X\| = \sum_{i=1}^{n} |X_i|$.*

Definition 2 *Given an n-dimensional vector X. The sum of the positive values is noted by $|X|^+ = \sum_{i=1}^{n} max(0, X_i)$.*

Proposition 1 *Given H and H^s two m-dimensional vectors and V and V^s two n-dimensional vectors. We have: $\|H^s - H\| = 2|H^s - H|^+ + \|H^s\| - \|H\|$ and $\|V^s - V\| = 2|V^s - V|^+ + \|V^s\| - \|V\|$.*

Proof 1 $\|H^s - H\| = |H^s - H|^+ + |H - H^s|^+ \quad (1)$.
For each employee i, we have $h_i^s = h_i + max(h_i^s - h_i, 0) - max(h_i - h_i^s, 0)$.
So we get $\|H^s\| = \|H\| + |H^s - H|^+ - |H - H^s|^+ \quad (2)$.
By combining (1) *and* (2), *we get* $\|H^s - H\| = 2|H^s - H|^+ + \|H^s\| - \|H\|$.
Similarly, we prove that $\|V^s - V\| = 2|V^s - V|^+ + \|V^s\| - \|V\|$

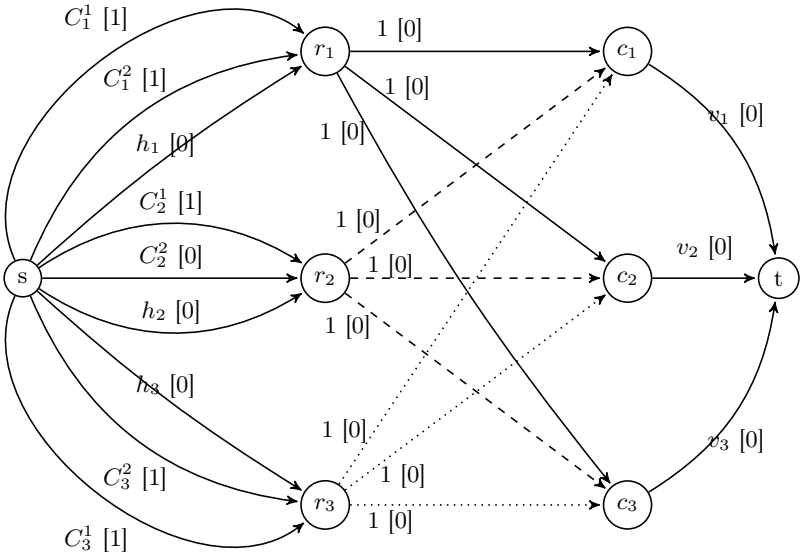


Fig. 3. Modelling the general flexible offer problem through a max flow problem

4 The flexible demand problem: $P3$

A second alternative to reconstruct a schedule when the basic problem does not have a solution is to relax the demand constraint. We seek to exactly satisfy the load offer while minimizing the uncovered demand ($|V - V^s|^+$) over the planning horizon. We have $\|V^s\| = \|H\|$ since we exactly satisfy the load offer of each employee.

From proposition 1, $|V - V^s|^+ = \|V\| - \|H\| + |V^s - V|^+$ because $\|H\| = \|V\| + |V^s - V|^+ - |V - V^s|^+$. Since $\|V\|$ and $\|H\|$ are constant, we deduce, that minimizing the sum of the uncovered demand is equivalent to minimizing the total surplus of employees. Thus $P3$ can also be modeled by a min-cost max-flow problem in a bipartite network $G3$ (see Figure 4).

We build a new graph $G3$ in the following way. We take a copy of G and we set to zero the cost of each edge. We replace every edge (c_j, t) by two parallel edges (see Figure 4). The first arc has a capacity of v_j and a zero cost. The second arc has an infinite capacity and a unit cost witch denotes the surplus of employees on day d_j. We note that in the maximal flow, all the arc outgoing from the source are saturated because of the unbounded arc incoming to the sink. Thus we verify that $\|V^s\| = \|H\|$.

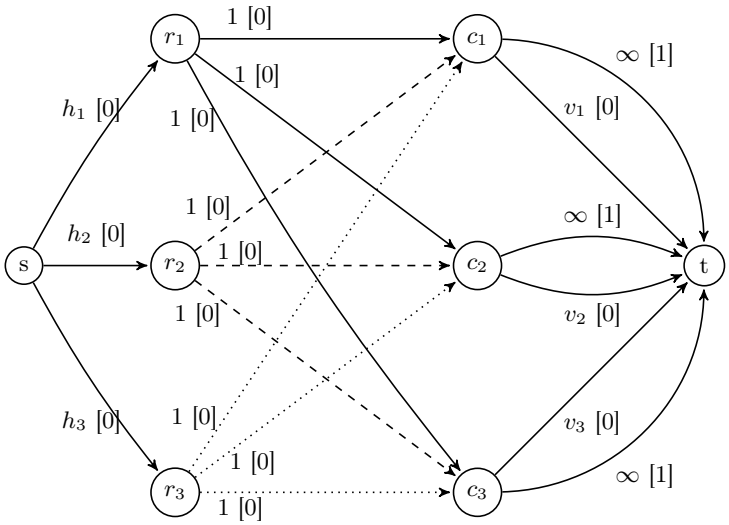


Fig. 4. Modelling the flexible demand problem through a max flow problem

5 The general flexible requirement problem: $P4$

Here we consider a more generalization of the basic problem under flexible daily demand and load offer. We seek to satisfy as best as possible the requirements instead of exactly satisfying them. The objective of this problem is to minimize the slack between the vectors H and H^s; and the vectors V and V^s.

We will construct a schedule S with L ($\|H^s\| = \|V^s\| = L$) working days while minimizing $\|H - H^s\| + \|V - V^s\|$. We note that if $\sum_{i=1}^{m} h_i = \sum_{j=1}^{n} v_j$, then we choose $L = \sum_{j=1}^{n} v_j$, otherwise we may set $L = \frac{\sum_{i=1}^{m} h_i + \sum_{j=1}^{n} v_j}{2}$, i.e the mean of the total demand and the total load.

By proposition1, we state that the flexible problem is equivalent to finding a schedule with L working days minimizing the sum of the positive gaps on the demand and the requirement.

Again this problem can be modeled by a min-cost max-flow problem in a bipartite network $G4$ (see Figure 5). The graph $G4$ is built from G in the as follows. We take a copy of G and we set to zero the cost of each edge. Each edge (s, r_i) is replaced by two parallel edge (see Figure 5). The first edge has a capacity of h_i and a zero cost. The second edge has an infinite capacity and unit cost such that an extra working days denotes a non zero flow on this edge. Similarly each edge (c_j, t) is replaced by two parallel edge (see Figure 5). The first edge has a capacity of v_j and a zero cost. The second edge has an infinite capacity and a unit cost with denotes an extra working employee on this day. To ensure L working days, we impose a constraint of L units of flow on vertices s and t.

6 Conclusion

In this paper, we have studied a general days off scheduling problem where the number of worked days for each employee and the staffing demand are taken into account. We have proposed polynomial time algorithms based on network flow to solve the problems under three alternatives of flexible requirements. It would be also interesting to study our basic problem under other days-off allocation strategies.

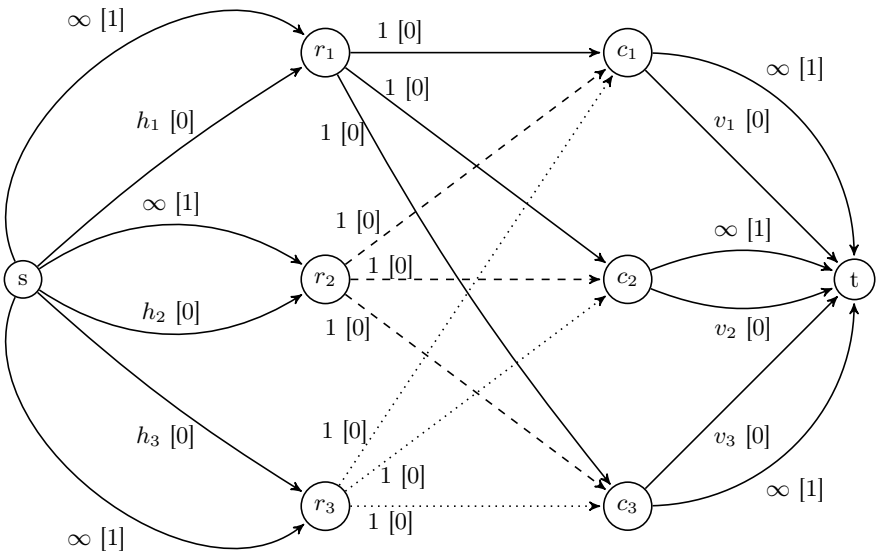


Fig. 5. Modelling the general flexible requirement problem through a max flow problem

References

1. Alfares, H.: Four day workweek scheduling with two or three consecutive days off. *Journal of Mathematical Modeling and Algorithms, 2(1), 67 -80 (2003).*
2. Alfares, H.: Survey, Categorization, and Comparison of Recent Tour Scheduling Literature. *Annals of Operations Research, 127, 147-177 (2004).*
3. Baker, K.R.: Scheduling a Full-Time Workforce to Meet Cyclic Staffing Requirements. *Management Science, 20(12), 1561-1568 (1974).*
4. Baker, K.R. and Magazine, M.T.: Workforce Scheduling with Cyclic Demands and Days-off Constraints. *Management Science, 24, 161-167 (1977).*
5. Burns, R.N. and Carter, M.W.: Workforce Size and Single Shift Schedules with Variable Demands. *Management Sciences*, 31(5), 599-607 (1985).
6. Costa, M.C. Jarray, F. and Picouleau, C.: An acyclic days-off scheduling problem. *4OR: Quarterly Journal of Operations Research, 4, 73-85 (2006).*
7. Elshafei, M. and Alfares, H.: A dynamic programming algorithm for daysoff scheduling with sequence dependent labor costs. *Journal of Scheduling, 11, 85-93 (2008).*
8. Emmons, H. and Fuh, D.: Sizing and scheduling a full-time and part-time workforce with off-day and off weekend constraints. *Annals of operations research, 70, 473-492 (1997).*
9. Hung, R.: Single-shift off-day scheduling of a hierarchical workforce with variable demands. *European Journal of Operational Research, 78, 49-57 (1994).*
10. Jarray, F.: A 4-day or a 3-day workweeks scheduling problem with a given workforce size . *APJOR: Asian-Pacific Journal of Operational Research, 26(5), 1-12 (2009).*
11. Jarray, F., Solving problems of discrete tomography: application in workforce scheduling. *4OR: Quarterly Journal of Operations Research* 3, 337-340 (2005).
12. Jarray, F., A lagrangean-based heuristics for the target covering problem in wireless sensor network.*Appl Math Model* 37:6780-6785
13. S. Brunetti, M.C. Costa, A. Frosini, F. Jarray, C. Picouleau, Reconstruction of binary matrices under adjacency constraints, *in: G.T. Herman, A. Kuba (Eds.), Advances in Discrete Tomography and its Applications, BirkhÃ¤user, Boston,* 2007, pp. 125-150.
14. Maenhout, B. and Vanhoucke, M.: A resource type analysis of the integrated project scheduling and personnel staffing problem. *Annals of Operations Research volume 252, pages 407â433 (2017).*
15. Shahnazari-Shahrezaei, P. Tavakkoli-Moghaddam, R. and Kazemipoor, H.: Solving a new fuzzy multi-objective model for a multi-skilled manpower scheduling problem by particle swarm optimization and elite tabu search. *International Journal of Advanced Manufacturing Technology, 64, 1517-1540 (2013).*

16. Shahnazari, S.H., and Tavakkoli-Moghaddam, R.: Solving a multi-objective multi-skilled manpower scheduling model by a fuzzy goal programming approach. *Applied Mathematical Modeling, 37, 5424-5443 (2013).*
17. Tiberwala, R. Philippe, D. and Browne, J.: Optimal scheduling of two consecutive idle periods. *Management science, 19(1), 71-75 (1972).*
18. Van den Bergh, J. Belien, J. De Bruecker, P. Demeulemeester, E. and De Boeck, L.: Personnel scheduling: A literature review. *European Journal of Operational Research, 226, 367-385 (2013).*
19. Sophie Van Veldhoven, S. Post, G. Van der Veen, E. and Curtois, T.: An assessment of a days off decomposition approach to personnel scheduling. *Annals of Operations Research, volume 239, pages 207-223 (2016).*
20. Zolfaghari, S. Quan, V. El-Bouri, A. Khashayardoust, M.: Application of a Genetic Algorithm to staff scheduling in retail sector. *International journal of Industrial and Systems Engineering, 5(1), 20-47 (2010).*

Mixed-variable surrogate-based optimization using probability features for categorical variables

C. Beauthier[1], S. Pool Marquez[2], C. Sainvitu[1], A. Sartenaer[2]

[1] Cenaero
Rue des frères Wright, 29
B-6041 Gosselies, Belgium
[2] University of Namur, naXys Institute
Rue de Bruxelles, 61
B-5000 Namur, Belgium
charlotte.beauthier@cenaero.be

1 Introduction

A globally effective approach to high-fidelity optimization problems based on computationally expensive analysis lies in the exploitation of surrogate models, also known as metamodels or response surface models. They act as cheap-to-evaluate alternatives to the original high-fidelity models reducing the computational cost, while still providing improved designs. The underlying principle of Surrogate-Based Optimization (SBO) consists in accelerating the optimization process by essentially exploiting surrogates for the objective and constraint evaluations, with a minimal number of function calls to the high-fidelity models for keeping the computational time within affordable limits [Forrester and Keane, 2009]. In more detail, the SBO design cycle consists in several major elements. First of all, a Design of Experiments (DoE) is defined using an a priori space filling technique Latinized CVT (`LCVT`), see [Gunzburger et al., 2007]. After the evaluation of the DoE by the high-fidelity models, the surrogate models are trained based on the available information in the database, and an evolutionary optimization step is launched to generate new best candidates for the given optimization problem. These candidates are then evaluated by the high-fidelity models and their accurate performance is checked afterwards. Finally, the new candidates are added to the database and the online SBO is repeated until a satisfactory performance is achieved.

In the literature, the vast majority of described SBO is limited to a purely continuous design space, i.e., an optimization context where each design variable is a continuous variable. However for real engineering problems, design variables can have different natures:

- *continuous* variables, defined over an interval in $\mathbb{R}$;
- *integer* variables, defined over an interval in $\mathbb{Z}$;
- *discrete* variables, only defined for a finite set of elements in $\mathbb{R}$ or $\mathbb{Z}$ (non-exclusive);
- *categorical* variables, defined for a set of strings where the elements are unordered (nominal).

One of the main challenges is the handling of the non-continuous variables within the whole optimization process. We present here a SBO framework handling mixed variables with a focus on the management of the categorical ones. More precisely, we propose an adaptation of the strategy that [Wang et al., 2021] have developed to handle categorical variables within a Particle Swarm Optimization (`PSO`) algorithm, to an Evolutionary Algorithm (`EA`) as inner optimizer of the SBO loop. Our strategy is implemented in the integrated optimization platform Minamo, Cenaero's in-house design space exploration and multi-disciplinary optimization platform, see [Sainvitu et al., 2010] for more information. The next section presents our proposed strategy to adapt the genetic operators (mainly mutation and crossover operators) inside the `EA` in order to better deal with categorical variables.

2 Proposed strategy for the `EA` of Minamo

By default, the `EA` of Minamo manages categorical variables within the mutation and crossover operators by using a random selection. For the mutation, the new value is chosen randomly among all the possible ones for the concerned variable while for the crossover, the new value is taken

randomly from the ones of the parents. In our adaptation of [Wang et al., 2021], a probability is associated to each possible value of each categorical variable. The initialization of these probabilities is done by defining an equivalent amount for each possible value of each categorical variable. If we consider a categorical variable x_j with n_j possible values, the associated probabilities will be initialized to $P_{j,k}(0) = 1/n_j, 1 \leq k \leq n_j$, where $P_{j,k}$ represents the probability associated to the k-th possible value of the j-th categorical variable. During the `EA`, the probabilities are updated using the following rule:

$$P_{j,k}(t+1) = \alpha P_{j,k}(t) + (1-\alpha)\frac{C_{j,k}(t)}{n(t)}, \tag{1}$$

where t is the current iteration of the algorithm. The quantity $C_{j,k}(t)$ is the number of individuals in the population whose j-th categorical variable at iteration t has the k-th value, while $n(t)$ is the current population size. Rule (1) is directly inspired from [Wang et al., 2021], where the real value $\alpha \in [0, 1]$ is used as a trade-off parameter between the historical value of the probability and the actual state of the search.

In our work, we consider a different tuning of the parameter α from the ones proposed by [Wang et al., 2021], which use either a Cauchy or a Gaussian distribution. We use a decreasing sigmoid function where the value of α starts close to one and ends close to zero. Doing so, the impact of the historical value of the probability is stronger at the beginning of the `EA` and at the end, the current state of the search has more impact for the new values of the probabilities. During the `EA`, these probabilities are used to handle categorical variables for the genetic operators. The mutation of such variables is done by making a selection of their possible values according to their probabilities. More precisely, a value will have a better chance of being chosen if its associated probability is high. The crossover operator is not referenced in [Wang et al., 2021] because it is not used in `PSO` algorithms. We propose to use the probabilities to build a crossover operator able to handle categorical variables. More precisely, let us consider two parents with their value and associated probability, $(v_1, prob_1)$ for parent 1 and $(v_2, prob_2)$ for parent 2. We define $\nu = \frac{\min\{prob_1, prob_2\}}{\max\{prob_1, prob_2\}}$, and take a random value u, uniformly distributed in $[0, 1]$. If this value is greater than ν, then the offspring value is set to the one corresponding to the greatest probability. Otherwise, the offspring value is set to the one corresponding to the smallest probability. With this crossover strategy, the smaller the minimum probability compared to the maximum probability, the more the value associated with the better probability will be promoted. If the probabilities are equal, the offspring value is set to the one corresponding to the parent with the best objective function value.

3 Numerical results for the SBO process of MINAMO

We present the results obtained with the modified `EA` in the SBO process of MINAMO where the sigmoid tuning for the parameter α and the adapted mutation and crossover strategies we mentioned in the previous section have been used. This version is called `SBO-probaCat` versus the default version of MINAMO, called `SBO-default`. Results obtained for a pure `EA` optimization process without surrogate model has been presented in [Beauthier et al., 2023].

To compare both versions, a set of 7 test problems has been used. All of them involve, at least, continuous and categorical variables and some of them also consider discrete and/or integer variables. Furthermore, this set contains constrained and unconstrained problems. These test problems are `car side impact design`, `reinforced concrete beam design` and `welded beam design` coming from [Gandomi et al., 2011]. We also consider `pressure vessel design`, `speed reducer design` and `tension/compression spring design` from [Cagnina et al., 2008] and `piston` from [Zhang et al., 2018]. In order to achieve a fair comparison, the two versions are executed in the same conditions : 100 independent runs, started from an initial database (LCVT) of size $n+1$, n being the number of design variables and using the auto-adapted radial basis function networks (TunedRBFN) implemented in MINAMO as surrogate model (see [Sainvitu et al., 2010]).

For the sake of analyzing the global performance of the tested versions, we use the *performance profiles* introduced by [Dolan and Moré, 2002]. The goal of this tool is to display the amount of solved problems by each version according to a value τ. This value represents the proportion of the budget used by a version to solve a problem, compared to the minimal budget required by all versions to solve the same problem. In this work, the budget is defined as the number of function evaluations and each run of each test problem is considered as a problem to solve.

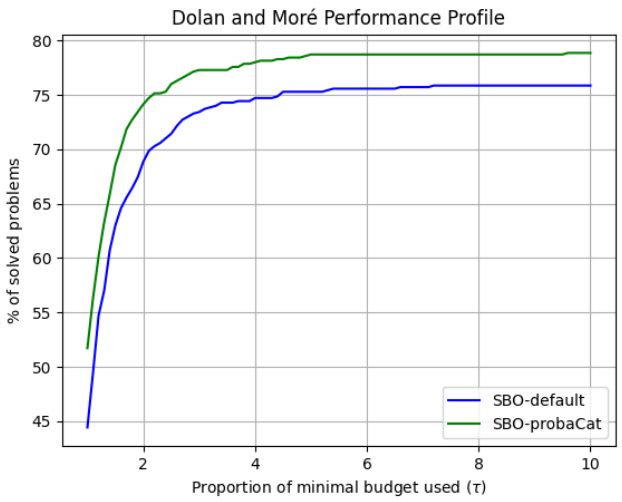


Fig. 1. Performance profile for an alternative version of the `SBO` used in MINAMO on a benchmark of 7 test problems. A zoom has been applied to the y-axis on the portion where the performance profile values start and end.

The performance profiles for the `SBO-default` and `SBO-probaCat` versions are shown in Figure 1. The abscissa axis of this graphic represents the proportion of the minimal budget used and the ordinate axis shows the percentage of successfully solved problems. We can observe that the modified version for mixed variables outperforms the default one. Indeed, for any proportion of the minimal budget, the `SBO-default` version is never able to solve more test problems than the `SBO-probaCat` version. However, the scale used for the y-axis increases the differences between the two curves. It would be interesting to use statistical tests to confirm that the observed differences are sufficiently relevant. Encouraged by the promising results for this first implementation of our adaptation, we would like to pursue our investigation of using probabilities features in the `EA` and also enrich the set of problems used for the benchmark. Moreover, we would like to add the proposed `PSO` algorithm by [Wang et al., 2021] in the benchmark in order to compare its results with those of the default and modified `EA`.

References

[Beauthier et al., 2023] Beauthier, C., Pool Marquez, S., Sainvitu, C., and Sartenaer, A. (May 3-5 2023). Impact of mixed-variable management by probability features in an evolutionary algorithm. *Proceedings of The International Conference in Optimization and Learning (OLA2023)*.

[Cagnina et al., 2008] Cagnina, L. C., Esquivel, S. C., and Coello, C. A. C. (2008). Solving engineering optimization problems with the simple constrained particle swarm optimizer. *Informatica (Slovenia)*, 32:319–326.

[Dolan and Moré, 2002] Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213.

[Forrester and Keane, 2009] Forrester, A. and Keane, A. (2009). Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1-3):50–79.

[Gandomi et al., 2011] Gandomi, A. H., Yang, X.-S., and Alavi, A. H. (2011). Mixed variable structural optimization using firefly algorithm. *Computers & Structures*, 89(23):2325–2336.

[Gunzburger et al., 2007] Gunzburger, M., Saka, Y., and Burkardt, J. (2007). Latinized, improved lhs, and cvt point sets in hypercubes. *International Journal of Numerical Analysis and Modeling*.

[Sainvitu et al., 2010] Sainvitu, C., Iliopoulou, V., and Lepot, I. (2010). Global Optimization with Expensive Functions - Sample Turbomachinery Design Application. In *Recent Advances in Optimization and its Applications in Engineering*, pages 499–509, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Wang et al., 2021] Wang, F., Zhang, H., and Zhou, A. (2021). A particle swarm optimization algorithm for mixed-variable optimization problems. *Swarm and Evolutionary Computation*, 60:100808.

[Zhang et al., 2018] Zhang, Y., Tao, S., Chen, W., and Apley, D. W. (2018). A latent variable approach to gaussian process modeling with qualitative and quantitative factors. *Technometrics*, 62:291 – 302.

Harnessing Collective Intelligence: Integrating Particle Swarm Optimization and Reinforcement Learning for Efficiently Solving Binarized Coverage Problems

Marcelo Becerra-Rozas[1 [0000−0003−0426−0144]], Broderick Crawford[1 [0000−0001−5500−0188]], Ricardo Soto[1 [0000−0002−5755−6929]], El-ghazali Talbi[2 [0000−0003−4549−1010]], and Giovanni Giachetti[3 [0000−0003−2809−5120]]

[1] Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile
{marcelo.becerra.r}@mail.pucv.cl
{broderick.crawford,ricardo.soto}@pucv.cl
[2] University of Lille & INRIA, Lille, France
el-ghazali.talbi@univ-lille.fr
[3] Universidad Andres Bello, Santiago, Chile
giovanni.giachetti@unab.cl

Abstract

This article proposes the utilization of a classical metaheuristic in our framework for selecting binarization schemes based on reinforcement learning. Our selector enables the binarization of continuous metaheuristic, allowing for its application in binary domains. In our previous works, reviewers commonly questioned why we didn't use more popular metaheuristics like PSO. Finally, we present the implementation and the results obtained, which demonstrate the successful performance of PSO, whether using QL or BQSA, in solving a coverage problem.

Keywords: **PSO · Q-Learning · Backward Q-Learning · Coverage Problem**

1 Introduction

In the literature, it has been conclusively demonstrated in recent years that metaheuristic optimization algorithms (MH) are highly effective in addressing a variety of optimization tasks. These applications span from robotics [20], power systems [12], to neural network training [3]. Some classical examples of such MH include Particle Swarm Optimization [22], Cuckoo Search [45], and Genetic Algorithm [16]. In the literature, population-based MH are more widely used than single-solution MH.

Metaheuristics (MH) are widely used algorithms for solving optimization problems. While most MH algorithms have been developed for continuous domains, there are techniques that can operate in both binary and continuous domains, such as genetic algorithms [31] and certain variations of Ant Colony

Optimization (ACO) [28]. However, these techniques have not been able to match the performance of continuous MH algorithms that utilize an operator capable of transforming continuous solutions into the binary space. Recent literature has shown a growing interest in the development of new binarization operators, drawing inspiration from genetic algorithms, including crossover techniques [1]. Furthermore, alternative approaches have been explored, such as machine learning-based techniques. These include unsupervised clustering methods, for instance, K-means [38] and DB-scan [13], reinforcement learning (RL) techniques like Q-Learning [10] and SARSA [26], as well as their combinations like Backward Q-Learning [6]. Additionally, inspirations from quantum computing [24] and logical operators [42] have been investigated. Moreover, classical approaches like the two-step method have been employed, involving the normalization of continuous values using a transfer function (step 1) and subsequent binarization using an approximation rule, resulting in binary values of 0 or 1 (step 2) [9]. It is crucial to continue exploring novel variations of binarization techniques, as their influence on the performance of MH algorithms has been well-documented [2].

This work proposes to binarize PSO using the developed framework called Binarization Scheme Selector (BSS) to solve 45 instances of the Set Covering Problem (SCP). BSS has the capability to adapt any continuous MH to the binary domain and is based on the classical two-step technique. In the first step, our intelligent operator carefully selects a suitable transfer function. This function is essential for normalizing the continuous values within the range of [0, 1] and preparing them for the subsequent stage. The proper choice of the transfer function is crucial to ensure accurate and efficient adaptation. In the second step, our intelligent operator chooses the optimal binarization criterion. This criterion determines how the normalized values will be transformed into the binary domain $\{0, 1\}$. Utilizing an intelligent and adaptive approach, our operator selects the best binarization criterion for each specific problem.

The paper is structured as follows: Section 2 presents the related work on metaheuristic techniques, binarization, hybridizations, and machine learning. In Section 3, we introduce the binarization using BSS. Finally, in Sections 4 and 5, we present the obtained results for problem solving and the respective conclusions.

2 Related Work

2.1 Metaheuristics

MH are widely used to solve optimization problems, employing strategies that go beyond local search. These techniques rely on two key components: exploration and exploitation. Exploration involves generating diverse solutions to explore the search space at a global level, while exploitation focuses on finding optimal solutions within a local region. Striking the right balance between these components is crucial to achieve global optimization. Selecting the best solutions ensures convergence towards the optimum, while diversification through randomization allows for the exploration of the entire search space, increasing solution diversity.

MH offer a significant advantage by generating near-optimal solutions in reduced computational time, unlike exact methods, and they have the capability to adapt to different problems compared to heuristic methods. These MH are generally designed for continuous domains, reflecting their development and classification. Various approaches exist for classifying MH, with popular options proposed by Heidari et al. [15], and Cuevas et al. [11], which categorize them into trajectory-based and population-based. Another relevant taxonomy presented in [19] delves into key components of these algorithms, including solution evaluation, parameters, encoding, initialization of agents or population, population management, operators, and local search. In this work, our focus is on swarm-based MH, specifically Particle Swarm Optimization (PSO). PSO, originally proposed by Kennedy and Eberhart in 1995 [22], simulates the movement of particles in a search space to find optimal solutions.

2.2 Particle Swarm Optimization

The original PSO is based on a swarm S consisting of n particles ($S = 1, 2, ..., n$). Each particle i in the swarm has its position vector $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,j}, ..., x_{i,m})$ and its velocity vector $v_i = (v_{i,1}, v_{i,2}, ..., v_{i,j}, ..., v_{i,m})$ in an m-dimensional continuous solution space. Initially, the particle positions and velocities are randomly assigned within predefined limits. At each iteration, the particles update both their position and velocity. The position of a particular particle depends solely on its velocity, meaning that in the $t - th$ iteration, the position of particle i is determined by:

$$x_{i,d}^{t+1} = x_{i,d}^{t} + v_{i,d}^{t+1} \tag{1}$$

$$v_{i,d}^{t+1} = w^{t} \cdot v_{i,d}^{t} + C_1 \cdot rand_1 \cdot (pbest_{i,d}^{t} - x_{i,d}^{t}) + C_2 \cdot rand_2 \cdot (gbest_{d}^{t} - x_{i,d}^{t}) \tag{2}$$

Where $v_{i,d}^{t}$ and $v_{i,d}^{t+1}$ represent the current velocity and the next velocity of the i-th particle in the d-th dimension. $x_{i,d}^{t}$ and $x_{i,d}^{t+1}$ denote the current position and the future position in the continuous search space. w is the inertia weight that balances exploration and exploitation, $C1$ and $C2$ are acceleration coefficients, $rand_1$ and $rand_2$ are two random numbers in the range of $[0, 1]$. Finally, $pbest_{i,d}^{t}$ and $gbest_{d}^{t}$ refer to the best position found by the i-th particle in the d-th dimension and the best position found by the swarm up to the current moment in the d-th dimension.

2.3 Binarization

Population-based MH algorithms have often been developed and utilized for optimization problems in continuous domains. However, when dealing with binary optimization problems, a specific adaptation to the context of 0 and 1 values becomes necessary. Consequently, over the decades, several authors have proposed various approaches to adapt MH algorithms for effective application in the binary domain [5].

2.4 Discrete and Binary Particle Swarm Optimization

The authors of the original PSO in 1997 [23] published a binary adaptation using the sigmoid function, which allows PSO to generate values in the range [0,1] (Eq. (1)). In order to have the next particle position in the binary search space ($xb_{i,d}^{t+1}$), the rule (4) is applied.

$$Sigmoid\ \ TF(v_{i,d}^{t+1}) = \frac{1}{1 + e^{-v_{i,d}^{t+1}}} \tag{3}$$

$$xb_{i,d}^{t+1} = \begin{cases} 1 & \text{if } rand < Sigmoid\ \ TF(v_{i,d}^{t+1}) \\ 0 & \text{if } rand \geq Sigmoid\ \ TF(v_{i,d}^{t+1}) \end{cases} \tag{4}$$

Where $rand$ is a uniform number between $U \sim (0, 1)$. The sigmoid function applied in the binary version of PSO is classified as a type-S transfer function, while the rule used is referred to as a standard binarization technique. Transfer functions play a crucial role in mapping the continuous search space to a binary one. Similarly to Kennedy and Eberhart, many other authors have proposed their own methods of discretization and binarization. In [7], the authors introduce some V-type transfer functions. Wang et al. [39], on the other hand, propose a probability-based BPSO using a linear transfer function. Engelbrecht and Pampara [34] proposed a binary differential evolution. Islam et al. [18] introduced time-varying S-shaped and V-shaped transfer functions, while Mirjalili et al. proposed several new functions, including S-type [30], V-type [30], and U-type [32]. Guo et al. [14] presented Z-type transfer functions, and other applications can be found in [35].

2.5 Hybrid-Metaheuristics

In the literature, a hybrid MH has been described as the fusion of a metaheuristic algorithm and a different learning algorithm [37, 21], such as combining MH with machine learning or RL techniques [36]. Within the realm of RL techniques, there are two groups: RL supporting MH and MH supporting RL. In the first group, Garcia et al. [13] discuss two lines of research. The first line involves integrating RL techniques as replacements for operators, such as population management, local searches, or parameter tuning. The second line uses RL as a selector for a set of MH, enabling the choice of the most suitable one depending on the problem at hand.

When RL is used as a selector, we can further divide this category into three groups. The first group is algorithm selection, where a choice is made among a set of techniques to address the problem in order to improve performance on a set of similar instances [27]. Secondly, there are hyperheuristic strategies, which utilize MH to cover a set of problems. Lastly, cooperative strategies combine algorithms sequentially with the aim of enhancing solution robustness.

2.6 Reinforcement-Learning Techniques

The main objective of these techniques is to enable the agent to learn a policy that maximizes long-term rewards by interacting with the environment and relying on its experience. The value function provides information about the utility of actions taken from a state, i.e., how favorable the reward is. The expected reward function, denoted as R_t, is composed of both the current obtained rewards and discounted future rewards. The future reward for time instant t is calculated using Equation (5).

$$R_t = \sum_{j=0}^{n} \gamma^j \cdot r_{t+j+1} \tag{5}$$

Where the total return is calculated by summing the future rewards weighted by a discount factor (γ) over a time range determined by the index (j) ranging from 0 to n. Each reward (r_{t+j+1}) represents the reward received at a step forward in time from time t.

2.7 Q-Learning

Q-learning, widely recognized in the field of RL [41], is an algorithm that operates independently of the environment in which it is executed. The agent selects the action a believed to generate the highest value. When the agent performs an action in the environment, a perturbation occurs. The impact of this perturbation is evaluated based on the obtained reward or punishment (r), determining the next state s_{t+1} of the environment. Eq. (6) provides a mathematical representation of how the value is updated in Q-learning.

$$Q_{new}(s_t, a_t) = (1 - \alpha) \cdot Q_{old}(s_t, a_t) + \alpha \cdot [r_n + \gamma \cdot maxQ(s_{t+1}, a_{t+1})] \tag{6}$$

In the presented equation, $Q_{new}(s_t, a_t)$ represents the reward of the action taken in state s_t, r_n is the reward received when performing action a_t, and $maxQ(s_{t+1}, a_{t+1})$ is the maximum action value for the next state. The values of α and γ are important factors in this equation. α is the learning factor and must satisfy $0 < \alpha \leq 1$. On the other hand, γ is the discount factor and must satisfy $0 \leq \gamma \leq 1$. As γ approaches 0, more importance is given to immediate reward, while as it approaches 1, greater emphasis is placed on future reward relative to immediate reward.

2.8 Backward Q-Learning

Backward Q-Learning is another RL technique, proposed by Wang et al. [40], that introduces a backward update in the learning process. Unlike conventional Q-Learning, Backward Q-Learning takes into account past rewards when updating the Q-values, allowing for more comprehensive feedback and improving the estimation of action-state values.

The retrospective update involves updating not only for the actions taken in the current state but also for the previous actions that led to the current state. This backward update allows the agent to consider the past consequences of its actions and adjust the Q-values accordingly. In this process, we define a goal state s_0 and a terminal state s_n. When the agent reaches the goal state, the technique requires the agent to update the Q-function N times from the initial state to the terminal state.

$$Q_{new}(s_t^i, a_t^i) \longleftarrow (1-\alpha) \cdot Q(s_t^i, a_t^i) + \alpha \cdot [r_{t+1}^i + \gamma \cdot max_a Q(s_{t+1}^i, a_{t+1}^i)] \quad (7)$$

where $i = 1, 2, \ldots, N$ is the number of times the Q-function will be updated in the current episode. In turn, the agent simultaneously records the four events in M^i, represented mathematically with the Equation (8):

$$M^i \longleftarrow s_t^i, a_t^i, r_t^i, s_{t+1}^i \quad (8)$$

Once the agent reaches the terminal state, the agent will backward update the Q-function based on the information obtained from Equation (8) as follows (Equation (9)).

$$Q_{new}(s_t^j, a_t^j) \longleftarrow (1-\alpha) \cdot Q(s_t^j, a_t^j) + \alpha_b \cdot [r_{t+1}^j + \gamma_b \cdot maxQ(s_{t+1}^j, a_{t+1}^j)] \quad (9)$$

where $j = N, N-1, N-2, \ldots, 1$, α_b and γ_b are the learning and discount factors respectively for the backward update of Q-function.

2.9 Reward Function

A good balance of reward and punishment results in an equal variety in action selection, which makes the optimal action identified more trustworthy. For this reason, we will use a simplified version of Xu and Pi's work [44]. The actions of our smart selector will be rewarded based on this version, which considers as reward value +1 when fitness is improved or 0 otherwise. In equation (10) contained in Table (1) we can see the above.

Table 1: Rewards used in BSS

Reference	**Reward Function**
[44]	$r_n = \begin{cases} +1, & if\ the\ current\ action\ improves\ fitness \\ 0, & otherwise. \end{cases}$ (10)

3 RL-PSO: An Smart Selection Strategy

PSO is a swarm MH of great interest to researchers for solving complex optimization problems [29]. As mentioned in previous sections, MH is developed to

work in continuous domains, and certain modifications and adaptations are required to work in binary domains. In this instance, unlike the adapted versions in Section 2.4, we propose utilizing a framework that aims to autonomously and intelligently select transfer functions and binarization rules based on a chosen MH to solve binary problems.

3.1 A Reinforcement Learning-Based Binarization Scheme Selector

As mentioned in Section 1, we propose to binarize PSO using the developed framework called Binarization Scheme Selector (BSS) based on the aforementioned two-step technique. This combination of RL with MH follows the framework proposed by Talbi et al. in [37]. The notable aspect of this intelligent selector is its ability to autonomously learn through the reward function provided at the end of an episode during the evaluation of a transfer function and binarization rule in the optimization process. Figure 1 provides a general overview of how BSS is applied.

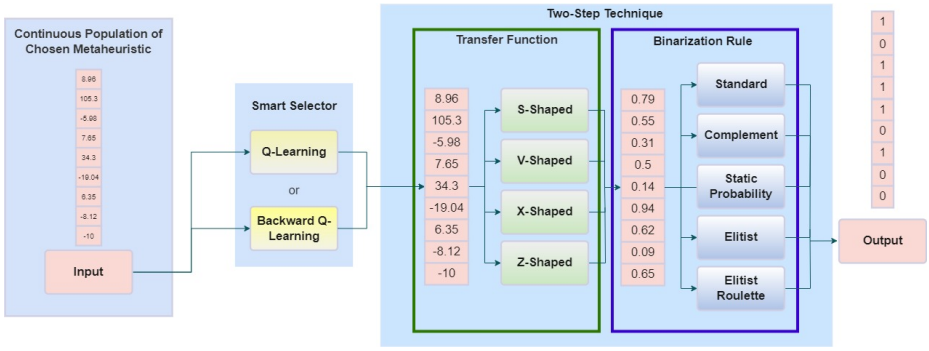


Fig. 1: Binarization Scheme Selector.

To explain the flow of our BSS (Binarization Scheme Selector), we need to choose the problem, the metaheuristic (MH), and the intelligent selector to use. In this particular case, we have chosen the Set Covering Problem (SCP) as the problem and Particle Swarm Optimization (PSO) as the MH. For the selector, we will use two different configurations: Q-Learning and Backward Q-Learning. This means that we will produce two distinct binary versions of PSO to solve the SCP.

Now, following the flow, we begin by initializing the continuous population and our Q-table. During the optimization process, in each iteration, we select a transfer function and a binarization rule. In the context of the intelligent selector, these selections correspond to a single action. PSO will be binarized using a single action from the intelligent selector. As the iterations progress and adapt to the problem, the action chosen by the selector will be modified.

In summary, the flow of BSS involves initializing the continuous population and the Q-table, selecting actions (transfer function and binarization rule) from the intelligent selector in each iteration, and using these actions to binarize PSO, which evolves over the iterations to better adapt to the problem. The aforementioned process is illustrated in Algorithm 1.

Algorithm 1 PSO WITH THE BSS

```
Initialize a random swarm
Initialize Q-table, M and N (If applicable)
for iteration (t) do
    Select action a_t for s_t from the Q-table
    for solution (i) do
        for dimension (d) do
            Randomly generate the value of r_1 and r_2
            Update the particle position with Eq. 1 and 2
            Update the pbest
        end for
    end for
    Binarization x_{i,d}^t with action a_t
    Get immediate reward r_t
    Get the maximum Q-value for the next state s_{t+1}
    Update Q-table using Equation (6) or Equation (7)
    Update the current state s_t ⟵ s_{t+1}
    if Using Backward Q-Learning then
        Record the four events in M
        if t = N then
            Backward update Q-Table usin Eq. 9
        end if
    end if
    Update gbest
end for
Return the updated swarm
```

4 Experimental Results

In order to validate the performance of our proposal, a comparison was made among 6 classical algorithms found in the literature [43] that solve the Set Covering Problem using different approaches. The algorithms BCL-PSO and MIR-PSO are algorithms that we define as static. For BCL, it uses the V4-Elitist functions [25], and for MIR, it uses the V4-Complement functions [30]. The reference instances of the Set Covering Problem solved are those proposed in the OR library by Beasley [4]. In particular, we solved 45 instances provided in this library, and the configuration of the instances is detailed in Table 2.

The algorithms were developed using Python 3.7 as the programming language and were executed on Google Colaboratory, a free service platform. The

Table 2: Configuration details from SCP instances employed in this work

Instance set	m	n	Cost range	Density (%)
4	200	1000	[1,100]	2
5	200	2000	[1,100]	2
6	200	1000	[1,100]	5
A	300	3000	[1,100]	2
B	300	3000	[1,100]	5
C	400	4000	[1,100]	2
D	400	4000	[1,100]	5

obtained results were stored and processed using databases provided by Google Cloud Platform. To assess the performance of the algorithms, the recommendations from the authors in the previous study [25] were followed, which suggested making 40,000 calls to the objective function. To achieve this, a population of 40 individuals was used, and 1000 iterations were performed in all PSO algorithm runs. Furthermore, 31 independent runs were conducted for each evaluated instance, ensuring a thorough and robust evaluation of the obtained results. This information is represented in Table 3.

Table 3: Parameters' setting.

Parameter	Value
Independent runs	31
Number of populations	40
Number of iterations	1000
parameter C_1 of PSO	2
parameter C_2 of PSO	2
parameter α of backward Q-learning, Q-learning	0.1
parameter γ of backward Q-learning, Q-learning	0.4
parameter N of backward Q-learning	10

Table 4 provides detailed computational results on 45 different instances for the 8 types of algorithms. To explain the different columns and rows of the table: the first column, titled "Inst.," represents the problem instance being solved, the second column, titled "Opt.," displays the known optimal value for each instance. The subsequent columns are organized into pairs, representing the different algorithms used to solve the SCP. Each pair of columns contains the algorithm's name and then shows the results for that algorithm in terms of the best solution found ("Best") and the average of all solutions found ("Avg"). Some cells are highlighted in bold, indicating that the corresponding algorithm found the best solution for that instance.

Table 4: Detailed computational results on 45 instances.

		BCL-PSO		MIR-PSO		GA		ACL		BCSO		HEMA		QL-PSO		BQSA-PSO	
Inst.	Opt.	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
41	429	430.0	433.73	513.0	535.27	**429**	429.7	**429**	429.0	432	443.0	**429**	429.0	430.0	433.52	430.0	434.65
42	512	513.0	530.36	780.0	848.91	**512**	512.0	**512**	512.0	**513**	538.5	**512**	512.0	519.0	530.84	515.0	530.48
43	516	**516.0**	520.64	859.0	904.36	**516**	516.0	**516**	516.0	520	554.5	**516**	516.0	521.0	528.9	520.0	528.74
44	494	495.0	506.0	713.0	745.18	**494**	494.8	**494**	494.0	495	512.5	**494**	494.0	495.0	508.16	497.0	509.1
45	512	514.0	519.91	821.0	875.73	**512**	512	**512**	512.0	**512**	526.5	**512**	512.0	515.0	525.71	518.0	525.94
46	560	563.0	570.55	985.0	1036.91	**560**	560	**560**	560.0	**560**	567.5	**560**	560.0	562.0	567.16	564.0	569.94
47	430	432.0	434.82	596.0	650.55	**430**	430.2	**430**	430.0	**430**	437.0	**430**	430.0	432.0	437.23	432.0	437.97
48	492	493.0	497.18	835.0	877.82	**492**	492.1	**492**	492.0	**492**	522.0	**492**	492.0	493.0	499.1	493.0	499.39
49	641	652.0	668.45	1129.0	1183.18	**641**	643.1	**641**	641.0	654	675.5	**641**	641.0	652.0	673.35	653.0	674.29
410	514	**514.0**	521.73	774.0	825.0	**514**	514.0	**514**	514.0	517	526.5	**514**	514.0	516.0	522.06	516.0	523.0
51	253	254.0	258.36	395.0	432.82	**253**	253.0	**253**	253.0	256	262.0	**253**	253.0	256.0	262.06	255.0	261.65
52	302	309.0	316.27	590.0	640.18	**302**	303.5	**302**	302.0	303	315.5	**302**	302.0	311.0	321.55	315.0	323.87
53	226	228.0	229.27	369.0	397.91	228	228.0	**226**	226.0	**226**	232.0	**226**	226.0	229.0	230.26	228.0	230.55
54	242	243.0	246.36	382.0	412.82	**242**	242.1	**242**	242.0	**242**	246	**242**	242.0	244.0	248.84	246.0	248.58
55	211	**211.0**	214.36	284.0	307.45	**211**	211.0	**211**	211.0	216	221.0	**211**	211.0	**211.0**	215.1	**211.0**	214.45
56	213	216.0	219.09	323.0	374.45	**213**	213.0	**213**	213.0	**213**	226.0	**213**	213.0	**213.0**	219.19	**213.0**	219.35
57	293	294.0	300.64	467.0	514.18	**293**	293.0	**293**	293.0	**293**	443.0	**293**	293.0	298.0	302.94	295.0	302.9
58	288	289.0	293.55	502.0	532.27	**288**	288.8	**288**	288.0	**288**	305.0	**288**	288.0	289.0	292.97	**288.0**	293.23
59	279	280.0	284.36	487.0	535.64	**279**	279.0	**279**	279.0	280	281.0	**279**	279.0	280.0	283.55	280.0	283.84
510	265	266.0	270.09	441.0	475.82	**265**	265.0	**265**	265.0	268	276.5	**265**	265.0	**265.0**	271.48	**265.0**	271.16
61	138	141.0	143.91	428.0	487.91	**138**	138.0	**138**	138.0	143	148.0	**138**	138.0	140.0	142.45	140.0	143.65
62	146	147.0	149.55	649.0	720.27	**146**	146.2	**146**	146.0	**146**	155.0	**146**	146.0	**146.0**	150.42	**146.0**	150.16
63	145	**145.0**	148.36	588.0	684.27	**145**	145.0	**145**	145.0	147	152.0	**145**	145.0	146.0	148.42	**145.0**	148.29
64	131	**131.0**	133.27	389.0	434.27	**131**	131.0	**131**	131.0	132	135.0	**131**	131.0	**131.0**	132.94	**131.0**	133.19
65	161	**161.0**	168.64	667.0	733.27	**161**	161.3	**161**	161.0	164	170.5	**161**	161.0	**161.0**	169.58	**161.0**	167.58
a1	253	256.0	259.09	863.0	971.64	**253**	253.2	**253**	253.0	269	276.5	**253**	253.0	257.0	261.52	258.0	262.45
a2	252	259.0	262.36	779.0	887.09	**252**	252.0	**252**	252.0	259	265.5	**252**	252.0	257.0	263.13	256.0	263.94
a3	232	237.0	240.45	793.0	844.64	**232**	232.5	**232**	232.8	233	243.5	**232**	232.0	239.0	242.23	238.0	242.35
a4	234	236.0	240.64	777.0	824.27	**234**	234.0	**234**	234.0	237	244.0	**234**	234.0	236.0	241.35	237.0	243.06
a5	236	**236.0**	238.73	800.0	841.09	**236**	236.0	**236**	236.0	**236**	239.0	**236**	236.0	238.0	241.94	239.0	242.55
b1	69	**69.0**	70.18	881.0	1026.82	**69**	69.0	**69**	69.0	**70**	74.0	**69**	69.0	**69.0**	70.0	**69.0**	70.32
b2	76	**76.0**	78.09	912.0	1035.09	**76**	76.0	**76**	76.0	79	84.0	**76**	76.0	**76.0**	77.03	**76.0**	76.87
b3	80	**80.0**	81.18	1282.0	1343.36	**80**	80.0	**80**	80.0	**80**	83.0	**80**	80.0	**80.0**	81.29	**80.0**	81.42
b4	79	**79.0**	81.82	1073.0	1184.82	**79**	79.0	**79**	79.0	81	84.0	**79**	79.0	**79.0**	81.58	**79.0**	81.52
b5	72	**72.0**	72.91	990.0	1076.73	**72**	72.0	**72**	72.0	73	73.0	**72**	72.0	**72.0**	72.87	**72.0**	72.94
c1	227	232.0	233.82	1160.0	1193.91	**227**	227.2	**227**	227.0	231	235.0	**227**	227.0	230.0	236.16	231.0	236.48
c2	219	223.0	226.0	1281.0	1350.0	**219**	220	**219**	219.0	221	231.0	**219**	219.0	222.0	228.97	225.0	229.71
c3	243	244.0	250.36	1514.0	1592.73	**243**	246.4	**243**	243.0	251	264.0	**243**	243.0	246.0	251.87	247.0	251.55
c4	219	225.0	230.09	1308.0	1340.36	**219**	219.1	**219**	219.0	225	240.0	**219**	219.0	222.0	228.55	221.0	228.26
c5	215	**215.0**	218.73	1212.0	1267.45	**215**	215.1	**215**	215.0	219	228.0	**215**	215.0	218.0	221.74	218.0	221.26
d1	60	**60.0**	61.55	1473.0	1553.55	**60**	60	**60**	60.0	**60**	65.0	**60**	60.0	61.0	62.26	61.0	62.52
d2	66	**66.0**	67.64	1719.0	1810.36	**66**	66	**66**	66.0	69	70.0	**66**	66.0	67.0	67.68	67.0	67.55
d3	72	73.0	74.91	1846.0	1978.82	**72**	72.2	**72**	72.0	76	79.0	**72**	72.0	73.0	74.81	73.0	75.16
d4	62	**62.0**	63.82	1517.0	1634.45	**62**	62	**62**	62.0	63	66.5	**62**	62.0	**62.0**	62.55	**62.0**	62.94
d5	61	**61.0**	62.45	1513.0	1628.55	**61**	61	**61**	61.0	63	65.0	**61**	61.0	**61.0**	62.23	**61.0**	62.58
		255.51	259.87	859.09	923.38	253,82	254,1	253,78	253,80	256,38	268,49	253,78	253,78	256.0	261.06	256.16	261.36

On the other hand, we present convergence plots (Figs. 2a-2d) and exploration-exploitation states (Figs. 2e-2h) of the implemented algorithms (BLC-PSO, MIR-PSO, QL-PSO, and BQSA-PSO). For the state plots, we have utilized the equations described in [17], which allow us to determine diversity [8] and subsequently calculate the exploration or exploitation states of the algorithm [33]. The plots are defined as follows: on the X-axis, the total number of iterations is presented. For the convergence plots, the Y-axis represents the fitness, while for states plots, it corresponds to the percentage related to exploration or exploitation. The equations for diversity determination and exploration-exploitation calculation are as follows:

$$Div = \frac{1}{l \cdot n} \sum_{d=1}^{l} \sum_{i=1}^{n} |\bar{x}^d - x_i^d|, \tag{11}$$

Where Div represents the diversity state determination, $\bar{x}^d$ denotes the mean of individuals in dimension d, x_i^d is the value of the i-th individual in the d-th dimension, n is the number of individuals in the population, and l is the size of the individuals' dimension.

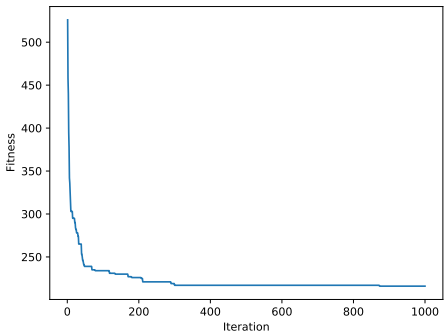
$$XPL\% = \frac{Div}{Div_{max}} \cdot 100, \tag{12}$$

$$XPT\% = \frac{|Div - Div_{max}|}{Div_{max}} \cdot 100. \tag{13}$$

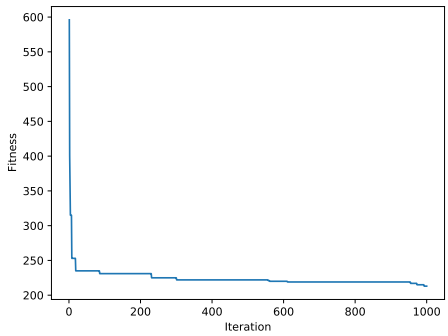
Where Div represents the diversity state determined by Eq. 11 and Div_{max} denotes the maximum value of the diversity state discovered throughout the optimization process.

5 Conclusion

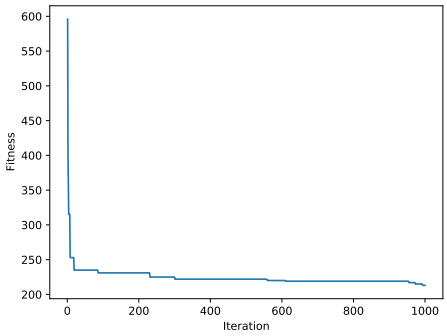
In this article, we have presented a study and evaluation of various algorithms, such as the application of binarized metaheuristic algorithms with our BSS framework, for solving a combinatorial optimization problem. Our flexible framework allows for choosing the number of actions and the intelligent selector to use. A comparison among different metaheuristic algorithms demonstrates that we can achieve different behaviors and scalability depending on the combinations of actions used. The results demonstrate the effectiveness of metaheuristic algorithms in solving optimization problems and also highlight that some algorithms have a greater impact on performance. During the analysis, it is worth mentioning that algorithms like HEMA or ACL have a 5 million call target, whereas ours has only 40,000. The exploration and exploitation graphs provide a different perspective on behavior during the search process, giving us information about the diversity among individuals, as defined in [33]. The rapid convergence of the implementations brings efficiency to our solution, which translates into more iterations and allows for rapid exploration of different regions of the search space, increasing the possibility of finding optimal solutions or improving already found solutions through exploitation. These results can be useful for future research in the field of combinatorial optimization. Future work will focus on implementing and integrating a new intelligent selector called Multi-armed Bandit into the framework. Additionally, the option to evaluate other metaheuristic algorithms from the literature with similar exploration and exploitation behaviors as presented in other binary domain problems is considered, in order to validate that the incorporation of reinforcement learning techniques has the same effect on them.



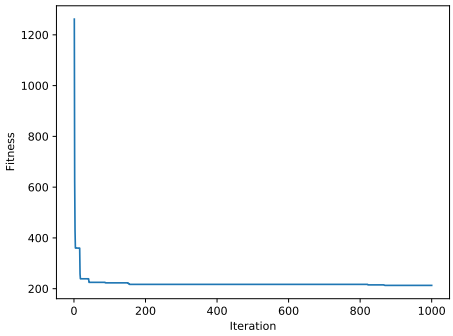
(a) BCL



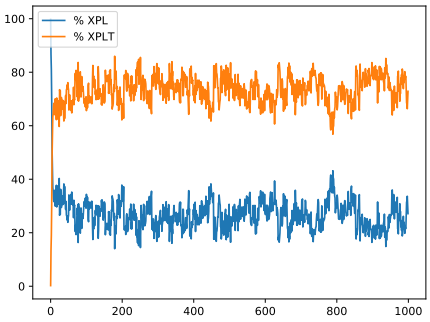
(b) MIR



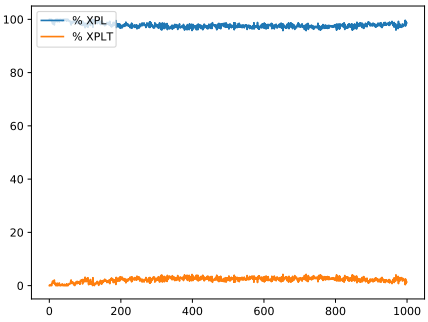
(c) QL



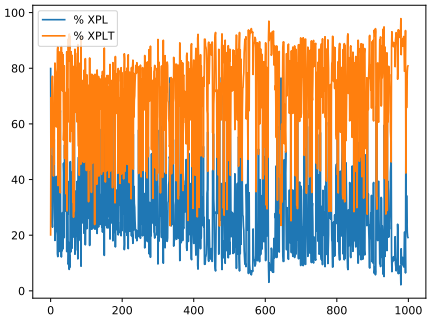
(d) BQSA



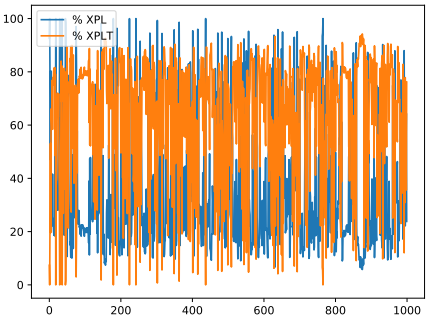
(e) BCL



(f) MIR



(g) QL



(h) BQSA

Fig. 2: Convergence and Exploration-Explotation plots of Instance 56 for the different approaches used.

6 Acknowledgements

Broderick Crawford and Ricardo Soto are supported by Grant ANID/FONDECYT/REGULAR/1210810. Marcelo Becerra-Rozas is supported by the National Agency for Research and Development (ANID)/Scholarship Program/DOCTORADO NACIONAL/2021-21210740.

References

1. Abdel-Basset, M., Mohamed, R., Elkomy, O.M., Abouhawwash, M.: Recent metaheuristic algorithms with genetic operators for high-dimensional knapsack instances: A comparative study. Computers & Industrial Engineering **166**, 107974 (2022)
2. Agrawal, P., Ganesh, T., Oliva, D., Mohamed, A.W.: S-shaped and v-shaped gaining-sharing knowledge-based algorithm for feature selection. Applied Intelligence **52**(1), 81–112 (2022)
3. Awadallah, M.A., Abu-Doush, I., Al-Betar, M.A., Braik, M.S.: Metaheuristics for optimizing weights in neural networks. In: Comprehensive Metaheuristics, pp. 359–377. Elsevier (2023)
4. Beasley, J., Jörnsten, K.: Enhancing an algorithm for set covering problems. European Journal of Operational Research **58**(2), 293–300 (1992)
5. Becerra-Rozas, M., Lemus-Romani, J., Cisternas-Caneo, F., Crawford, B., Soto, R., Astorga, G., Castro, C., García, J.: Continuous metaheuristics for binary optimization problems: An updated systematic literature review. Mathematics **11**(1), 129 (2022)
6. Becerra-Rozas, M., Lemus-Romani, J., Cisternas-Caneo, F., Crawford, B., Soto, R., García, J.: Swarm-inspired computing to solve binary optimization problems: A backward q-learning binarization scheme selector. Mathematics **10**(24), 4776 (2022)
7. Beheshti, Z., Shamsuddin, S.M., Hasan, S.: Memetic binary particle swarm optimization for discrete optimization problems. Information Sciences **299**, 58–84 (2015)
8. Choi, S.S., Cha, S.H., Tappert, C.C.: A survey of binary similarity and distance measures. Journal of systemics, cybernetics and informatics **8**(1), 43–48 (2010)
9. Crawford, B., Soto, R., Astorga, G., García, J., Castro, C., Paredes, F.: Putting continuous metaheuristics to work in binary search spaces. Complexity **2017** (2017)
10. Crawford, B., Soto, R., Lemus-Romani, J., Becerra-Rozas, M., Lanza-Gutiérrez, J.M., Caballé, N., Castillo, M., Tapia, D., Cisternas-Caneo, F., García, J., et al.: Q-learnheuristics: Towards data-driven balanced metaheuristics. Mathematics **9**(16), 1839 (2021)
11. Cuevas, E., Fausto, F., González, A.: New Advancements in Swarm Algorithms: Operators and Applications. Springer (2020)
12. Farag, M.M., Alhamad, R.A., Nassif, A.B.: Metaheuristic algorithms in optimal power flow analysis: A qualitative systematic review. International Journal on Artificial Intelligence Tools (2023)
13. García, J., Moraga, P., Valenzuela, M., Crawford, B., Soto, R., Pinto, H., Peña, A., Altimiras, F., Astorga, G.: A db-scan binarization algorithm applied to matrix covering problems. Computational intelligence and neuroscience **2019** (2019)

14. Guo, S.s., Wang, J.s., Guo, M.w.: Z-shaped transfer functions for binary particle swarm optimization algorithm. Computational Intelligence and Neuroscience **2020** (2020)
15. Heidari, A.A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., Chen, H.: Harris hawks optimization: Algorithm and applications. Future generation computer systems **97**, 849–872 (2019)
16. Holland, J.H.: Genetic algorithms. Scientific american **267**(1), 66–73 (1992)
17. Hussain, K., Zhu, W., Salleh, M.N.M.: Long-term memory harris' hawk optimization for high dimensional and optimal power flow problems. IEEE Access **7**, 147596–147616 (2019)
18. Islam, M.J., Li, X., Mei, Y.: A time-varying transfer function for balancing the exploration and exploitation ability of a binary pso. Applied Soft Computing **59**, 182–196 (2017)
19. Jourdan, L., Dhaenens, C., Talbi, E.G.: Using datamining techniques to help metaheuristics: A short survey. In: International Workshop on Hybrid Metaheuristics. pp. 57–69. Springer (2006)
20. Kalimuthu, M., Pathmakumar, T., Hayat, A.A., Veerajagadheswar, P., Elara, M.R., Wood, K.L.: Optimal morphologies of n-omino-based reconfigurable robot for area coverage task using metaheuristic optimization. Mathematics **11**(4), 948 (2023)
21. Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A.M., Talbi, E.G.: Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. European Journal of Operational Research **296**(2), 393–422 (2022)
22. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks. vol. 4, pp. 1942–1948. IEEE (1995)
23. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: 1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation. vol. 5, pp. 4104–4108. IEEE (1997)
24. Lai, X., Hao, J.K., Fu, Z.H., Yue, D.: Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. Expert Systems with Applications **149**, 113310 (2020)
25. Lanza-Gutierrez, J.M., Crawford, B., Soto, R., Berrios, N., Gomez-Pulido, J.A., Paredes, F.: Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization. Expert Systems with Applications **70**, 67–82 (2017)
26. Lemus-Romani, J., Becerra-Rozas, M., Crawford, B., Soto, R., Cisternas-Caneo, F., Vega, E., Castillo, M., Tapia, D., Astorga, G., Palma, W., et al.: A novel learning-based binarization scheme selector for swarm algorithms solving combinatorial problems. Mathematics **9**(22), 2887 (2021)
27. de León, A.D., Lalla-Ruiz, E., Melián-Batista, B., Moreno-Vega, J.M.: A machine learning-based system for berth scheduling at bulk terminals. Expert Systems with Applications **87**, 170–182 (2017)
28. Ma, W., Zhou, X., Zhu, H., Li, L., Jiao, L.: A two-stage hybrid ant colony optimization for high-dimensional feature selection. Pattern Recognition **116**, 107933 (2021)
29. Ma, Z., Wu, G., Suganthan, P.N., Song, A., Luo, Q.: Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms. Swarm and Evolutionary Computation **77**, 101248 (2023)

30. Mirjalili, S., Lewis, A.: S-shaped versus v-shaped transfer functions for binary particle swarm optimization. Swarm and Evolutionary Computation **9**, 1–14 (2013)
31. Mirjalili, S., Song Dong, J., Sadiq, A.S., Faris, H.: Genetic algorithm: Theory, literature review, and application in image reconstruction. Nature-Inspired Optimizers: Theories, Literature Reviews and Applications pp. 69–85 (2020)
32. Mirjalili, S., Zhang, H., Mirjalili, S., Chalup, S., Noman, N.: A novel u-shaped transfer function for binary particle swarm optimisation. In: Soft Computing for Problem Solving 2019: Proceedings of SocProS 2019, Volume 1. pp. 241–259. Springer (2020)
33. Morales-Castañeda, B., Zaldivar, D., Cuevas, E., Fausto, F., Rodríguez, A.: A better balance in metaheuristic algorithms: Does it exist? Swarm and Evolutionary Computation p. 100671 (2020)
34. Pampara, G., Engelbrecht, A.P., Franken, N.: Binary differential evolution. In: 2006 IEEE international conference on evolutionary computation. pp. 1873–1879. IEEE (2006)
35. Saha, S., Kole, A., Dey, K.: A modified continuous particle swarm optimization algorithm for uncapacitated facility location problem. In: Information Technology and Mobile Communication: International Conference, AIM 2011, Nagpur, Maharashtra, India, April 21-22, 2011. Proceedings. pp. 305–311. Springer (2011)
36. Talbi, E.G.: Combining metaheuristics with mathematical programming, constraint programming and machine learning. Annals of Operations Research **240**(1), 171–215 (2016)
37. Talbi, E.G.: Machine learning into metaheuristics: A survey and taxonomy. ACM Computing Surveys (CSUR) **54**(6), 1–32 (2021)
38. Valenzuela, M., Moraga, P., Causa, L., Pinto, H., Rubio, J.M.: A machine learning whale algorithm applied to the matrix covering problem. In: Data Science and Intelligent Systems: Proceedings of 5th Computational Methods in Systems and Software 2021, Vol. 2. pp. 413–422. Springer (2021)
39. Wang, L., Wang, X., Fu, J., Zhen, L.: A novel probability binary particle swarm optimization algorithm and its application. J. Softw. **3**(9), 28–35 (2008)
40. Wang, Y.H., Li, T.H.S., Lin, C.J.: Backward q-learning: The combination of sarsa algorithm and q-learning. Engineering Applications of Artificial Intelligence **26**(9), 2184–2193 (2013)
41. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)
42. Xiong, G., Yuan, X., Mohamed, A.W., Zhang, J.: Fault section diagnosis of power systems with logical operation binary gaining-sharing knowledge-based algorithm. International Journal of Intelligent Systems **37**(2), 1057–1080 (2022)
43. Xu, F., Li, J.: A hybrid encoded memetic algorithm for set covering problem. In: 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI). pp. 552–557. IEEE (2018)
44. Xu, Y., Pi, D.: A reinforcement learning-based communication topology in particle swarm optimization. Neural Computing and Applications pp. 1–26 (2019)
45. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: 2009 World congress on nature & biologically inspired computing (NaBIC). pp. 210–214. Ieee (2009)

Trend–Risk in Complex Portfolio Selection Strategy

D. Neděla[1], S. Ortobelli[2] and T. Tichý[3]

[1] Department of Finance, VSB – Technical University of Ostrava, Sokolská třída 33, Ostrava, 70200, Czech Republic
david.nedela@vsb.cz

[2] Department of Management, University of Bergamo, Via dei Caniana 2, Bergamo, 24127, Italy
sergio.ortobelli@unibg.it

[3] Department of Finance, VSB – Technical University of Ostrava, Sokolská třída 33, Ostrava, 70200, Czech Republic
tomas.tichy@vsb.cz

Abstract

In a portfolio optimization theory, a rationally minded investor focuses on maximizing potential profit with respect to possible portfolio losses (risk incurred). In general, the question of how to adequately estimate and evaluate the risk of a large-scale portfolio or a single asset remains an important problem in financial risk management. Furthermore, the approach to portfolio optimization is an important aspect as well, where different types of optimization processes may be employed. For this reason, metaheuristic algorithms are also finding increasing use in the field of computational finance.

The major contribution of this work is to provide a time-dependent risk (dispersion) measure which is an alternative to the "accrued returns variability" (ARV) introduced by Ruttiens, [7]. Moreover, we enhance the existing literature focused on portfolio theory and trend-dependent risk measures, [2], [3], [4], [5], [8]. In the original work of Ruttiens, the ARV is calculated as the standard deviation of spreads between cumulative returns and a trend risk free variation of these returns at each time leading to the same final cumulative returns. Rather, the definition of the non-volatile linear alternative formulation could correspond to the analysis of trend dispersion. The mathematical formulation of the presented dynamic risk measure can be more precise when we consider the mean of squared spreads of cumulative return series with respect to the non-volatile benchmark. Therefore, our "modified accrued returns variability" (modARV) is more accurate in the portfolio optimization framework while minimizing distortion with respect to the predefined zero-risk linear trend. We also define modified dependency measures, such as covariance and correlation, derived from the proposed risk measures.

In the context of portfolio optimization, we apply newly proposed trend–risk measures and dependency matrices in complex mean–variance portfolio selection strategies to demonstrate their properties in the portfolio theory. In particular, to evaluate the effect of trend-dependent risk measures, we compare the mean–variance optimization strategy with a new compounded double optimization framework (strategy). This double optimization strategy consists of two steps. In the first step, we fit optimal portfolios of the mean–variance efficient frontier. In the second step, we determine the minimum modified Ruttiens risk measure fixing the expected mean and the final wealth of the optimal mean–variance portfolios in order to minimize the deviations with respect to the trend. To reduce the dimensionality of the portfolio, we use parametric and nonparametric returns approximation techniques with PCA applied to linear or trend-dependent correlation matrices, [1], [3], [6]. According to this empirical analysis, the newly proposed approach leads to the mitigation of shortcomings and improves the ex-post portfolio statistics compared to the mean–variance scenarios.

The empirical results showed that using the nonparametric RW approximation, the wealth is smoother during the investment and return series are less variable, with insignificant differences in the profitability. Moreover, we have demonstrated that:

- the integration of time-dependent or trend–risk measures as an alternative in the optimization process expands our insight into the issue of portfolio selection strategy;
- a trend–risk double optimization portfolio strategy outperforms the profitability of the simple mean–variance selection strategy;

– in general, these risk measures appear to be more useful in the finance sense as well as attractive to risk-avoiding investors.

The potential of this concept is to integrate different types of function (e.g. the exponential) to capture trend preferences or replace it with the market trend. Furthermore, we could consider replacing the mean–variance model for the first optimization with a max-ratio model.

References

1. N. Kouaissah and A. Hocine. Forecasting systemic risk in portfolio selection: The role of technical trading rules. *Journal of Forecasting*, 40(4):708-729, 2021.
2. H. M. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77-91, 1952.
3. S. Ortobelli, N. Kouaissah, and T. Tichý. On the use of conditional expectation in portfolio selection problems. *Annals of Operations Research,* 274(1):501-530, 2019.
4. S. Rachev, S. Ortobelli, S. Stoyanov, F. J. Fabozzi, and A. Biglova. Desirable properties of an ideal risk measure in portfolio theory. *International Journal of Theoretical and Applied Finance,* 11(1):447-469, 2008.
5. R. Rockafellar, and S. Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26:1443-1471, 2022.
6. D. Ruppert, and M. Wand. Multivariate locally weighted least squares regression. *The Annals of Statistics*, 22(3):1346-1370, 1994.
7. A. Ruttiens. Portfolio risk measures: the time's arrow matters. *Computational Economics*, 41(3):407-424, 2013.
8. G. Szegö. Measures of risk. *Journal of Banking & Finance*, 26(7):1253-1272, 2022.

A Study About Meta-Optimizing the NSGA-II Multi-Objective Evolutionary Algorithm

José F. Aldana-Martín[2], Antonio J. Nebro[1,2], Juan J. Durillo[3], and María del Mar Roldán García[1,2]

[1] Departamento de Lenguajes y Ciencias de la Computación. University of Málaga, 29071 Málaga, Spain
[2] ITIS Software, University of Málaga, 29071, Málaga, Spain
jfaldanam@uma.es, ajnebro@uma.es, mrgarcia@uma.es
[3] Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities, Germany
durillo@lrz.de

Abstract. The automatic design of multi-objective metaheuristics is an active research line aimed at, given a set of problems used as training set, to find the configuration of a multi-objective optimizer able of solving them efficiently. The expected outcome is that the auto-configured algorithm can be used of find accurate Pareto front approximations for other problems. In this paper, we conduct a study on the meta-optimization of the well-known NSGA-II algorithm, i.e., we intend to use NSGA-II as an automatic configuration tool to find configurations of NSGA-II. This search can be formulated as a multi-objective problem where the decision variables are the NSGA-II components and parameters and the the objectives are quality indicators that have to be minimized. To develop this study, we rely on the jMetal framework. The analysis we propose is aimed at answering the following research questions: RQ1 - how complex is to build the meta-optimization package?, and RQ2 - can accurate configurations be found? We conduct an experimentation to give an answer to these questions.

Keywords: Multi-objective optimization, auto-configuration of metaheuristics, NSGA-II

1 Introduction

The quality of the Pareto front approximations found by multi-objective evolutionary algorithms is affected on the values of their control parameters. This means that, given a set of problems to be optimized and a given algorithm, the user has to fine tune the algorithms parameters to get accurate results. The approach commonly adopted to carry out this task is to try to adjust the parameters manually by conducting pilot tests, which is a trial-and-error strategy. Furthermore, this process requires knowledge of the algorithm, which is not usually the case of the users expert in the problems. The consequence is that those users are likely to end up selecting a well-known algorithm, typically NSGA-II [1], with default settings.

In this context, an active research line is automatic algorithm configuration [2], consisting in taking a set of problems as training set to find a particular parameter configuration of the parameters to a produce version of the algorithm that, configured with them, can solve those problems efficiently. An extension of this idea is automatic algorithm design, where not only parameters but also algorithmic components can be combined to design a new algorithmic variant. An advantage of these approaches is that they can be supported by tools that help to find the configurations automatically, such as irace [3], paramILS [4], GSF [5] and SMAC3 [6]. Focusing on multi-objective evolutionary algorithms, irace has been applied in several works [7][8][9].

In this paper, we conduct a study about the use of NSGA-II to find configurations of NSGA-II, i.e., using NSGA-II as meta-optimizer. The basic idea is to consider the auto-design of NSGA-II as a multi-objective problem, where the decision variables represent parameters and components and the objectives can be combinations of quality indicators [10].

Our motivation stems, first, from our experiences in automatic design of multi-objective metaheuristics, which are based on combining the jMetal optimization framework [11, 12] with irace to find configurations of NSGA-II [13][14] and particle swarm optimizers [15]. Second, a recent survey [2] that remarked as future research prospects easy-to-use algorithm tuning and multi-objective approaches. Although our proposal does not include a toolbox (as suggested the mentioned survey [2]), we design a package based on jMetal, so we do not need to use external tools such as

irace, thus simplifying the auto-design process in case the optimization problems are implemented with that framework.

We define two research questions that we intend to answer in our our study:

- RQ1: how complex is to build the meta-optimization package?. We are interested in a simple and easy-to-use software solution.
- RQ2: can accurate configurations be found? The search capabilities of the meta-optimizer must be validated by conducting representative experiments.

The rest of the paper is organized as follows. Section 2 provides a detailed description of the presented approach for the automated design of a meta-optimizer for NSGA-II. In Section 3, we present the results of three experiments conducted to validate our proposal. The findings and implications of these experiments are discussed in Section 4, while Section 5 provides conclusions about the effectiveness and usefulness of our study.

2 Meta-Optimization Approach

The process of auto-designing evolutionary algorithms requires three elements: the design space, an algorithmic template, and an auto-design tool. We describe these elements next, including how we cope with them.

2.1 Design space

The design space is composed of the algorithm parameters and components, their types, allowed values, and, optionally, constraints. In the case of NSGA-II, we consider a flexible definition of it, in which a multi-objective evolutionary algorithm adopting a replacement strategy based on dominance ranking and the crowding distance density estimator is considered a NSGA-II variant. We define the design space detailed in Table 1, which is similar to the ones used in former works [9][14] (please refer to these references for a detailed explanation of the parameters and components).

Parameter/Component	Type	Domain	
algorithmResult	c	{externalArchive, population}	
populationSizeWithArchive	i	[10, 200]	s.t. algorithmResult == externalArchive
externalArchive	c	{crowdingDistance, unbounded}	s.t. algorithmResult == externalArchive
offspringPopulationSize	i	[1, 400]	
selection	c	{tournament, random}	
selectionTournamentSize	i	[2, 10]	s.t. selection == tournament
createInitialSolutions	c	{random, latinHypercubeSampling, scatterSearch}	
crossover	c	{SBX, BLX_ALPHA, wholeArithmetic}	
crossoverProbability	r	[0.0, 1.0]	
crossoverRepairStrategy	c	{random, round, bounds}	
sbxDistributionIndex	r	[5.0, 400.0]	s.t. crossover == SBX
blxAlphaCrossoverAlphaValue	r	[0.0, 1.0]	s.t. crossover == BLX_ALPHA
mutation	c	{uniform, polynomial, linkedPolynomial, nonUniform}	
mutationProbabilityFactor	r	[0.0, 2.0]	
mutationRepairStrategy	c	{random, round, bounds}	
polynomialMutationDistributionIndex	r	[5.0, 400.0]	s.t. mutation ∈ {polynomial, linkedPolinomial}
uniformMutationPerturbation	r	[0.0, 1.0]	s.t. mutation == uniform
nonUniformMutationPerturbation	r	[0.0, 1.0]	s.t. mutation == nonUniform

Table 1: Design space of the configurable NSGA-II in jMetal. Types: (c)ategorical, (i)nteger, (r)eal.

An example of parameter is the offspring population size, which is an integer variable taking values in the range [1, 400] (a value of 1 would lead to a steady-state version of NSGA-II). Examples of components are the crossover and mutation operators. Each operators can in turn also have specific parameters, such as the distribution index for SBX crossover.

A design decision is whether NSGA-II uses an external archive (i.e., an auxiliary population) or not. If the population size is P, the idea is that any evaluated solution is inserted into the archive, which keeps only non-dominated solutions, and the result of the algorithm would be P solutions from the archive; in this case, the population size is not fixed and it can take a value between 10

and 200. The external archive can be bounded (the crowding distance is used as density estimator to remove solutions when the archive size is greater than P) or unbounded (in this case, all the evaluated solutions are inserted and, when the algorithm finishes, P evenly spread solutions are returned).

2.2 Algorithmic template

Since release 6.0, jMetal includes a *jmetal-auto* package containing an implementation of NSGA-II, called AutoNSGAII, which can take any valid combination of the parameters and components of Table 1, generating different NSGA-II versions.

The input of AutoNSGAII is a string containing all the parameter names and their values. This string is parsed internally and AutoNSGAII is configured with the parameter values and the components specified in the string. An example of a subset of this string is the following: "*–archiveResult externalArchive –offpringPopulation 40 –selection tournament ...*"

2.3 Meta-optimizer

In our previous works combining jMetal with irace [13][14], the finding of configurations is based on running irace, which generates combinations of valid configurations according to the design space. For each configuration, irace runs AutoNSGAII, which returns as a result the value of a quality indicator; this value is taken by irace as a measure of the quality of the configuration.

As we intend to replace irace by the NSGA-II algorithm implemented in jMetal, which would act as meta-optimizer, we have to formulate and implement the optimization problem that would to be solved by the meta-optimizer. This problem has the following parameters:

- List of problems used as training set.
- List of quality indicators, being each indicator an objective to be minimized.
- The population size of AutoNSGAII.
- The stopping condition of AutoNSGAII (in terms of number of evaluations).
- Number of independent runs of AutoNSGAII for each configuration to be evaluated.

To define the problem encoding, the approach we have adopted is simple: every parameter of Table 1 is represented as a real value in the range $[0.0, 1.0]$, so the solutions are composed of 18 decision variables. When a solution has to be evaluated, the variables are decoded to construct the parameter string that is used when calling AutoNSGAII. The decoding is done as follows:

- Real parameter: the value is scaled up from $[0.0, 1.0]$ to the range of the parameter (e.g.,. $[5.0, 400.0]$ in the case of the SBX distribution index).
- Integer parameter: same procedure as for real parameters, but the resulting value is truncated.
- Categorical parameter: the interval $[0, 0, 1, 0]$ is divided into sub-intervals according to the number of parameter values, and the index of the sub-interval is used to obtain the actual categorical value.

Once the parameter string is decoded, AutoNSGAII is called to solve all the problems of the training set as many times as the number of independent runs. For each obtained front, the quality indicators are computed and the resulting objectives values of evaluating a configuration is the median of the median of the quality indicators of all the problems of the training set.

We now look at the pros and cons of this approach. Starting by the cons, we are not considering parameter constraints, so all the elements of design space are included although some of them may be ignored (e.g., the uniform perturbation is useless if the selected mutation operator is polynomial), and the discretization of categorical parameters using sub-intervals can lead to different solutions being equivalent if all variables have the same values except one corresponding to a categorical parameter whose values are in the same sub-interval. As advantages, the encoding is very simple and any multi-objective algorithm in jMetal able of solving continuous problems can be used as meta-optimizer.

3 Experimentation

We aim to empirically validate our approach with a set of experiments grouped into two different scenarios. These experiments are described below, detailing their purpose, expected outcomes and results.

The meta-optimizer is configured with the additive epsilon (EP) and normalized hypervolume (NHV) quality indicators as the objective functions to be minimized. The first indicator measures the convergence of a Pareto front approximation while the second one takes into account both convergence and diversity [10]. We use NHV instead of plain hypervolume as for this latter, the bigger its value the better, while jMetal minimizes objective functions by default. NHV is defined as 1.0 minus the hypervolume of the front divided by the hypervolume of the reference front.

For the meta-optimizer, we have configured it with common NSGA-II parameter values. The population size is 50 and the variation operators are SBX crossover (with probability 0.9 and a distribution index value of 20.0) and polynomial mutation (with probability $1/n$, being n the number of decision variables of the problem, and a distribution index value of 20.0). We set the stopping to condition to 3000 function evaluations. The NSGA-II implementation in jMetal can be executed in parallel both using a synchronous or an asynchronous scheme [16].

Next, we define two scenarios and three experiments.

3.1 Scenario 1: Finding Configurations for Single Problems

The first scenario is aimed at determining whether our meta-optimization approach is able of finding well-performing configurations of NSGA-II for single problems. For that, we focus on experimenting with two problems:

- **Experiment 1 - problem ZDT4**: this problem [17] is a bi-objective multi-frontal problem, whose default configuration consists of 10 decision variables. The standard NSGA-II has difficulty in providing Pareto front approximations with a uniform spread of solutions. Previous studies [13] have shown that using a steady-approach can significantly improve the diversity of the fronts. Pilot tests also indicate that comparable improvements can be achieved when using an external bounded archive.
- **Experiment 2 - problem DTLZ3**: This problem belongs to the DTLZ benchmark [18]. It is formulated with a default configuration consisting of twelve decision variables and three objectives. DTLZ3 is also a multi-modal problem with a convex Pareto front. The study presented in [9] showed that both, NSGA-II and the AutoNSGAII, configured with irace were unable to find accurate approximated fronts in terms of convergence and diversity for this problem.According to other works [19], NSGA-II is able of finding accurate fronts for problem DTLZ2 when using an external unbounded archive and retrieving from it a subset of evenly distributed solutions. DLTZ2 is not multi-modal but shares many similarities with DTLZ3 (i.e., convex Pareto front, three objectives, and twelve decision variables). Our aim here is twofold: 1) to determine whether the meta-optimizer is able of finding a configuration to effectively solve DTLZ3; and, 2) to check if that configuration includes an unbounded archive.

3.2 Scenario 2: Finding Configurations for Sets of Problems

The above scenario must validate the potential of auto-configuration applied to optimize single problems. The found configurations may be, however, too specific to that particular problem, and perform poorly for other problems (overfitting). Our second scenario addresses this issue by auto-configuring the algorithm on a set of problems instead of just one. Additionally, the obtained configurations are validated using different sets of problems.

- **Experiment 3 - WFG benchmark**: This experiment aims to replicate the study presented in [9]. NSGA-II is configured for optimizing the nine problems of the WFG suite [20], which are *training set*. The found configurations are later used to solve both the WFG problems and the seven instances of the DTLZ family problems–the *validation set*. All the problems in this experiment are formulated as bi-objective ones.

Parameter	NSGA-II	Exp. 1	Exp. 2	Exp. 3
populationSize	100	100	100	100
createInitialSolutions	random	LHS	scatterSearch	random
algorithmResult	population	externalArchive	externalArchive	externalArchive
externalArchive	-	CD	unboundedArchive	CD
populationSizeWithArchive	-	106	58	61
offspringPopulationSize	100	60	130	68
crossover	SBX	SBX	SBX	BLX_ALPHA
crossoverProbability	0.9	0.991	0.942	0.858
crossoverRepairStrategy	random	round	random	bounds
sbxDistributionIndexValue	20.0	5.11	70.479	-
blxAlphaCrossoverAlphaValue	-	-	-	0.547
mutation	polynomial	polynomial	uniform	linkedPolynomial
mutationProbabilityFactor	1	0.76	0.699	0.161
mutationRepairStrategy	random	bounds	round	round
polynomialMutationDistributionIndex	20	32.23	-	-
linkedPolynomialMutationDistributionIndex	-	-	-	11.335
uniformMutationPerturbation	-	-	0.417	-
selection	tournament	tournament	random	tournament
selectionTournamentSize	2	9	-	4

Table 2: Best configuration found for the NSGA-II on each experiment. (LHS; latinHypercubeSampling, CD; crowdingDistanceArchive)

3.3 Results

We report and analyze the results obtained on the three defined experiments. In all the cases, the number of independent runs per configuration is set to 3.

Experiment 1 We set the stopping condition of AutoNSGAII to perform a total 15000 function evaluations. Fig. 1 shows computed fronts by the meta-optimizer at 1000, 2000, and 3000 evaluations. As shown, the final front is composed of only one solution, and the figure suggests that the meta-optimizer might not have converged in the performed evaluations. The found design in this experiment (see Table 2) includes a bounded external archive with crowding distance; as commented before, the use of this kind of archive is known to be beneficial for converging and for achieving a front of evenly spread solutions.

AutoNSGAII with the obtained configuration is compared with NSGA-II with default settings next. We set the stopping condition to 25000 function evaluations in both cases and compare the Pareto front approximations computed by both algorithms. We observe that the front computed with the found configuration (Fig. 2 right) has a noticeable better convergence and spread than the approximation computed by NSGA-II with standard the default setting (Fig. 2 left).

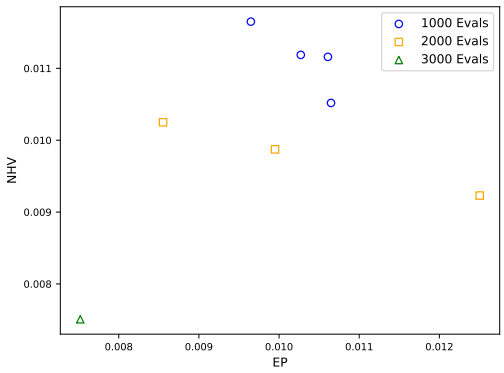


Fig. 1: Problem ZDT4. Evolution of the front generated by the meta-optimizer.

Experiment 2 In this experiment, the stopping criterion for AutoNSGAII has been raised to 20000 evaluations.

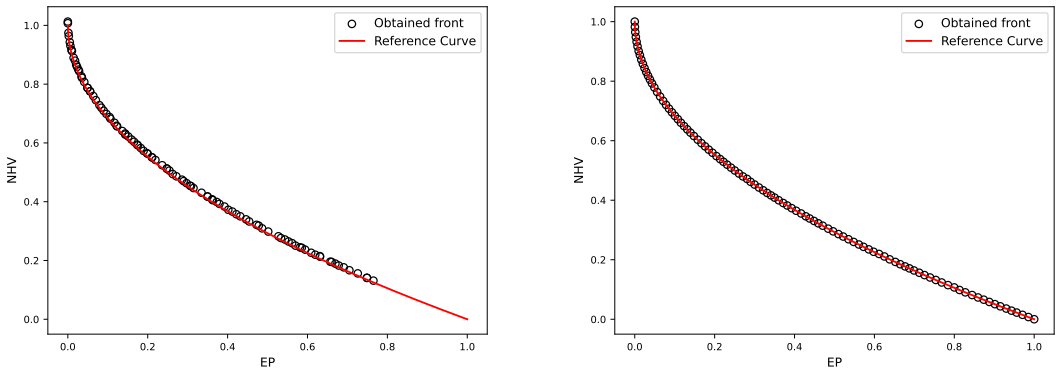


Fig. 2: Problem ZDT4. Pareto front approximation found by the standard NSGA-II (left), and Pareto front approximation found by the auto-designed NSGA-II (right).

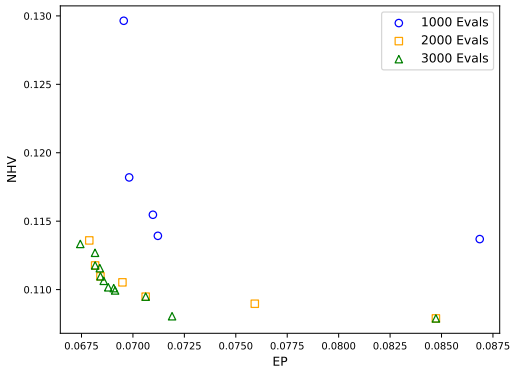


Fig. 3: Problem DTLZ3. Evolution of the front generated by the meta-optimizer.

Fig. 3 shows the approximation fronts computed after 1000, 2000 and 3000 evaluations. In this case, the figure suggest that the meta-optimizer has almost converged after performing the 3000 function evaluations. The computed approximation front consists of twelve points. The configuration corresponding to the point with the lowest NHV value (on the right end) is included in Table 2. As expected, the configuration found by the meta-optimizer uses the unbounded external archive.

In Fig. 4, we compare the approximation front computed with NSGA-II and the one computed with the configuration found the meta-optimizer using AutoNSGAII. In both cases, we use 40000 function evaluations as stopping criterion. The graph shows remarkable differences between the front computed by NSGA-II (poor convergence and coverage of the Pareto front approximation) and AutoNSGAII.

Experiment 3 In alignment with existing work [9], we set the stopping criterion of AutoNSGAII to 25000 evaluations for this experiment. The evolution of the fronts over different number of evaluations is shown in Fig. 5. As in the previous experiment, the point with the minimum NHV value is taken and its corresponding configuration is used to compare with the results reported in [9]. The comparison in this case includes NSGA-II, and SMPSO [21] with their default settings and AutoNSGAII with the mentioned configuration.

The chosen configuration from the meta-optimizer is summarized in Table 2. Interestingly, this configuration is similar to the one computed in [9]: both share the use use BLX_ALPHA crossover and an external bounded archive).

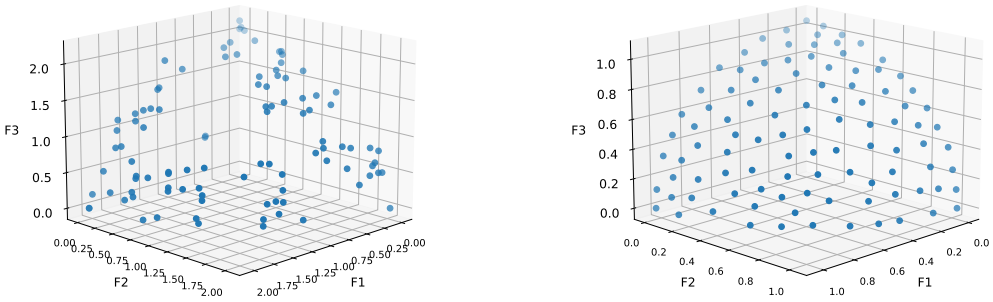


Fig. 4: Problem DTLZ3. Pareto front approximation found by the standard NSGA-II (left), and Pareto front approximation found by the auto-designed NSGA-II (right).

Table 3 showcase the validation results of the auto-designed NSGA-II for the WFG and DTLZ benchmarks. Tables (a) and (b) contains the Hypervolume indicator values and Tables (c) and (d) the Epsilon ones. As a general remark, the configurations found by our proposal yield similar indicator values (each cell includes the median of 25 independent runs) than those presented in previous work [9]. Additionally, this results are also supported by the Wilcoxon rank sum statistical test for significance. The results of the Wilcoxon test are included in Table 4. We can observe that statistical confidence has been found in most of the results.

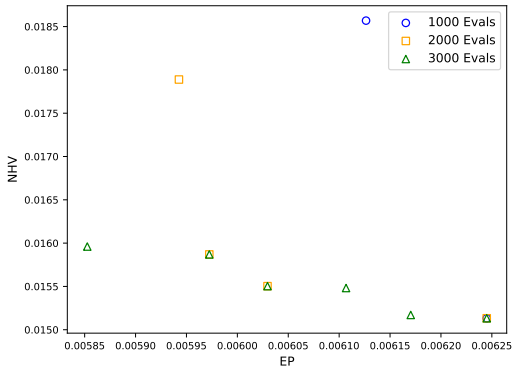


Fig. 5: WFG problem family. Evolution of the front generated by the meta-optimizer.

4 Discussion

Once we have conducted the two defined experiments, we revisit the two formulated research questions in the introduction, and we attempt to answer them based on the obtained results.

4.1 Research Questions

RQ1 - approach complexity: Our meta-optimization package relies only on jMetal code, so it does not require any external tool. The AutoNSGAII template within jMetal, designed and used in former studies, combined with irace simplifies the formulation of the auto-design of NSGA-II as a continuous optimization problem. We hope that researchers familiar with jMetal can benefit from the use of the meta-optimizer with little effort.

	NSGAII	SMPSO	AutoNSGAII
WFG1	$4.35e-01_{1.8e-01}$	$1.17e-01_{8.0e-03}$	$6.34e-01_{1.8e-05}$
WFG2	$5.61e-01_{2.3e-03}$	$5.61e-01_{1.6e-03}$	$5.64e-01_{9.2e-05}$
WFG3	$4.92e-01_{8.3e-04}$	$4.92e-01_{6.1e-04}$	$4.95e-01_{6.1e-05}$
WFG4	$2.17e-01_{3.7e-04}$	$2.03e-01_{2.4e-03}$	$2.18e-01_{1.5e-03}$
WFG5	$1.95e-01_{3.5e-04}$	$1.96e-01_{7.8e-05}$	$1.96e-01_{9.8e-05}$
WFG6	$2.01e-01_{1.3e-02}$	$2.09e-01_{5.0e-04}$	$2.02e-01_{1.4e-02}$
WFG7	$2.09e-01_{5.5e-04}$	$2.09e-01_{2.7e-04}$	$2.11e-01_{3.5e-05}$
WFG8	$1.47e-01_{2.7e-03}$	$1.47e-01_{3.4e-03}$	$1.40e-01_{3.1e-03}$
WFG9	$2.37e-01_{1.8e-03}$	$2.35e-01_{5.8e-04}$	$2.39e-01_{2.0e-03}$
DTLZ1	$4.88e-01_{7.9e-03}$	$4.94e-01_{2.7e-04}$	$0.00e+00_{4.9e-01}$
DTLZ2	$2.09e-01_{4.7e-04}$	$2.10e-01_{1.6e-04}$	$2.11e-01_{3.6e-05}$
DTLZ3	$0.00e+00_{1.8e-02}$	$2.10e-01_{1.2e-01}$	$0.00e+00_{0.0e+00}$
DTLZ4	$2.09e-01_{2.1e-01}$	$2.10e-01_{8.7e-05}$	$2.11e-01_{7.2e-05}$
DTLZ5	$2.11e-01_{3.4e-04}$	$2.12e-01_{2.2e-04}$	$2.12e-01_{4.5e-05}$
DTLZ6	$1.82e-01_{3.6e-02}$	$2.12e-01_{8.1e-05}$	$2.12e-01_{4.2e-05}$
DTLZ7	$3.34e-01_{3.0e-04}$	$3.35e-01_{1.2e-04}$	$3.35e-01_{9.2e-05}$

(a) Current study results using as objective the Hypervolume.

	NSGAII	SMPSO	AutoNSGAII
WFG1	$4.49e-01_{7.6e-02}$	$1.16e-01_{7.7e-03}$	$6.34e-01_{2.6e-05}$
WFG2	$5.64e-01_{9.5e-04}$	$5.62e-01_{1.2e-03}$	$5.65e-01_{5.1e-05}$
WFG3	$4.41e-01_{3.8e-04}$	$4.41e-01_{2.2e-04}$	$4.42e-01_{1.1e-05}$
WFG4	$2.17e-01_{7.6e-04}$	$2.03e-01_{2.4e-03}$	$2.17e-01_{3.0e-03}$
WFG5	$1.95e-01_{2.9e-04}$	$1.96e-01_{7.5e-05}$	$1.96e-01_{1.0e-04}$
WFG6	$2.03e-01_{8.9e-03}$	$2.09e-01_{4.3e-04}$	$2.08e-01_{1.3e-02}$
WFG7	$2.09e-01_{3.5e-04}$	$2.09e-01_{3.2e-04}$	$2.11e-01_{3.1e-05}$
WFG8	$1.48e-01_{2.3e-02}$	$1.48e-01_{1.0e-03}$	$1.39e-01_{2.3e-03}$
WFG9	$2.37e-01_{2.9e-03}$	$2.35e-01_{8.2e-04}$	$2.39e-01_{1.9e-03}$
DTLZ1	$4.66e-01_{1.6e-01}$	$4.94e-01_{1.9e-04}$	$0.00e+00_{0.0e+00}$
DTLZ2	$2.09e-01_{2.7e-04}$	$2.10e-01_{1.5e-04}$	$2.11e-01_{4.1e-05}$
DTLZ3	$0.00e+00_{0.0e+00}$	$2.10e-01_{6.3e-02}$	$0.00e+00_{0.0e+00}$
DTLZ4	$2.10e-01_{7.1e-04}$	$2.10e-01_{1.5e-04}$	$2.11e-01_{4.2e-05}$
DTLZ5	$2.11e-01_{3.5e-04}$	$2.12e-01_{1.3e-04}$	$2.12e-01_{4.1e-05}$
DTLZ6	$1.89e-05_{1.4e-03}$	$2.12e-01_{6.9e-05}$	$2.12e-01_{5.6e-05}$
DTLZ7	$3.29e-01_{2.8e-04}$	$3.30e-01_{9.8e-05}$	$3.30e-01_{7.3e-05}$

(b) Results obtained from [9] with irace using as objective the Hypervolume.

	NSGAII	SMPSO	AutoNSGAII
WFG1	$2.94e-01_{2.7e-01}$	$4.56e-01_{1.3e-02}$	$6.19e-03_{5.6e-04}$
WFG2	$1.81e-01_{1.7e-01}$	$6.70e-03_{2.8e-03}$	$4.03e-03_{5.0e-04}$
WFG3	$1.33e-02_{2.8e-03}$	$7.39e-03_{8.4e-04}$	$5.40e-03_{2.2e-04}$
WFG4	$1.21e-02_{3.6e-03}$	$2.19e-02_{2.6e-03}$	$6.33e-03_{1.1e-03}$
WFG5	$3.31e-02_{2.8e-03}$	$2.78e-02_{3.9e-04}$	$2.76e-02_{1.8e-04}$
WFG6	$1.49e-02_{1.0e-02}$	$6.19e-03_{5.9e-04}$	$1.00e-02_{9.0e-03}$
WFG7	$1.20e-02_{4.4e-03}$	$6.34e-03_{9.1e-04}$	$5.18e-03_{2.6e-04}$
WFG8	$2.44e-01_{1.0e-01}$	$1.75e-01_{2.1e-02}$	$2.45e-01_{1.0e-03}$
WFG9	$1.47e-02_{2.7e-03}$	$1.12e-02_{1.2e-03}$	$7.19e-03_{1.6e-03}$
DTLZ1	$1.60e-02_{5.2e-03}$	$6.30e-03_{7.0e-04}$	$5.16e-01_{1.0e+00}$
DTLZ2	$1.23e-02_{2.8e-03}$	$5.59e-03_{3.7e-04}$	$5.23e-03_{2.2e-04}$
DTLZ3	$1.14e+00_{1.4e+00}$	$6.20e-03_{7.0e-01}$	$1.16e+01_{8.4e+00}$
DTLZ4	$1.18e-02_{9.9e-01}$	$5.68e-03_{3.7e-04}$	$5.38e-03_{3.0e-04}$
DTLZ5	$1.14e-02_{2.5e-03}$	$5.25e-03_{2.5e-04}$	$5.05e-03_{2.1e-04}$
DTLZ6	$2.77e-02_{2.4e-02}$	$5.22e-03_{6.6e-04}$	$5.09e-03_{1.9e-04}$
DTLZ7	$1.02e-02_{3.5e-03}$	$4.68e-03_{4.6e-04}$	$4.50e-03_{3.5e-04}$

(c) Current study results using as objective the Epsilon.

	NSGAII	SMPSO	AutoNSGAII
WFG1	$4.52e-01_{2.4e-01}$	$4.55e-01_{9.8e-03}$	$5.99e-03_{7.3e-04}$
WFG2	$5.41e-03_{2.4e-03}$	$6.04e-03_{1.1e-03}$	$3.88e-03_{5.5e-04}$
WFG3	$3.34e-01_{5.3e-04}$	$3.34e-01_{1.9e-04}$	$3.33e-01_{2.7e-07}$
WFG4	$1.29e-02_{2.2e-03}$	$2.16e-02_{3.4e-03}$	$6.72e-03_{1.2e-03}$
WFG5	$3.31e-02_{3.7e-03}$	$2.77e-02_{1.9e-04}$	$2.76e-02_{1.6e-04}$
WFG6	$1.50e-02_{5.9e-03}$	$6.37e-03_{4.9e-04}$	$6.13e-03_{7.2e-03}$
WFG7	$1.28e-02_{4.0e-03}$	$6.52e-03_{4.9e-04}$	$5.20e-03_{1.9e-04}$
WFG8	$1.68e-01_{1.0e-01}$	$1.69e-01_{1.7e-02}$	$2.45e-01_{1.3e-03}$
WFG9	$1.42e-02_{2.0e-03}$	$1.10e-02_{1.9e-03}$	$7.39e-03_{1.2e-03}$
DTLZ1	$3.53e-02_{1.5e-01}$	$6.30e-03_{5.5e-04}$	$3.08e+01_{1.7e+01}$
DTLZ2	$1.12e-02_{4.2e-03}$	$5.53e-03_{3.2e-04}$	$5.28e-03_{2.5e-04}$
DTLZ3	$1.00e+01_{6.4e+00}$	$5.97e-03_{3.5e-01}$	$1.05e+02_{3.6e+01}$
DTLZ4	$1.18e-02_{5.4e-03}$	$5.61e-03_{3.1e-04}$	$5.32e-03_{1.9e-04}$
DTLZ5	$1.01e-02_{2.2e-03}$	$5.12e-03_{3.5e-04}$	$5.03e-03_{2.7e-04}$
DTLZ6	$3.72e-01_{5.3e-02}$	$5.15e-03_{4.5e-04}$	$5.06e-03_{2.9e-04}$
DTLZ7	$7.78e-03_{2.6e-03}$	$4.30e-03_{2.3e-04}$	$4.06e-03_{2.8e-04}$

(d) Results obtained from [9] with irace using as objective the Epsilon.

Table 3: The cells include the median and interquartile range of 25 independent runs. The dark-grey and light-grey background cells indicate, respectively, the best and second best indicator values.

	SMPSO	AutoNSGAII
NSGAII	▲ – ▽ ▲ ▽ ▽ ▽ – ▲ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ – ▽ – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ – ▲ ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ –

(a) Current study results using as objective the Hypervolume.

	SMPSO	AutoNSGAII
NSGAII	▲ ▲ ▽ ▲ ▽ ▽ – – ▲ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ – ▽ – ▽ ▲ ▽ ▲ ▽ – ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ – – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ ▽

(b) Results obtained from [9] with irace using as objective the Hypervolume.

	SMPSO	AutoNSGAII
NSGAII	▲ ▽ ▽ ▲ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▲ ▽ – ▽ ▲ ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ –

(c) Current study results using as objective the Epsilon.

	SMPSO	AutoNSGAII
NSGAII	– – ▽ ▲ ▽ ▽ ▽ – ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ ▽ ▽ – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ – ▽

(d) Results obtained from [9] with irace using as objective the Epsilon.

Table 4: Wilcoxon rank sum test results. The symbols in each cell correspond to problems WFG1-9 and DTLZ1-7. The symbols indicate: "–" no stadistical significance, "▲" the algorithm in the row has a better indicator value than the algorithm in the row with confidence and "▽" the algorithm in the row has a worse indicator value than the algorithm in the row with confidence

RQ2 - finding of accurate designs: We have adopted a simple encoding for the NSGA-II configurations consisting in codifying each parameters as a floating point value in the range $[0.0, 1.0]$, and these values are further decoded into a string that used as the input of the AutoNSGAII template. This configuration has been proved effectively by our empirical experiments. The first two experiments showed that the meta-optimizer has been able to generate the expected key components required by NSGA-II to converge to the Pareto front of the selected problems. For these experiments we provided visual evidence. Additionally, the generalization capabilities of the autotuner were challenged by requiring it to find an accurate configuration for a training set composed of nine problems, and validating the found configurations on a set of additional seven problems. The experiment we have accomplished shows almost identical results to the previously published work were irace was used as auto-configuration tool.

4.2 Further Remarks

Our empirical evaluation also revealed a few issues that are worth discussing:

- The formulation of searching designs for NSGA-II as a continuous problem opens the opportunity of using most of the metaheuristics provided by jMetal as meta-optimizers. This enable the easy development comparative studies based on configuring AutoNSGAII with different training sets.
- Although we have used two quality indicators, EP and NHV, as objectives for guiding the search, the inclusion of additional ones (e.g., spread, inverted generational distance, etc.) could reveal new insights regarding the configurations for solving different problems.
- We used NSGA-II with standard settings as the meta-optimizer. The obtained results in this paper could be used in order to analyze whether its performance could be improved if using different parameter settings.
- We have performed only a run of the meta-optimizer in the experiments. A deeper study should be carried out by performing a number of independent runs and making statistical analysis of the results.

5 Conclusions and Future Work

In this paper we have presented a study in which the NSGA-II algorithm is used as a meta-optimizer, i.e., as a tool that, given a set of problems as training set, is aimed at finding configurations that include NSGA-II parameters and components. By using a simple encoding scheme and the features existing in jMetal that were developed in former studies, our proposal is 100% developed in jMetal, so no external tools are required.

We have defined an experimentation to validate our proposal considering two scenarios and three experiments to cover both automatic search of NSGA-II designs for single and multi-problem training sets. The outcomes of these experiments reveal that the meta-optimizer is able of finding configurations of NSGA-II that successfully achieve the defined goals.

We have indicated a number of open research lines in the discussion section. Additionally, to reduce the computing time of the meta-optimization, we have set in Experiments 1 and 2 a number of function evaluations which is lower than the used when validating the configurations; we are interested in studying to what extent the number of evaluations can be reduced in the search while the resulting NSGA-II designs are able of solving the problems efficiently.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Innovation via Grant PID2020-112540RB-C41 (AEI/FEDER, UE). José F. Aldana-Martín is supported by Grant PRE2021-098594 (Spanish Ministry of Science, Innovation and Universities).

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6** (2002) 182–197
2. Huang, C., Li, Y., Yao, X.: A survey of automatic parameter tuning methods for metaheuristics. IEEE Trans. Evol. Comput. **24** (2020) 201–216
3. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3** (2016) 43–58
4. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: An automatic algorithm configuration framework. J. Artif. Int. Res. **36** (2009) 267–306
5. Yi, W., Qu, R., Jiao, L., Niu, B.: Automated design of metaheuristics using reinforcement learning within a novel general search framework. IEEE Transactions on Evolutionary Computation (2022) 1–1
6. Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., Hutter, F.: Smac3: A versatile bayesian optimization package for hyperparameter optimization. Journal of Machine Learning Research **23** (2022) 1–9
7. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: Automatic component-wise design of multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation **20** (2016) 403–417
8. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T. In: Automatic Configuration of Multi-objective Optimizers and Multi-objective Configuration. Springer International Publishing, Cham (2020) 69–92
9. Nebro, A.J., López-Ibáñez, M., Barba-González, C., García-Nieto, J.: Automatic configuration of NSGA-II with jMetal and irace. Genetic and Evolutionary Computation Conference (2019) 1374–1381
10. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation **7** (2003) 117–132
11. Durillo, J.J., Nebro, A.J.: jMetal: A java framework for multi-objective optimization. Advances in Engineering Software **42** (2011) 760–771
12. Nebro, A.J., Durillo, J.J., Vergne, M.: Redesigning the jMetal multi-objective optimization framework. Genetic and Evolutionary Computation Conference (2015) 1093–1100
13. Durillo, J.J., Nebro, A.J., Luna, F., Alba, E.: On the effect of the steady-state selection scheme in multi-objective genetic algorithms. In Ehrgott, M., Fonseca, C.M., Gandibleux, X., Hao, J.K., Sevaux, M., eds.: Evolutionary Multi-Criterion Optimization, Berlin, Heidelberg, Springer Berlin Heidelberg (2009) 183–197
14. Nebro, A.J., Galeano-Brajones, J., Luna, F., Coello Coello, C.A.: Is nsga-ii ready for large-scale multi-objective optimization? Mathematical and Computational Applications **27** (2022)
15. Doblas, D., Nebro, A.J., López-Ibáñez, M., García-Nieto, J., Coello Coello, C.A.: Automatic design of multi-objective particle swarm optimizers. In Dorigo, M., Hamann, H., López-Ibáñez, M., García-Nieto, J., Engelbrecht, A., Pinciroli, C., Strobel, V., Camacho-Villalón, C., eds.: Swarm Intelligence, Cham, Springer International Publishing (2022) 28–40
16. Durillo, J.J., Nebro, A.J., Luna, F., Alba, E.: A study of master-slave approaches to parallelize nsga-ii. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. (2008) 1–8
17. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation **8** (2000) 173–195
18. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable Test Problems for Evolutionary Multi-Objective Optimization. In Abraham, A., Jain, L., Goldberg, R., eds.: Evolutionary Multiobjective Optimization. Theoretical Advances and Applications. Springer (2001) 105–145
19. Ishibuchi, H., Pang, L.M., Shang, K.: A new framework of evolutionary multi-objective algorithms with an unbounded external archive. In Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J., eds.: ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020). Volume 325 of Frontiers in Artificial Intelligence and Applications., IOS Press (2020) 283–290
20. Huband, S., Hingston, P., Barone, L., While, L.: A review of multi-objective test problems and a scalable test problem toolkit. IEEE Transactions on Evolutionary Computation **10** (2006) 477–506
21. Nebro, A., Durilllo, J., García-Nieto, J., Coello Coello, C., Luna, F., Alba, E.: SMPSO: a new pso-based metaheuristic for multi-objective optimization. In: 2009 IEEE Symposium on Computational Intelligence in Multi-criteria decision-making (MCDM 2009), IEEE Computer Society (2009) 66–73

EVADyR: a new dynamic resampling algorithm for optimizing noisy expensive systems

S. Robert[1], S. Zertal[2], and P. Couvée[1]

[1] Atos BDS RD Data Management
Echirolles, France
sophie.robert@atos.net
philippe.couvee@atos.net
[2] University of UPSacaly-UVSQ
Guyancourt, France
soraya.zertal@uvsq.fr

Abstract. Black-box auto-tuning methods have been proven to be efficient for tuning configurable computer appliance. However, because of the shared nature and the complexity of the software and hardware stack of some systems such as cloud or HPC systems, the measurement of the performance function can be tainted by noise during the tuning process, which can reduce and sometime prevent, the benefit of auto-tuning. An usual choice is to add a resampling step at each iteration to reduce uncertainty, but this approach can be time-consuming. In this paper, we propose a new resampling and filtering algorithm called EVADyR (Efficient Value Aware Dynamic Resampling). This algorithm is able to tune efficiently a prefetching strategy in the case of multiple parallel accesses. Because it finds a better exploration versus exploitation trade-off by resampling only promising parametrizations and increases the level of confidence around the suggested solution as the tuning process advances, it outperforms state of the art dynamic resampling by reducing the distance to the optimum by 93.5%, as well as speed-up the experiment duration by 45.8% because less iterations are needed to reach the found optimum. An additional proof of this study is the demonstration of the importance of using noise reduction strategies for the optimization of highly shared resources such as HPC or cloud systems.

Mots-Clefs. Auto-tuning, Black-box optimization, High Performance Computing, Stochastic Optimization, Resampling

1 Introduction

Most of the software of modern computer systems come with many configurable parameters that control the system's behavior and its interaction with the underlying hardware. These parameters are challenging to tune by solely relying on field insight and user expertise, due to huge parametric spaces and complex, non-linear system behavior and environment variations. Consequently, users often have to rely on the default parameters and do not take advantage of the possible performance that the system could deliver for their applications. As users are not easily able to take adequate decisions for the parametrization of complex systems, new tuning methods, usually called *auto-tuning* methods, have emerged from the optimization and machine learning fields to automate parameter selection depending on the current workload. They have been successfully applied to a wide range of systems, such as storage systems, database management systems and compilers.

Among existing auto-tuning methods, black-box optimization has been demonstrated to be successful for tuning a wide range of computer systems. However, these studies were done under the assumption that the tuned system is deterministic, *i.e.* a given parametrization will always yield the same execution time. While this hypothesis is valid when working in a controlled and exclusive test environment, it does not always hold for systems that rely on shared resources, such as cloud or HPC systems. Due to the high cost of dedicating exclusive resources, complex system auto-tuning often occurs in shared environments, requiring automatic tuning methods to consider potential interference, which we will call "noise", that can degrade the performance of traditional auto-tuning heuristics, as the performance of the system being tuned can vary due to fluctuations in resource availability or the presence of competing tasks, independently from the tested parameters. The black-box optimizer risks matching the input parameters with the random variations in the performance, rather than the performance itself. Sub-optimal parameter configuration that appears good due to

favorable conditions, such as low system activity, can be retained, while good parameter configurations may be rejected if they were evaluated under challenging circumstances, such as high usage or storage backup events. Consequently, to be used on modern shared systems, auto-tuning algorithms must be able to distinguish between the true system behavior and random variations, while remaining as sparse as possible in terms of iterations.
In this paper, we consider the noise present when auto-tuning a highly configurable appliance aimed at reducing I/O contention in High Performance Computing (HPC) systems, called the *Small Read Optimizer* (SRO), and suggest a new resampling algorithm, called EVADyR (**E**fficient **V**alue **A**ware **Dy**namic **R**esampling), and show that it outperforms already existing state-of-the-art.
The main contributions of this paper are:

- The proposition of the original EVADyR resampling algorithm based on dynamic resampling but specifically tailored to the optimization of noisy and expensive function;
- The highlighting of the importance of using noise reduction when tuning systems in highly shared production environment;
- The validation of this new algorithm on a real-life I/O accelerator with different noisy context, which compared to state-of-the-art dynamic resampling provides an improvement of the quality of convergence of 93.5% and a speed-up of the experiment duration by 45.76% for the tuned system.

This paper is organized as follows. Section 2 covers related works and section 3 the main principles of Bayesian Optimization and some of the state of the art noise reduction techniques. Section 4 presents the proposed EVADyR algorithm to improve resampling algorithms, section 5 the experiment plan and section 6 the obtained results. Section 7 concludes this study by giving some insights into some future work.

2 Related works

In the literature, black-box optimization is a popular choice for tuning different reconfigurable systems, and it has been particularly helpful in computer science to look for the optimal configurations of various software and hardware components. In the field of Big Data Processing systems, software that are notoriously hard to tune such as Hadoop, Spark and Storm, have benefited from black-box optimization. It is for example the methodology chosen by Liao et al. in their Gunther framework [24] which relies on Genetic Algorithms to find the optimal parametrization of Hadoop. In [20], Jamshidi et al. use Bayesian Optimization to optimize the stream processing system Storm. In [9], Desani et al. compare two derivative-free methods (Bounded Optimization BY Quadratic Approximation method and Constrained Optimization BY Linear Approximation method) to find the optimal configuration of the Hadoop framework. In Database Management Systems (DBMS), tuning relying on surrogate modeling with expected improvement and pruning strategies have been suggested through the iTuned framework designed by Duan et al. in [11]. This framework has significantly improved the performance of PostgreSQL for different workloads. Also, using black-box optimization to optimize query has also proven its efficiency using simulated annealing [19] and genetic algorithms [5].
Storage systems are hard to tune as they have both a very large parametric space with a strong dependence on the running workload [7]. The efficiency of black-box optimization for tuning storage systems when faced with different workloads has been explored by Cao et al. in [8], where they propose a thorough analysis of the behavior of each heuristic. Reinforcement learning has also been successfully used as an auto-tuner to optimize the performance of the Lustre filesystem in data center storage systems by Li et al. in [23] and an optimal parametrization for the several layers of HDF5 library was found using genetic algorithms by Behzad et al. in [4]. An extension of this auto-tuner which selects the best parameters according to the I/O pattern is described in [3] by the same authors.
Within the HPC community, auto-tuning has gained a lot of attention for tuning particular HPC application and improve their portability across architectures. In [35], Seymour et al. provide a comparison of several random-based heuristic searches (Simulated annealing, genetic algorithms ...) that have provided some good results when used for code auto-tuning. Bergstra has had good results as well in this field with surrogate modeling using boosted regression trees [6]. In [26], Menon et al. use Bayesian Optimization and suggest the framework HiPerBOT to tune application parameters as well as compiler runtime settings. HPC systems energy consumption can also benefit from Bayesian Optimization, as Miyazaki et al. have shown in [27] where an auto-tuner based on a combination of Gaussian Process regression and the Expected Improvement acquisition function has raised their cluster to the Green500 list. A scheduling algorithm using genetic algorithms has been introduced by

Kassab et al. in [21] and manages to schedule jobs under a limited energy power constraint. The MPI community has also shown the superiority of a hill-climbing black-box algorithm over an exhaustive sampling of the parametric space in [12].

The literature addressing the problem of noise when tuning real systems is surprisingly sparse, as most of the works detailed in the previous section do not study the potential interference on the tuned system and do not mention their tuner resilience to possible interference in shared settings. While this is not critical when working on single user systems, such as local hard drives for personal computers [8], it cannot be ignored when working on highly parallel shared systems [28]. Several studies do acknowledge their system's noise [7], but do not provide any practical solution to make the tuner resilient, other than computing the mean or the median of the found best parametrization repeated several times [34] [17]. To our knowledge, the only notable exception is the Baloo framework [16] developed by Grohmann et al. for tuning distributed database systems, which performs adaptive sampling until the confidence interval around the mean is smaller than a set threshold, or until the maximum number of reevaluation allowed per parametrization is reached.

3 Bayesian Optimization for noisy systems

3.1 Problem formalization

Formally, let S be the system to optimize and θ_i its parametrization from the possible parametrizations of the tunable system $\Theta = \{\theta_i\}_{i\in\mathbb{N}}$. Let $\mathcal{A}$ be the optimized application and $\mathcal{E}$ the execution context (the underlying hardware, the number of nodes...) for which we optimize the application.

Let $f : (\mathcal{A},\ \mathcal{E}, \Theta) \longrightarrow \mathbb{R},\ \theta \longrightarrow f(\theta)$ represent the execution time for parametrization θ. Because of the possible random interference depending on the system's state, we only have access to the observed execution time $F(\theta)$. These interference, which we will refer to as noise, can depend on the parametrization or the optimization step and is thus a function of θ and n, represented by $\epsilon(\theta, n)$. The vector corresponding to the different sampled values at parameter θ will be denoted as $F(\theta)_{1\leq j\leq j_\theta}^{j}$, j_θ being the number of samples for parameter θ. The estimation of f at θ is denoted $\hat{f}(\theta)$.

With these notations, the optimizer must solve the following problem:

$$min\ \mathbb{E}(F(\theta)),\ \theta \in \Theta$$
$$F(\theta) = f(\theta) + \epsilon(\theta, n)$$

This formula holds in the case of minimization (for example when minimizing the execution time) and in the case of maximization (for example of the throughput), one can simply minimize $-f$. From this definition, we find that in this noisy framework the optimal parametrization is not the one leading to the quickest run but the one corresponding to the optimal execution time on average. This ensures that the optimum is not a result of chance. Throughout the remainder of this paper, the term "optimum" will refer to this average optimal parametrization.

3.2 Bayesian optimization for auto-tuning

Black-box optimization is a method for optimizing a function with unknown properties that can only be evaluated a limited number of times. In the context of tuning computer systems, it involves optimizing the performance of an application based on the relationship between input parameters (configurable component and execution context) and the output (application performance) without having any insight into the internal workings of the system.

Two distinct steps are necessary for black-box optimization: the selection of initial parameters, such as Latin Hypercube Designs [39], and an optimization heuristic selecting at each iteration the most promising configuration.

Many heuristics have been designed and tested, among which Bayesian Optimization (also known as Sequential Model Based Optimization). It is an efficient and versatile method for tuning complex systems, which uses a cheaper to evaluate probabilistic model to approximate the performance function. Bayesian Optimization requires two components: an acquisition function that selects the next data point to evaluate and a probabilistic model that represents the performance function. In our experiments, based on previous study results in [33], we will use Expected Improvement [39] as an acquisition function and Gaussian processes [31] as a probabilistic model. The optimization process will continue until the improvement from the current best objective function value is below a set threshold over a number of iterations.

3.3 Stochastic black-box optimization

While almost non-existent in the system's tuning community, black-box optimization with noisy fitness is a very proficient field when it comes to theoretical research on synthetic benchmarking function. The available research can be broadly split into two main categories : *heuristic specific noise reduction* and *heuristic agnostic noise reduction*.
In the case of Bayesian Optimization, a possible improvement is the modification of acquisition functions in order to make them handle noisy observations better. For instance, Gramacy et al. in [15] and Vàsquez et al. in [40] use respectively the mean and a quantile as an estimation of the performance function through Gaussian Processes when using Bayesian Optimization. Letham et al. propose in [22] a novel way of defining expected improvement, called *Noisy Expected Improvement*, using Quasi Monte Carlo simulation. In [18], Huang et al. suggest using the *Augmented Expected Improvement*, which uses a robust estimation of the best performing parametrization by defining the best solution as the one with the lowest β-quantile, with β a configurable value. A similar quantile based approach is proposed by Picheny et al. in [30] where they suggest using the *Expected Quantile Improvement* to select the next data point to evaluate. In [14], Forrester et al. suggest using a reinterpolation procedure which uses the results of a Gaussian Process regressor on noisy observations into another interpolation model which will be used to compute the Expected Improvement. While these different methods have proven to be efficient when facing different noises on different benchmarking functions [29], they have the major drawback of being specific to the selected optimization and cannot be generalized to other heuristics, such as genetic algorithms or simulated annealing, even though there is no single best performing heuristic for every optimization problem [33, 7, 41]. For this reason, we decide to take another, more versatile, approach that can be used with any optimization heuristic: resampling [2] [13] [37] [36] which is one of the most popular method in this category. We focus on it in this paper as there is no single best performing heuristic for every optimization problem [42], as demonstrated in some of our previous works [33], and we aim to improve noise reduction methods that allow switching heuristics depending on the context and optimization use-case.

3.4 Resampling techniques

Resampling consists in adding a "resampling filter" by using a set logical rule to select which parametrization to reevaluate, as shown in figure 1. Its goal is to reduce the standard deviation of the mean of an objective value in order to augment the knowledge of the impact of the parameter on the performance. Resampling is a trade-off between having a better knowledge of the space and wasting some computing times on re-evaluation. Many strategies exist in order to efficiently reevaluate a parametrization, and we present here two of the most popular: simple and dynamic resampling using SEDR (Standard Error Dynamic Resampling).

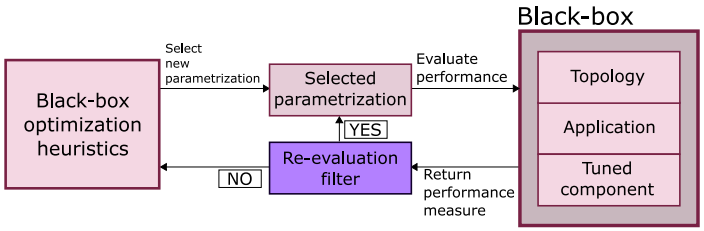


Fig. 1. Schematic representation of resampling algorithms

Simple resampling Simple resampling consists in evaluating for a fixed number of times the fitness value of the selected parametrization [13], regardless of the parametrization and its fitness. In our case, it consists in launching a fixed number of times the application and the tuned system with the same selected parametrization. Its main drawbacks is its lack of adaptivity to the noise.

Dynamic resampling Because the need for resampling is often not homogeneously distributed throughout the search space [36] (especially for shared and distributed systems), dynamic resampling methods that consider the performance variance around each data point have been introduced. Among the most popular, *Standard Error Dynamic Resampling* (SEDR) [10] adapts the number of samples to the noise strength measured at each parametrization by using a different number of evaluations n_θ for each one. The parametriza-

tion is re-evaluated until the 95% confidence interval around the mean is below a fixed threshold τ_{se}.
SEDR has proven its efficiency in theoretical [10] and practical problems [38] where the fitness value is not taken into account whenever setting a confidence interval threshold and require the same certainty regardless of the performance value. However, when working with real-systems, especially for applications which can take a long time to run, the length of the confidence interval should take into account the application's performance. Because of this, the authors in [32] suggest the definition of an interval width proportional to the currently measured mean for this parametrization, and it is this version of SEDR resampling that will be the basis for our EVADyR algorithm.

4 The EvaDyr algorithm: improving resampling methods

Existing resampling methods have proven to be efficient but still present several drawbacks, because:

1. **Resampling can take too many iterations on a single parametrization**, causing a waste of resources, because methods based on noise values can become focused on a particular parametrization with large deviation, even if the noise is temporary (data backup, unusual traffic on the system,..). This variation is already included in [16].
2. **Dependence on hyperparameters**: The success of dynamic and static resampling methods depends greatly on their hyperparameters (the number of resamples for simple resampling and the interval confidence width for dynamic resampling). If the threshold is set too high, no resampling will occur, and if it is too low, excessive resampling will occur, hindering exploration of the parametric space.
3. **No comparison with already tested parametrization**: Resampling does not consider the comparison between the current parametrization's fitness and the previously tested ones. This can lead to resampling slow parametrizations multiple times, slowing the convergence process and wasting the systems' resource.

To address these limitations, we suggest the EVADyR algorithm (**E**fficient **VA**lue **A**ware **Dy**namic **R**esampling), with several improvements:

Setting a limit to the number of resamples We add a floor limit N to the allowed number of resamples on a given parametrization. The number of resamples n_θ for parametrization θ is thus located between 2 (because each parametrization is resampled at least twice in order to measure the noise on this data point) and N. We set for the experiments the maximum number of resamples per parametrization to 10% of the total maximum allowed budget.

Dynamic confidence intervals Another improvement is the removal of the dependence on the hyperparameter threshold. To do so, we make the size of the confidence interval inversely proportional to the number of elapsed steps and use a bounded decreasing exponential to reduce the ratio of the confidence interval at each step. We also make it proportional to the mean measured for this parametrization, to consider the fact that the required precision on the mean estimator is dependent on the value of the mean.
The required confidence around the mean becomes smaller as the number of iterations raises, as we verify:

$$ci\_width(\theta) = 2 \times 1.96 \times \frac{\hat{\sigma}(\theta)}{\sqrt{j_\theta}} \leq FC(j_\theta) \times \hat{\mu}(\theta) = max(0.99^{j_\theta}, 0.1) \times \hat{\mu}(\theta)$$

This ensures that at the beginning of the optimization process, the algorithm is more lax about the precision of the true performance value to test many different parametrization, ensuring a satisfactory exploration of the parametric space. However, at the end of the optimization process, it is more precise about the knowledge we have about the parametrization ensuring an adequate exploitation of already known parametrizations.

Performance based resampling filter The last change we suggest is to introduce a dynamic filtering component before performing the resampling. Its goal is to filter the most promising parametrizations before submitting them to the resampling which reduces its time cost. The filtering process runs as follow:

1. Each time the heuristic suggests a new parametrization, it is evaluated at least twice
2. If the median of the related performance is inferior to a certain ratio of the current median, move to the next step, otherwise move to step 4
3. Keep this parametrization and submit it to the resampling process.
4. Discard this unpromising parametrization, go back to step 1 if the budget is not empty.

The ratio of the median used in step 2 can either be a fixed value or can be computed dynamically as a decreasing function of the number of elapsed iterations, similarly to dynamic interval definition. As the optimization progresses, this filter ensures that the algorithm is more and more strict about the quality of the resampled solutions, so that last iterations are not wasted on parametrization that are not promising. As the optimization process draws to an end, we make sure that we do not waste any of the remaining resources.

5 Evaluation of EVADyR's performance

To evaluate the relevance of noise reduction and compare EVADyR to the state-of-the-art on real-life test case, we perform the tuning on a notoriously noisy and highly shared environment by tuning an I/O accelerator, aimed at reducing I/O contention on large scale HPC systems. This I/O accelerator, called the Small Read Optimizer (`SRO`) is a dynamic data preload strategy that prefetches frequently accessed file chunks into the memory of the compute node. It detects repeatedly accessed zones in a file and loads the entire zone into memory. This method takes into account both temporal and spatial factors, making it more effective than the Linux read-ahead strategy, which only loads one zone at a time. The behavior of the software can be greatly impacted by the four positive and discrete dimensions in its parametric space [33] [32].

5.1 Interference generation

To validate the relevance of our algorithm and compare it to state of the art, we run an I/O heavy benchmark performing pseudo-random read accesses, in a noisy setting created by performing concurrent accesses on the file read by the benchmark. This type of direct interference is common when running applications with different concurrent nodes that require using the same data, which is common in shared systems [25]. It creates an intense, localized noise, that affects periodically and negatively the system's performance.

Tuned application and potential auto-tuning improvement An I/O intensive application was selected that performs 4k operations on a 500GB file with 500 hotspots of width 50MB. The hotspots have 10000 random accesses before moving on to the next file zone. This access pattern is common in the case of applications accessing different zones of a data file (such as in a 3D model), performing many random accesses within a zone if it's considered interesting, before moving on to the next zone.
The benchmark runs on a single compute node. It consists in an Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz with 16 physical cores (32 logical cores), bi-socket, and 82 GB of DDR4-DRAM. The back-end parallel filesystem is a Lustre bay of 40TB. The storage system is isolated from the rest of the users in order to have a fine control over the generated noise and ensure the precision of our study.
To determine the optimal parameters of the accelerator for the application to use as the ground truth for comparison in a noiseless settings, auto-tuning experiments were run 20 times using Bayesian Optimization on isolated nodes and storage systems to eliminate interference. The maximum number of iterations is set to 100 and the optimization process stops if there is less than 5% improvement in the optimum over 15 iterations.
The collected statistical estimators over the 20 noiseless experiments show that all optimization runs converge towards the same optimal performance and the average potential tuning improvement, corresponding to the distance to the default parametrization, is located around 75.33%, with a standard error of 1.15%. The retained best performance, used as ground truth for the optimum performance of the application, is taken to be the average value of the best performance over all 15 experiments: 8.88 seconds, for an average improvement of 75.33%.

Noise generation methodology Noise is introduced by running parallel applications on other computing nodes that perform random operations on the same data file. Three nodes are used, with two nodes performing write operations and one performing read operations, corresponding to concurrent random accesses (reads or writes) across the input data file. The elapsed time between the arrival of each interference is defined as the time interval between the beginning and the end of the concurrent noise application and is expressed in seconds. Three different noise frequencies were tested, selected to match a certain frequency of arrival relative to the duration of the I/O benchmark with default parameters. We denote the experiments as `SRO.n`, n being the noise arrival time. For example, `SRO.60` will refer to the SRO experiment with interference coming every 60 seconds.
The impact of noise on the constant default parametrization is shown in figure 2. The noise profile varies depending on the frequency of arrival. At 60 seconds, the noise is constant, with each run translated to a higher value. At 120 seconds, the noise is less constant but the value

of the application does not return to its original value between runs, creating a challenging environment for the optimizer. At 300 seconds, a Cauchy-type noise is observed, with a run taking longer when hit by the noise but returning to the original value between runs. This test environment provides different types of challenges for the optimizer to validate its relevance in dealing with constant noise (60 seconds arrival), unstable noise (120 seconds arrival), and impulse-type noise (300 seconds arrival).

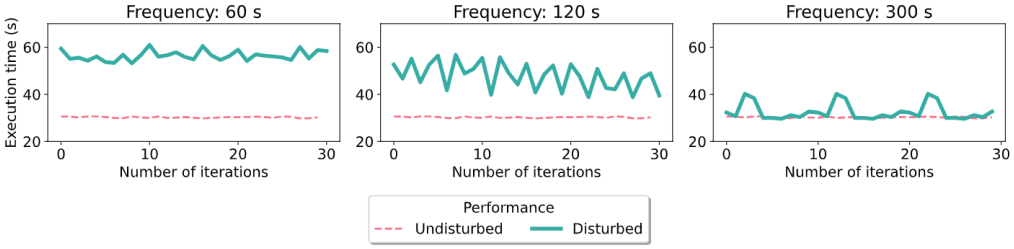


Fig. 2. Impact of noise on constant parametrization

Tested methods The black-box optimization heuristic is SMBO, using Gaussian Processes as the regression method and Expected Improvement as the acquisition function. The initialization method uses LHS, the number of initialization runs is set to 10. The maximum number of optimization runs is set to 100, resulting in a maximum total number of iterations of 110. The stop criterion stops the experiment automatically when there is less than a 5% improvement over the last 15 iterations. Each optimization process is repeated 5 times to average some of its random behavior.
The EVADyR algorithm is compared to static resampling (testing 3 and 5 resamples) and SEDR (using an interval width of 10% and 30% of the mean), as well as in the absence of any noise reduction strategy. As one of the major advantage of our methods is the absence of hyperparameters, we do not test different hyperparameters values for our solution.
All the code used to run the experiment is bundled in the SHAMan optimization framework, fully available as an Open-Source framework [1], and our experiments are thus fully reproducible.

5.2 Evaluation metrics

To evaluate the relevance of each optimization heuristic, we compute the different metrics described in table 1.

Table 1. Evaluation metrics for noisy optimization

Name	**Abbreviation**	**Description**
Distance to optimum parametrization	AvgDistOpt	Difference between the *average* of the found parametrization to the *best time on average*
Improvement compared to the default parametrization	ImprovDefault	Difference between the *average* of the found parametrization to the *average performance found for the default parametrization*
Total duration	Duration	Total duration of the experiment as the sum of the execution times at each iteration

The first metric *AvgDistOpt* is computed by measuring the distance between the average performance corresponding to the found parametrization and the optimum. This metric corresponds to measuring the asymptotic quality of the optimizer. It consists in running 15 times the parametrization found in the noisy case on the noiseless cluster and computing the mean. The metric $ImprovementDefault$ corresponds to the distance between the asymptotic value of the parametrization returned by the tuner and the average performance measured

at the default parametrization. It represents the interest of using an auto-tuner rather than simply using the default parametrization.
The third metrics *Duration* reflect the time taken by the optimization experiment before the automatic stop criterion stops it in terms of elapsed time. There is a trade-off between the convergence speed and the quality of the optimization, as the more we perform evaluations and resampling, the more we gain insight on the system's behavior, but the more expensive resources are spent. We are thus looking for an equilibrium between having a solution close to the optimum and minimizing the resources cost.

6 Results and discussion

6.1 On the importance of noise reduction for tuning noisy systems

The values of the different metrics for the experiments when using Bayesian Optimization without any noise reduction technique are presented in table 2.

Table 2. Optimization results without noise reduction

Experiment ID	Avg Dist Opt (%)	Improv. Default (%)	Duration (s)
SRO.60	75.54	56.70	1200.98
SRO.120	319.92	-3.58	1079.57
SRO.300	66.25	58.99	827.03

These experiments show that stochasticity makes the auto-tuner ineffective without noise reduction, resulting in wasted time and resources: in the case of `SRO.120`, the parametrization returned by the optimization process performs worse than the default parametrization, and in the case of `SRO.60` and `SRO.300` an improvement is observed compared to the default parametrization of respectively 56.70% and 58.99%, but nothing as high as the 75% that can be expected. The experiments show that the noise makes the optimization problem more difficult, even in the case of lower frequency noise, such as `SRO.300`, and affects the quality of the optimizer, causing the stop criterion to be reached and the optimization to be stopped, as can be seen by looking at the convergence speed in table 2. The noise impact on the regression model and the acquisition function is also noted as a reason for the decreased optimization results.

6.2 Using state of the art's algorithms

Impact of static resampling The metrics computed for the different number of resamples in the case of static resampling are available in table 3. The results of using static resampling show improvement in the quality of optimization performed by the auto-tuner compared to not using any noise reduction. For every experiment, there is a number of resamples that improves significantly the performance of the tuner when compared to not using any resampling process in terms of distance to the ground truth. In the case of the less noisy experiment `SRO.300`, it even comes close to the optimum and the maximum observed potential improvement. However, the optimization process is slowed down by the re-evaluation of parametrizations that are not very impacted by noise, which are re-evaluated the same number of times as others that are more subject to noise. The experiments confirm the drawbacks of using static resampling, as it is hard to find the right exploration-exploitation trade-off and some iterations are wasted on parametrization with low noise.

Table 3. Metrics on static resampling

Resamp.	Exp.	Avg DistOpt (%)	Improv. Default (%)	Duration (s)
3	**SRO.60**	23.57	69.52	2430.698
	SRO.120	30.92	67.71	1411.23
	SRO.300	38.65	65.80	1211.79
5	**SRO.60**	15.98	71.39	1946.672
	SRO.120	27.37	68.58	1843.915
	SRO.300	2.84	74.63	1632.083

Impact of dynamic resampling Dynamic resampling does not show any benefits compared to static resampling and in some cases performs worse than without using any noise reduction strategy, as shown in figure 4. The reason behind this is the difficulty of finding the right parametrization of the algorithm depending on the noise setting. Optimization trajectories show in the case of a 10% interval width that the parametrization can be resampled too much or too little depending on the frequency of the noise. In the case of a 30% interval width, the resampling interval is too large and the algorithm does not have enough data to have a precise estimation of the performance function. These results highlight the advantage of dynamic resampling over static resampling in the case of regular Cauchy noise and confirm some of the results found in the state of the art.

Table 4. Metrics on dynamic resampling

Width (%)	Exp.	Avg Dist Opt (%)	Improv. Default (%)	Duration (s)
10%	**SRO.60**	53.99	62.02	4304.41
	SRO.120	112.40	47.61	3131.82
	SRO.300	28.21	68.37	1620.60
30%	**SRO.60**	309.31	-0.96	2240.62
	SRO.120	308.15	-0.68	1994.496
	SRO.300	205.76	24.58	2014.72

6.3 Using EVaDyR

The values of the metrics computed for the experiments using the EVADyR algorithm with all the improvements suggested in section 4 are available in table 5. Using dynamic intervals and bounded dynamic resampling combines the benefits of both dynamic and static resampling. It eliminates the need for selecting a specific hyperparameter for the algorithm and adjusts the resampling rate to the requirements of the noisy setting. This leads to a significant improvement in the distance to the optimum compared to standard dynamic resampling: it results in a decrease of the distance from 53.99% to 8.91% for `SRO.60`, from 112.40% to 5.12% for `SRO.120` and from 28.21% to 2.39% for `SRO.300`. This is reflected as well in the improvement compared to the default parametrization, which is above 73%.

Table 5. Metrics value using EVaDyR

Exp.	Avg Dist Opt (%)	Improv. Default (%)	Duration (s)
SRO.60	5.81	73.90	2045.52
SRO.120	4.27	74.28	1556.82
SRO.300	2.57	74.70	1311.26

In terms of distance to the ground truth, adding a resampling filter allows to find results very close to the ground truth optimum, which brings a strong improvement compared to the default parametrization. Even in the case of the most noisy optimization problem `SRO.60`, the performance measured at the returned parametrization is only 5.81% away from the ground truth, with an absolute difference of 0.5 seconds, reaching almost the full optimization potential of the auto-tuner. In the case of `SRO.120`, the performance measured at the returned parametrization is 4.27% away from the optimum, corresponding to a 0.36 seconds difference between the ground truth and the found performance. For `SRO.300`, the returned performance is 2.57% away, for an absolute difference of 0.15 seconds. Overall experiments, we find that the distance to the optimum is very small and almost negligible to users.
The most notable advantage of adding a resampling filter to dynamic bounded intervals is the improvement of the convergence speed, by preventing the evaluation of uninteresting parametrizations. This leads to a reduction in the number of iterations and total duration of the experiment, as we observe a time gain of 22% for `SRO.60`, 16% for `SRO.120`, 14% for `SRO.300`.

6.4 Summary of results

The results summary of using EVADyR algorithm compared to static resampling and SEDR, as well as not using any noise reduction strategy are available in table 6. They show that

EVADyR outperforms the state-of-the-art algorithms in terms of distance to the optimum and improvement from the default parametrization: static resampling is outperformed by 72.6% and dynamic resampling by 93.5%. The same improvement is seen when looking at the distance of the turned optimum from the default parametrization. EVADyR also provides a significant improvement in convergence quality, with a 97.46% improvement compared to not using any noise reduction. EVADyR thus both improves the convergence property of noise reduction algorithms and removes the need for hyperparameters, primarily by adding bounded dynamic interval resampling to the tuning process.
Another important result of our study is the importance of noise reduction: when not using any, black-box optimization is unable to perform the tuning in a noisy environment, as the returned parametrization brings little to no improvement compared to the default parametrization. EVADyR thus improves the convergence quality of the auto-tuner by 97.46% rather than when not taking the noise into account. In terms of convergence speed and experiment duration, the results show a time gain in terms of experiment duration compared to the state-of-the-art, because of the added resampling decision filter. Indeed, compared to static resampling, EVADyR brings a time gain of 9.38 %, and compared to dynamic resampling, EVADyR brings a time gain of 45.76 %.

Table 6. Comparison of our solution to the state of the art in terms of:

	None	**Static**	**Dynamic**	**EVADyr**
Distance to the ground truth (%)	153.90	15.39	64.86	4.21
Improvement to default (%)	37.37	71.53	59.33	73.88
Experiment duration (s)	1035.33	1807.55	3018.94	1637.86

7 Conclusion

In conclusion, this paper discusses the impact of noisy interference on the performance of black-box optimization auto-tuners. We prove that the noise cannot be neglected in production systems, as it significantly impacts the performance of the optimizer. To increase resilience, we suggest a new resampling technique called EVADyR and show that it improves convergence by 93.5% and reduces experiment duration by 45.76% compared to state of the art dynamic resampling. We also emphasize the importance of using noise reduction strategies in the case of highly shared systems, as we found a 97.46% increase in the optimum performance.
Planned improvement of this work consists in testing noise with a random time of arrival rather than the fixed tested intervals. This would allow to study the behavior of the noise reduction resampling algorithms when faced with more uncertainty in terms of noise arrival. Further works also include the comparison of our results to heuristic-specific noise resilient Bayesian Optimization, by using acquisition functions tailored for stochastic optimization, such as *Noisy Expected Improvement*, *Augmented Expected Improvement* and *Expected Quantile Improvement*.
-–

References

1. The SHAMan application, https://github.com/bds-ailab/shaman
2. Aizawa, A.N., Wah, B.W.: Scheduling of Genetic Algorithms in a Noisy Environment. In: Evolutionary Computation. vol. 2, pp. 97–122 (1994)
3. Behzad, B., Byna, S., Prabhat, M., Snir, M.: Pattern-driven parallel i/o tuning. In: Proceedings of the 10th Parallel Data Storage Workshop. pp. 43–48 (2015)
4. Behzad, B., Luu, H.V.T., Huchette, J., Byna, S., Prabhat, Aydt, R., Koziol, Q., Snir, M.: Taming parallel i/o complexity with auto-tuning. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. SC '13
5. Bennett, K., Ferris, M., Ioannidis, Y.: A genetic algorithm for database query optimization. In: Proceedings of the fourth International Conference on Genetic Algorithms. pp. 400 – 407 (1991)

6. Bergstra, J., Pinto, N., Cox, D.: Machine learning for predictive auto-tuning with boosted regression trees. In: 2012 Innovative Parallel Computing (InPar). pp. 1–9 (2012)
7. Cao, Z.: A Practical , Real-Time Auto-Tuning Framework for Storage Systems. Ph.D. thesis, State University of New York at Stony Brook (2018)
8. Cao, Z., Tarasov, V., Tiwari, S., Zadok, E.: Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In: Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference. pp. 893–907. USENIX ATC '18 (2018)
9. Desani, D., Costa, V.G., Marcondes, C.A.C., Senger, H.: Black-box optimization of hadoop parameters using derivative-free optimization. 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP) pp. 43–50 (2016)
10. Di Pietro, A., While, L., Barone, L.: Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. In: Proceedings of the 2004 Congress on Evolutionary Computation. vol. 2, pp. 1254–1261 (2004)
11. Duan, S., Thummala, V., Babu, S.: Tuning database configuration parameters with ituned. In: Proceedings of the VLDB Endowment. pp. 1246 – 1257 (2009)
12. Faraj, A., Yuan, X.: Automatic generation and tuning of mpi collective communication routines. In: Proceedings of the 19th Annual International Conference on Supercomputing. pp. 393 – 402 (2005)
13. Fitzpatrick, J.M., Grefenstette, J.J.: Genetic algorithms in noisy environments. In: Machine Learning. vol. 3, pp. 101–120 (1988)
14. Forrester, E.I.J., Keane, A.J., Bressloff, N.W.: Design and analysis of 'noisy' computer experiments. AIAA Journal pp. 2331–2339
15. Gramacy, R., Lee, H.: Optimization under unknown constraints. In: Bayesian Statistics. vol. 9 (2010)
16. Grohmann, J., Seybold, D., Eismann, S., Leznik, M., Kounev, S., Domaschka, J.: Baloo: Measuring and modeling the performance configurations of distributed dbms. In: 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 1–8 (2020)
17. Gur, Y., Yang, D., Stalschus, F., Reinwald, B.: Adaptive multi-model reinforcement learning for online database tuning. In: International Conference on Extending Database Technology (2021)
18. Huang, D., Allen, T., Notz, W., Miller, R.A.: Sequential kriging optimization using multiple-fidelity evaluations. In: Structural and Multidisciplinary Optimization. vol. 32, pp. 369–382 (2006)
19. Ioannidis, Y., Wong, E.: Query optimization by simulated annealing. In: SIGMOD Record. pp. 9 – 22 (1987)
20. Jamshidi, P., Casale, G.: An uncertainty-aware approach to optimal configuration of stream processing systems. In: arXiv (2016)
21. Kassab, A., Nicod, J.M., Philippe, L., Rehn-Sonigo, V.: Assessing the use of genetic algorithms to schedule independent tasks under power constraints. In: 2018 International Conference on High Performance Computing Simulation (HPCS). pp. 252–259 (2018)
22. Letham, B., Karrer, B., Ottoni, G., Bakshy, E.: Constrained bayesian optimization with noisy experiments. In: ArXiv (2017)
23. Li, Y., Chang, K., Bel, O., Miller, E.L., Long, D.D.E.: Capes: Unsupervised storage performance tuning using neural network-based deep reinforcement learning. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 42:1–42:14. SC '17 (2017)
24. Liao, G., Datta, K., Willke, T.: Gunther: Search-based auto-tuning of mapreduce. In: Euro-Par. vol. 8097, pp. 406 – 419 (2013)
25. Melo Alves, M., de Assumpção Drummond, L.M.: A multivariate and quantitative model for predicting cross-application interference in virtual environments. Journal of Systems and Software **128**, 150–163 (2017)
26. Menon, H., Bhatele, A., Gamblin, T.: Auto-tuning parameter choices in hpc applications using bayesian optimization. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 831 – 840 (2020)
27. Miyazaki, T., Sato, I., Shimizu, N.: Bayesian optimization of hpc systems for energy efficiency. In: High Performance Computing. pp. 44–62 (2018)
28. Ozer, G., Netti, A., Tafani, D., Schulz, M.: Characterizing hpc performance variation with monitoring and unsupervised learning. In: Jagode, H., Anzt, H., Juckeland, G., Ltaief, H. (eds.) High Performance Computing. pp. 280–292 (2020)

29. Picheny, V., Wagner, T., Ginsbourger, D.: A benchmark of kriging-based infill criteria for noisy optimization. In: Structural and Multidisciplinary Optimization. vol. 48, pp. 607–626 (2013)
30. Picheny, V., Ginsbourger, D., Richet, Y., Caplin, G.: Quantile-based optimization of noisy computer experiments with tunable precision. In: Technometrics. vol. 55 (2012)
31. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning) (2005)
32. Robert, S., Zertal, S., Vaumourin, G.: Using genetic algorithms for noisy systems' autotuning: an application to the case of burst buffers. In: Proceedings of the International Conference on High Performance Computing Simulation (HPCS) (2020)
33. Robert, S., Zertal, S., Vaumourin, G., Couvée, P.: A comparative study of black-box optimization heuristics for online tuning of high performance computing i/o accelerators. In: Concurrency and Computation: Practice and Experience (2021)
34. Schmied, T., Didona, D., Döring, A., Parnell, T., Ioannou, N.: Towards a general framework for ML-based self-tuning databases (2020)
35. Seymour, K., You, H., Dongarra, J.: A comparison of search heuristics for empirical code optimization. In: 2008 IEEE International Conference on Cluster Computing. pp. 421–429 (2008)
36. Siegmund, F., Ng, A., Deb, K.: A comparative study of dynamic resampling strategies for guided evolutionary multi-objective optimization. In: 2013 IEEE Congress on Evolutionary Computation. pp. 1826–1835 (2013)
37. Stagge, P.: Averaging efficiently in the presence of noise. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.P. (eds.) Proceedings of the 5th Parallel Problem Solving from Nature. pp. 188–197. Springer Berlin Heidelberg (1998)
38. Syberfeldt, A., Ng, A., John, R.I., Moore, P.: Evolutionary optimisation of noisy multiobjective problems using confidence-based dynamic resampling. In: European Journal of Operational Research. vol. 204, pp. 533–544 (2010)
39. V. K. Ky, C. D'Ambrosio, Y.H., Liberti, L.: Surrogate-based methods for black-box optimization. International Transactions in Operational Research (24) (2016)
40. Vázquez, E., Villemonteix, J., Sidorkiewicz, M., Walter, E.: Global optimization based on noisy evaluations: An empirical study of two statistical approaches. In: Journal of Physics Conference Series. vol. 135 (2008)
41. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)
42. Zheng, W., Fang, J., Juan, C., Wu, F., Pan, X., Wang, H., Sun, X., Yuan, Y., Xie, M., Huang, C., Tang, T., Wang, Z.: Auto-tuning mpi collective operations on large-scale parallel systems. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). pp. 670 – 677 (2019)

Machine Learning-based Per-Instance Algorithm Selection for High-Performance Subgraph Isomorphism Enumeration

Syed Ibtisam Tauhidi[1], Arindam Karmakar[2], Thai Son Mai[1], and Hans Vandierendonck[1]

[1] Queen's University Belfast, Belfast, Northern Ireland, UK
[2] Tezpur University, Tezpur, Assam, India
stauhidi01@qub.ac.uk

Abstract. The Subgraph Isomorphism Enumeration (SIE) problem requires discovering all the embeddings of a *pattern* (or *query*) subgraph in a given *data* graph. The problem is NP-complete, and numerous exact heuristic algorithms have been proposed to speed up the execution of the problem. Based on the observation that no heuristic is the fastest for all pattern and data graph pairs, we design a metaheuristic for *per-instance algorithm selection* to determine the fastest heuristic for each graph pair. We hypothesise that the connections and properties of vertices in the graph pair are indicative of the best-performing heuristic algorithm. As such, we design a Machine Learning (ML)-based metaheuristic algorithm and investigate how well various types of graph features and ML algorithms predict performance. Our best-performing metaheuristic improves the execution speed of the SIE problem by up to 1.54 times across 8 data graphs compared to any single heuristic algorithm. The analysis furthermore identifies remaining challenges unique to specific data graphs in the SIE problem.

1 Introduction

Graphs are ubiquitous in computer science. Data from various domains, such as astronomy, bioinformatics, and social and computer networks, are often represented as graphs. Graph analytics performs analysis of such graph representations. The `Subgraph Isomorphism Enumeration` (SIE) problem is widely used in graph analytics, including in graph-based database systems, anomaly detection, subcircuit identification, astrophysics, etc. The SIE problem involves discovering all *embeddings* of a given *pattern* (or *query*) graph in an associated *data* graph. Although polynomial-time solutions for special cases of the SIE problem are known [6], the problem is NP-complete in general, making it computationally intensive. However, several proposed algorithms [2, 3, 8, 16, 4, 9, 11], called *heuristics*, have used characteristics of the graph (e.g., labels, degree, regions, etc.) to speed up the execution by pruning the exhaustive search space.

An open challenge in the design of metaheuristic solutions to the SIE problem, as with many other NP-complete problems [10, 20, 14], is that different heuristics perform faster on different problem instances. While some heuristics are, on average, faster than others over a number of problem instances, none is consistently the fastest. In this work, we aim to ameliorate this problem by designing a metaheuristic algorithm that identifies, for a particular problem instance, an appropriate heuristic that minimises execution time.

Metaheuristic algorithms for per-instance heuristic selection using Machine Learning (ML) are promising to speedup NP-complete problems [18, 12, 13, 20, 19]. The application of ML to metaheuristic design, however, is faced with several challenges. A plethora of machine learning algorithms exist, and there is little knowledge available in the literature on what machine learning algorithms are effective for this task. Moreover, machine learning algorithms predict on the basis of extracted features, where the set of features must be chosen to be predictive of the desired outcome, in this case identifying the fastest heuristic for a given instance. Often, features are specific to the problem, e.g., the SIE problem has categorical labels on vertices, whereas the Travelling Salesman Problem has edge weights. As such, graph features are often specific to the problem, and we perform a thorough study of different graph features. Our main contributions are:

1. Feature selection: We investigate three types of features for each pattern graph in each dataset, namely (summary) Graph Features (`GF`), Label Features (`LF`) and Graphlet Features (`GLF`). To the best of our knowledge, this is the first study that uses `GLF` for *per-instance algorithm selection*. We observe that applying `GLF` results in better performance than `GF` but need to be applied in conjunction with LF.

2. Metaheuristic Approach: We propose a metaheuristic system using multiple ML algorithms individually to choose the predicted fastest heuristic algorithm for any given pattern graph in a dataset. We analyse the use of both a Random Forest and a voting-based ensemble technique for developing a high-level metaheuristic strategy for the per-instance algorithm selection task. Experimental evaluation of eight data graphs and seven heuristics shows that our proposed metaheuristic approach achieves a speedup of up to 1.54 times over any static heuristic.

The remainder of the paper is organised as follows. Section 2 defines the SIE problem and discusses related works in the literature. Section 3 motivates the work through the analysis of collected data. Section 4 discusses the methodology in our proposed approach in the context of the feature selection, experimental setup, and evaluation metric. Section 5 discusses the experimental evaluation. Section 6 concludes the paper.

2 Preliminaries and related works

Definition Given a graph $G = (V, E)$ and a set of labels, A, $\alpha : V \to A$ and $\beta : E \to A$ are two labelling functions that assign labels to vertices and edges, respectively, of G. Let $G_d = (V_d,\ E_d)$ and $G_p = (V_p,\ E_p)$ be two graphs. The `Subgraph Isomorphism Enumeration` (SIE) problem is to find an injective mapping, $M : V_p \to V_d$, while preserving edge adjacency. If vertex $u \in G_p$ is mapped to $u' \in G_d$, then $\alpha(u) = \alpha(u')$. Additionally, if $v \in G_p$ is mapped to $v' \in G_d$ and there exists $(u, v) \in E_p$, then $(u', v') \in E_p$ and $\beta((u, v)) = \beta((u', v'))$.

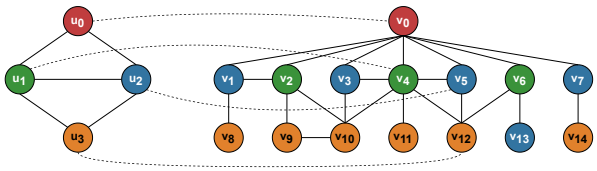


Fig. 1: Dashed lines showing a mapping corresponding to a subgraph isomorphism embedding between a pattern graph (left) and a data graph (right). The colours indicate the vertex labels.

Fig. 1 shows two graphs. The smaller *pattern* graph is shown on the left, and the larger *data* graph is shown on the right. A mapping corresponding to a subgraph isomorphism is $\{(u_0,\ v_0),\ (u_1,\ v_4),\ (u_2,\ v_5),\ (u_3,\ v_{12})\}$ and is shown in the figure using dashed lines. Another valid mapping is $\{(u_0,\ v_0),\ (u_1,\ v_4),\ (u_2,\ v_3),\ (u_3,\ v_{10})\}$. There is no other valid mapping between the vertices of the two graphs. An SIE search algorithm would enumerate both mappings.

Multiple studies have applied nature-inspired algorithms to develop metaheuristic strategies for the SIE problem. However, such algorithms are usually not the best approach towards developing a metaheuristic for exact search. Most of the nature-inspired algorithms have shown positive performance in finding approximate solutions to the SIE problem but failed to find exact solutions. For example, Farhani et al. [7] used genetic algorithms iteratively to improve local searches by perturbing a solution matrix. However, experimental results have shown that it does not achieve an exact match. Yun et al. [21] used a strategy based on harmony search to find approximate subgraphs in graphs with up to 150 nodes. The approach succeeded in finding optimal solutions in small pattern graphs; however, it failed when the graphs grew in size.

The application of ML in developing metaheuristic algorithms was studied by Talbi [18], who classified its usage into three categories, one of which is to use it as a high-level metaheuristic to select a heuristic from a portfolio of heuristics. This is the approach that has been taken in this study. Karimi et al. [12] used the terminology *meta-learning* to describe the ML-based metaheuristics that are used for *per-instance algorithm selection* and divides the process into two stages: *meta-data extraction* and *meta-model creation*.

Kerschke et al. [13] surveyed the application of ML-based metaheuristics for various problems, including SAT, AI Planning, and TSP. They have emphasized three characteristics essential for *meta-data extraction*, i.e., feature selection: (i) Features should be *informative* and *interpretable*,

(ii) features should be *easily computable*, and (iii) features should be *generally applicable* to all input cases. These characteristics have been considered in the feature selection of this study.

SATZILLA [20] was one of the pioneering studies that used ML for *per-instance algorithm selection* for the Satisfiablity (SAT) problem. Tornede et al. [19] studied the application of various ensemble methods to develop a metaheuristic, which they tested on the SAT problem by comparing it against ML models and an `ORACLE` algorithm. This study takes a similar approach toward improving the runtime of the SIE problem, albeit with a larger number of features and datasets.

3 Motivation

For runtime data collection, seven heuristics, specifically `CECI` [2], `CFL` [3], `DPiso` [8], `QuickSI` [16], `RI` [4], `TurboISO` [9], and `VF2++` [11], were used through the open-source In-Memory Subgraph Matching (`IMSM`) [17] codebase to collect runtime data on eight different datasets, namely `DBLP`, `EU2005`, `HPRD`, `HUMAN`, `PATENTS`, `WORDNET`, `YEAST` and `YOUTUBE`. Each dataset contains one data graph and 1800 pattern graphs. We performed the SIE search for all the 1800 pattern graphs in each of the eight datasets using all seven heuristics with a timeout of 30 minutes. Table 1 provides an overview of the data graphs in each dataset.

Dataset	Vertex Count	Edge Count	Mean Degree	Label Count
DBLP	317080	1049866	6.622	15
EU2005	862664	16138468	37.415	40
HPRD	9460	34998	7.399	307
HUMAN	4674	86282	36.92	44
PATENTS	3774768	16518947	8.752	20
WORDNET	76853	120399	3.133	5
YEAST	3112	12519	8.046	71
YOUTUBE	1134890	2987624	5.265	25

Table 1: Overview of data graphs

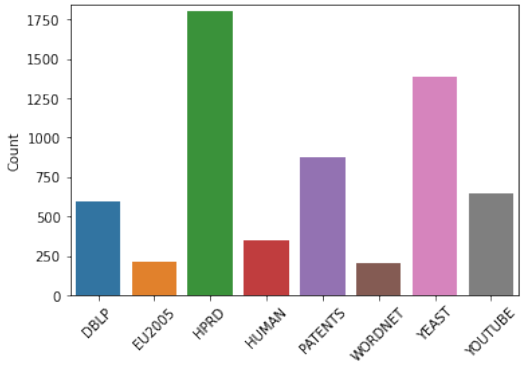


Fig. 2: Count of pattern graphs that completed execution within the specified timeout

At the end of the 30 min timeout, all 1800 pattern graphs in the `HPRD` dataset had completed execution using all seven heuristics. However, many pattern graphs in the other datasets did not complete execution within the timeout. Fig. 2 shows the count of pattern graphs for each dataset in which all the heuristics completed the SIE search successfully. For further evaluation, we consider only the pattern graphs that had completed execution on all heuristics. Fig. 3 shows the mean runtime of the different heuristic algorithms on the considered pattern graphs in the eight datasets.

Fig. 4 shows the count of pattern graphs in each dataset in which each of the heuristics showed the fastest runtime execution. We can observe that the sum of the count for each dataset exceeds

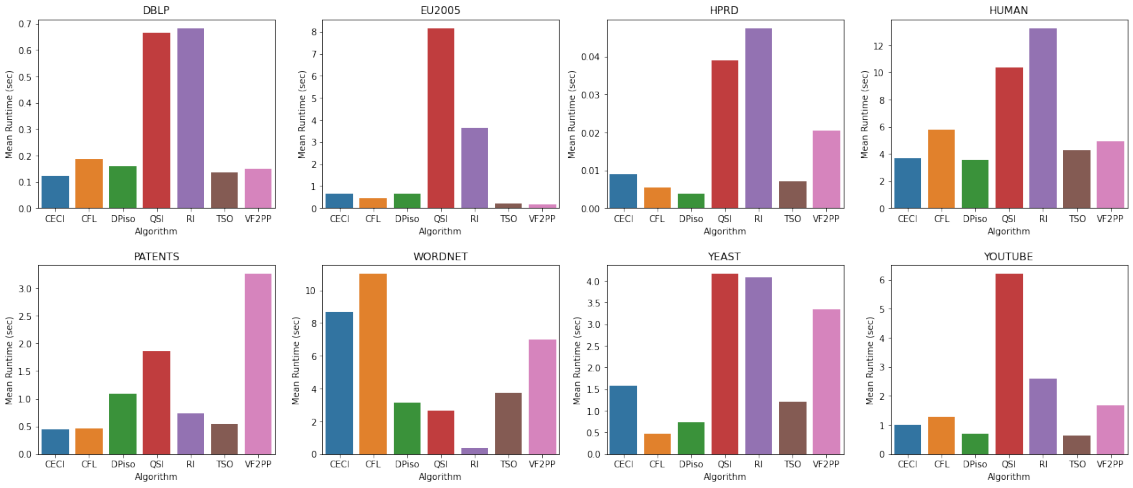


Fig. 3: Mean runtime of the seven different heuristic algorithms on the eight datasets

the total number of pattern graphs in the dataset. For example, the sum of the fastest execution run for each of the heuristics in the `HPRD` dataset is 3366, despite the dataset having only 1800 pattern graphs. This is because, in a single pattern graph, multiple heuristics, when measured with millisecond-scale granularity, might have the same fastest runtime.

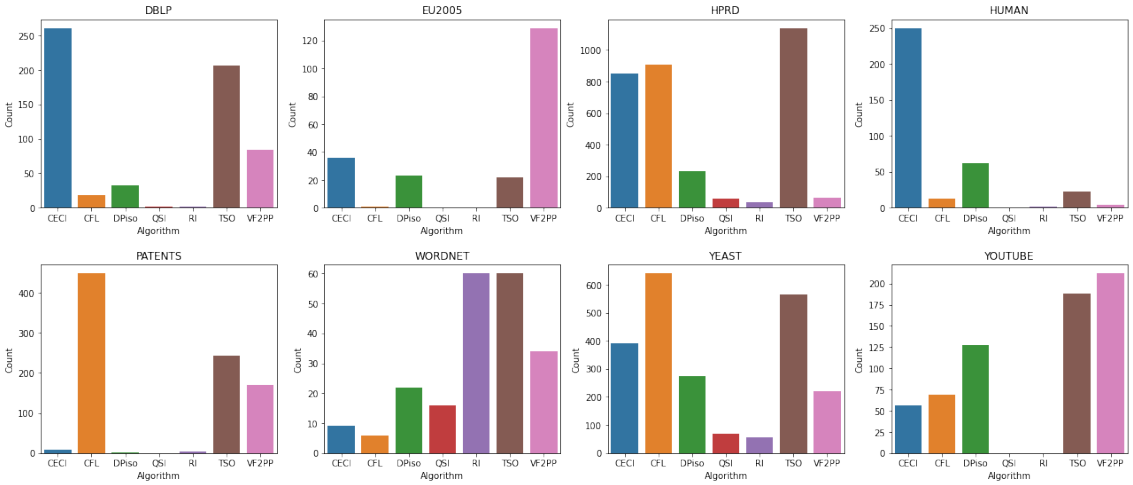


Fig. 4: Count of fastest execution by different heuristics

Additionally, it can be observed that the heuristic with the fastest mean runtime in a dataset is not necessarily the fastest algorithm in most pattern graphs. For example, we can observe in Fig. 3 that in the `HPRD` dataset, the `DPiso` algorithm has the fastest mean execution time, followed by `CFL` and `TSO`. However, Fig. 4 shows that `TSO` (1131) has the fastest execution in the highest number of pattern graphs, followed by `CFL` (901) and `CECI` (843). The `DPiso` heuristic, which has the fastest mean execution time, ranks fourth in the count (233) of the pattern graphs in which it has the fastest execution time. Similar observations can be made in other datasets.

A hypothetical metaheuristic algorithm that would always select the fastest heuristic (or one of the multiple fastest heuristics) for each pattern graph in a dataset would result in a faster execution than using any single heuristic statically for all pattern graphs. We call such a hypothetical metaheuristic the `ORACLE` algorithm. Fig. 5 shows the mean runtime of the `ORACLE` algorithm on the eight datasets and compares it with the fastest heuristic for each dataset.

In this study, we are motivated to approximate the `ORACLE` algorithm, i.e., we propose a metaheuristic algorithm that would predict and select the fastest heuristic for any given pattern

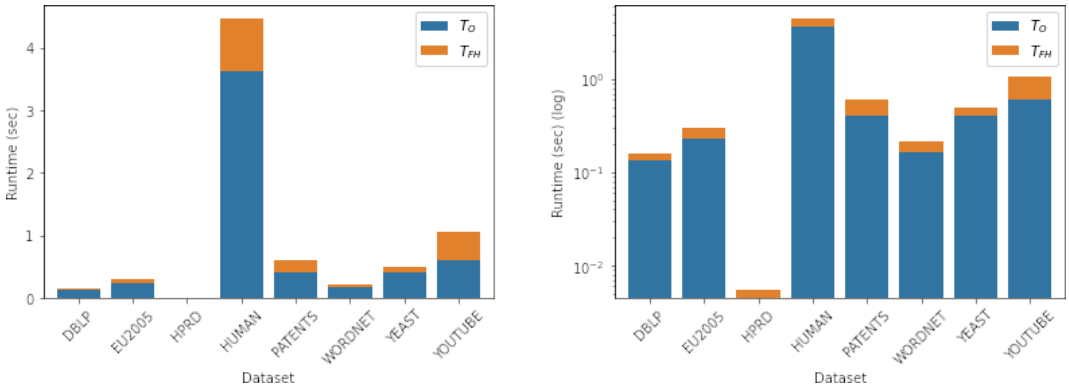


Fig. 5: Runtime comparison of the ORACLE algorithm (T_O) and the fastest heuristic (T_{FH}) shown in linear (left) and logarithmic (right) scale.

graph in a dataset. In ML terminology, we want to perform multiclass classification using the features of a given pattern graph. Such a task involves predicting one output class (i.e., heuristic) from the target set (i.e., set of all heuristics). Thus, despite a pattern graph having potentially multiple heuristics showing the fastest execution time, only one of them can be chosen as the output. If any pattern graph has multiple fastest heuristics, the heuristic with the worst mean runtime in the dataset has been chosen as the target. This ensures that the heuristics that have shown degraded performance in the overall datasets are favoured whenever they have shown the best performance, thereby enabling the balancing of the datasets. This balancing prevents the fastest heuristic in the overall dataset from dominating the target set, thereby preventing the ML algorithms from becoming biased toward the faster heuristics. Fig. 6 shows the distribution of the count of the pattern graphs for which each heuristic was chosen as the target.

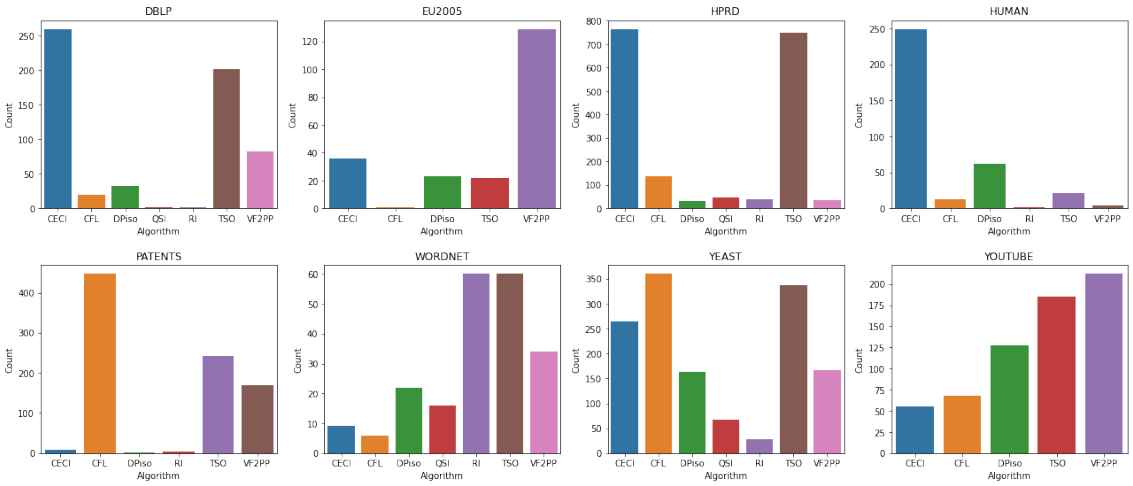


Fig. 6: Distribution of the count of the pattern graphs for each heuristic in the target set

4 Our Proposed Approach

4.1 Feature Extraction

Multiple feature selection techniques have been used for each pattern graph in each of the eight datasets. These are described as follows.

Vertex Count	Edge Count	Diameter
Girth	Mean Degree	Maximum Degree
Minimum Degree	Min. Eigenvalue Centrality	Density
Is graph bipartite?	Unique Label Count	Max. Label Count
Mean Label Count	-	Label Count Std. Dev.

Table 2: Graph Features (GF) retained for each pattern graph after feature selection

Graph Features (GF): Initially, for each pattern graph, we extracted 25 features using the `igraph` [5] software package. We performed feature reduction manually to address the *curse of dimensionality* problem. The features `Maximum Eigenvector Centrality`, `Is graph connected?`, and `Minimum Label` were eliminated based on variance-based thresholding. Furthermore, `Radius`, `Mean Path Length`, `Mean Eccentricity`, `Maximum Eccentricity` and `Minimum Eccentricity` were eliminated because of high correlation ($\geq$ 0.9) with `Diameter`. Additionally, `Edge Connectivity` and `Vertex Connectivity` were discarded because of high correlation with `Minimum Degree`, and `Mean Eigenvector Centrality` because of high correlation with `Density`. Only features with a linear correlation with a retained feature have been eliminated. For example, `Density` has been retained despite being a nonlinear combination of `Vertex Count` and `Edge Count`. The features retained after manual feature selection are listed in Table 2.

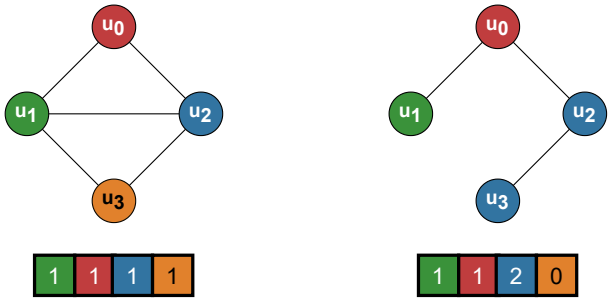


Fig. 7: Pattern graphs and their associated `LF` feature vectors

Label Features (LF): The Label Feature (`LF`) vector for each pattern graph is a vector of size equal to the count of unique labels in the associated data graph. For each pattern graph, each column in the `LF` vector records the count of the occurenoccurrence of the corresponding label in the pattern graph. Fig. 7 shows two pattern graphs and their associated `LF` vectors, given the data graph in Fig. 1. Unlike the Graph Features (`GF`), the length of the `LF` vectors varies across the different datasets and equals the count of the unique labels in the data graphs (given in Table 1), i.e., the `LF` vector ranges from 5 in `WORDNET` to 307 in `HPRD`. As discussed previously, a large feature count gives rise to the *curse of dimensionality* problem. Therefore, we used Principal Component Analysis (PCA) with the `LF` vectors to reduce the dimensions of the feature vectors. Fig. 8 shows the explained variance ratio and the count of selected principal components for each dataset. The count of the selected principal components is the final length of the `LF` vector for each dataset.

Graphlet Features (GLF): Graphlets (or motifs) are recurring subgraph patterns that occur with a statistically significant frequency within a large graph. Counting the number of embeddings of the graphlets in a graph enables the creation of a profile of the underlying local subgraph structure of the graph. Two graphs with similar Graph Features (`GF`) will have different Graphlet Features (`GLF`) if their underlying structures are different. In this study, we used the `Parallel Graphlet Decomposition (PGD) Library` toolkit [1] to extract the count of the occurrences of seventeen graphlets, shown in Fig. 9. All the extracted graphlets, including the 4-clique (K_4), are planar graphs, and their count can be efficiently computed in linear time.

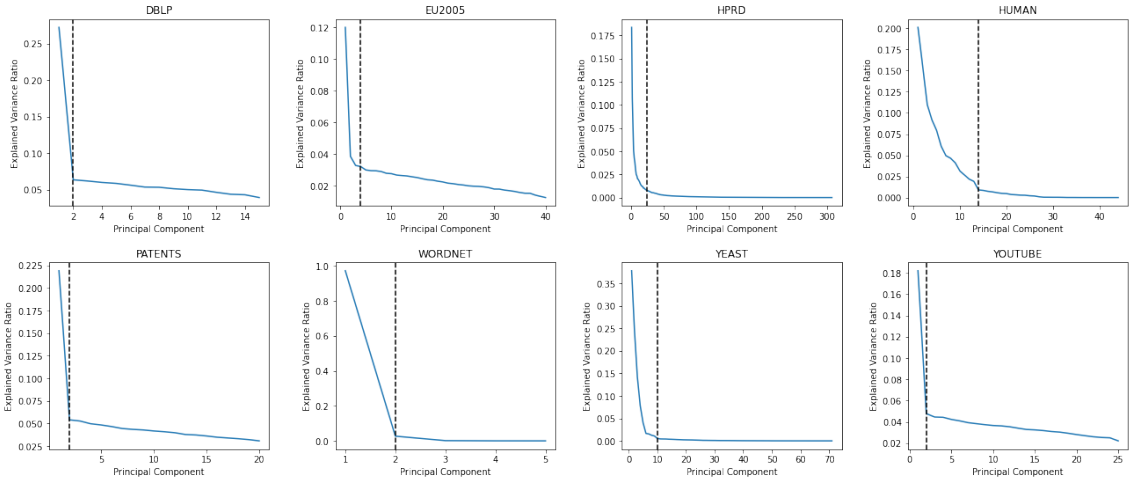


Fig. 8: Components selected after dimensionality reduction using PCA on Label Features (`LF`)

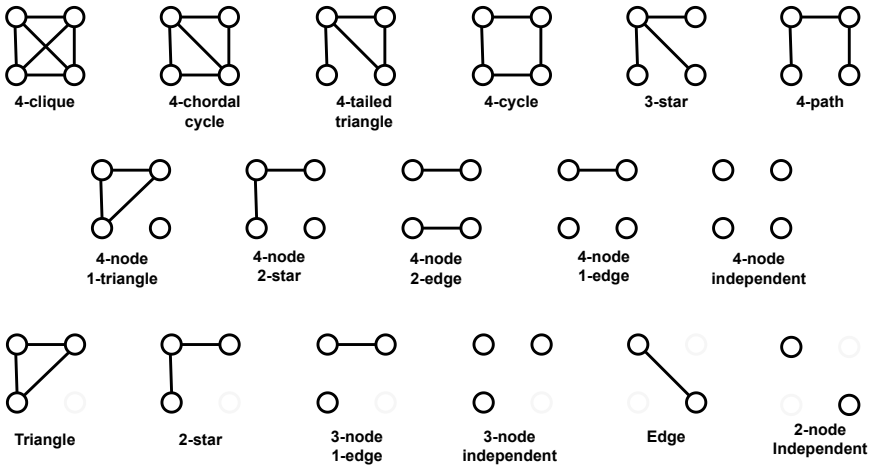


Fig. 9: The seventeen Graphlet Features (`GLF`) extracted for each pattern graph

4.2 Experimental Setup

We used a single core of an `AMD EPYC 7702 64-Core Processor` with an allotment of 4 GB RAM and a runtime threshold of 30 minutes to run one instance of the SIE search. We used a High-Performance Computing (HPC) infrastructure to run sequential execution of the seven heuristic algorithms on each of the 1800 pattern graphs in all eight datasets in parallel in separate cores.

For extraction of the Graph Features (`GF`) and Label Features (`LF`), we used the `igraph` software package. Additionally, we used the `PGD library` toolkit to extract the Graphlet Features (`GLF`). As discussed, redundant and low-variance features were removed from `GF` using manual analysis. We used PCA to perform dimensionality reduction on the `LF` using the *elbow* method.

Finally, we used fifteen ML algorithms, including tree-based methods (Decision Tree, Random Forest, Extra Trees, AdaBoost, Gradient Boost), feed-forward neural network (with 1, 2, and 3 hidden layers), Stochastic Gradient Descent, k-Nearest Neighbours, Naïve Bayes, and Support Vector Machine (with Radial Basis Function, Linear, Sigmoid and Polynomial kernels) to predict the fastest heuristic for each pattern graph in the eight datasets. We have chosen algorithms across all popular paradigms (e.g., tree-based, probabilistic, lazy, etc.) of ML algorithms. Additionally, we used 10-fold cross-validation to evaluate the performance of the ML models. We exhaustively used all combinations of the three feature types for the prediction. We used the `Scikit-learn` [15] library to perform the PCA and the ML classification (training and inferring) tasks.

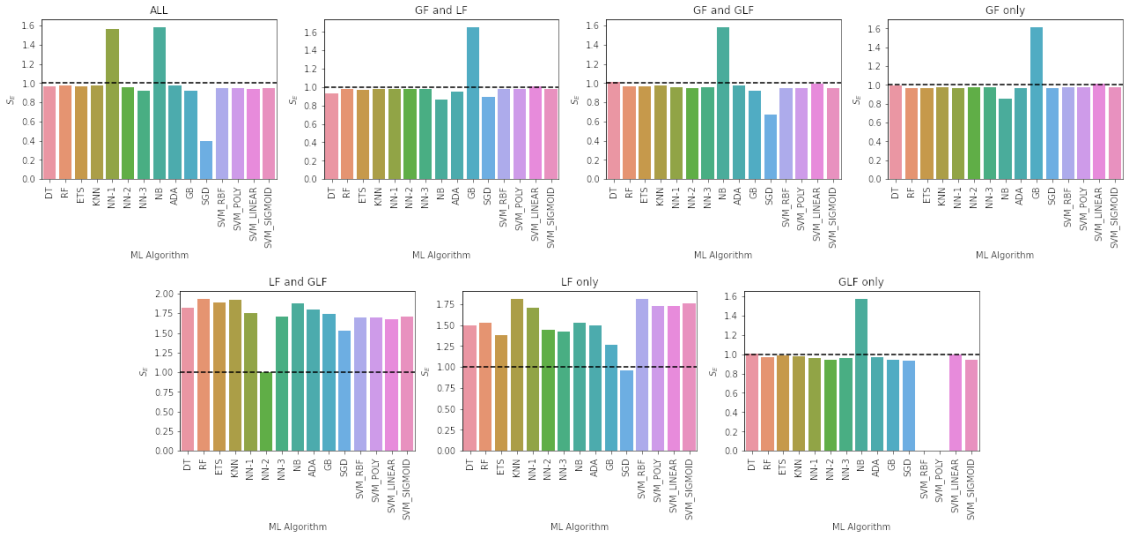


Fig. 10: Speedup on the PATENTS dataset using all modes

4.3 Evaluation Metric

Since the objective of this study is to determine whether ML algorithms can be used to improve the runtime of solving the SIE problem using a *per-instance algorithm selection* strategy, the traditional metrics used to evaluate the performance of classification (viz. accuracy, confusion matrix and its derivatives, etc.) are not sufficient for the evaluation in this study. For example, the traditional metrics would not be able to account for the correct prediction when the ML algorithm predicts a heuristic that is not the target for any particular pattern graph but instead predicts another heuristic that is as fast as the target. Therefore, we define a metric to evaluate the performance of our proposed metaheuristic system. The metric is described below.

Effective Speedup (S_E): This is the speedup obtained using the metaheuristic compared to the fastest heuristic algorithm. If T_{FH} is the execution time of the fastest heuristic on the dataset, and T_{ML} is the execution time using the ML algorithm, then the relative speedup, S_E, is defined as -

$$S_E = \frac{T_{FH}}{T_{ML}}$$

The metric $S_E = 1$ when the metaheuristic would have a performance equal to the fastest heuristic algorithm. If the metaheuristic algorithm is slower than the fastest heuristic, then S_E would have a value of less than one. The faster the metaheuristic algorithm is compared to the fastest heuristic, the higher would be the value of S_E. It must be noted that no metaheuristic algorithm can be faster than the ORACLE algorithm, and therefore, the S_E is constrained by the runtime of the oracle. In other words, we have the highest value of S_E when $T_{ML} = T_O$, where T_O is the runtime of the ORACLE algorithm.

5 Experimental Evaluation

The performance of the various ML algorithms using all combinations of the feature types to predict the fastest heuristic on the PATENTS dataset is shown in Fig. 10. We observe that the performance of all ML algorithms improves when GF features are discarded. In the PATENTS dataset, the best performances are observed when LF features are used in isolation or in combination with GLF features. We observe similar performance characteristics in all other datasets.

Fig. 11 shows the performance of the different ML algorithms on the eight datasets using only LF and GLF features. We see that in the WORDNET dataset, none of the ML algorithms show a speedup. Also, only Random Forest (RF) shows a speedup in the YEAST dataset. Nevertheless, in all other

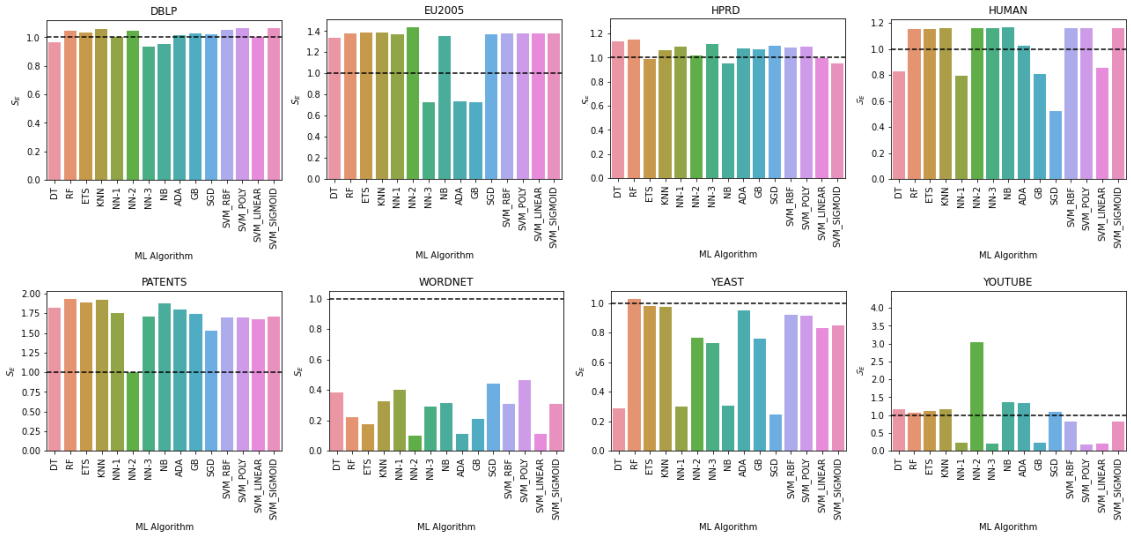


Fig. 11: Effective Speedup on the different datasets using only `LF` and `GLF` features. The dashed line indicates the baseline speedup using the fastest heuristic.

datasets, multiple ML algorithms show speedup using only the `LF` and `GLF` feature combination. Fig. 12 shows the performance of the different ML algorithms when all three feature types (`GF`, `LF` and `GLF`) are used on the eight datasets. We see that apart from the `DBLP` dataset, all other datasets exhibit degraded performance when we use all the features. This corroborates our observations in Fig. 10 that the inclusion of `GF` features results in performance degradation.

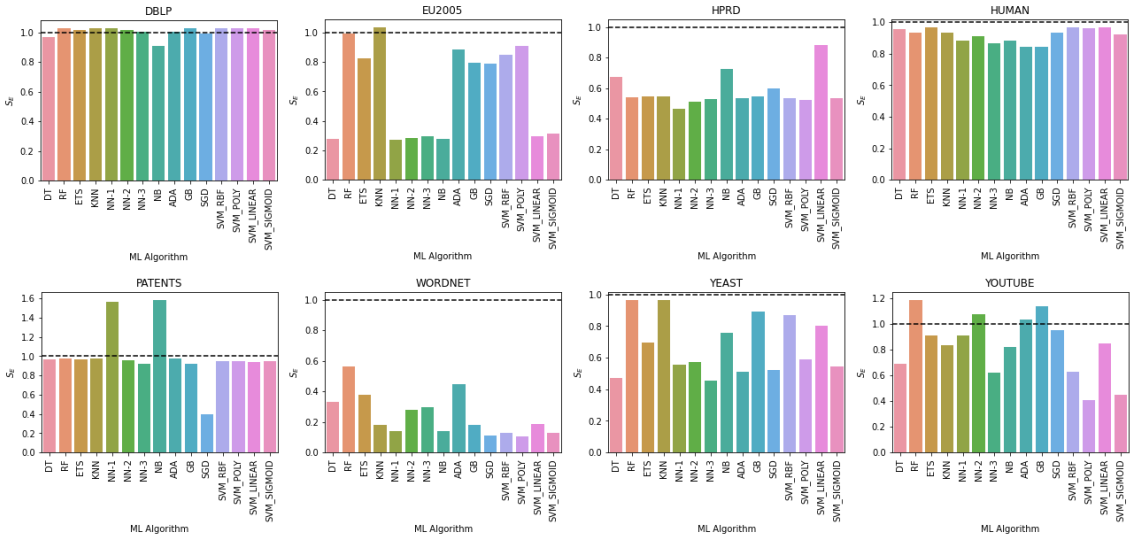


Fig. 12: Speedup on the different datasets using `ALL (GF, LF and GLF)` features

We observe that the Label Features (`LF`) are the most crucial feature types for all ML algorithms. All the heuristics considered in this study can be divided into three stages. In the first stage, all the heuristic algorithms use `LF` at the core for filtering out incompatible vertices in the data graph for each vertex in the pattern graph, thereby reducing the search space. Each heuristic has a different mechanism to perform the filtering. Therefore, it fits that the label distributions given by the `LF` features are essential for ML algorithms. In fact, the `WORDNET` dataset, which has shown the worst performance, also has the lowest number of labels.

We observe that the `GLF` features have shown higher performance than applying `GF` features. Pattern graphs with similar `GF` features can have very different internal structures. The `GF` features

cannot capture the details of the internal structures of the pattern graphs. On the other hand, the graphlet (or motif) count captured using GLF features can describe the structure of subgraphs of the pattern graphs. Thus, used along with LF features, the GLF features provide a better description of the structure of the pattern graphs for the ML algorithms to make more accurate predictions.

In all datasets, except WORDNET and YEAST, multiple ML algorithms have shown a runtime speedup. However, we observe that no ML algorithm has shown a consistent speedup across all datasets, with only RF showing an improved performance in a majority of datasets.

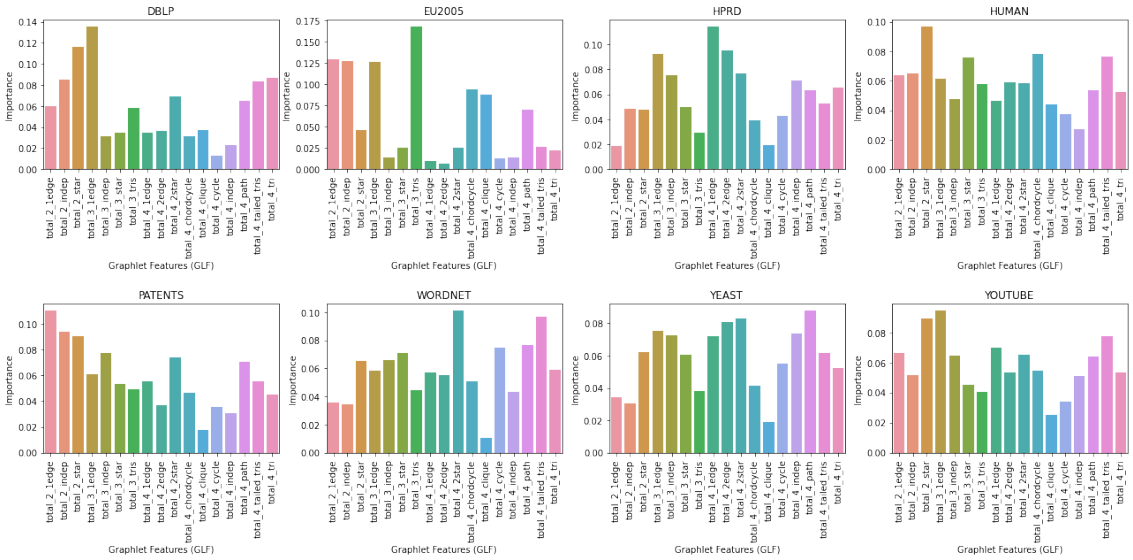


Fig. 13: Gini Feature Importance using Random Forest

Given that RF has shown improved performance in almost all the datasets, we propose a naïve metaheuristic algorithm that uses only RF for *per-instance algorithm selection*. As discussed, we consider only the LF and GLF features while discarding the GF features because of the degradation in performance that it introduces. The dimensionality reduction performed through PCA ensures that only LF features with significant importance are selected. Fig. 13 shows the Gini importance of the different GLF features when performing the classification using RF.

While RF has shown improved performance in all datasets, except WORDNET, we can see that the GLF features used for the construction of the decision trees in the RF have different importance for different datasets. This variation in feature importance indicates that the RF algorithm performs the prediction task differently for each dataset, i.e., based on the structure of the data and pattern graphs, different structural information is used by the different RF models for different datasets. Thus, a generic RF that generalises over multiple datasets would not be as efficient as a per-dataset RF. Each dataset in, say, a database of data graphs would require a separate RF to be constructed.

While we proposed the naïve metaheuristic algorithm using RF because of its improved performance in almost all the datasets, we observe that there are other ML algorithms that have shown better performance in various datasets. Therefore, we hypothesise that a high-level metaheuristic algorithm that uses multiple low-level ML algorithms would perform better than using a single ML algorithm (RF, in this case). We use a voting-based ensemble technique using the output of all fifteen ML algorithms as our metaheuristic algorithm. If ML_i is the output of the i^{th} ML algorithm, then the output, $\hat{y}$, of the metaheuristic algorithm is given as -

$$\hat{y} = \underset{i \to 1}{\overset{15}{\text{mode}}} [ML_i]$$

Fig. 14 shows the speedup on the different datasets using the metaheuristic algorithm developed with the voting-based ensemble method. The metaheuristic algorithm using the voting-based ensemble method shows a trend similar as seen in Fig. 11. This is expected as the former is essentially an amalgamation of the latter. Datasets like PATENTS, where most of the ML algorithms have shown a high runtime speedup, have shown increased performance using the metaheuristic

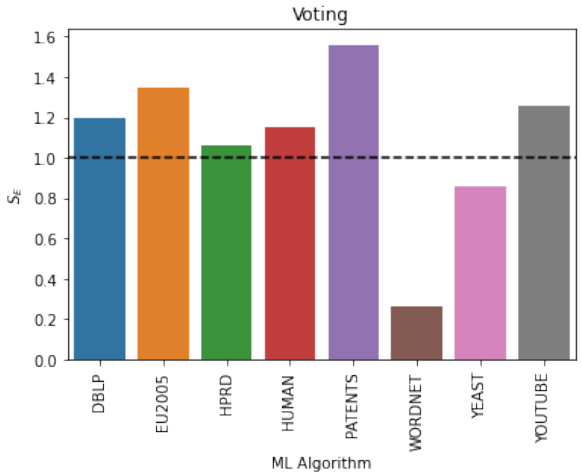


Fig. 14: Metaheuristic algorithm using voting-based ensemble method

algorithm, whereas datasets like `WORDNET`, in which all the ML algorithms have shown a degraded performance, have shown a degraded performance. In the `YEAST` dataset, where only `RF` has shown a positive speedup, the voting-based ensemble method has shown a degradation in performance because of the degraded performance of all the other ML algorithms.

6 Conclusion

Studies in the literature have applied ML-based metaheuristic techniques to several NP-complete problems like SAT, TSP, AI planning, etc. In this study, we focus on the Subgraph Isomorphism Enumeration (SIE) problem that finds application in a wide variety of domains, including, but not limited to, pattern finding and graph database query resolution. While most of the studies on the other problems in the literature have focused on using only `GF` and `LF` features, we have extended the feature types in our study by additionally using the `GLF` features, resulting in better performance using `GLF` features than `GF` features. Moreover, we have evaluated our metaheuristic technique using eight different datasets, each containing one data graph and 1800 pattern graphs. This allowed us to analyse the performance of our technique in a broader range of datasets.

We observed that metaheuristic techniques, either using a naïve single-ML `RF` algorithm or a voting-based ensemble method, can help improve SIE search performance compared to statically using any single heuristic algorithm. While this is not universally true, as can be seen from the degraded performance in the `WORDNET` and `YEAST` datasets, the application of the metaheuristic techniques improved the runtime performance on all the other datasets.

In this study, we are constrained by the limited count of the available pattern graphs in general and completely executed pattern graphs in particular. ML models require a large number of data instances to fit the training data with high generalisability while avoiding overfitting. In each of our eight datasets, we have 1800 pattern graphs, but not all of them completed execution within the given runtime threshold. For example, in the `WORDNET` dataset, only 207 pattern graphs had finished execution using all seven heuristics. Increasing the runtime threshold can help generate more data instances, but because of being NP-complete, some of the problem instances can require a copious amount of time, making it unfeasible to increase the threshold beyond a certain limit.

Contrary to the studies in the literature, we found that the metaheuristic technique does not always result in improved performance. This might be because we have evaluated our technique in a broader range of datasets. Further investigations should be performed into problems like SAT and TSP to assess if metaheuristic techniques result in degraded performance in some datasets if a higher number of datasets are considered.

Acknowledgment

The authors thank the NI-HPC Centre, funded by the Engineering and Physical Sciences Research Council (EPSRC) under Grant no. EP/T022175/1, for access to the Kelvin2 High-Performance Computing (HPC) facility for performing this study. Additionally, we thank the Ministry of Education, Govt. of India, for funding the collaboration between Queen's University Belfast (QUB) and Tezpur University (TU) under which this study was conducted.

References

1. N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*, pages 1–10. IEEE, 2015.
2. B. Bhattarai, H. Liu, and H. H. Huang. CECI: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1447–1462, 2019.
3. F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1199–1214, 2016.
4. V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(7):1–13, 2013.
5. G. Csardi, T. Nepusz, et al. The igraph software package for complex network research. *InterJournal, complex systems*, 1695(5):1–9, 2006.
6. D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Graph Algorithms and Applications I*, pages 283–309. World Scientific, 2002.
7. M. M. Farahani and S. K. Chaharsoughi. A genetic and iterative local search algorithm for solving subgraph isomorphism problem. In *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, pages 1–6. IEEE, 2015.
8. M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1429–1446, 2019.
9. W.-S. Han, J. Lee, and J.-H. Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2013.
10. I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. Asín-Achá. Improving the state-of-the-art in the traveling salesman problem: an anytime automatic algorithm selection. *Expert Systems with Applications*, 187:115948, 2022.
11. A. Jüttner and P. Madarasi. VF2++—an improved subgraph isomorphism algorithm. *Discrete Applied Mathematics*, 242:69–81, 2018.
12. M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.
13. P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.
14. L. Kotthoff, C. McCreesh, and C. Solnon. Portfolios of subgraph isomorphism algorithms. In *Learning and Intelligent Optimization: 10th International Conference, LION 10, Ischia, Italy, May 29–June 1, 2016, Revised Selected Papers*, pages 107–122. Springer, 2016.
15. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
16. H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1):364–375, 2008.
17. S. Sun and Q. Luo. In-memory subgraph matching: An in-depth study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1083–1098, 2020.
18. E.-G. Talbi. Machine learning into metaheuristics: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(6):1–32, 2021.
19. A. Tornede, L. Gehring, T. Tornede, M. Wever, and E. Hüllermeier. Algorithm selection on a meta level. *Machine Learning*, pages 1–34, 2022.
20. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.
21. H. Yun, Y. Joe, B.-O. Jung, H. Bang, and D. Shin. Solving the subgraph isomorphism problem using harmony search. In *Advanced Multimedia and Ubiquitous Engineering*, pages 151–158. Springer, 2017.

Opposition Based Local Escaping Marine Predators Algorithm for Continuous Optimization

Manish Kumar and Kusum Deep

Indian Institute of technology Roorkee, Uttrakhand, India
mkumar1@mt.iitr.ac.in, kusum.deep@ma.iitr.ac.in

Abstract. The marine predators algorithm (MPA) is a recently developed metaheuristic algorithm that is inspired by the foraging behavior of marine predators. It has been widely used to solve real-life optimization problems. However, it frequently gets trapped in a local optima since it is unable to have a diversified population in the early stages of optimization. To overcome the premature convergence problem of MPA, this paper introduces an improved version of the marine predators algorithm named as opposition-based local escaping marine predators algorithm (OLMPA). There are two ways in which the improvement is carried out. The first improvement uses opposition based learning (OBL) assures the diversity of solutions in the search space. The second improvement uses the local escaping operation, which creates new solutions that replace the worst solutions to estimate the best solution. These enhancements are designed to address the imbalance between exploration and exploitation. The proposed OLMPA is tested on 23 benchmark functions. The numerical and statistical experimental results show that the proposed algorithm overperformed classical MPA.

Keywords: Nature Inspired Optimization, Metaheuristics, Marine Predators Algorithm, Local escaping operator, Opposition based learning

1 Introduction

In recent decades, many new meta-heuristic optimization algorithms have been proposed to address the increasing complexities and challenges of real-world problems. These stochastic approaches utilize random operators to globalize the search space and avoid local optima in order to estimate the best solutions for a range of optimization problems. However, while they offer several benefits, such as being gradient-free, adaptable, and problem-independent, these strategies do not guarantee a global solution will be found once convergence is achieved. Metaheuristic optimization algorithms rely on two key strategies - exploration/diversification and exploitation/intensification - to enhance their performance. Exploration is used to investigate the entire search space, while exploitation is used to improve quality locally. Achieving optimal performance for an algorithm involves striking a balance between these two strategies, which is typically accomplished through a combination of different operators and processes that vary between population-based algorithms.

The existing literature on these algorithms can be broadly classified into three categories: improving existing algorithms, hybridizing different algorithms, and developing new algorithms. Each of these categories is active, and there is a significant body of research and applications in each one. However, as per the No Free Lunch theorem [1], there is no single optimization technique that can solve all optimization problems, and hence researchers do not rely on a single algorithm. One such attempt occurred when the marine predators algorithm was introduced in [2]. Numerous researchers have experimented with various approaches to improve the search performance of MPA. In order to accomplish this, numerous variants of MPA have been developed. Some of them are as follow: A modified marine predators algorithm is proposed by hybridization of MPA and teaching-learning-based algorithm (TLMPA) in [3], the multi-objective MPA (MOMPA) is formulated in [5], a logistic opposition-based learning (LOBL) and self-adaptive updating methods are introduced into a new version of marine predators algorithm (MMPA) in [6], an improved marine predators algorithm (ODMPA) has been developed in [7], four new versions are developed of multi-objective MPA in [8],

In order to more effectively address present problems or find answers to brand-new ones, we must modify current algorithms or propose brand-new ones. This is what prompted us to propose OLMPA, a new optimization technique. A review work on newly developed nature inspired algorithms is done in [14]. The MPA is chosen as a study because it is a newly proposed algorithm

and it attracted our attention because of its simplicity. Therefore, we tried to improve it by incorporating opposition based learning and local escaping operator. The present study contributes the following:

- Opposition based learning is used to assure the diversity of solutions in the search space.
- Local escaping operator is used which creates new solutions that replace the worst solutions to estimate the best solution.
- Developed algorithm is compared with classical MPA on 23 well known classical benchmark functions.

The article is structured as follows: Section 2 provides a brief overview of the classical MPA. Section 3 describes the proposed OLMPA. The performance of the OLMPA algorithm is evaluated on a set of benchmark functions in Section 4, and the results are discussed. Finally, Section 5 summarizes the conclusions of the paper and presents recommendations for future research.

2 Marine Predators Algorithm

The marine predators algorithm is a novel nature-inspired meta-heuristic algorithm that draws its inspiration from different tactics employed by predators to boost the rate at which they seek for prey. Predators search for their food, as well as the prey. MPA algorithm based on this concept is designed in [11].

Similarly to many other meta-heuristic algorithms, the MPA is also a population based-optimization algorithm which is designed to solve nonlinear optimization problems of the type

$$\begin{aligned} &\text{Min/Max} f(P),\ f : R^d \to R\ \text{, where } P \in R^d \\ &\text{subject to}\ \ P_{min} \le P \le P_{max} \end{aligned}$$

where d represents the dimension of the problem. The initial population P_0 is generated using the random initialization approach. This can be formulated as follows:

$$P_0 = P_{min} + rand(\ P_{max} - P_{min}) \tag{1}$$

Where P_{min} is the lower bound and P_{max} is the upper bound of the problem, and $rand$ in [0,1] is a uniformly distributed random number. This method controls the behaviour of two groups of feasible solution candidates, which are represented by the E and P matrices.

$$E = \begin{bmatrix} P^1_{1,1} & P^1_{1,2} & \dots & P^1_{1,d} \\ P^1_{2,1} & P^1_{2,2} & \dots & P^1_{2,d} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ P^1_{n,1} & P^1_{n,2} & \dots & P^1_{n,d} \end{bmatrix} \qquad P = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,d} \\ P_{2,1} & P_{2,2} & \dots & P_{2,d} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ P_{n,1} & P_{n,2} & \dots & P_{n,d} \end{bmatrix}$$

P^1 denotes the vector of the top predator, d is the dimension of the problem and n represents the number of solutions. $P_{i,j}$ denotes the jth dimension of the ith prey in P matrix. On the basis of a range of velocity ratios, MPA simulation has three major phases.

Phase-1 In the first phase of the algorithm, the prey moves faster than the predator with a high-velocity ratio. Prey migrate in a brownian distribution, and the ideal predator approach is to never move. As a result, when exploration is required during the initial iteration of the process, this phase is employed. The step size is supposed to be large in the first phase for better exploration. This stage is denoted mathematically as :

$$\text{while } Iter < 1/3\ Iter_{max}$$

$$\begin{aligned} SL_i &= B_R \otimes (\ E_i - B_R \otimes P_i) \\ P_i &= P_i + T \times R \otimes SL_i, \quad \text{where } i = 1, \dots n \end{aligned} \tag{2}$$

where brownian motion is shown by the random values called B_R , which is based on the normal distribution. $\otimes$ shows entry-wise multiplication. The value of T is equal to 0.5. R is a vector of uniform random numbers in closed interval [0,1].

Phase-2 In the second phase of the algorithm, both predator and prey search for their own food moving at the same velocity. This phase is also known as the unit velocity ratio phase. Exploration and exploitation both are happened equally in the second phase of algorithm. As a result, the population is split evenly in half. Exploration takes place in the first segment, while exploitation takes place in the second. Exploration is used by prey, while exploitation is used by predators. Prey moves with levy distribution, while predators move with brownian distribution. This stage is denoted mathematically as :

$$\text{while } 1/3Iter_{max} < Iter < 2/3Iter_{max}$$

$$\begin{aligned} SL_i &= L_R \otimes (\ E_i - L_R \otimes P_i) \\ P_i &= P_i + T \times R \otimes SL_i, \quad \text{where } i = 1, \dots n/2 \end{aligned} \tag{3}$$

where L_R is a lévy distributed random numbers, represents the lévy movement.

$$\begin{aligned} SL_i &= B_R \otimes (\ B_R \otimes E_i - P_i) \\ P_i &= E_i + T \times CF \otimes SL_i, \quad \text{where } i = n/2, \dots n \end{aligned} \tag{4}$$

where $CF = (1 - Iter/Iter_{max})^{(2Iter/Iter_{max})}$ is defined as an adaptive parameter to regulate the predator's movement's step size.

Phase-3 In this phase, predator moves faster than the prey with the low-velocity ratio, which happens in the last phase of the algorithm. The best strategy for predators is to move with the levy distribution. The mathematical formula for this phase is

$$\text{while } Iter > 2/3\ Iter_{max}$$

$$\begin{aligned} SL_i &= L_R \otimes (\ L_R \otimes E_i - P_i) \\ P_i &= E_i + T \times CF \otimes SL_i, \text{where } i = 1, \dots n \end{aligned} \tag{5}$$

Eddy formulation and FADs' effect The process of foraging may be affected by the eddy current formation of fish aggregation(FADs) [11]. So, to prevent trapping into local optima, large stepsizes are used. The jumping mode is defined as:

$$P_i = \begin{cases} P_i + CF[\ P_{min} + R \otimes (\ P_{max} - P_{min})\ ] \otimes U & when\ \alpha \leq FADs \\ P_i + [\ FADs(\ 1 - \alpha)\ + \alpha]\ (\ P_{r_1} - P_{r_2}) & when\ \alpha > FADs \end{cases} \tag{6}$$

where $FADs = .2$ shows the probability of FADs effect on the optimization process. U is a binary vector that contains an array of zeros and ones. α is a uniformly distributed random number in [0, 1]. Subscripts r_1 and r_2 stand for the P matrix's random indexes.

Memory saving in MPA Marine predators are known for their good memory. Therefore, the current solution and the solution obtained from the previous iteration are compared at the end of each iteration. The solution with higher fitness is chosen to increase population quality.

3 Proposed opposition-based local escaping marine predators algorithm (OLMPA)

The proposed algorithm OLMPA has been described in details in this section. The motivation behind the improvement is stagnation of solution in local optima. To overcome this problem local escaping operator and opposition based learning is imposed on classical MPA.

3.1 Opposition based learning

Opposition-based learning (OBL) is inspired by the opposite relationship among entities. The idea of opposite numbers was introduced in [12]. According to this concept, a solution and its opposite solution are both used in the search process to search in both directions. Hence, this concept has the potential to enhance the convergence process. Different types of computing algorithms, like artificial neural networks, optimization methods, and reinforcement learning, use the concept of OBL to improve performance.

Consider a point $p \in [$ c,d $]$ subset of real numbers R, then the opposite of p is denoted by $\widehat{p}$ and defined as $\widehat{p} = c + d - p$. where c and d are the lower and upper bound of p respectively. Also, this concept can be extended from one dimension to multiple dimensions which is defined as

Consider $P = (\ p_1, p_2, \ldots, p_m)\ \in R^m$, where R^m is m- dimensional search space. Then opposite point of P is $\widehat{P} = (\ \widehat{p_1}, \widehat{p_2}, \ldots, \widehat{p_m})\ \in R^m$ and defined as $\widehat{p_i} = a_i + b_i - p_i$, for $i = 1, 2, \ldots m$, where lower and upper bounds of p_i are a_i and b_i respectively. Let $P \in R^m$ be a feasible point and $\widehat{P}$ is its opposition point, $f(\ P\ )$ and $f(\ \widehat{P}\ )$ are objective function values at P and $\widehat{P}$ respectively. P is replaced by $\widehat{P}$ if $f(\ \widehat{P}\ ) \leq f(\ P\ )$, otherwise P is used for the rest of the generation.

3.2 Local escaping operator

In the MPA process, there is a lack of any inter-dependence of solutions; therefore, there would be an opportunity to accept solution generated by the external procedures. So, new feasible solution candidates that are generated by external computational can be injected into the population. This procedure may help the population be more diverse. Consequently, it may help to lead to the emergence of better solution conditions.

Let $S = \{P_{best}, P_{avg}, P^{rep}_{best}, P^{rep}_{worst}, P_{rnd}\}$ be the set of representative solutions, where P_{best} represent the best solution in current iteration, P_{avg} represent the average solution candidate, based on mutual similarities P^{rep}_{best} and P^{rep}_{worst} are two candidates that represent the population, P_{rnd} represent the random solution in the population.

Now, define a matrix

$$M = \begin{bmatrix} 0 & dist(\ P_1, P_2) & \ldots & dist(\ P_1, P_n) \\ dist(\ P_2, P_1) & 0 & \ldots & dist(\ P_2, P_n) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ dist(\ P_n, P_1) & dist(\ P_n, P_2) & \ldots & 0 \end{bmatrix} \tag{7}$$

where $dist()$ represent the euclidean distance, P^{rep}_{best} and P^{rep}_{worst} will be generated by the help of M. Let $S_i = \frac{1}{n}\sum_{j=1}^{n} M_{i,j}$, for $i = 1, 2, 3 \ldots n$, P^{rep}_{best} is the candidate having smallest S and P^{rep}_{worst} is the candidate having largest S. P_{avg} is calculated as $P_{avg} = \frac{1}{n}\sum_{i=1}^{n} P_i$. Now, The new candidate solution P_{in} is calculated as

$$P_{in} = P_{best} + N(\ 0, (\ 1 - \frac{Iter}{Iter_{max}})\ )\ *(\ P_{best} - P_{avg}) \tag{8}$$

where $Iter$ denote the current iterations and $Iter_{max}$ denote the maximum number of iterations to be performed. N represent the standard deviation of the normal distribution with zero mean. For more exploration and exploitation P_{in}, can be modified. for this, generate two candidates P_{h1} (9) and P_{h2} (10)

$$P_{h1} = P_{best} + N(\ 0, (\ 1 - \frac{Iter}{Iter_{max}})\ )\ *(\ P_{best} - P^{rep}_{best}) \tag{9}$$

$$P_{h2} = P_{best} + N(\ 0, (\ 1 - \frac{Iter}{Iter_{max}})\ )\ *(\ P_{best} - P^{rep}_{worst}) \tag{10}$$

and a new candidate can also be generated using bounds determined by the P matrix

$$P_{rnd} = P_{min} + rand(\ P_{max} - P_{min}) \tag{11}$$

where rand is vector containing uniformly distributed random numbers between 0 and 1. P_{rnd} is a random solution, Since P_{rnd} is generated based on the values of the P matrix. P_{in} is modified solution based on selected and generated solutions. It only works if the randomly generated number is smaller than .66. Consider P_{in} as P^1_{in} to differentiate the candidates for further analysis. Modification of P^1_{in} are obtained consecutive numbers in superscript. Hence

$$\begin{cases} P^2_{in} = rn * P^1_{in} + (\ 1 - rn)\ * P_{rnd} + R, \text{if rand} < 5 \\ P^3_{in} = rn * P_{best} + (\ 1 - rn)\ * P_{rnd} + R, \text{otherwise} \end{cases} \tag{12}$$

where rn is a uniform number in [0,1] . R is calculated as

$$R = N(\ 0, (\ 1 - \frac{Iter}{Iter_{max}})\ )\ * (\ P_{h1} - P_{h2}) \tag{13}$$

In the proposed procedure, candidate solutions generated by LEO [13] are replaced with k worst solution candidates. The pseudo code of OLMPA is presented in algorithm [1].

Algorithm 1 The OLMPA

Input $Iter_{max}$: maximum number of Iteration to be performed
n: number of feasible solutions
k: number of solutions replaced by the LEO, k < n Randomly create the population, P
calculates the fitness of each solution
construct Elite matrix by optimal solution
for $Iter = 1$ *to* $Iter_{max}$
    if $Iter < 1/3\ Iter_{max}$
        Phase-1: P update by Eq(2)
    else if *while* $1/3\ Iter_{max} < Iter < 2/3\ Iter_{max}$
        Phase-2: Update first half P by Eq(3)
            Update second half P by Eq(4)
    else if $Iter > 2/3 Iter_{max}$
        Phase-3: P update by Eq.(5)
    end
Sort P, Apply opposition-based learning to the first half population.
Marine memory saving: Replace present solutions with better (Iter -1) solutions.
Apply FADs, Eq.(6)
Sort P, determine the k worst solutions for the replacement.
Select representative solutions P^{rep}_{best} and P^{rep}_{worst} based on Eq. (7)
    for l=1 to k
        Create new solution (P_{in}) based on P_{best} and the created P_{avg}, Eq. (8)
        Create solutions P_{h_1}, P_{h_2} and P_{rnd}, Eq. (9) - Eq. (11)
        if $rand < 0.66$
            if $rand < 0.5$
                Create P^2_{in}, P_{in}= P^2_{in}, Eq. (12)
            else
                Create P^3_{in}, P_{in}= P^3_{in}, Eq. (12)
            end
        end
        Replace the P_1 with P_{in}
    end
end

Table 1: Mean, STD, Max, Min of top predator's fitness values in 30 runs of MPA, and its proposed variant OLMPA for Unimodal test functions for 10 dimensions.

Test function [10D]	algorithm	mean	STD	max	min
F1	MPA	5.29E-30	1.25E-29	6.77E-29	9.49E-33
	OLMPA	**0.0000**	**0.0000**	**0.0000**	**0.0000**
F2	MPA	5.89E-17	6.70E-17	2.63E-16	1.72E-18
	OLMPA	**2.92E-184**	**0.0000**	**8.12E-183**	**3.76E-208**
F3	MPA	2.79E-14	6.10E-14	3.15E-13	4.59E-19
	OLMPA	**1.55E-312**	**0.0000**	**4.67E-311**	**0.0000**
F4	MPA	1.36E-12	1.17E-12	4.36E-12	4.54E-14
	OLMPA	**1.83E-167**	**0.0000**	**5.10E-166**	**4.28E-194**
F5	MPA	1.81E+00	3.70E-01	2.66E+00	1.14E+00
	OLMPA	3.44E+00	4.80E-01	4.52E+00	2.26E+00
F6	MPA	1.48E-11	8.53E-12	3.69E-11	1.45E-12
	OLMPA	**1.21E-12**	**1.56E-12**	**6.42E-12**	**4.09E-18**
F7	MPA	7.80E-04	4.77E-04	2.01E-03	8.76E-05
	OLMPA	**1.80E-04**	**1.02E-04**	**3.79E-04**	**1.87E-05**

4 Numerical results and analysis

To benchmark, the performance of the proposed OLMPA algorithm, a variety of test functions [16] are chosen. Such functions are essential for validating and comparing new optimization techniques. In this paper, all the experiments are performed on MATLAB 2020b with 4 GB RAM system.

4.1 Benchmark test functions and parameter settings

Based on their properties, the benchmark problems are divided into three categories namely unimodal functions, multimodal functions, and fixed-dimension multimodal functions. Because the unimodal function has just one optimum, these functions can demonstrate diversification while also accelerating convergence. On the other hand, multimodal functions comprise a large number of optima, including many local optima and single or multiple global optima. These functions can be used to verify the proposed algorithm's ability to avoid local optima. As a result, these functions are more complex, and they should be capable of avoiding local optima stagnation by promoting exploration. Out of 23 benchmark functions F1-F7 are unimodal functions and F8-F13 are multimodal functions. Non-scalable fixed-dimensional multimodal functions are depicted by functions F14-F23.

4.2 Analysis of OLMPA for unimodal functions

The results of unimodal test functions for 10 and 30 dimensions are shown in Tables [1] and [2] respectively. Unimodal functions have a single global optimum so these functions can be used to assess exploitation potential. Except for function P5, the average values show significant gains for the proposed approach. The standard deviations show that the suggested algorithm's superiority is stable. It is also clear from the p-values of the Wilcoxon rank sum test, as given in Tables [6] and [8], that the suggested approach is statistically significant when compared to the classical MPA. It demonstrates that the suggested opposition-based model and local escaping operator implemented in OLMPA have the potential to significantly improve MPA exploration and exploitation.

4.3 Analysis of OLMPA for multimodal functions

Table [2] and Table [4] exhibit the results of multimodal test functions for 10 and 30 dimensions, respectively. Since multimodal functions include a high number of local optima that increase in

Table 2: Mean, STD, Max, Min of top predator's fitness values in 30 runs of MPA, and its proposed variant OLMPA for multimodal test functions for 10 dimensions.

Test function [10D]	**Method**	**Mean**	**STD**	**Max**	**Min**
F8	MPA	-3.56E+03	229.4403	-3140.586	-4071.39
	OLMPA	**-3.72E+03**	**2.31E+02**	**-3360.76**	**-4071.39**
F9	MPA	3.26E-11	1.78E-10	9.77E-10	0.0000
	OLMPA	**0.0000**	**0.0000**	**0.00E+00**	**0.0000**
F10	MPA	4.68E-15	1.30E-15	7.99E-15	8.88E-16
	OLMPA	**8.88E-16**	**0.0000**	**8.88E-16**	**8.88E-16**
F11	MPA	1.59E-11	8.69E-11	4.76E-10	0.0000
	OLMPA	**0.0000**	**0.0000**	**0.0000**	**0.0000**
F12	MPA	1.01E-11	6.15E-12	2.36E-11	6.21E-13
	OLMPA	**2.56E-13**	**3.53E-13**	**1.40E-12**	**7.90E-18**
F13	MPA	4.39E-11	3.00E-11	1.27E-10	8.72E-12
	OLMPA	1.37E-06	7.48E-06	4.10E-05	6.66E-15

Table 3: Mean, STD, Max, Min of top predator's fitness values in 30 runs of MPA, and its proposed variant OLMPA for Unimodal test functions for 30 dimensions.

Test function [30D]	**Method**	**Mean**	**STD**	**Max**	**Min**
F1	MPA	7.16E-23	9.28E-23	3.94E-22	3.51E-24
	OLMPA	**0.0000**	**0.0000**	**0.0000**	**0.0000**
F2	MPA	2.02E-13	2.09E-13	9.46E-13	9.65E-16
	OLMPA	**1.22E-172**	**0.0000**	**3.58E-171**	**2.09E-195**
F3	MPA	1.71E-04	3.08E-04	1.47E-03	1.74E-07
	OLMPA	**3.44E-286**	**0.0000**	**1.03E-284**	**0.0000**
F4	MPA	2.93E-09	1.79E-09	6.93E-09	4.26E-10
	OLMPA	**1.67E-158**	**9.14E-158**	**5.01E-157**	**3.16E-183**
F5	MPA	2.53E+01	4.35E-01	2.62E+01	2.43E+01
	OLMPA	**2.50E+01**	**5.06E-01**	**2.62E+01**	**2.39E+01**
F6	MPA	3.82E-08	1.90E-08	1.16E-07	1.07E-08
	OLMPA	9.90E-03	3.53E-02	1.55E-01	3.21E-09
F7	MPA	1.15E-03	6.07E-04	2.70E-03	3.38E-04
	OLMPA	**2.37E-04**	**1.78E-04**	**6.59E-04**	**3.37E-05**

number as the size of the dimension increases, they are employed to evaluate an algorithm's capacity for exploration. These tasks are capable of identifying the algorithm's exploratory behavior.

The Wilcoxon rank-sum test p-values in Table [7] and Table [9] demonstrate the significance of the suggested OLMPA algorithm.

4.4 Analysis of OLMPA for fixed dimensional functions

The table [5] shows the results of 500 iterations of fixed-dimensional test functions. These functions are also multimodal having fixed dimensions, whereas multimodal functions' dimensions can vary based on the demands of the designer. As a result, their exploration behaviour differs from that of the multimodal functions P8–P13. The outcomes of all these functions for OLMPA and the original MPA are the same. All these functions show improvement in exploration. The p-values of the Wilcoxon rank sum test in Table [10] show that the suggested OLMPA method outperforms the old one.

Table 4: Mean, STD, Max, Min of top predator's fitness values in 30 runs of MPA, and its proposed variant OLMPA for multimodal test functions for 30 dimensions.

Test function [30D]	Method	Mean	STD	Max	Min
F8	MPA	-9.08E+03	4.61E+02	-8.25E+03	-1.01E+04
	OLMPA	**-9.47E+03**	**5.39E+02**	**-8.40E+03**	**-1.10E+04**
F9	MPA	0.0000	0.0000	0.0000	0.0000
	OLMPA	**0.0000**	**0.0000**	**0.0000**	**0.0000**
F10	MPA	1.71E-12	1.85E-12	1.06E-11	2.82E-13
	OLMPA	**8.88E-16**	**0.0000**	**8.88E-16**	**8.88E-16**
F11	MPA	0.0000	0.0000	0.0000	0.0000
	OLMPA	**0.0000**	**0.0000**	**0.0000**	**0.0000**
F12	MPA	2.20E-06	1.03E-05	5.65E-05	1.85E-09
	OLMPA	2.33E-04	1.18E-03	6.44E-03	2.46E-10
F13	MPA	8.28E-03	1.09E-02	4.39E-02	2.76E-08
	OLMPA	4.09E-01	4.48E-01	1.80E+00	3.69E-05

Table 5: Mean, STD, Max, Min of top predator's fitness values in 30 runs of MPA, and its proposed variant OLMPA for fixed dimensional test functions

Test function	Method	Mean	STD	Max	Min
F14	MPA	9.9800E-01	1.3983E-16	9.9800E-01	9.9800E-01
	OLMPA	9.9800E-01	**4.1233E-17**	9.9800E-01	9.9800E-01
F15	MPA	3.0749E-04	4.6170E-15	3.0749E-04	3.0749E-04
	OLMPA	3.9905E-04	2.7940E-04	1.2230E-03	3.0749E-04
F16	MPA	-1.0316E+00	4.6100E-16	-1.0316E+00	-1.0316E+00
	OLMPA	-1.0316E+00	3.5635E-15	-1.0316E+00	-1.0316E+00
F17	MPA	3.9789E-01	6.4496E-15	3.9789E-01	3.9789E-01
	OLMPA	3.9789E-01	1.1567E-13	3.9789E-01	3.9789E-01
F18	MPA	3.0000E+00	2.0031E-15	3.0000E+00	3.0000E+00
	OLMPA	3.0000E+00	1.2345E-15	3.0000E+00	3.0000E+00
F19	MPA	-3.8628E+00	2.4057E-15	-3.8628E+00	-3.8628E+00
	OLMPA	-3.8628E+00	2.7101E-15	-3.8628E+00	-3.8628E+00
F20	MPA	-3.3220E+00	1.6031E-11	-3.3220E+00	-3.3220E+00
	OLMPA	-3.2744E+00	5.9241E-02	-3.2031E+00	-3.3220E+00
F21	MPA	-1.0153E+01	2.8132E-11	-1.0153E+01	-1.0153E+01
	OLMPA	-1.0153E+01	**7.0129E-15**	-1.0153E+01	-1.0153E+01
F22	MPA	-1.0403E+01	2.7979E-11	**-1.0403E+01**	**-1.0403E+01**
	OLMPA	-1.0403E+01	**7.3759E-16**	-1.0403E+01	-1.0403E+01
F23	MPA	-1.0536E+01	4.7180E-11	-1.0536E+01	-1.0536E+01
	OLMPA	-1.0536E+01	**2.1377E-15**	-1.0536E+01	-1.0536E+01

4.5 Convergence analysis

To analyse the convergence behaviour of a few functions from unimodal, multimodal, and fixed dimensions, their convergence graphs are drawn in figures [1-3]. The number of generations is displayed on the X-axes, and the best objective function values are determined on the Y-axes for each function.

The figure [1] depicts the convergence graph of six unimodal functions (P1, P2, P3, P4, P5, P7). This unimodal functions' convergence graph demonstrates that the proposed OLMPA exhibits convergence behaviour beginning with the first generation and increasing significantly as the number of iterations increases. Compared to the classical MPA, convergence is occurring significantly more rapidly. Because unimodal functions can attest to a function's exploitation and convergence behavior, the significant increases in the convergence of these functions clearly demonstrate that proposed OLMPA outperforms over the MPA.

The figure [2] depicts the convergence behavior of four multimodal functions (P9, P10, P11, P12) and the figure [3] shows the convergence behaviour of four fixed-dimension test functions (P15, P21, P22, P23). Because multimodal test functions have several local optima, an algorithm should be able to traverse the search region quickly such that it avoids local optima and tends to the

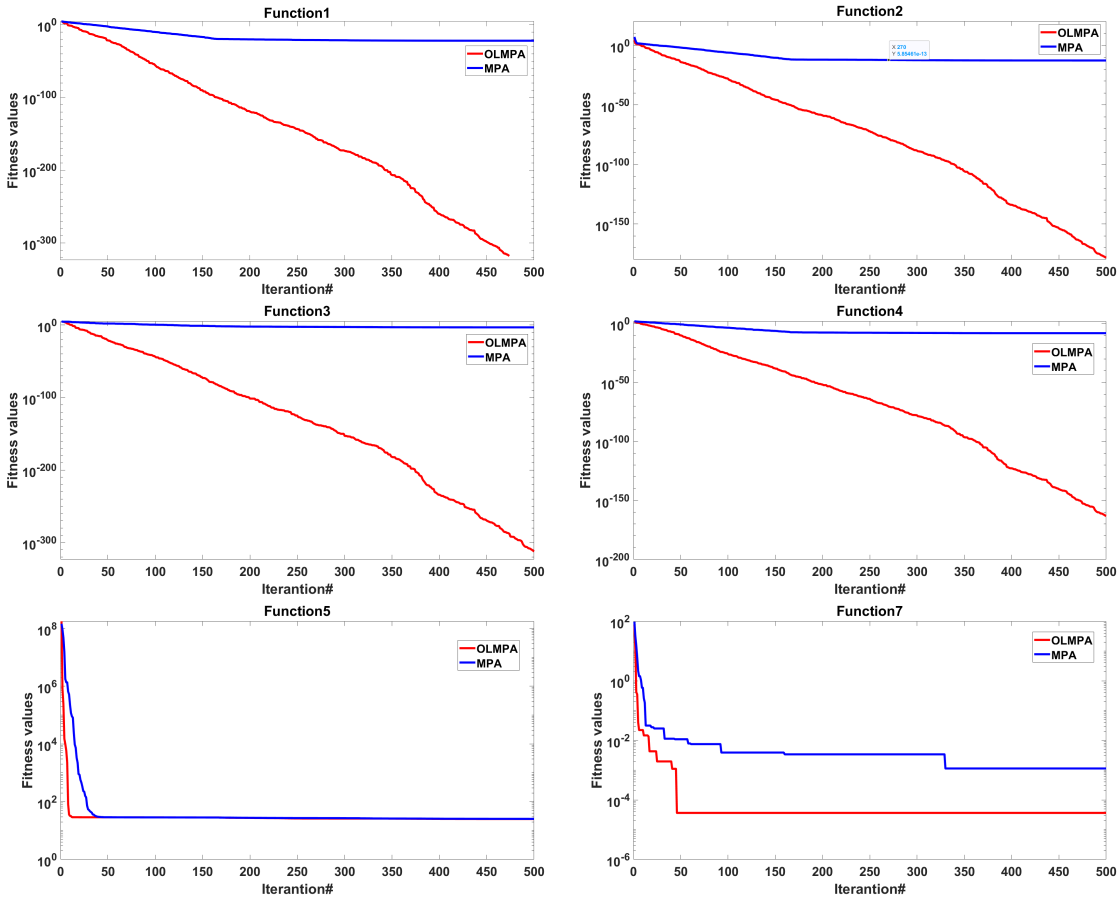


Fig. 1: Convergence of six unimodal test functions P1, P2, P3, P4, P5, and P7 using MPA and OLMPA

global optimal solution. The convergence graphs for three multimodal functions demonstrate that the proposed OLMPA converges substantially more quickly in the initial generations. The figure [3] shows the convergence behaviour of fixed dimensional functions and shows that the exploration rate of OLMPA is better than the classical MPA. This investigation demonstrates that the proposed OLMPA algorithm is quite capable of locating global optima.

4.6 Statistical analysis

The non-parametric pair-wise Wilcoxon test was used to assess the statistical validity of the MPA and OLMPA results. The test is carried out at 5 percent level of significance. Based on the p-values obtained from the Wilcoxon rank-sum test, the proposed OLMPA is worse than the original MPA for (p-value > 0.1) , slightly worse than the original MPA for (p-value ≤ 0.1) , equal to the original MPA for (p-value $= 0.1$) , considerably better than the original MPA for (p-value < 0.05) , and high significant than the original MPA for (p-value ≤ 0.01) , the grades are awarded as "C", "C+", "B", "A", and "A+" respectively. Tables [6 - 10] contain the results of p-value calculations.

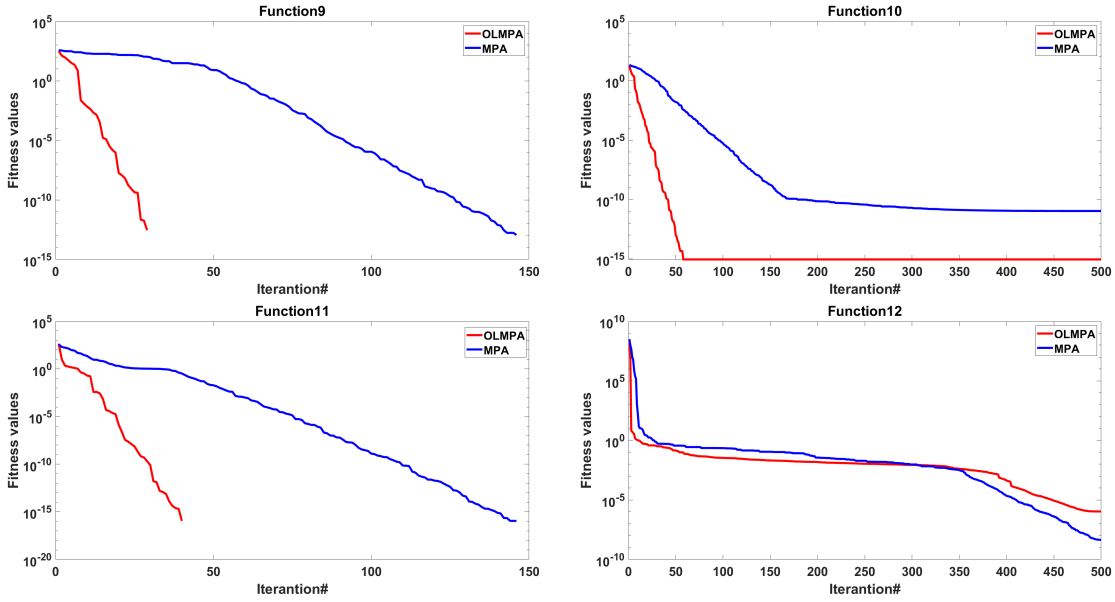


Fig. 2: Convergence of four multi-modal test functions P9, P10, P11, and P12 using MPA and OLMPA

Table 6: Statistical results on unimodal test functions by Wilcoxon Rank sum for 10 dimensions.

Test function [10D]	p-value(Wilcoxon test)	Conclusion
F1	1.21E-12	A+
F2	3.02E-11	A+
F3	4.11E-12	A+
F4	3.02E-11	A+
F5	4.50E-11	A+
F6	1.61E-10	A+
F7	4.31E-08	A+

Table 7: Statistical results on multi-modal test functions by Wilcoxon Rank sum for 10 dimensions

Test function [10D]	p-value (Wilcoxon test)	Conclusion
F8	4.23E-03	A+
F9	3.34E-01	A+
F10	3.60E-13	A+
F11	0.04192	A
F12	5.49E-11	A+
F13	4.80E-07	A+

Table 8: Statistical results on unimodal test functions by Wilcoxon Rank sum for 30 dimensions.

Test function [30D]	p-value (Wilcoxon test)	Conclusion
F1	1.21E-12	A+
F2	1.21E-12	A+
F3	2.95E-11	A+
F4	3.02E-11	A+
F5	1.11E-03	A+
F6	3.83E-05	A+
F7	9.76E-10	A+

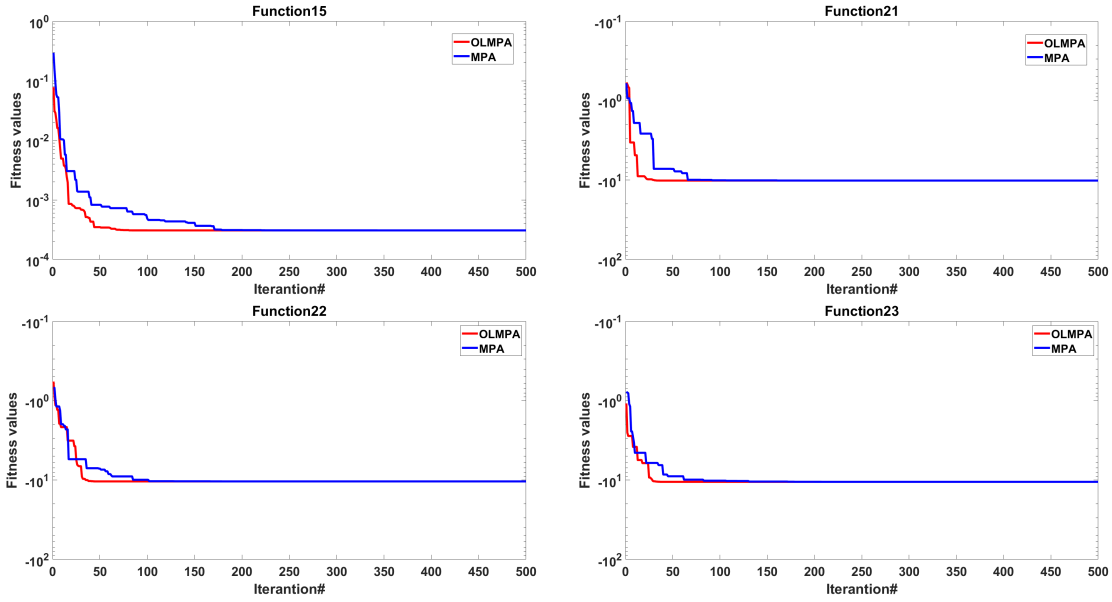


Fig. 3: Convergence of four fixed dimensional test functions P15, P21, P22, and P23 using MPA and OLMPA

Table 9: Statistical results on multi-modal test functions by Wilcoxon Rank sum for 30 dimensions.

Test function [30D]	p-value(Wilcoxon test)	Conclusion
F8	4.03E-03	A+
F9	NaN	NaN
F10	1.21E-12	A+
F11	NaN	NaN
F12	2.28E-05	A+
F13	4.62E-10	A+

Table 10: Statistical results on fixed dimensional test functions by Wilcoxon Rank sum.

Test function	p-value(Wilcoxon test)	Conclusion
F14	NaN	NaN
F15	2.47E-08	A+
F16	2.15E-02	A
F17	5.87E-02	C+
F18	5.36E-03	A
F19	NaN	NaN
F20	1.78E-01	C
F21	1.21E-12	A+
F22	1.21E-12	A+
F23	1.21E-12	A+

5 Conclusion

This research offers a modified version of the previously established marine predators algorithm, named OLMPA. The local escaping operator and opposition-based learning are used for this purpose on the classical MPA. These operators are capable of improving macro search in the early stages of development while simultaneously providing balance for micro search in the latter stages of generations. The effective combination of these operators accelerates convergence and ensures the achievement of a global optimum. The proposed OLMPA's performance is validated against standard benchmark functions consisting of unimodal functions, multimodal functions, and fixed-

dimensional multimodal functions issues of varying dimensions and complexity levels. The obtained results are validated using the Wilcoxon rank-sum test and analysed with the help of convergence curves. To test the robustness of the proposed OLMPA algorithm, a good set of 23 common benchmark problems are used. The comparison between the proposed OLMPA and the classical MPA demonstrates that OLMPA is really competitive with the classical MPA. This proposed algorithm offers a new direction for improving search capabilities so that real-world application problems can be solved.

In the coming decades, OLMPA may be used to solve a wide range of optimization problems, such as constrained optimization problems, integer programming problems, and so on.

Authors' contributions Manish Kumar and Kusum Deep conceived this research and designed experiments. Manish Kumar participated in editing and drafting the article.

References

1. Wolpert, David H., and William G. Macready.:No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1, no.1, 67-82(1997)
2. Faramarzi, Afshin, Mohammad Heidarinejad, Seyedali Mirjalili, and Amir H. Gandomi.:Marine Predators Algorithm: A nature-inspired metaheuristic. Expert systems with applications 152, 113377(2020)
3. Zhong, Keyu, Qifang Luo, Yongquan Zhou, and Ming Jiang.:TLMPA:Teaching-learning-based Marine Predators algorithm. Aims Mathematics 6, no. 2 : 1395-1442(2021)
4. Hans, R., Kaur, H.:Opposition-based enhanced grey wolf optimization algorithm for feature selection in breast density classification. International Journal of Machine Learning and Computing, 10(3), 458-464,(2020)
5. Jangir, Pradeep, Hitarth Buch, Seyedali Mirjalili, and Premkumar Manoharan.:Multi-objective marine predator algorithm for solving multi-objective optimization problems. Evolutionary Intelligence 16, no. 1 169-195(2023)
6. Fan, Qingsong, Haisong Huang, Qipeng Chen, Liguo Yao, Kai Yang, and Dong Huang.:A modified self-adaptive marine predators algorithm: framework and engineering applications. Engineering with Computers : 1-26(2021)
7. Hu, Gang, Xiaoni Zhu, Guo Wei, and Ching-Ter Chang.:An improved marine predators algorithm for shape optimization of developable Ball surfaces. Engineering Applications of Artificial Intelligence 105 104417(2021)
8. Abdel-Basset, Mohamed, Reda Mohamed, Seyedali Mirjalili, Ripon K. Chakrabortty, and Michael Ryan.:An efficient marine predators algorithm for solving multi-objective optimization problems: analysis and validations, IEEE Access 9 42817-42844(2021)
9. Shaheen, Abdullah M., Abdallah M. Elsayed, Ragab A. El-Sehiemy, Salah Kamel, and Sherif SM Ghoneim.:A modified marine predators optimization algorithm for simultaneous network reconfiguration and distributed generator allocation in distribution systems under different loading conditions. Engineering Optimization 54, no. 4, 687-708(2022)
10. Sobhy, Mohamed A., Almoataz Y. Abdelaziz, Hany M. Hasanien, and Mohamed Ezzat.:Marine predators algorithm for load frequency control of modern interconnected power systems including renewable energy sources and energy storage units. Ain Shams Engineering Journal 12, no. 4, 3843-3857(2021)
11. Faramarzi, Afshin, Mohammad Heidarinejad, Seyedali Mirjalili, and Amir H. Gandomi.: Marine Predators Algorithm: A nature-inspired metaheuristic. Expert systems with applications 152 113377(2020)
12. Tizhoosh, Hamid R.:Opposition-based learning: a new scheme for machine intelligence. In International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06), vol. 1, pp. 695-701, IEEE(2005).
13. Oszust, Mariusz.:Enhanced marine predators algorithm with local escaping operator for global optimization. Knowledge-Based Systems 232 107467(2021)
14. Rajwar, Kanchan, Kusum Deep, and Swagatam Das.:An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. Artificial Intelligence Review 1-71(2023)
15. Kononova, Anna V., David W. Corne, Philippe De Wilde, Vsevolod Shneer, and Fabio Caraffini.:Structural bias in population-based algorithms. Information Sciences 298, 468-490(2015)
16. Gupta, Shubham, and Kusum Deep.:Improved sine cosine algorithm with crossover scheme for global optimization. Knowledge-Based Systems 165, 374-406 (2019)

An improved multi-objective genetic algorithm for the neural architecture search problem

Eloy Bedia-García[1] and Enrique Domínguez[1]

Dept. of Computer Science
University of Malaga
eloybg97@uma.es, enriqued@lcc.uma.es

1 Introduction

In recent years, there is a great interest in automating the process of searching for neural network topology. This problem is called Neural Architecture Search (NAS), which can be seen as a 3-gear mechanism: the search space, the error estimation and the search strategy.

The search space defines what kind of neural architectures can be reached, and it can be divided into three types:

- A search space restricted to sequential models. These models consist of consecutive layers $(L_0.L_1, ..., L_n)$, where layer L_i receives as input the output of layer L_{i-1}
- A search space with more complex architectures where the input of the i-th layer is defined by the function $g_i(L_0, ..., L_{i-1})$. Examples of this search space are the Residual Networks [4] or the DenseNets [5].
- The search space of block-based architectures [12], [11]. These are based on a design pattern that consists of the consecutive repetition of a set of layers called blocks.

To guide the selected strategy throughout the search space, we need a metric to help us. The simplest way is to evaluate the error obtained in the validation set, however, due to the long computation times required, alternative methods are being searched for, such as: reducing the training set [12], reducing the number of epochs [10, 12], using less filters [7, 12] or using lower resolution images [2].

Different kind of search strategies have been proposed in the literature mainly based on reinforcement learning with a reinforce policy [9], a proximal policy optimization [12] or Q-learning [1]. Other strategies based on evolutionary algorithms have been also proposed with different types of genetic operators, such as using selection by tournament [7, 8], eliminating the worst individual in each generation [8], eliminating the oldest individual [7], or using Lamarkian inheritance [3] for the offspring generation.

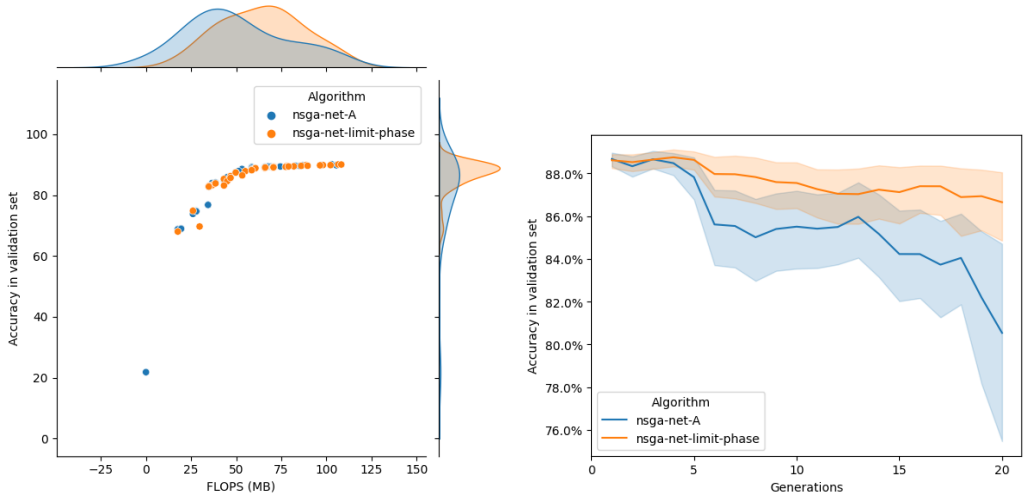
In this paper, we propose an improved version of the NSGA-Net algorithm [6], which is a multi-objective genetic algorithm for the NAS problem. One of the drawbacks is the limited diversity that can be generated by the original crossover operator, which generates only one offspring keeping the common genomes, and leaving the rest randomly. In order to avoid this limitation, we proposed a new 2-point crossover restricting the possible cutoff points only to the block limits.

The rest of the paper is organized as follows. In the next section, some preliminary results are shown on the well-known CIFAR-10 dataset. Finally, some remarks and future works are presented in the conclusions section.

2 Experiments

In this section, we present the empirical results to show the efficacy of the proposed operators for the NSGA-Net algorithm to automate the NAS process on the CIFAR-10 benchmark. Two objectives were considered to guide our NSGA-Net based algorithm: the classification error on the validation set and the computational complexity measured as the number of floating point operations (FLOPs) needed to execute the forward pass of the neural network. The CIFAR-10 dataset was considered for the classification task, splitting the original training set into our training (80%) and validation set (20%) for the neural architecture search. The original testing set was only used to obtain the test accuracy of the final models.

Regarding to the genetic parameters, the population was randomly initialized with size fixed to 40 during the 20 generations for exploration and 10 generations for exploitation. The coding of the genomes consists in a sequence of 3 blocks, each containing a maximum of 4 nodes. The training of the neural networks is carried out for 25 epochs, with a learning rate of 0.025 descending according to the cosine annealing scheme, and a batch size of 128.



(a) Objectives space distribution (b) Evolution of the population accuracy

Fig. 1: NSGA-Net vs LimitPhase on CIFAR-10

Figure 1 shows the comparative result between the original NSGA-Net and our proposed algorithm on the CIFAR-10 dataset. The distribution of the solutions provided by both algorithms is shown in figure 1a, where our proposal (NSGA-LimitPhase) presents a higher accuracy concentration in values around 90%, although with greater complexity (FLOPs). This information is also reflected in figure 1b, where the evolution of the accuracy of the population models of NSGA-LimitPhase presents a better performance. Additionally, in this figure we can see the deterioration of the NSGA-Net population throughout the generations, which it is accentuated at the end. However, our proposal presents accuracies more stable at high values with a smooth downward trend at the end.

3 Conclusions

In this work, we have proposed the use of a new crossover to improve the NSGA-Net algorithm for the NAS problem. Experimental results show a better accuracy distribution in the objectives space, and a better accuracy evolution on the CIFAR-10 dataset. These results are very promising for the treatment of other datasets/benchmarks in order to design a good algorithm for solving the NAS problem. Further works include both the application of the proposed algorithm to other datasets and the comparison to other state-of-the-art algorithms for the NAS problem.

Acknowledgement

The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the SCBI (Supercomputing and Bioinformatics) center of the University of Málaga. They also gratefully acknowledge the support of NVIDIA Corporation with the donation of a RTX A6000 GPU with 48Gb.

References

1. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning
2. Chrabaszcz, P., Loshchilov, I., Hutter, F.: A downsampled variant of ImageNet as an alternative to the CIFAR datasets https://www.semanticscholar.org/paper/A-Downsampled-Variant-of-ImageNet-as-an-Alternative-Chrabaszcz-Loshchilov/e644a409b4a4c6eaedffe27efbc5c76280b34c61
3. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution. https://doi.org/10.48550/arXiv.1804.09081, http://arxiv.org/abs/1804.09081
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778. https://doi.org/10.1109/CVPR.2016.90, ISSN: 1063-6919
5. Huang, G., Liu, Z., Weinberger, K.: Densely connected convolutional networks p. 12
6. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: Neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 419–427. GECCO '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3321707.3321729, https://doi.org/10.1145/3321707.3321729
7. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Aging evolution for image classifier architecture search. https://www.semanticscholar.org/paper/Aging-Evolution-for-Image-Classifier-Architecture-Real-Aggarwal/7bac3d11824fabe0dc3ac2fee9bfb667e82fba9c
8. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q., Kurakin, A.: Large-scale evolution of image classifiers. https://doi.org/10.48550/arXiv.1703.01041, http://arxiv.org/abs/1703.01041
9. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning **8**(3), 229–256. https://doi.org/10.1007/BF00992696, https://doi.org/10.1007/BF00992696
10. Zela, A., Klein, A., Falkner, S., Hutter, F.: Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. https://doi.org/10.48550/arXiv.1807.06906, http://arxiv.org/abs/1807.06906
11. Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C.L.: Practical block-wise neural network architecture generation. pp. 2423–2432. https://doi.org/10.1109/CVPR.2018.00257
12. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710. https://doi.org/10.1109/CVPR.2018.00907

Success Rate Based Scaling Factor Adaptation in Dual-Population Differential Evolution

V. Stanovov[1] and E. Semenkin[1]

[1]Siberian Federal University, 79 Svobodny pr., 660041, Krasnoyarsk, Russian Federation
vladimirstanovov@yandex.ru, eugenesemenkin@yandex.ru

1 Introduction

The heuristic numerical optimization methods, such as evolutionary algorithms (EA) and swarm intelligence (SI) approaches are an important part of the computational intelligence (CI) field, as they are often used for solving real-world problems, and serve as parts of other algorithms. However, their applicability is often limited due to the large number of parameter values that should be set in order to achieve desirable performance on a problem at hand. Hence, the development of parameter adaptation mechanisms and algorithms, which are less sensitive to parameter settings, is an important research direction.

The differential evolution (DE) is currently one of the most popular evolutionary algorithms for numerical optimization due to simplicity and high efficiency. However, since the original proposal in [8], it is known that DE is highly sensitive to parameter settings, thus a huge number of studies is aimed at proposing novel parameter adaptation schemes for scaling factor F and crossover rate Cr, as well as population size [11]. One of the most influential studies [6] proposed the success-history based adaptive DE (SHADE) algorithm, variations of which are used in numerous prize-winning algorithms. Most attempts to improve SHADE's parameter adaptation have not resulted in significant breakthrough, also some development is clearly observed [1].

In this study a new strategy is proposed for scaling factor F adaptation in DE. It is based on the success rate value, i.e. the number of successful solutions generated within one generation divided by total population size, and is inspired by the findings presented in [2], where the genetic programming was applied to design parameter adaptation schemes. In this study these ideas are further developed, and applied to a recently proposed L-NTADE algorithm [3]. It is shown that on two sets of benchmark functions, used in Congress on Evolutionary Computation 2017 [4] and 2022 [5], the modified algorithm is capable of achieving much better results, and competing with the best proposed algorithms, while using the same set of parameters.

The paper is organized as follows: section 2 provides the related work and describes parameter adaptation schemes in differential evolution, section 3 contains the description the proposed success rate based method, section 4 contains the experimental setup and results, as well as their discussion, and section 5 concludes the paper.

2 Related Work: Differential Evolution

Differential Evolution is a population-based heuristic numerical optimization method, proposed by Storn and Price in [8]. The simplicity of its algorithmic scheme, small number of control parameters and high efficiency for local and global optimization have made it popular among researchers [10]. The name of the algorithm comes from the mutation operator, where new solutions are generated using difference between solutions in the population.

The initialization creates population, i.e. a set of NP individuals $x_i = (x_{i,1}, x_{i,2}, ..., x_{i,D})$, where D is the problem dimension. In most studies random generation of solutions with uniform distribution is used:

$$S = \left\{ x_i \in R^D | x_i = (x_{i,1}, x_{i,2}, ..., x_{i,D}) : x_{i,j} \in [x_{lb,j}, x_{ub,j}] \right\} \quad (1)$$

where $x_{lb,j}$ and $x_{ub,j}$ are the lower and upper bounds for variable j.

After initialization the mutation step is performed. Nowadays the most popular mutation scheme is called current-to-pbest, proposed in JADE algorithm [9]:

$$v_{i,j} = x_{i,j} + F \times (x_{pbest,j} - x_{i,j}) + F \times (x_{r1,j} - x_{r2,j}), \quad (2)$$

where v_i is the mutant vector, F is the scaling factor parameter, *pbest* is the index of one of the $pb * 100\%$ best individuals, $r1$ and $r2$ are uniformly generated random indexes from $[1, NP]$, $i = 1, 2, ...NP$, $j = 1, 2, ...D$. Indexes $r1$, $r2$ and *pbest* are generated to be different from each other and index i.

After mutation the trial vector is generated using crossover, which combines the trial vector x_i and mutant vector v_i with probability Cr. Most studies use binomial crossover:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } rand(0,1) < Cr \text{ or } j = jrand \\ x_{i,j}, & \text{otherwise} \end{cases}. \tag{3}$$

where $jrand$ is a random index in range $[1, D]$, required to make sure that at least one component is taken from mutant vector - otherwise the u_i could be the same as x_i.

After the new solution u_i is generated, the bound constraint handling method is applied and the target function value $f(u_i)$ is calculated. Next, the selection (replacement) operation is performed as follows:

$$x_i = \begin{cases} u_i, & \text{if } f(u_i) \leq f(x_i) \\ x_i, & \text{if } f(u_i) > f(x_i) \end{cases}. \tag{4}$$

The selection in DE replaces the target individual x_i with a new one only if it is better in terms of objective function.

The DE has only three main parameters: F, Cr and NP, but the algorithm is highly sensitive to their settings. This lead to many studies, which proposed parameter adaptation schemes, mainly for scaling factor and crossover rate. One of the most important is the SHADE algorithm [6], where success-history based adaptation (SHA) was proposed, being a further development of JADE. In SHADE there are H memory cells, each containing a pair of values $(M_{F,h}, M_{Cr,h})$, $h = 1, 2, ...H$. These values are used to generate new parameters in the following way:

$$\begin{cases} F = randc(M_{F,k}, 0.1) \\ Cr = randn(M_{Cr,k}, 0.1) \end{cases}. \tag{5}$$

where $randc(l, s)$ is a Cauchy distributed random value, and $randn(l, s)$ is a normally distributed random value, l is the location parameter, s is the scale parameter. For generation one of the memory cells with index k is randomly chosen, $k \in [1, H]$. If the generated value $F > 1$, then it is set to $F = 1$, and if $F < 0$, it is generated again until it falls within $(0, 1]$. The Cr value is truncated to $[0, 1]$.

The sampled F and Cr values are used to create new trial solution, and if this solution is successful, i.e. better than target vector, then F and Cr are stored into arrays S_F and S_{Cr}. In addition, the improvement value is stored: $S_{\Delta f} = f(u_i) - f(x_i)$. At the end of the generation these arrays are used to update the one of the H memory cells using weighted Lehmer mean:

$$mean_{wL} = \frac{\sum_{j=1}^{|S|} w_j S_j^{pm}}{\sum_{j=1}^{|S|} w_j S_j^{pm}}, \tag{6}$$

where $w_j = \frac{S_{\Delta f_j}}{\sum_{k=1}^{|S|} S_{\Delta f_k}}$, $S_{\Delta f_j} = |f(u_j) - f(x_j)|$ and S is either S_{Cr} or S_F, pm is set to 2. The new memory cell value is calculated in the following way:

$$\begin{cases} M_{F,k}^{t+1} = 0.5(M_{F,k}^t + mean_{(wL,F)}) \\ M_{CR,k}^{t+1} = 0.5(M_{Cr,k}^t + mean_{(wL,Cr)}) \end{cases}, \tag{7}$$

where t is the generation number.

The SHADE algorithm has received significant attention, starting with the L-SHADE [7], which proposed the Linear Population Size Reduction to control the third main parameter NP:

$$NP^g = round(\frac{NP_{min} - NP_{max}}{NFE_{max}} NFE) + NP_{max} \tag{8}$$

where g is the generation number, NP_{min} is usually set to 4, NFE and NFE_{max} are the current and total number of function evaluations, NP_{max} is the initial population size.

Other studies have proposed specific rules for parameter adaptation in jSO [12], rank-based selective pressure in L-SHADE-RSP [13], hybrids with CMA-ES algorithm in LSHADE-SPACMA [15], non-linear population size reduction and adaptive archive usage in NL-SHADE-RSP [14], and many others. However, most of them relied on a mostly similar main scheme with one population, external archive and success-history based adaptation for F and Cr.

Recently the L-NTADE algorithm was proposed [2], in which two populations are used, one with the newest individuals x_i^{new}, $i = 1, \ldots, N_{max}$, and the other - with the top individuals x_i^{top}, $i = 1, \ldots, NP_{max}$, i.e best known from the whole search process. During initialization, both populations are set to have the same random solutions. Inspired by the unbounded DE [16], where all generated individuals are stored and used to generate new solutions, the L-NTADE uses specific mutation strategy, which is derived from current-to-pbest and uses both populations and called r-new-to-ptop/n/t:

$$v_{i,j} = x_{r1,j}^{new} + F \times (x_{pbest,j}^{top} - x_{i,j}^{new}) + F \times (x_{r2,j}^{new} - x_{r3,j}^{top}) \tag{9}$$

where $r1$, $r2$ and $r3$ are randomly chosen indexes from corresponding populations, the terms *r-new* stands for the choice of a random individual from the newest population as a target solution, *ptop* for the choice of one of the $pb\%$ best individuals from the top population.

The crossover step in L-NTADE is unchanged, however, the selection step is different. Once a new solution u_i is generated by crossover, it is compared to the randomly chosen for mutation with index $r1$:

$$x_{nc} = \begin{cases} u_i, & \text{if } f(u_i) \leq f(x_{r1}^{new}) \\ x_{nc}, & \text{if } f(u_i) > f(x_{r1}^{new}) \end{cases}. \tag{10}$$

The main idea of an update here is similar: if the trial vector is better than the target, it should be saved, however, the new solutions are always saved to the newest population, but with different index nc, which is iterated between 1 and NP_{cur}, where NP_{cur} is the current population size, controlled by Linear Population Size Reduction, applied to both newest and top populations. Such selection mechanism means that the newest population is continuously updated, and better solutions could be replaced by worse ones, but only if there is some improvement compared to other solution.

In addition, all newly generated solutions are stored in a temporary pool x^{temp}, and at the end of a generation the top population is updated by sorting a joined set of x^{temp} and x^{top} and selecting the best NP_{cur} solutions. This means that x^{top} always contains the best known solutions so far.

L-NTADE also uses exponential rank-based selection was implemented by selecting an individual depending on its fitness in a sorted array, with the ranks assigned as follows:

$$rank_i = e^{frac - kp \cdot iNP}, \tag{11}$$

where kp is the parameter controlling the pressure. The selective pressure is applied when generating the $r2$ index, i.e. for the newest population.

The L-NTADE algorithm was shown to be a highly competitive algorithm, and will be used as a baseline approach in this study.

3 Proposed approach: success rate based scaling factor adaptation

The development of new parameter adaptation schemes is a challenging task, as it requires new ideas on how to connect some of the statistical characteristics of an algorithm during its run with the parameter values that should be used. In a recent study on automatic design of parameter adaptation strategies in [2], the genetic programming (GP) was used to generate equations, which would combine input variables, such as current spent resource, current success rate, as well as values generated by SHA to set the location parameter for sampling F and Cr values. The performed experiments have shown that GP is capable of designing efficient parameter adaptation schemes which are different from those created by researchers. One of the important results of that study was that GP solutions relied on the success rate, i.e. the number of successful solutions divided by the total population size.

Based on this insight revealed by GP, here the following parameter adaptation technique is proposed. First, the success rate SR is calculated as follows:

$$SR = \frac{NS}{NP_{cur}} \tag{12}$$

where NS is the number of successful solutions, equal to $|S|$, i.e. cardinality of set S_F or S_{Cr}, used in success history adaptation. The success rate SR is then used to set the mean scaling factor value MF:

$$MF = SR^{\frac{1}{c}} \tag{13}$$

where c is the parameter value. Such equation is used, as in [2] the values generated by GP were usually larger than success rate. The MF value is further used to sample scaling factor values F for mutation:

$$F = randc(MF, 0.1) \tag{14}$$

In other words, the proposed method could be simplified to a single equation:

$$F = randc((\frac{NS}{NP_{cur}})^{\frac{1}{c}}, 0.1) \tag{15}$$

Figure 1 shows the different MF values, which will be used for F sampling with different c parameter values.

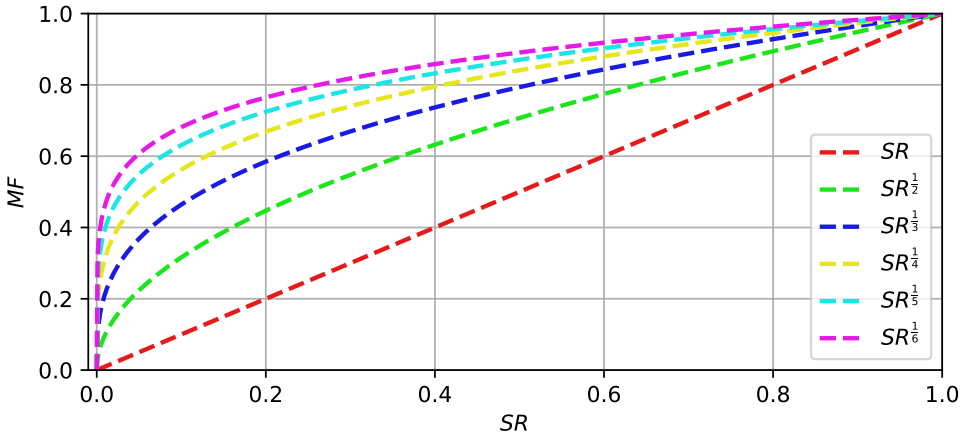


Fig. 1. Mean MF for scaling factor sampling with different c parameter values

As can be seen from Figure 1, with increased c the MF values are shifted up, and small success rates result in relatively high MF values, i.e. for example $SR = 0.05$ may lead to MF close to 0.5.

The proposed method will be further referred to as success rate based adaptation (SRA), and applied to the L-NTADE algorithm, resulting in L-NTRDE approach (Linear population size reduction Newest and Top success Rate adaptive Differential Evolution). The rest of the algorithm stays unchanged, only F generation is replaced. Same as in SHADE, if $F > 1$, it is set to 1, and if $F < 0$, it is sampled again. For Cr the success history based adaptation is used.

4 Experimental Setup and Results

To evaluate the performance of the proposed SRA approach, two sets of benchmark functions were used, namely the Congress on Evolutionary Computation competition on single-objective optimization from 2017 [4] and 2022 [5]. These benchmarks are used as they have different settings, i.e. different number of problems, different dimensions and different computational resource limitations.

The CEC 2017 contains 30 test functions for dimensions $10D$, $30D$, $50D$ and $100D$, and the computational resource is set to $NFE_{max} = 10000D$ evaluations. In CEC 2022 benchmark there are 12 test functions, dimensions $10D$ and $20D$, and the resource NFE_{max} is set to 200000 for $10D$, and 1000000 for $20D$. For CEC 2017 there are 51 independent runs required to compare algorithms, and for CEC 2022 - 30 independent runs.

The proposed L-NTRDE algorithm was implemented in C++, compiled with GCC under Ubuntu 20.04, and ran on an OpenMPI-powered cluster of 8 AMD Ryzen 3700 PRO processors. The post-processing of the results was performed using Python 3.8.

The following parameters were used for the L-NTRDE algorithm: $N_{max} = 20D$, $N_{min} = 4$, $H = 5$, $M_{Cr,r} = 1$, $pb = 0.3$, selective pressure parameter $kp = 3$, Lehmer mean parameter $pm = 4$. For the L-NTADE the memory cells were set as $M_{F,r} = 0.3$.

In the first set of experiments the influence of different c parameter values are evaluated, for this purpose the L-NTRDE is tested with $c = 1, 2, 3, 4, 5, 6$ and compared to L-NTADE. For comparison the Mann-Whitney rank sum statistical test with normal approximation and tie-breaking was applied, the significance level was set to $p = 0.01$, which corresponds to the threshold Z score of $|2.58|$ (two-tailed test). Estimating Z-score simplifies the reasoning and estimating the difference between algorithm variants. Table 1 contains the comparison on CEC 2017 benchmark functions, in every cell are the number of wins/ties/losses. The values in the brackets represent the total standard score, i.e., sum of all standard scores of the Mann–Whitney test over all test functions.

Table 1. Comparison of L-NTRDE to L-NTADE with different c parameter values, CEC 2017, Mann–Whitney tests and total standard score.

Algorithms	**10D**	**30D**	**50D**	**100D**
L-NTRDE ($c = 1$) vs	2/10/18	0/8/22	0/4/26	1/2/27
L-NTADE	(-105.68)	(-192.68)	(-218.98)	(-219.12)
L-NTRDE ($c = 2$) vs	6/15/9	3/14/13	4/8/18	5/3/22
L-NTADE	(-11.94)	(-65.01)	(-108.84)	(-124.60)
L-NTRDE ($c = 3$) vs	9/15/6	12/15/3	8/17/5	8/9/13
L-NTADE	(22.08)	(43.24)	(19.67)	(-6.39)
L-NTRDE ($c = 4$) vs	11/14/5	14/15/1	15/11/4	15/11/4
L-NTADE	(37.30)	(75.39)	(69.26)	(70.63)
L-NTRDE ($c = 5$) vs	10/16/4	14/15/1	17/10/3	15/11/4
L-NTADE	(44.53)	(73.95)	(67.77)	(65.51)
L-NTRDE ($c = 6$) vs	11/17/2	9/19/2	13/11/6	11/12/7
L-NTADE	(36.40)	(48.88)	(25.88)	(24.52)

As can be seen from Table 1, small c values make L-NTRDE algorithm much worse than L-NTADE with success history adaptation, however, setting c to 4 or 5 results in significant performance gain, and L-NTRDE becomes better than the baseline algorithm. In particular, with $c = 5$ L-NTRDE is capable of outperforming L-NTADE on up to 17 functions, i.e. more than half of the test functions. Table 2 contains the comparison of L-NTRDE with L-NTADE on the CEC 2022 benchmark set. Note that the ranking during Mann-Whitney test here takes into consideration not only the final achieved result, but also convergence speed - if the problem is solved successfully (with 10^{-8} accuracy), then the number of function evaluations required to achieve this is recorded. This comparison mechanism was proposed in the CEC 2022 benchmark.

The results in Table 2 show that L-NTRDE is better than L-NTADE on $10D$ functions in most cases, however, the performance on $20D$ functions is worse. However, similar dependence on c can be observed, i.e. $c = 4$ appears to be the best setting for this benchmark.

In Tables 3 and 4 the L-NTRDE is compared to alternative approaches on the CEC 2017 and CEC 2022 benchmark sets, the c parameter is set to 4 in both cases

The results in Table 3 show that L-NTRDE is capable of outperforming all methods in the comparison, and only LSHADE-SPACMA has similar performance in $100D$ case. Compared to other approaches, L-NTRDE is better on most of the test functions.

As for the CEC 2022 results, here L-NTRDE is better than most of the methods in $10D$ case, except EA4eigN100 (ranked first), NL-SHADE-LBC (ranked second) and MLS-LSHADE (proposed

Table 2. Comparison of L-NTRDE to L-NTADE with different c parameter values, CEC 2022, Mann–Whitney tests and total standard score.

Algorithms	**10*D***	**20*D***
L-NTRDE ($c = 1$) vs	5/3/4	3/2/7
L-NTADE	(11.08)	(-22.99)
L-NTRDE ($c = 2$) vs	6/4/2	3/3/6
L-NTADE	(25.53)	(-6.99)
L-NTRDE ($c = 3$) vs	6/4/2	3/5/4
L-NTADE	(25.45)	(1.07)
L-NTRDE ($c = 4$) vs	5/5/2	2/6/4
L-NTADE	(26.17)	(-1.82)
L-NTRDE ($c = 5$) vs	3/8/1	2/6/4
L-NTADE	(13.29)	(-10.64)
L-NTRDE ($c = 6$) vs	2/6/4	2/7/3
L-NTADE	(-10.49)	(-12.62)

Table 3. Comparison of L-NTRDE to different approaches, CEC 2017, Mann–Whitney tests and total standard score.

Algorithms	**10*D***	**30*D***	**50*D***	**100*D***
L-NTRDE vs	9/18/3	17/6/7	14/3/13	14/1/15
LSHADE-SPACMA [15]	(25.76)	(75.97)	(17.98)	(1.50)
L-NTRDE vs	6/21/3	19/10/1	22/6/2	24/1/5
jSO [12]	(7.27)	(139.25)	(153.40)	(147.83)
L-NTRDE vs	3/19/8	16/9/5	18/7/5	22/2/6
EBOwithCMAR [17]	(-37.99)	(79.00)	(110.30)	(131.61)
L-NTRDE vs	6/22/2	19/10/1	18/10/2	20/5/5
LSHADE-RSP [21]	(18.25)	(133.79)	(119.75)	(108.01)
L-NTRDE vs	13/8/9	24/3/3	29/1/0	29/0/1
NL-SHADE-RSP [14]	(20.26)	(175.19)	(248.78)	(236.40)
L-NTRDE vs	6/20/4	24/5/1	28/2/0	27/2/1
NL-SHADE-LBC [22]	(2.61)	(180.57)	(230.02)	(220.26)
L-NTRDE vs	11/14/5	14/15/1	15/11/4	15/11/4
L-NTADE [3]	(37.30)	(75.39)	(69.26)	(70.63)

Table 4. Comparison of L-NTRDE to different approaches, CEC 2022, Mann–Whitney tests and total standard score.

Algorithms	**10*D***	**20*D***
L-NTRDE vs	5/3/4	4/5/3
LSHADE-RSP [21]	(10.84)	(5.59)
L-NTRDE vs	5/4/3	7/4/1
NL-SHADE-RSP [14]	(18.28)	(42.71)
L-NTRDE vs	0/5/7	6/2/4
NL-SHADE-LBC [22]	(-40.59)	(2.34)
L-NTRDE vs	2/6/4	5/3/4
EA4eigN100 [18]	(-15.49)	(5.51)
L-NTRDE vs	3/4/5	6/2/4
NL-SHADE-RSP-MID [19]	(-5.98)	(17.59)
L-NTRDE vs	8/2/2	10/0/2
APGSK-IMODE [20]	(36.86)	(52.60)
L-NTRDE vs	4/2/6	4/1/7
MLS-LSHADE [23]	(-9.67)	(-19.78)
L-NTRDE vs	8/2/2	8/2/2
MadDE [24]	(39.05)	(43.60)
L-NTRDE vs	5/5/2	2/6/4
L-NTADE [3]	(26.17)	(-1.82)

for CEC 2021 benchmark). In $20D$ case it was outperformed by MLS-LSHADE. However, it should be noted that L-NTRDE was not tuned for this benchmark, and used the same parameters as for CEC 2017 benchmark.

For a better understanding of the process of parameter adaptation based on success rate, Figures 2 and 3 shows the graphs of SR and MF values, as well as the average of $M_{F,k}$ values for the L-NTADE algorithm with the same settings.

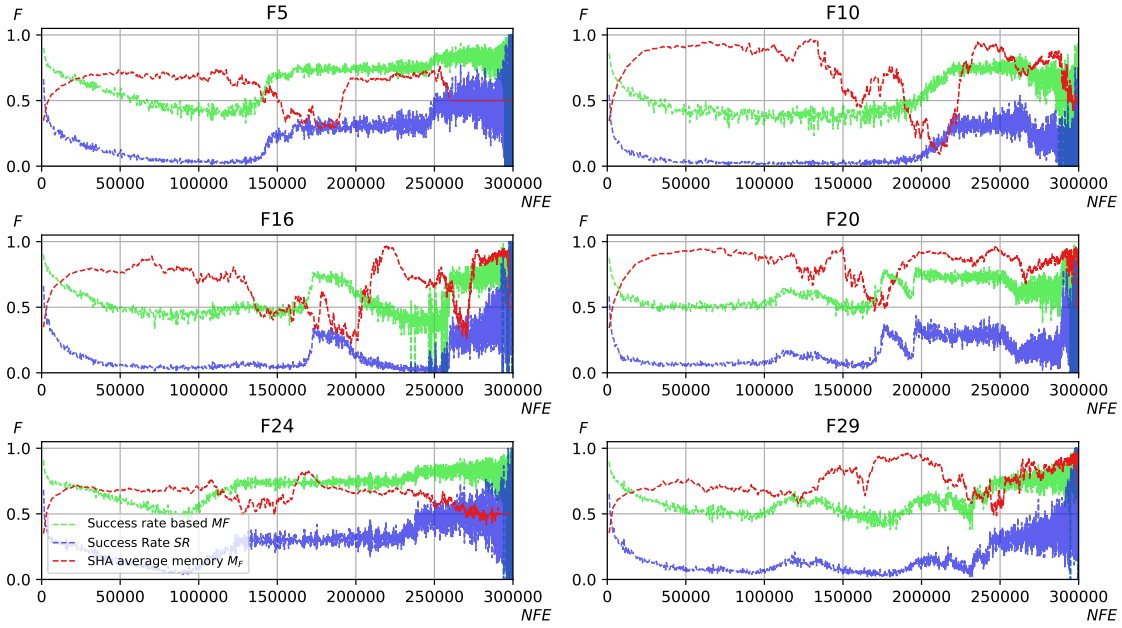


Fig. 2. Process of parameter adaptation in L-NTRDE and L-NTADE, CEC 2017, several selected functions, $30D$ case

Considering the graphs in Figure 2, it can be observed that the behaviour of success history adaptation and success rate adaptation is significantly different. In fact, sometimes there are opposite trends. For example, at the beginning usually the values in the memory cells M_F generated by SHA increase, while success rate adaptation decreases the MF value. Moreover, if at some point during the optimization process SHA values goes down, MF increases, for example at around 200000 function evaluations on F16, around 125000 evaluations on F29 and so on. If the success rate starts oscillating, this results in larger spread of generated MF values, leading to more diverse scaling factors F being sampled.

As for the $50D$ case shown in Figure 3, similar tendencies can be observed, however, there are cases when both methods have almost the same effect, for example like on F30.

Table 5 contains the mean values achieved by L-NTADE and L-NTRDE on every function of the CEC 2017 benchmark set, as well as Mann-Whitney tests, "+" means that L-NTRDE was better, "-" means that it was worse, and "=" denotes a tie.

As can be seen from Table 5, the difference between L-NTADE and L-NTRDE is sometime quite large, for example on functions 5, 10, and 20.

According to the presented experimental results, the proposed success rate based adaptation works very different from the baseline approach, used in many algorithms nowadays, i.e. success history adaptation. The reasons of such efficiency of the proposed algorithm could be the following. If the success rate is low, this means that the algorithm is probably stuck in a local optimum (or different parts of the population in different optima). In such situation the scaling factor values should be sampled with a mean close to 0.5. With the mutation strategy used this means that trial vectors will be generated in the direction of better *pbest* solutions, but not very close to them, i.e. with moderate variation due to the second part of the mutation equation. This means that the algorithm will focus more on exploration rather than exploitation. Such sampling of F is possible

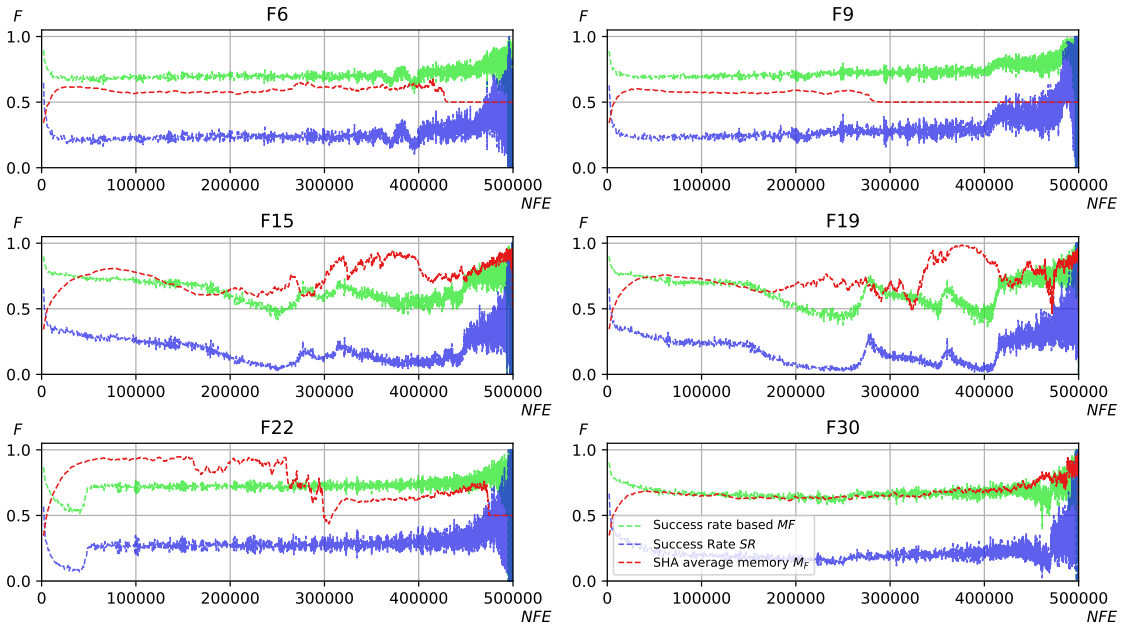


Fig. 3. Process of parameter adaptation in L-NTRDE and L-NTADE, CEC 2017, several selected functions, $50D$ case

Table 5. Comparison of L-NTRDE and L-NTADE on 30 functions, CEC 2017, Mann–Whitney tests

F	L-NTADE	L-NTRDE	MW	L-NTADE	L-NTRDE	MW	L-NTADE	L-NTRDE	MW	L-NTADE	L-NTRDE	MW
1	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=	0.0e+0	1.360e-6	-	4.075e-2	6.494e+1	-
2	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=	2.024e+4	8.241e+1	=	4.064e+41	9.496e+41	=
3	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=	1.735e+1	4.145e-1	+
4	0.0e+0	0.0e+0	=	4.07e+1	5.774e+1	=	3.839e+1	5.897e+1	-	1.034e+2	2.020e+2	-
5	4.955e+0	1.814e+0	+	9.984e+0	4.897e+0	+	1.450e+1	9.735e+0	+	2.739e+1	2.286e+1	+
6	0.0e+0	0.0e+0	=	0.0e+0	6.709e-10	=	1.175e-9	2.713e-8	-	9.619e-4	1.521e-6	+
7	1.573e+1	1.262e+1	+	4.275e+1	3.547e+1	+	6.591e+1	5.877e+1	+	1.275e+2	1.137e+2	+
8	4.936e+0	1.580e+0	+	9.479e+0	5.384e+0	+	1.570e+1	1.05e+1	+	2.622e+1	2.283e+1	+
9	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=	0.0e+0	0.0e+0	=
10	2.778e+2	1.289e+1	+	2.810e+3	7.775e+2	+	4.771e+3	1.296e+3	+	1.131e+4	4.197e+3	+
11	1.561e-1	1.951e-2	=	2.421e+0	2.051e+0	=	2.105e+1	2.101e+1	=	6.609e+1	5.959e+1	=
12	3.020e-1	2.979e-1	=	5.928e+0	1.084e+1	=	6.236e+2	5.103e+2	=	2.422e+4	3.497e+4	-
13	7.297e-1	1.668e+0	=	1.172e+1	9.213e+0	+	2.203e+1	1.120e+1	+	8.636e+1	9.057e+1	=
14	4.292e-1	3.902e-2	+	6.184e+0	4.429e+0	+	2.554e+1	2.374e+1	+	3.181e+1	3.144e+1	=
15	1.381e-1	2.126e-1	=	1.992e+0	8.072e-1	+	2.072e+1	1.996e+1	=	6.178e+1	6.768e+1	=
16	3.164e-1	6.097e-1	-	2.513e+1	8.012e+0	+	1.412e+2	1.355e+2	+	2.293e+2	2.061e+2	=
17	1.586e+0	6.926e-1	+	1.952e+1	1.909e+1	+	2.214e+2	1.255e+2	+	2.794e+2	1.198e+2	+
18	1.368e-1	2.828e-1	-	1.419e+1	1.942e+1	=	2.188e+1	2.093e+1	+	6.698e+1	4.698e+1	+
19	2.021e-2	1.757e-2	+	3.336e+0	2.037e+0	+	6.533e+0	4.998e+0	+	3.289e+1	2.838e+1	+
20	6.121e-3	2.081e-1	-	1.949e+1	1.227e+0	+	1.620e+2	2.529e+1	+	1.110e+3	1.921e+2	+
21	1.363e+2	1.521e+2	=	2.095e+2	2.040e+2	+	2.151e+2	2.104e+2	+	2.483e+2	2.452e+2	=
22	1.02e+2	1.0e+2	+	1.0e+2	1.0e+2	=	1.0e+2	1.0e+2	=	1.235e+4	4.213e+3	+
23	3.035e+2	3.04e+2	+	3.437e+2	3.420e+2	=	4.162e+2	4.135e+2	=	5.60e+2	5.570e+2	=
24	2.857e+2	3.057e+2	+	4.212e+2	4.192e+2	+	4.899e+2	4.889e+2	=	8.260e+2	8.364e+2	-
25	4.085e+2	4.077e+2	-	3.867e+2	3.867e+2	=	5.209e+2	4.853e+2	+	7.583e+2	6.981e+2	+
26	3.0e+2	3.0e+2	=	3.384e+2	7.396e+2	-	5.580e+2	7.898e+2	-	2.168e+3	2.598e+3	=
27	3.924e+2	3.932e+2	-	4.730e+2	4.746e+2	=	4.858e+2	4.824e+2	=	5.676e+2	5.545e+2	+
28	3.056e+2	3.0e+2	=	3.065e+2	3.043e+2	=	4.684e+2	4.588e+2	+	5.506e+2	5.349e+2	+
29	2.312e+2	2.288e+2	+	4.143e+2	4.041e+2	+	3.223e+2	3.142e+2	+	7.873e+2	7.460e+2	+
30	3.945e+2	3.948e+2	=	1.984e+3	1.984e+3	=	5.878e+5	5.898e+5	=	2.252e+3	2.239e+3	=

due to the fact that $c = 4$ results in a 4-th order root curve, where even small success rates SR result in MF close to 0.5. This means that the algorithm will rarely use MF smaller than 0.5, which was observed during experiments in Figures 2 and 3. However, if the success rate is high, that means that the algorithm is probably moving solutions in the right direction, so they should be sampled closer to some of the *pbest* solutions in the population. And this is exactly what success rate based adaptation does: the more successful points are generated, the higher are sampled F values, i.e. more solutions are generated next to better ones. The presented explanation is only our vision of the reasons of high efficiency of L-NTRDE, and there could be some other underlying processes.

5 Conclusion

In this study the success rate based scaling factor adaptation method was proposed for the differential evolution algorithm. The modification is much simpler to implement than the widely used success history based adaptation, and at the same time it allows achieving significantly better results compared to the state-of-the art algorithms. Here this method was used only for modification of L-NTADE algorithm, however, it can be applied to any other differential evolution, as it relies only on the improvement rate, which can be calculated for any DE-based approach. Further studies in this direction may include evaluating the performance of success rate based adaptation in other modern algorithms, as well as proposing novel crossover rate adaptation techniques, which are based on success rate value.

6 Acknowledgments

This research was funded by the Ministry of Science and Higher Education of the Russian Federation, Grant No. 075-15-2022-1121.

References

1. Piotrowski, A.P., and Napiorkowski, J.J. (2018). Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure? Swarm Evol. Comput., 43, 88-108. https://doi.org/10.1016/j.swevo.2018.03.007
2. Stanovov, V., Akhmedova, S., and Semenkin, E. (2022). The automatic design of parameter adaptation techniques for differential evolution with genetic programming. Knowl. Based Syst., 239, 108070. https://doi.org/10.1016/j.knosys.2021.108070
3. Stanovov, V., Akhmedova, S., Semenkin, E. (2022). Dual-Population Adaptive Differential Evolution Algorithm L-NTADE. Mathematics. 2022; 10(24):4666. https://doi.org/10.3390/math10244666
4. Awad, N.; Ali, M.; Liang, J.; Qu, B.; Suganthan, P. Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization; Technical Report; Nanyang Technological University: Singapoure, 2016.
5. Kumar, A.; Price, K.V.; Mohamed, A.W.; Hadi, A.A.; Suganthan, P.N. Problem Definitions and Evaluation Criteria for the CEC 2022 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization; Technical Report, Nanyang Technological University: Singapoure, 2021.
6. R. Tanabe and A.S. Fukunaga. 2013. Success-history based parameter adaptation for differential evolution. In Proceedings of the IEEE Congress on Evolutionary Computation. IEEE Press, 71–78. https://doi.org/10.1109/CEC.2013.6557555
7. R. Tanabe and A.S. Fukunaga. 2014. Improving the search performance of SHADE using linear population size reduction. In Proceedings of the IEEE Congress on Evolutionary Computation, CEC. Beijing, China, 1658–1665. https://doi.org/10.1109/CEC.2014.6900380
8. R. Storn and K. Price. 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 4 (1997), 341–359. https://doi.org/10.1023/A:1008202821328
9. J. Zhang and A. C. Sanderson. 2007. JADE: Self-adaptive differential evolution with fast and reliable convergence performance. In 2007 IEEE Congress on Evolutionary Computation. 2251–2258. https://doi.org/10.1109/CEC. 2007.4424751
10. K. Price, R.M. Storn, and J.A. Lampinen. 2005. Differential evolution: a practical approach to global optimization (1st. ed.). Springer.

11. S. Das, S.S. Mullick, and P.N. Suganthan. 2016. Recent advances in differential evolution – an updated survey. Swarm and Evolutionary Computation 27 (2016), 1–30. https://doi.org/10.1016/j.swevo.2016.01.004
12. J. Brest, M.S. Maučec, and B. Boškovic. 2017. Single objective real-parameter optimization algorithm jSO. In Proceedings of the IEEE Congress on Evolutionary Computation. IEEE Press, 1311–1318. https://doi.org/10.1109/CEC.2017.7969456
13. Stanovov, V.; Akhmedova, S.; Semenkin, E. Selective Pressure Strategy in differential evolution: Exploitation improvement in solving global optimization problems. Swarm Evol. Comput. 2019, 50, 100463. https://doi.org/10.1016/J.SWEVO.2018.10.014
14. Stanovov, V.; Akhmedova, S.; Semenkin, E. NL-SHADE-RSP Algorithm with Adaptive Archive and Selective Pressure for CEC 2021 Numerical Optimization. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Krakow, Poland, 28 June–1 July 2021; pp. 809–816. https://doi.org/10.1109/CEC45853.2021.9504959.
15. Mohamed, A.; Hadi, A.A.; Fattouh, A.; Jambi, K. LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia-San Sebastián, Spain, 5–8 June 2017; pp. 145–152. https://doi.org/10.1109/CEC.2017.7969307
16. Kitamura, T.; Fukunaga, A. Differential Evolution with an Unbounded Population. In Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, 18–23 July 2022. https://doi.org/10.1109/CEC55065.2022.9870363
17. Kumar, A.; Misra, R.K.; Singh, D. Improving the local search capability of Effective Butterfly Optimizer using Covariance Matrix Adapted Retreat Phase. In Proceedings of the IEEE Congress on Evolutionary Computation, Donostia, Spain, 5–8 June 2017; pp. 1835–1842. https://doi.org/10.1109/CEC.2017.7969524.
18. Bujok, P.; Kolenovsky, P. Eigen Crossover in Cooperative Model of Evolutionary Algorithms Applied to CEC 2022 Single Objective Numerical Optimisation. In Proceedings of the IEEE Congress on Evolutionary Computation, Padua, Italy, 18–23 July 2022; pp. 1–8. https://doi.org/10.1109/CEC55065.2022.9870433.
19. Biedrzycki, R.; Arabas, J.; Warchulski, E. A Version of NL-SHADE-RSP Algorithm with Midpoint for CEC 2022 Single Objective Bound Constrained Problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Padua, Italy, 18–23 July 2022; pp. 1–8. https://doi.org/10.1109/CEC55065.2022.9870220.
20. Mohamed, A.W.; Hadi, A.A.; Agrawal, P.; Sallam, K.M.; Mohamed, A.K. Gaining-Sharing Knowledge Based Algorithm with Adaptive Parameters Hybrid with IMODE Algorithm for Solving CEC 2021 Benchmark Problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Kraków, Poland, 28 June–1 July 2021; pp. 841–848. https://doi.org/10.1109/CEC45853.2021.9504814.
21. Stanovov, V.; Akhmedova, S.; Semenkin, E. LSHADE Algorithm with Rank-Based Selective Pressure Strategy for Solving CEC 2017 Benchmark Problems. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8. https://doi.org/10.1109/CEC.2018.8477977.
22. Stanovov, V.; Akhmedova, S.; Semenkin, E. NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 Numerical Optimization. In Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, 18–23 July 2022. https://doi.org/10.1109/CEC55065.2022.9870295.
23. Cuong, L.V.; Bao, N.N.; Binh, H.T.T. Technical Report: A Multi-Start Local Search Algorithm with L-SHADE for Single Objective Bound Constrained Optimization; Technical Report; SoICT, Hanoi University of Science and Technology: Hanoi, Vietnam, 2021.
24. Biswas, S.; Saha, D.; De, S.; Cobb, A.D.; Das, S.; Jalaian, B. Improving Differential Evolution through Bayesian Hyperparameter Optimization. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Krakow, Poland, 28 June–1 July 2021; pp. 832–840. https://doi.org/10.1109/CEC45853.2021.9504792

Metaheuristic Algorithms for Circle Packing Problem: A Comprehensive Review

Yogesh Kumar [1] and Kusum Deep [2]

[1] Department of Mathematics, Indian Institute of Technology Roorkee, Roorkee, India
yogesh_k@ma.iitr.ac.in

[2] Department of Mathematics, Indian Institute of Technology Roorkee, Roorkee, India
kusum.deep@ma.iitr.ac.in

Abstract

The Circle Packing Problem (CPP) is a well-known optimization problem with a wide range of applications. It is difficult to identify exact solutions for the CPP as it is a NP-hard problem. As a result, metaheuristic algorithms are efficient approach to address this challenging problem. This paper presents an extensive literature review of the role and effectiveness of metaheuristic algorithms in solving the CPP. The objective of this paper is to examine the applications, advancements, and potential of metaheuristic algorithms in solving the CPP. Moreover, case studies and real-world examples of how metaheuristics have been used to address the CPP. Finally, some suggested possible future areas are also highlighted for the researchers who want to utilize metaheuristic algorithms for solving the CPP and related optimization problems.

Keywords: Metaheuristic Algorithms, Circle Packing Problem

1. Introduction

Placing a specified collection of circles into a 2D container with the shape of a circle, triangle, rectangle, polygon, etc., is known as the Circle Packing Problem (CPP). Usually, the objective is:

(a) To reduce the size of the container and the waste area as a result.

(b) To get the highest density, pack the container with the most circles possible.

Two different categories of constraints apply to this arrangement:

(i) Boundary Constraints: It guarantees that every circle is contained inside the container.

(ii) Non-overlapping Constraints: All the circles in the container must be not overlapped.

2. Literature Review

Metaheuristics [1] are a collection of clever tactics to increase the effectiveness of heuristic techniques. Nowadays, there are many metaheuristic algorithms have been developed, such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE), Grey Wolf Optimization (GWO), Sine Cosine Optimization (SCO), and many more. In recent years, lots of review paper has been written on metaheuristic algorithms. Kanchan et al. [2] show that more than 500 algorithms have been developed. Moreover, [3] intended to provide a quick review of evolutionary algorithms, their benchmarks, and their most recent, effective applications. [4] gives a thorough literature overview on the use of recent (2009–2019) metaheuristic algorithm development to solve the feature selection challenge. [5] includes some of

the metaheuristic algorithms that are developed between 2014 and 2020 and summarizes the algorithms and changes made thus far.

3. Circle Packing Problem are solved with the help of Metaheuristic Algorithms

An efficient iterated dynamic neighborhood search (IDNS) approach for the equal circle packing problem on a sphere (ECPOS) problem [6]. The technique includes an adjustment method for the lowest distance between points on the unit sphere, a broad dynamic neighborhood search approach, a multi-stage local optimization method, and more.

With the help of straightforward combinatorial developments and robust geometrical analytical forms, this polynomial decoding approach enables the transition from the permutations search space to the packings search space. The packing approach is incorporated into the simulated annealing (SA) and variable neighborhood search (VNS), two well-known metaheuristics [7].

A particular instance of the two-dimensional spatial organization problem known as irregular bin packing is discussed. HPA (Heuristic placement algorithm), based on GA and GWO, after thoroughly examining the features and difficulty of the problem [8].

The two-dimensional circle bin packing problem (2D-CBPP) provides an adaptive local search strategy to resolve it [9]. The algorithm applies our greedy constructive algorithm's simulated annealing search. The greedy algorithm constructs the initial solution. Then a partial solution by randomly picking two bins of rectangular regions, eliminating the circles that cross those areas, and utilizing the greedy method to finish incomplete solutions.

A hybrid metaheuristic Adaptive Tabu search and Variable Neighbourhood Descent (ATS-VND) for Packing Unequal Circles into a Square (PUCS) is developed [10]. Variable Neighbourhood Descent (VND) and Tabu search (TS) are adaptively combined to form the metaheuristic. The Packing Unequal Circles into a Square (PUCS) problem is used to apply the metaheuristic, which is then improved using an Iterated Local Search (ILS) framework to produce the final IATS-VND method.

Effectively integrated tabu search with simulated annealing to create a hybrid solution for the packing circle problem [11]. This algorithm's major feature is a potent way to escape local minima.

For the layout optimization challenge [12], provide a unique order-based placement method. A layout may be created by describing the placement of the circles in a permutation. The GA is an ideal approach to utilize to explore such a vast space because there are potential permutations for circles. The GA is utilized to develop the order in which the circles are placed.

An evolution approach with a crossover operator (ESCO) to address the restricted circle packing problem [13]. By using the crossover operator from genetic algorithms, the suggested ESCO extends a classical ES to handle combinatorial optimization. It aims to swap out the locations of circles to get a better packing scheme. Its objective is to solve the general CPP, and they gauge the packing's effectiveness by looking at factors like container size and the weighted average pair-wise distance between circles.

To solve the two-dimensional packing problem with restrictions, introduce the layout pattern-based particle swarm optimization technique (LPPSO) [14]. In the LPPSO optimization procedure, certain individuals are constructed using non-isomorphic layout patterns and then added to the current population of the PSO algorithm to replace the worst individuals, resulting in the development of a new population.

4. Conclusion

It is essential to remember that several variables, including parameter choices, problem size, and problem complexity, can have an impact on how well metaheuristic algorithms work. To increase the quality of the solutions and computing effectiveness for the circle packing problem, further study might concentrate on optimizing the algorithms, examining hybrid strategies, or looking at different metaheuristic methods. Overall, the article provides the groundwork for future research in this field and shows how metaheuristic algorithms may solve challenging geometric optimization problems. Using metaheuristic algorithms creates new opportunities for tackling real-world problems in circle packing problems and related fields, advancing optimization approaches and their use in real situations.

References

1. Beheshti, Z., & Shamsuddin, S. M. H. (2013). A review of population-based meta-heuristic algorithms. *International Journal of Adv. Soft Comput. Appl, 5(1), 1-35*.
2. Rajwar, K., Deep, K., & Das, S. (2023). An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artificial Intelligence Review, 1-71*.
3. Wong, W. K., & Ming, C. I. (2019, June). A review on metaheuristic algorithms: recent trends, benchmarking and applications. *In 2019 7th International Conference on Smart Computing & Communications (ICSCC) (pp. 1-5). IEEE.*
4. Agrawal, P., Abutarboush, H. F., Ganesh, T., & Mohamed, A. W. (2021). Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019). *IEEE Access, 9, 26766-26791*.
5. Khanduja, N., & Bhushan, B. (2021). Recent advances and application of metaheuristic algorithms: A survey (2014–2020). *Metaheuristic and evolutionary computation: algorithms and applications, 207-228*.
6. Lai, X., Yue, D., Hao, J. K., Glover, F., & Lü, Z. (2023). Iterated dynamic neighborhood search for packing equal circles on a sphere. *Computers & Operations Research*, *151*, 106121.
7. Bouzid, M. C., & Salhi, S. (2020). Packing rectangles into a fixed size circular container: Constructive and metaheuristic search approaches. *European Journal of Operational Research*, *285*(3), 865-883.
8. Luo, Q., Rao, Y., & Peng, D. (2022). GA and GWO algorithm for the special bin packing problem encountered in field of aircraft arrangement. Applied Soft Computing, 114, 108060.
9. He, K., Tole, K., Ni, F., Yuan, Y., & Liao, L. (2021). Adaptive large neighborhood search for solving the circle bin packing problem. *Computers & Operations Research*, *127*, 105140.
10. Zeng, Z., Yu, X., He, K., Huang, W., & Fu, Z. (2016). Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European Journal of Operational Research*, *250*(2), 615-627.
11. Zhang, D. F., & Deng, A. S. (2005). An effective hybrid algorithm for the problem of packing circles into a larger containing circle. *Computers & Operations Research*, *32*(8), 1941-1951.
12. Xu, Y. C., Xiao, R. B., & Amos, M. (2007, September). A novel genetic algorithm for the layout optimization problem. In *2007 IEEE Congress on Evolutionary Computation* (pp. 3938-3943). IEEE.
13. Shi, Y. J., Liu, Z. C., & Ma, S. (2010). An improved evolution strategy for constrained circle packing problem. In *Advanced Intelligent Computing Theories and Applications: 6th International Conference on Intelligent Computing, ICIC 2010, Changsha, China, August 18-21, 2010. Proceedings 6* (pp. 86-93). Springer Berlin Heidelberg.
14. Yan Jun, S., Yi Shou, W., Long, W., & Hong Fei, T. (2012). A layout pattern-based particle swarm optimization for constrained packing problems. *Information Technology Journal*, *11*(12), 1722.

An optimised version of differential evolution heuristic for feature selection

Thibault Anani[2], François Delbot[1.2] and Jean-François Pradat-Peyre[1.2]

[1] University Paris Nanterre, Nanterre, France
[2] LIP6, Sorbonne University, Paris, France
{thibault.anani-agondja,francois.delbot,jean-francois.pradat-peyre}@lip6.fr

1 Introduction

It is common practice to apply machine learning to real-life data, for example from clinical trials. These datasets have a wide disparity in the number of features, not all of which are necessarily relevant to the learning task in hand. The very nature of these data implies redundancy, omissions, errors and a degree of subjectivity, all of which will have a negative impact on the quality of the models trained.

Feature selection aims to optimise the quality of machine learning models by identifying a feature subset that are most suitable in predicting the target feature, while discarding redundant, noisy or harmful features. This is a crucial step in the construction of statistical and machine learning models, because by removing unnecessary features it is possible to simplify the model considerably. This improves the interpretability and robustness of the proposed solutions, avoiding problems such as overfitting. To carry out this feature selection, the ideal approach would be to evaluate all possible combinations of subsets of features to find the best possible one. However, this approach is impractical because of the combinatorial explosion that results when the number of features is large[3]. The feature selection problem is an NP-hard problem [1] which has been the subject of a great deal of research in recent decades [2, 3, 4].

Various methods have been proposed, ranging from filter methods to strategies based on heuristics. However, despite the progress made, feature selection remains an open problem. The fundamental reason for this complexity can be attributed to the "No Free Lunch" theorem [5], which states that in optimisation, no universal method offers superior performance for all problems. This means that an optimisation method that performs very well on one specific set of problems (or data) is likely to underperform on another. Consequently, the idea that a feature selection method running in reasonable time can be better than all other methods whatever the data provided seems unlikely. Instead, the emphasis should be on selecting an appropriate method based on the particularities and requirements of the problem to be solved and the quality of the data available.

For the purposes of this study, we are focusing specifically on data from the medical field. These data come from various studies and clinical trials. These studies involve numerous institutions, medical teams and patients. Each clinical trial collects data relating to the subject of the study. These studies often have different focuses, which may explain the presence or absence of certain data for different patients. Collecting this data is a long and tedious process that requires direct contact with patients before reporting, which can lead to omissions or data entry errors. Doctors sometimes use metrics that are not always objective, depending on their own perception or that of the patients.

Due to the small amount of data, and in order to increase the quality of scientific research, databases have been set up by pooling data from different studies. As these studies come from different geographical areas, biases in the data and errors when integrating the data can occur. For example, the units of measurement used to calculate a patient's height or weight may differ (metric or imperial). Finally, with regard to the temporality of data, during clinical trials, patients

[3] Indeed, with n explanatory features, there would be 2^n different subsets to explore, which in the current state of knowledge cannot be achieved in a reasonable time.

are monitored for varying time periods, ranging from several days to several years, and information about them is collected periodically. However, the gaps between these periods are not always constant for each patient, which can also introduce negative biases.

In this context, we are interested in Amyotrophic Lateral Sclerosis (ALS), also known as Charcot's disease. We are attempting to predict patients' one-year survival based on data available after the first 3 months of the disease [6]. The data available to us are unusual in that they come from clinical trials for a rare disease. Thus, we have a low volume of data, missing data, noisy and/or useless data. We present an improvement of a heuristic (differential evolution) that we use for feature selection. Our heuristic is based on an ad-hoc mutation strategy, an initialisation based on a filter method and an optimisation of the different parameters of the heuristic. We show experimentally that our heuristic outperforms (in many cases) the most common feature selection techniques [1, 7] and outperforms them on ALS data in particular.

2 Standard feature selection methods

2.1 Embedded methods

Embedded feature selection methods are approaches that integrate feature selection directly into the model learning process. Instead of treating feature selection as a separate step, these methods seek to automatically identify and use the most informative features during the learning of the model itself. The most commonly used methods rely on regularisation or penalty mechanisms built into the objective function of the learning algorithm. This encourages the model to assign lower weights to certain features, or even to remove them altogether. Techniques such as Lasso [8] and Elastic net [9] are often used in linear classification methods such as logistic regression, ridge regression and support vector machines. Embedded methods also include approaches based on decision trees, such as the random forest [10]. In these methods, the importance of each feature is assessed by measuring its impact on the accuracy of the algorithm's predictions, by calculating the average information gain obtained by using this feature in the construction of the trees.

2.2 Filter methods

Filter methods use statistical techniques or predefined criteria to evaluate each explanatory feature. By using these statistical criteria, they are able to identify and filter out the least relevant features independently of the learning method used and prior to the construction of a model.
Spearman Correlation Coefficient (SCC) evaluates a monotonic relationship between an explanatory feature and the target feature, whether linear or not, unlike the Pearson Correlation Coefficient (PCC) [3]. It is represented by a correlation coefficient between -1 and 1. A correlation of 1 indicates a perfect positive monotonic relationship, a correlation of -1 indicates a perfect negative monotonic relationship, and a correlation of 0 indicates the absence of a monotonic relationship between the two features. In a complex dataset where the relationship between features may not be strictly linear, the SCC can provide more robust indications of the associations between features.
Analysis of variance (ANOVA) compares the means of several groups to determine whether they are significantly different from each other. It is often used to determine whether an explanatory feature has a significant effect on a target feature. It works by dividing the total variation in the data into two components: variation between features and variation within features. If the between-feature variation is significantly greater than the within-feature variation, this indicates that the features differ significantly from one another [3].
Mutation Information (MI) quantifies the amount of information shared between two features. It measures the reduction in uncertainty of one of the features thanks to the knowledge of the other feature. It is the difference between the entropy of the combined features and the entropy of the individual features. Entropy measures the amount of uncertainty or disorder in a feature, while mutual information measures the amount of information shared by two features. [3].
Maximum Relevance Minimum Redundancy (MRMR) evaluates the relevance of features using the mutual information between the explanatory features and the target feature. There are two steps. First, it calculates the relevance of each feature to the target feature using the mutual

information. Second, it uses a search heuristic to select a subset of features that maximises relevance and minimises redundancy [11, 3].
ReliefF works by calculating weight scores for each feature based on the difference between the values of that feature for neighbouring samples belonging to the same class and those belonging to different classes. Features that contribute most to class distinction are assigned higher weights [12, 3].

2.3 Wrapper methods

Wrapper methods involve creating a series of models with different subsets of explanatory features. These models are then evaluated according to different selection criteria appropriate for evaluating each combination. Unlike embedded and filter methods, the purpose of these methods is to find the feature subset that optimises the performance of a specific predictive model.

2.3.1 Sequential feature selection

Sequential feature selection methods work by following an iterative scheme, where features are added or removed one by one, depending on their contribution to the accuracy or overall performance of the model.
Forward Feature Selection (FFS) starts with an empty model and iterates, sequentially adding the features that significantly improve the model. The features are evaluated individually, and the one that brings the greatest improvement at each iteration is added. [13].
Backward Feature Selection (BFS) starts with a model that includes all the features and sequentially removes those that are least relevant to the prediction [13].

2.3.2 Heuristics

Heuristics are optimisation techniques used to find an approximate solution to a difficult optimisation problem. Since feature selection is an NP-hard problem, heuristics are often used to explore the space of possible feature combinations when the number of features is large, more efficiently in terms of the number of calculations than sequential wrapper methods. Each potential feature subset represents a solution. A subset of features can be represented as a vector of booleans. In effect, each boolean represents whether or not a feature will be included in the composition of a subset in order to carry out learning and design a machine learning model that can then be tested and evaluated.
Random Search (RS) iteratively generates solutions with a random number of features and evaluates them. The basic idea behind random search is to cover a wide spectrum of potential solutions without following any specific scheme or logic. It allows a search space to be explored quickly, but without any guarantee of finding an optimal solution, and can be useful when the objective function is complex, non-linear or when the search space is high-dimensional. It serves as a reference approach for comparison with the following more sophisticated methods.
Taboo Search (TS) progressively improves an initial solution by iteratively exploring neighbouring solutions. Improving solutions are accepted, while solutions that do not lead to an improvement are generally rejected. Taboo search also uses a list of forbidden solutions, called the 'taboo list', to keep track of solutions that have already been explored and avoid returning to the same configurations. This avoids cycles and encourages the exploration of a wider search space [14, 15].
Genetic Algorithm (GA) is a heuristic based on a population composed of different individuals (solutions) themselves composed of chromosomes (in this case booleans). It uses evolutionary operations to create new generations of individuals. These operations include the selection of the more adapted individuals, the reproduction of selected individuals by crossing (genetic recombination) and the random modification of certain genes (mutation). Individuals are generally selected on a probabilistic basis, favouring those most competent for reproduction. Across the generations, the most competent individuals tend to spread through the population, leading to a convergence towards individuals of better quality [16].

Population-based incremental learning (PBIL) combines elements of genetic algorithms and probabilistic model learning. PBIL begins by initialising a population of potential solutions represented as binary vectors. The algorithm then uses a probabilistic estimate that describes the probability of selecting a specific feature in the model. During each iteration, new solutions are generated using these estimates. These new solutions are evaluated and the probabilities of selecting each feature are updated using the best-fit individuals. [17].
Differential evolution (DE) is a heuristic inspired by the genetic algorithm. It is based on mechanisms found in nature (e.g. the evolution of a species) and defines a succession of population generations. In each generation, the population is made up of several individuals, themselves made up of chromosomes that will try to survive the next generation. In particular, differential evolution uses the diversity present between individuals in the population to try to find the best ones by using mutation operations [18].

The population is composed of N individuals denoted $P_G = \{X_1^G, X_2^G, ..., X_N^G\}$ at generation G (where $G \in [1, G_{max}]$ and $X_{i,j}^G$ is the j-th, $j \in [1, D]$, chromosome of the individual X_i^G). By carrying out mutation, crossover and selection operations, the population evolves through the different generations until the stopping criterion is reached. During the initial generation, these individuals are generated at random.

At each generation the algorithm first performs a mutation step using a mutation strategy which can be expressed as 'DE/x/y' where DE stands for differential evolution, x represents how an individual in the mutation operation is chosen and $y \in \mathbb{N}$ specifies the number of differential individuals in the mutation strategy. The best known and most widely used strategy for mutation is 'DE/rand/1' as shown below:

$$V_i^G = X_{r1}^G + F \times (X_{r2}^G - X_{r3}^G) \tag{1}$$

V_i^G is the mutant obtained by mutation, $i = \{1, 2, \ldots, N\}$, $r1, r2, r3$ are random numbers belonging to $\{1, 2, \ldots, N\}$ i.e. $r1 \neq r2 \neq r3 \neq i$ and where $F \in [0, 2]$ is a constant probability factor which controls the amplification of the differential variation. For the feature selection problem, the result of the equation 1 is rounded to the nearest value between 0 and 1. There are other mutation strategies such as 'DE/best/1' which use the best performing individual for the generation instead of one randomly chosen for X_{r1}^G. In this paper, the 'DE/best/1' strategy is used for better convergence in a limited number of generations.

The crossover step is performed to generate a new individual U_i^G which is the cross between the original individual X_i^G and the mutant V_i^G. The new individual U_i^G is generated as follows:

$$U_{i,j}^G = \begin{cases} V_{i,j}^G, & \text{if rand}(0,\ 1) \leq CR \text{ or } j = j_{rand} \\ X_{i,j}^G, & \text{otherwise} \end{cases} \tag{2}$$

$j_{rand} \in [1, D]$ is a random number to reduce the chances of the individual U_i^G being composed solely of elements of X_i^G and $CR \in [0, 1]$ is the probability of crossover. CR has a major influence on the diversity of the population created by the algorithm, since the number of elements that change will vary according to its value: the greater the value, the greater the variation.

The last step is to select the best performing individuals. To determine whether the individuals generated by the crossover stage will be kept, their score is compared with the score of the current individuals.

$$X_i^{G+1} = \begin{cases} U_i^G, & f(U_i^G) \geq f(X_i^G) \\ X_i^G, & \text{otherwise} \end{cases} \tag{3}$$

f represents the evaluation function of an individual. If an individual U_i^G has a better score than the individual X_i^G then this individual is kept for the next generation, otherwise it is rejected and the previous one is kept.

2.4 Limitations and challenges of these techniques

Embedded methods are entirely dependent on the learning method used, which distinguishes them from other feature selection approaches. Unlike those approaches, they do not allow the use of a

universal feature selection method that is independent of the learning method, because learning and feature selection are intertwined. In addition, non-embedded approaches offer the possibility of incorporating additional criteria to guide feature selection, such as expert considerations, specific domain knowledge, time or cost constraints. In this way, they offer greater flexibility in the selection process and allow essential factors other than the properties of the model alone to be taken into account. This is the reason why embedded feature selection methods is not taken into account when comparing methods.

Filter methods offer a quick and simple approach to feature selection compared to other methods. They are independent of the model itself and take into account only the data, which makes them easier to apply and interpret. However, these methods do not necessarily take into account the interactions between the explanatory features, nor the features that are not correlated with the target feature, nor the learning method used, although they could potentially have a significant influence on the quality of the model. In fact, the features considered relevant and their number for maximising predictive quality may differ depending on the learning method selected [6]. Furthermore, for these methods, it is necessary to specify in advance the number of features to be retained in order to obtain an accurate and reliable model. However, determining this number in advance can be complex. Setting an arbitrary number of features can lead to sub-optimal models if the number is not chosen wisely. It is therefore crucial to take these limitations into account when using filter methods for feature selection.

The main drawback of wrapper methods is that they are directly tied to model performance, which often leads to considerable computation times, especially when the number of features and instances is high. Indeed, these methods simultaneously evaluate several features subset, i.e. models, which can require significant resources. In addition, most of these methods have several parameters that need to be adjusted to optimise performance. This requires expertise and careful exploration of the parameter space to find the optimal values. It is therefore important to take these aspects into account when using sequential wrapper methods and heuristics to select features. However, despite these challenges, these methods generally offer a better ability to take into account interactions between features and to identify non-linear relationships, which can lead to better performing models [6].

3 A new variant of differential evolution

In this paper we propose a variant of differential evolution: **Tournament In Differential Evolution (TiDE)**. This version is based on an improved mutation strategy and an initialisation that is better adapted to the feature selection problem.

Generating the initial population is the first step in DE, as it is in most other population-based metaheuristics. Unlike the others, this stage is only carried out once, but it is nonetheless important for the progress of the algorithm. Indeed, it generates several useful individuals for the following steps. The classic DE method proposes to initialise the population by randomly generating individuals. The random generation technique is the most widely used and is often considered to be the most suitable for generating a population of individuals, as it allows the different zones of the search space to be explored uniformly. However, the method does not take into account the structure of this space or the knowledge about the optimal solution and in most cases generates poorly performing individuals, which slows down the speed of convergence of the algorithm as well as the chances of finding the optimal solution or at least an approximate solution. In the feature selection problem, it is possible to use filter methods, for example, to obtain more robust individuals as soon as the heuristic is initialised. The individual returned by the selected method is then integrated with the rest of the randomly generated initial population.

We propose to improve the mutation strategy by using the chromosomes of the individuals in the population. Each chromosome can take the values 1 or 0 to indicate whether a feature is selected for learning or not, respectively. A value of $F < 0.5$ means that the differences between the individuals are zero. In this case, the reference individual will always be selected, which will not introduce any differences in the population. this considerably limits the exploration of the search space, which can be pointless or even counter-productive, especially when the DE/best/1

strategy is used (equation 1). From this observation, it is possible to calculate the result of all the combinations of the initial mutation strategy DE/x/1 when $F \geq 0.5$ in this way:

$$V_{i,j}^{G} = \begin{cases} X_{r1,j}^{G}, & \text{if } X_{r2,j}^{G} = X_{r3,j}^{G} \\ X_{r2,j}^{G}, & \text{otherwise} \end{cases} \tag{4}$$

Carrying out this transformation removes the F factor from the equation. In addition, this approach makes it possible to use the chromosomes present in an individual directly without having to go through a conversion operation, which can be time-consuming when the size of a population is relatively large. The choice of mutation strategy is crucial to achieving good convergence. The 'DE/rand/1' strategy takes a single random individual in the population as a reference, which allows good exploration in the search space and maintains good diversity in the population. However, training can be time-consuming depending on the structure of the data and the training algorithm used, therefore the number of iterations of the algorithm is also limited. The 'DE/best/1' strategy takes the best individual as a reference, which favours exploitation and faster convergence at the risk of rapidly reducing the diversity between individuals and getting stuck in a local optimum. We can therefore conclude that determining the reference individual is important for the strategy to work. We propose a mutation strategy that offers a compromise between the two. At each generation a measure of diversity is calculated using the Shannon entropy [19]. The overall diversity present in the population at generation G is obtained by calculating the local diversity for each of the chromosomes $j \in [1, D]$ for all the individuals in the population as follows:

$$H_{j}^{G} = \begin{cases} -\sum_{x=1}^{n} \mathcal{P}_x \log_2 \mathcal{P}_x, & \mathcal{P}_x \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$\mathcal{P}_x$ is the probability that a specific j chromosome is present, i.e. the probability that $X_{i,j}^{G} = 1$, $x = \{0, 1\}$. The local entropies (H_j^G) are then averaged to obtain the global entropy (E^G). This entropy is used to automatically adapt the mutation strategy over the generations by introducing tournament selection to select the reference individual. Tournament selection is a strategy for selecting individuals from a population in which several individuals are chosen at random and their performances are compared to determine the winner of the tournament, the winner being the best-performing individual. In TiDE, the size of the tournament varies automatically as a function of entropy as follows:

$$p^{G} = (1 - E^{G}) \times (1 - \alpha) + \alpha \tag{6}$$

$\alpha \in [0, 1]$ is the minimum threshold for the percentage of individuals to be selected for the tournament. This formula performs a linear interpolation between the given α and the percentage of entropy in the population E^G. More precisely, it assigns a higher percentage when the entropy is low (close to 0) and a lower percentage when the entropy is high (close to 1). When the entropy between individuals in the population is high, this means that the solutions present in the population are diverse and potentially far apart. At this stage, giving priority to exploration can make it possible to explore the search space further and discover new regions that could contain better quality solutions. By introducing variations in the population, the algorithm is more likely to discover promising solutions that have not yet been explored. On the other hand, when the individuals in the population become similar, this indicates that the algorithm has already converged towards potentially good or optimal solutions. At this stage, exploitation is more beneficial. By exploiting the knowledge acquired so far, the algorithm can refine and improve existing solutions, and potentially converge on an optimal solution. By combining exploration and exploitation in an adaptive way throughout the execution of the algorithm, the advantages of both approaches can be exploited. Exploration avoids getting stuck in sub-optimal local regions, while exploitation allows solutions to be refined and improved as the algorithm approaches convergence.

Algorithm 1: Tournament in Differential Evolution (TiDE)

$P :=$ Population of size $N-1$ randomly generated individuals;
$P \leftarrow$ Solution obtained by filter method;
while *stopping criterion has not been reached* **do**
  $H^G := \emptyset$;
  for $j := 1$ *to* D **do**
    if $\mathcal{P}_x \neq 0$ **then**
      $H^G \leftarrow -\sum_{x=1}^{n} \mathcal{P}_x \log_2 \mathcal{P}_x$;
    else
      $H^G \leftarrow 0$;
    end
  end
  $E^G := \frac{1}{D} \sum_{j:=1}^{D} H_j^G$;
  $p^G := (1 - E^G) \times (1 - \alpha) + \alpha$;
  for $i := 1$ *to* N **do**
    $Xr1 :=$ Best individual from p^G randomly selected individuals from P;
    do
      Randomly select X_{r2}^G and X_{r3}^G from P;
    while $r2 \neq r3 \neq i$;
    for $j := 1$ *to* D **do**
      if $X_{r2,j}^G = X_{r3,j}^G$ **then**
        $V_{i,j}^G := X_{r1,j}^G$;
      else
        $V_{i,j}^G := X_{r2,j}^G$;
      end
    end
    Randomly select j_{rand} from $[1, D]$;
    Randomly select CR from $[0.3, 0.7]$;
    for $j := 1$ *to* D **do**
      if $rand(0,1) \leq CR_i$ *or* $j = j_{rand}$ **then**
        $U_{i,j}^G := V_{i,j}^G$;
      else
        $U_{i,j}^G := X_{i,j}^G$;
      end
    end
    if $f(U_i^G) > f(X_i^G)$ **then**
      $X_i^G := U_i^G$;
    end
  end
  $G := G + 1$
end

4 Experimental methodology

4.1 Datasets and properties

ALS data come from two different datasets. The PRO-ACT database, available online [20], is made up of clinical data on ALS. It was created to bring together data from different ALS clinical trials to provide a comprehensive overview of their results and to facilitate research into the disease. It is freely accessible to researchers and clinicians, making it a valuable tool. The data in this database is frequently used to evaluate the effectiveness of new treatments for ALS, as well as to better understand the properties of the disease and the associated risk factors. It is structured in several tables according to patient information such as age, sex, type of ALS, stage of disease and medical test results, as well as information on treatments received by patients, such as drugs, therapies and surgical interventions. It is the largest reference database on ALS, offering a unique opportunity to develop prognostic models. The Exonhit database is made up of patients from the Exonhit Therapeutics clinical trial [21]. It includes 400 patients who were followed for 18 months, from October 2002 to August 2004. The aim of this clinical trial was to evaluate the effect of pentoxifylline on ALS patients, with a view to developing a complementary treatment to riluzole if efficacy was observed. The study demonstrated that the use of pentoxifylline was not particularly beneficial to patients and should be avoided in combination with riluzole.

The aim of our study is to demonstrate the contribution of our feature selection method. The structure of a dataset can have strong consequences on the efficiency of different selection methods and it is common to have poorly structured data with several anomalies [4] e.g. ALS data. Therefore, we evaluate the performance of the different feature selection methods also in several scenarios with different structures that may represent a challenge for their proper functioning; Noisy data at class

level, Noisy data at feature level, Redundant features, Imbalance between classes, Large number of features, More features than instances, Several anomalies at the same time.

For each of these artificial datasets we specify the structure that the dataset must have, i.e. its number of instances and features, which are relevant and which are redundant, the distribution of instances between the different classes and the difficulty of the classification problem. In addition, we also evaluate the selection methods on the Madelon dataset so that we can compare the results we obtain with those in the state-of-the-art [3, 4]. This is an artificial dataset which has the property of having highly non-linear data and was designed specifically to evaluate feature selection methods for the NIPS 2003 feature selection challenge [22].

Table 1. Properties of the different datasets

Name	Instances (Train, Test)	Features	Relevant	Redundant	Class Noise	Attribute Noise	Minority class
ALS	(4556, 366)	62	?	?	0	0	0.18
Madelon	(2000, 600)	500	20	0	0	0	0.5
Baseline	(2400, 600)	100	10	0	0	0	0.5
Class Noise	(2400, 600)	100	10	0	0.2	0	0.5
Feature Noise	(2400, 600)	100	10	0	0	0.2	0.5
Redundant	(2400, 600)	100	10	10	0	0	0.5
Imbalanced	(2400, 600)	100	10	0	0	0	0.35
Features 1	(2400, 600)	500	10	0	0	0	0.5
Features 2	(600, 150)	1000	10	0	0	0	0.5
All	(2400, 600)	500	10	10	0.2	0.2	0.38

The 'Baseline' dataset is a dataset which has no special properties. This dataset is then used as the basis for creating the other datasets, the properties of which are given in Table 1. The 'Class Noise' dataset is obtained by randomly inverting a specific number of class instances in the training data. The 'Feature Noise' dataset is obtained by adding a random number to the values of randomly chosen features in the training data. The purpose of these datasets is to study the sensitivity of the methods to the noise present in the data. The 'Imbalanced' dataset has the same properties as the 'Baseline' dataset, with the exception of the unbalanced distribution of instances between the classes, which makes it possible to evaluate the impact of the ratio between the classes. The 'Features 1' and 'Features 2' datasets allow us to evaluate the effectiveness of our methods as the number of features increases. In 'Features 1' we add 400 more random features, while in 'Features 2' we add more features than instances. Finally, the last dataset, 'All', represents a dataset that contains several anomalies simultaneously.

4.2 Evaluation criteria

Predictive quality: To evaluate the performance of the models obtained, we calculate the balanced accuracy (BA) score $\in [0, 1]$ from the True Positives (TP), False Negatives (FN), True Negatives (TN) and False Positives (FP). The balanced accuracy is the average between the sensitivity ($\frac{TP}{TP+FN}$) and the specificity ($\frac{TN}{TN+FP}$). It represents the average percentage of prediction scores for each class weighted by the size of each class, allowing a model to correctly identify the examples in each class, without being influenced by class imbalance. A score of 1 indicates no error observed by the model.

Number of features selected: As a second criterion, we use the number of features selected. It is important to consider the number of features selected in addition to predictive performance when evaluating a feature selection method on a dataset, for several reasons. The greater the number of features, the more complex the model will often be, which can considerably reduce the model's interpretability. Furthermore, in real-world data such as ALS, some features may be costly or difficult to collect. If a large number of features are selected, this can lead to additional costs in terms of data collection and preparation. It is therefore important to consider whether the benefits obtained by adding these features justify the associated costs. In this study, we propose a variant

of balanced accuracy that also takes into account the number of features included in the model. This is also the score that the heuristics optimise during their execution:

$$score = BA - (10^{-3} \times \frac{\bar{D}}{D}) \tag{7}$$

$\bar{D}$ represents the number of features selected from the total D features. In this way, even if there are two subsets with the same predictive quality, the less complex model will be considered to perform better.

Stability of scores: Filter and sequential wrapper methods give the same results every time they are run, but this is not necessarily the case for heuristics. The stability of the predictive performance of a feature selection method indicates its ability to produce consistent and reliable results. If a feature selection method is unstable, the results may vary considerably from one data sample to another, calling into question the validity and robustness of the features selected. To assess the stability of the different methods, they were run 10 times for each dataset.

Convergence Speed: For heuristics it is important to evaluate the convergence speed. In many scenarios, it is essential to have analysis results available quickly to make decisions or meet time requirements. In addition, some learning methods can take from several seconds to several minutes to execute depending on the data properties. If a feature selection method converges slowly, it may not be practical or applicable in situations where quick results are required. The convergence speed therefore becomes a critical factor to consider when assessing the usefulness and feasibility of the method. This is why the stopping criterion for the heuristics has been set at two hours of computing time for each of the experiments.

4.3 Learning methods

Seven learning methods were used during the experiments: logistic regression (LR), ridge regression (RR), random forest (RF), support vector machine (SVM), K-nearest neighbour (KNN), gaussian naive bayes (GNB) and linear discriminant analysis (LDA). These learning methods were implemented using the python library scikit-learn (1.2.1) [23]. For the heuristics, the choice of learning method was integrated directly by associating a learning method with each solution (or individual) in addition to the booleans that compose. This increases the search space but allows the heuristics to converge towards the best performing of the seven available methods, thus guaranteeing the best predictive quality. In the case of filter and sequential methods, the subset returned corresponds to that obtained with the learning method offering the best predictive quality.

4.4 Settings and reproducibility

For the filter and sequential methods, the number of features to be selected at the end of the process is determined depending on the number of relevant features if known, and by the number of features of the best result obtained during our experiments with the other methods (ALS). Each heuristic was applied to a population (or neighbourhood) of 100 individuals (or solutions). For the TS heuristic, the distance to each neighbour varied randomly between 1 and 5. For GA, the number of mutations per individual is also randomly chosen between 1 and 5. For PBIL, the learning rate and the mutation shift are fixed at 0.1, while the mutation probability is 0.05. For DE, the values of F and CR are 1.0 and 0.5 respectively. The 'TIDE + ReliefF' method uses the ReliefF method as a filter during initialisation, while the 'TIDE' method performs a random initialisation. In both cases, the value of α is initialised to 0.9. For each dataset, the set of feature selection methods was run 10 times simultaneously on multiple processors, with a time limit of 2 hours as a stopping criterion (whenever appropriate). All the experiments were carried out on a machine running Windows 11 64-bit (version 10.0, build 22621). The processor used was an Intel(R) Core(TM) i7-10700KF with 16 cores, accompanied by 32 GB of RAM and an NVIDIA GeForce RTX 3060 Ti graphics card. For more details on the hyper-parameters of the learning methods, the code is available on GitHub [24].

5 Results

Table 2 presents the performance of each feature selection method on each dataset. The results demonstrate the significant impact of feature selection on the predictive quality of a model, particularly when the number of features is high (Madelon, Features 1, Features 2). For example, the 'TIDE+ReliefF' method achieves a score of 97.97% compared to 63.23% without selection, an improvement of 34.67 percentage points. In general, heuristics appear to outperform filter methods, especially DE and its variants, in all datasets, regardless of their structure. Furthermore, the results obtained surpass those reported in the state of the art [3, 4]. The hybrid heuristic 'TIDE+ReliefF' outperforms 'TIDE' and other methods in most cases, highlighting the value of using filter methods in conjunction with heuristics to obtain a more relevant feature subset. In addition to obtaining the best predictive quality, it also manages to converge on smaller feature subset than the rest of the methods (Table 3), is one of the most stable heuristics in terms of score (Table 4) and its convergence speed remains at the same level as the rest of the heuristics (Table 5). The final ranking of a feature selection method is obtained by summing the rankings of the feature selection method for each dataset. Based on our empirical results, we find that the use of differential evolution and in particular our variant 'TIDE+ReliefF' leads to a better predictive quality.

Table 2. Results of feature selection methods for the predictive quality criterion (Balanced Accuracy). The superscript number indicates the learning method that gave the best predictive quality: LR (1), RR (2), RF (3), SVM (4), KNN (5), GNB (6), LDA (7).

Method	ALS	Madelon	Baseline	Class Noise	Attribute Noise	Redundant	Imbalanced	Features 1	Features 2	All	Rank
No Selection	71.89% ^{1}	69.17% ^{5}	73.23% ^{5}	67.90% ^{2}	75.07% ^{5}	80.73% ^{5}	74.21% ^{5}	65.57% ^{5}	63.23% ^{6}	69.30% ^{4}	13
SCC	66.65% ^{6}	87.17% ^{5}	77.82% ^{5}	72.16% ^{2}	77.82% ^{5}	76.82% ^{5}	76.77% ^{5}	77.83% ^{5}	70.00% ^{5}	69.94% ^{5}	11
Anova	65.16% ^{6}	87.83% ^{5}	77.82% ^{5}	73.66% ^{1}	78.49% ^{5}	77.32% ^{5}	77.06% ^{5}	77.83% ^{5}	71.33% ^{5}	72.30% ^{5}	10
MI	66.47% ^{5}	76.67% ^{5}	71.66% ^{6}	64.82% ^{2}	70.66% ^{6}	77.32% ^{3}	77.06% ^{6}	77.83% ^{2}	71.33% ^{2}	72.30% ^{6}	14
MRMR	67.94% ^{6}	72.00% ^{5}	76.16% ^{5}	69.49% ^{6}	75.82% ^{5}	76.66% ^{5}	75.03% ^{5}	75.83% ^{5}	71.33% ^{6}	64.29% ^{6}	12
ReliefF	66.98% ^{6}	86.33% ^{5}	88.82% ^{5}	79.82% ^{5}	88.82% ^{5}	77.49% ^{5}	81.80% ^{5}	89.00% ^{5}	72.00% ^{3}	72.95% ^{5}	8
FFS	70.38% ^{6}	90.33% ^{5}	88.82% ^{5}	72.16% ^{3}	88.82% ^{5}	88.82% ^{5}	87.79% ^{5}	88.83% ^{5}	71.33% ^{6}	68.49% ^{3}	7
RG	81.07% ^{4}	78.05% ^{5}	79.70% ^{5}	74.38% ^{5}	79.88% ^{5}	82.99% ^{5}	78.92% ^{5}	72.07% ^{7}	72.10% ^{1}	71.41% ^{5}	9
TS	77.50% ^{4}	88.22% ^{5}	88.47% ^{5}	80.83% ^{5}	88.63% ^{5}	87.86% ^{5}	87.15% ^{5}	81.70% ^{6}	91.50% ^{6}	80.40% ^{5}	5
GA	81.37% ^{4}	88.07% ^{5}	88.49% ^{5}	80.59% ^{5}	88.74% ^{5}	87.84% ^{5}	86.96% ^{5}	81.04% ^{5}	89.43% ^{1}	80.42% ^{5}	6
PBIL	81.87% ^{4}	90.17% ^{5}	90.86% ^{5}	79.36% ^{4}	91.04% ^{5}	88.86% ^{5}	89.45% ^{5}	82.34% ^{6}	95.89% ^{1}	81.68% ^{5}	4
DE	81.83% ^{4}	95.13% ^{5}	90.85% ^{5}	81.08% ^{5}	90.87% ^{5}	88.90% ^{5}	89.62% ^{5}	85.75% ^{6}	97.70% ^{1}	**86.90%** ^{5}	3
TiDE	**82.04%** ^{4}	95.43% ^{5}	91.34% ^{5}	79.81% ^{1}	91.11% ^{5}	**89.45%** ^{5}	90.04% ^{5}	86.73% ^{6}	97.57% ^{1}	84.39% ^{5}	2
TiDE + ReliefF	81.96% ^{4}	**95.85%** ^{5}	**91.53%** ^{5}	**86.40%** ^{5}	**91.60%** ^{5}	**89.45%** ^{5}	**90.11%** ^{5}	**92.30%** ^{5}	**97.97%** ^{1}	84.54% ^{5}	1

Table 3. Results of the methods for the criterion of the number of features selected as a percentage of the total number of features

Method	ALS	Madelon	Baseline	Class Noise	Attribute Noise	Redundant	Imbalanced	Features 1	Features 2	All	**Average**
RG	74.52%	41.88%	64.10%	41.40%	58.80%	73.82%	55.60%	26.26%	30.03%	81.92%	54.83%
TS	57.74%	41.52%	35.50%	39.10%	32.50%	45.18%	39.70%	38.26%	30.66%	50.50%	41.07%
GA	62.42%	42.34%	39.70%	42.20%	40.40%	60.00%	50.60%	30.66%	35.18%	61.08%	46.46%
PBIL	**52.42%**	42.16%	28.00%	39.70%	27.50%	**44.00%**	32.90%	43.16%	45.29%	46.72%	40.18%
DE	55.65%	37.86%	30.00%	41.50%	29.90%	49.45%	36.60%	35.58%	32.02%	51.26%	39.98%
TiDE	53.39%	34.68%	27.10%	43.00%	25.40%	48.36%	33.20%	36.20%	32.28%	51.80%	38.54%
TiDE + ReliefF	60.16%	**19.86%**	**17.30%**	**18.80%**	**17.20%**	46.73%	**22.50%**	**4.06%**	**27.65%**	**41.42%**	**27.57%**

Table 4. Results of feature selection methods for the score stability criterion (standard deviation), for 10 experiments

Method	ALS	Madelon	Baseline	Class Noise	Attribute Noise	Redundant	Imbalanced	Features 1	Features 2	All	**Average**
RG	0.71	1.66	0.68	0.88	0.59	**0.42**	1.13	0.66	0.87	**0.32**	0.79
TS	2.88	1.09	0.51	1.76	0.35	0.64	0.57	0.82	2.73	0.69	1.20
GA	0.61	1.52	0.44	2.47	0.50	0.45	0.85	1.97	2.44	1.27	1.25
PBIL	0.32	0.77	0.47	0.41	0.46	0.68	0.59	1.57	0.80	1.15	0.72
DE	0.38	0.80	**0.24**	2.22	**0.25**	0.68	0.37	0.98	**0.56**	2.54	0.90
TiDE	**0.17**	1.15	0.34	0.34	0.36	0.51	**0.22**	1.12	0.78	2.21	**0.72**
TiDE + ReliefF	0.68	**0.49**	0.29	**0.24**	**0.25**	0.47	0.37	**0.37**	1.18	3.07	0.74

Table 5. Results of the methods for the convergence speed criterion

Method	ALS	Madelon	Baseline	Class Noise	Attribute Noise	Redundant	Imbalanced	Features 1	Features 2	All	**Average**
No selection	**00:00:01**	00:00:05	00:00:01	00:00:01	00:00:01	00:00:02	00:00:01	00:00:06	00:00:01	00:00:07	00:00:03
SCC	**00:00:01**	**00:00:01**	00:00:01	**00:00:00**	00:00:01	00:00:01	00:00:01	00:00:01	00:00:01	00:00:01	00:00:01
Anova	**00:00:01**	**00:00:01**	**00:00:00**	**00:00:00**	**00:00:00**	**00:00:00**	**00:00:00**	**00:00:00**	**00:00:00**	**00:00:00**	**00:00:00**
MI	00:00:03	00:00:07	00:00:02	00:00:02	00:00:02	00:00:02	00:00:02	00:00:07	00:00:05	00:00:07	00:00:04
MRMR	00:00:11	00:00:14	00:00:11	00:00:11	00:00:13	00:00:13	00:00:13	00:00:13	00:00:16	00:00:14	00:00:13
ReliefF	00:01:58	00:07:31	00:01:51	00:01:35	00:01:53	00:02:07	00:01:55	00:08:16	00:04:20	00:09:30	00:04:05
FFS	00:10:15	00:51:14	00:03:22	00:03:03	00:03:23	00:03:55	00:03:25	00:38:46	00:20:46	00:40:14	00:17:50
RG	00:47:51	01:00:30	00:59:34	01:02:15	01:08:28	01:08:09	01:06:10	01:16:12	00:53:32	01:17:05	01:03:59
TS	00:01:29	01:33:54	01:28:00	01:13:35	01:09:26	01:09:06	01:13:50	00:57:02	01:07:35	01:16:15	01:07:01
GA	01:26:30	01:24:18	01:05:08	00:55:42	00:58:34	01:03:53	00:53:08	01:12:51	01:02:52	00:54:01	01:05:42
PBIL	01:11:40	01:51:25	01:11:34	00:58:45	01:09:01	01:12:15	01:03:38	01:52:38	01:34:53	01:47:43	01:23:21
DE	01:45:43	01:09:00	00:58:58	01:05:06	01:14:12	01:05:38	00:44:55	01:26:50	01:03:55	01:53:34	01:14:47
TiDE	01:39:22	01:44:19	01:00:35	01:08:29	00:56:49	00:58:02	01:07:18	01:32:10	01:31:33	01:56:14	01:21:29
TiDE + ReliefF	01:17:38	01:15:55	00:53:24	00:51:07	01:10:20	00:55:40	00:59:44	01:04:36	01:40:22	01:55:38	01:12:26

6 Conclusion and prospects

In this work, we compared the experimental performance of different feature selection methods with the aim of optimising the predictive quality of trained models. In particular, we looked at data from the medical field, which poses a number of challenges (low data volume, missing data, noisy and/or useless data). The differential evolution heuristic seems to stand out. We have therefore proposed a new optimised version (population initialisation using a filter method, ad-hoc mutation strategy) which achieves even better results. This good performance is intriguing given its lack of popularity in the scientific literature for the specific problem of feature selection. We therefore recommend using differential evolution rather than other methods. In the future, it would be interesting to explain this apparent superiority from a theoretical point of view, or to identify datasets for which its performance would be less impressive.

References

[1] Avrim L. Blum and Pat Langley. "Selection of relevant features and examples in machine learning". In: *Artificial Intelligence* 97.1 (1997). Relevance, pp. 245–271. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(97)00063-5. URL: https://www.sciencedirect.com/science/article/pii/S0004370297000635.

[2] Laura Morán-Fernández et al. "Feature selection with limited bit depth mutual information for portable embedded systems". In: *Knowledge-Based Systems* 197 (2020), p. 105885. DOI: 10.1016/j.knosys.2020.105885. URL: https://doi.org/10.1016%2Fj.knosys.2020.105885.

[3] Andrea Bommert et al. "Benchmark for filter methods for feature selection in high-dimensional classification data". en. In: *Computational Statistics & Data Analysis* 143 (Mar. 2020), p. 106839. ISSN: 01679473. DOI: 10.1016/j.csda.2019.106839. (Visited on 09/26/2021).

[4] Konstantin Hopf and Sascha Reifenrath. *Filter Methods for Feature Selection in Supervised Machine Learning Applications – Review and Benchmark*. 2021. arXiv: 2111.12140 [cs.LG].

[5] D.H. Wolpert and W.G. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

[6] Thibault Anani, François Delbot, and Jean-François Pradat-Peyre. "Experimental Comparison of Metaheuristics for Feature Selection in Machine Learning in the Medical Context". In: *Artificial Intelligence Applications and Innovations*. Ed. by Ilias Maglogiannis et al. Cham: Springer International Publishing, 2022, pp. 194–205. ISBN: 978-3-031-08337-2.

[7] Girish Chandrashekar and Ferat Sahin. "A survey on feature selection methods". In: *Computers Electrical Engineering* 40.1 (2014). 40th-year commemorative issue, pp. 16–28. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2013.11.024. URL: https://www.sciencedirect.com/science/article/pii/S0045790613003066.

[8] Shuangge Ma and Jian Huang. "Penalized feature selection and classification in bioinformatics". In: *Briefings in Bioinformatics* 9.5 (June 2008), pp. 392–403. ISSN: 1467-5463. DOI: 10.1093/bib/bbn027. eprint: https://academic.oup.com/bib/article-pdf/9/5/392/743955/bbn027.pdf. URL: https://doi.org/10.1093/bib/bbn027.

[9] Hui Zou and Trevor Hastie. "Regularization and Variable Selection via the Elastic Net". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320. ISSN: 13697412, 14679868. URL: http://www.jstor.org/stable/3647580 (visited on 05/26/2023).

[10] Gérard Biau and Erwan Scornet. *A Random Forest Guided Tour*. 2015. arXiv: 1511.05741 [math.ST].

[11] Hanchuan Peng, Fuhui Long, and C. Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (2005), pp. 1226–1238. DOI: 10.1109/TPAMI.2005.159.

[12] Marko Robnik-Sikonja and Igor Kononenko. "Theoretical and Empirical Analysis of ReliefF and RReliefF". In: *Machine Learning* 53 (Oct. 2003), pp. 23–69. DOI: 10.1023/A:1025667309714.

[13] David W. Aha and Richard L. Bankert. "A Comparative Evaluation of Sequential Feature Selection Algorithms". In: *International Conference on Artificial Intelligence and Statistics*. 1995.

[14] Peng Hao et al. "Intra-platoon vehicle sequence optimization for eco-cooperative adaptive cruise control". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 1–6. DOI: 10.1109/ITSC.2017.8317879.

[15] Hongbin Zhang and Guangyu Sun. "Feature selection using tabu search method". In: *Pattern Recognit.* 35 (2002), pp. 701–711.

[16] John McCall. "Genetic algorithms for modelling and optimisation". en. In: *Journal of Computational and Applied Mathematics*. Special Issue on Mathematics Applied to Immunology 184.1 (Dec. 2005), pp. 205–222. ISSN: 0377-0427. DOI: 10.1016/j.cam.2004.07.034. URL: https://www.sciencedirect.com/science/article/pii/S0377042705000774 (visited on 09/26/2021).

[17] Shumeet Baluja. *Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning*. Tech. rep. CMU-CS-94-163. Pittsburgh, PA: Carnegie Mellon University, June 1994.

[18] Rainer Storn and Kenneth Price. "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. ISSN: 1573-2916. DOI: 10.1023/A:1008202821328. URL: https://doi.org/10.1023/A:1008202821328.

[19] Claude Elwood Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423. URL: http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf (visited on 04/22/2003).

[20] *Pooled Resource Open-Access ALS Clinical Trials Database*. https://ncri1.partners.org/proact.

[21] V Meininger et al. "Pentoxifylline in ALS: a double-blind, randomized, multicenter, placebo-controlled trial". In: *Neurology* 66.1 (2006), 88—92. ISSN: 0028-3878. DOI: 10.1212/01.wnl.0000191326.40772.62. URL: https://doi.org/10.1212/01.wnl.0000191326.40772.62.

[22] Isabelle Guyon et al. "Result Analysis of the NIPS 2003 Feature Selection Challenge". In: *Advances in Neural Information Processing Systems*. Ed. by L. Saul, Y. Weiss, and L. Bottou. Vol. 17. MIT Press, 2004. URL: https://proceedings.neurips.cc/paper_files/paper/2004/file/5e751896e527c862bf67251a474b3819-Paper.pdf.

[23] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *CoRR* abs/1201.0490 (2012). arXiv: 1201.0490. URL: http://arxiv.org/abs/1201.0490.

[24] Thibault Anani. *Tournament in Differential Evolution*. https://github.com/thibaultanani/Tournament-in-Differential-Evolution. 2023.

Memetic Algorithms for the Technician Routing and Scheduling Problem: real case study of Energy Distribution System Operator (DSO)

L. Cardinaël[1,2], W. Ramdane Cherif-Khettaf[1], and A. Oulamara[1]

[1] LORIA, UMR 7503, Lorraine University, Campus Scientifique, 615 Rue du Jardin-Botanique, 54506 Vandœuvre-lès-Nancy
[2] efluid SAS, 2 Bd du Pontiffroy, 57000 Metz, France

Abstract. This study addresses a real industrial problem faced by a company operating an energy distribution network: the efficient assignment of tasks to their agents. This problem involves determining the best routes for agents to visit multiple tasks, known in the literature as the Technician Routing and Scheduling Problem (TRSP). It considers a limited number of agents, and tasks located in different areas. Each task has a service duration, a time window during which the service should be executed, and a set of required skills. Additionally, agents possess different skills, multiple availability intervals, and a maximum daily working time. The objective is to minimize the total traveled distance, including penalties for unrouted tasks. To tackle this problem, we propose an order-first split-second approach that combines a memetic algorithm using a giant tour encoding of the chromosome with an extension of the optimal split method. Our primary objective is to evaluate the effectiveness of this approach on a real case of the TRSP problem in the context of energy network management. The second objective is to compare our genetic algorithm, which uses the giant tour encoding with a genetic algorithm scheme that employs an indirect encoding representation proposed in the literature. Il is worth noting that this indirect enconding has proved its effectiveness on real instances of an industrial problem, which is similar to our case study. We aim to test different implementations of the proposed genetic algorithms on real instances to evaluate the impact of the extended optimal split, local search, and encoding type.

1 Introduction

This paper is motivated by an application encountered by an Energy Distribution System Operator (DSO) related to efluid, a company specialized in developing an expert software package solution that covers all requirements of electricity, gas, and water stakeholders. The problem addressed in this paper involves the management of electricity and gas networks. For a network operator (DSO), issues such as the installation and maintenance of equipment in the energy network must be solved within a given period of time. Failure to allocate resources effectively (materials and staff) can result in penalties and a decline in customer service quality. Additionally, specific constraints within this environment must be taken into account. Tasks are generally constrained by time windows, and some tasks, such as changing high voltage lines, may require specific skills or the participation of multiple agents simultaneously. There can also be multiple interventions at the same site throughout the day, requiring the involvement of the same agent for effective knowledge of the location and environment. Furthermore, in certain situations like power cuts intervention, agents may need to focus on specific skills for a part of the day.

These various interventions require skilled agents and necessitate careful planning based on the availability of agents and the required skills for interventions. Thus, technicians routing and tasks assignment are crucial considerations in the context of energy network management. This issue is modeled in the literature as a Technician Routing and Scheduling Problem (TRSP)[14]. The TRSP belongs to the broader class of routing and scheduling problems called Workforce Routing and Scheduling Problems (WRSP)[6]. Specifically, WRSP refers to situations where workers need to travel to different customer locations using vehicles to perform assigned field operations. The tasks must be executed within their time windows, require certain skills for their completion, and are characterized by a service time, while the travel time between tasks can be significant. The number of activities at different sites is usually greater than the number of available employees, which means that employees must travel between sites to do their work. This results in a combination

of personnel scheduling and vehicle routing problems. Several objectives can be considered, such as reducing personnel travel time because travel time is considered as paid working time, reducing the cost of temporary personnel, and maximizing customer satisfaction. The WSRP has been extensively studied in the literature, with many variants corresponding to different application domains. For example, home care, with the scheduling of nurse's visits to their patient's homes [21], security agent's rounds in different locations, delivery services, the restaurant industry, and cleaning services [13]. All these applications involve scheduling and routing agents to ensure they reach designated locations on time to carry out assigned activities.

The TRSP problem is a special case of the WRSP and is specifically designed for applications where the objective is to optimize the routing of technicians or service personnel for maintenance or repair tasks. This problem typically involves determining the best routes for technicians to visit multiple customer locations or equipment sites, considering factors such as distance, travel time, service priorities, and technician skills. It is particularly relevant in industries such as telecommunications, utilities, and facilities management.While the home care and delivery domains have received a rich research coverage [25], the problem of technician routing is much less present in the literature, despite its strong functional component. Therefore, only the specificities that are found in particular for home care benefit from proven solutions.The TRSP models in the literature take into account classical constraints such as the management of technician's skills [10] and their availability [4], and time windows for interventions. In practice, the TRSP often involves working with complex business constraints [20] [11].

A review of the literature on TRSP and WRSP problems reveals that metaheuristics have been widely used to solve these problems. For instance, invasive weed pptimization [22], tabu search coupled with an adaptive memory [17], adaptive large neighborhood search [15], particle swarm optimization [18] [12] have been investigated. Genetic algorithms, in particular, have been successful, in addressing various variants of TRSP and WRSP [19][9][7].

In this paper, we propose an order-first split-second approach to address a real TRSP problem. The approach combines a memetic algorithm (genetic algorithm with local search as a mutation operator) using a giant tour encoding of the chromosome with an extension of the optimal split method [24] to solve real instances provided by efluid. This approach has proven its effectiveness for various Vehicle Routing Problem (VRP) variants [24] [16] [5]. Here, we propose an extension of the optimal split method to handle the specific constraints of our problem. Our primary objective is to evaluate the effectiveness of this approach on a real TRSP problem within the context of energy network management.

Further research into the literature revealed that real instances of WRSP, similar to our case study, were efficiently solved using a genetic algorithm with indirect encoding [3]. This encoding produced competitive feasible solutions for highly constrained WSRP. Hence, the second objective of this paper is to compare our genetic algorithm using the giant tour encoding with a genetic algorithm schema that uses the indirect encoding proposed by [3]. We aim to test different implementations to evaluate the impact of the extended optimal split, local search, and encoding type on the studied problem.

The remaining sections of this paper will be organized as follows: In Section 2, we will define our industrial problem and emphasize its significant characteristics. Section 3 will present two genetic algorithm approaches. The first approach is based on the indirect encoding technique from the literature, which we have adapted to suit our problem. The second approach is based on the giant tour representation and incorporates an extension of the optimal split method. Section 4 will present the industrial instances used in our study, along with a comparative analysis of several genetic algorithm implementations. This analysis will include the use of local search as a mutation operator (memetic algorithm), as well as the use of an optimal split method, and a simplified split based on a sequential technique. Finally, conclusions and perspectives will be given in section 5.

2 Problem definition

This paper focuses on an industrial application of the technician routing and scheduling problem in the context of energy network management. The problem is defined on complete directed graph $G = (V, A)$. The set of arcs is denoted by $A = \{(i, j) \mid i, j \in V\}$. Each arc (i, j) of A is described by distance $c_{i,j}$, travel time τ_{ij}. V denotes the set of vertices composed of a depot denoted by 0,

and n customers that each customer represents a unique task. A task defines a customer's request for an installation or maintenance operation on the energy network such as changing an electricity meter, installing and repairing equipment, and maintaining power lines. Each task $i \in V$ has an associated service duration s_i , a time window $[e_i, l_i]$, within which the service should be executed. A list r_i of required skills is defined for each task i.

A set $K = \{1, \ldots, nk\}$ of technicians are available to perform the tasks. Each technician is specialized in a number of skill domains. There are compatibility constraints between tasks and technicians, since different skills are required to perform different tasks and a technician does not necessarily have all those skills. Every technician departs from and returns to the depot, and has a maximum working time denoted by T. A specific constraint of our industrial problem, which to our knowledge has not been addressed in technician routing and scheduling problems, is the fact that technicians have multiple availability intervals. Outside these intervals, technicians have other specific tasks, which are outside the scope of our study. The aim is to assign a subset of tasks to each technician and to construct a route over each subset to minimize the total traveled distance. The routes must satisfy the compatibility constraints between tasks and technicians, the time bounds for the service of each task and for the return time of each technician at the depot, and the availability intervals of technicians. It is also assumed that not all tasks can be served by the technicians. Thus, a penalty, denoted by P is associated with each unrouted task and the minimization of the total penalty becomes part of the objective. The value of this penalty is set to $P = 200$ in our study. This corresponds to the maximum distance in kilometers that a technician would travel to perform a single task, which is the farthest one. A TRSP solution consists in finding a sequence of feasible tasks to be carried out by each technician according, while satisfying all the constraints of our problem. Each sequence begins and ends at the depot. The aim is to minimize the total distance covered to perform all the assigned tasks, and the total penalty distance for unperformed tasks.

3 Genetic algorithms

The Genetic Algorithm (GA) has been successfully used to deal with a large variety of combinatorial optimization problems, including the Technician Routing Problem [8] [14]. A genetic algorithm scheme with indirect encoding proposed in [3] has shown its effectiveness, and allow an efficient customised chromosome representation combined with problem-specific genetic operators for tackling real-world WSRP instances. The advantage of the proposed indirect encoding is the smaller and simpler search spaces that are produced, to ensure the feasibility of solutions.

Our aim is to compare two genetic algorithms schemes. The first one uses the indirect encoding and the crossover proposed in [3], referred to as GA-I, and the second schema, entitled GA-S, is based on a giant tour or task ordering chromosome representation and a splitting procedure used as a decoding process. In the context of VRP and its variants, several metaheuristics including splitting procedure reported very good results. The key feature of these metaheuristics consists in encoding solutions as giant tours and using a splitting procedure to extract the corresponding solution of the studied VRP [24] [16]. The split procedure computes a shortest path in an auxiliary graph, in which each arc models a possible trip [24]. Several extensions of the split procedure were proposed in the literature to deal with extensions of classical routing problems. In this paper, we propose a novel split extension to handle simultaneously several realistic constraints such as a limited heterogeneous fleet of vehicles, compatibility between tasks and resources, multiple availability time windows per resource, a time window per task, and thethe possibility to skip tasks in the giant tour sequence.

The general framework of the both GAs is the same and will be detailed in the following. The initial population of size N is generated randomly using one of the proposed encoding schemes. Subsequently, the GA runs for a number of iterations. In each iteration, the following steps take place. First, each individual in the population is evaluated based on its fitness after applying a decoding or split procedure to extract a feasible solution from the used encoding scheme. Second, $N/2$ of individuals are selected by tournament selection, and crossover operators are applied to this pair of parents to produce N solutions, called children. Thirdly, a mutation is applied to each generated child with a given probability. If the solution obtained after mutation is better than the solution it started from, it replaces the latter in the population; else, the solution before mutation or local search is retained. Finally, a replacement strategy based on binary tournament selection

of individuals from the new population formed by N parents and N children, is used to create the new population. The best solution is always preserved in the population for the next generation. The best solution is returned when the stopping criterion of the GA is met.

The main contribution of our GA-S method is the extension of the classical optimal split procedure to handle the constraint of our industrial problem, and the comparison between two efficient encoding schemes proposed in the literature. Following, we'll first outline the specific features of each genetic algorithm schema, then we present the common ingredients to both schemes as the mutation operators, and the selection and replacement strategy.

3.1 Genetic Algorithm with Indirect encoding (GA-I)

The study in [3] investigated a genetic algorithm with indirect representation schema for an industrial application of WSRP, that allow maintaining feasible solutions throughout the evolutionary process. The study presented solution encoding, that is capable of producing competitive feasible solutions for a highly constrained WSRP. We propose here a genetic algorithm schema including the main component of [3] which are the chromosome encoding representation and the crossover operator.

Indirect chromosome representation Indirect gene encoding is a lesser known approach for solving VRPs using genetic algorithms [2] [3]. It relies on a more indirect way of encoding genes, but allows generating genes without value restrictions. This enables a wider variety of operators without the need for specific repair procedures. To explain this part, we will use the concept of slot-agent. These are the cartesian product of the agents and the time windows. In [3], each individual is represented as a vector of integers of size n, which corresponds to the number of tasks in the schedule. The integer value at each position in the vector indicates the worker assigned to that particular task. To ensure temporal feasibility, the tasks in the chromosome are sorted in a non-decreasing order based on their start time. Then, for each task, the compatible agents are sorted using an heuristic. Then, for each task, we use the first integer of the gene as the index in wsl of the slot-agent which will do the task. If the slot-agent is already full, we use the next slot-agent in wsl. If we exhaust all slot-agents left in wsl, we consider the task is non-assigned.

We have proposed two adaptations of the representation described above to deal with the constraints of our problem. The first one is the use of a vector of two integers. The first integer has the same goal than the one from the literature. The second helps to place the task inside a time window. All tasks in the same time window are ordered according to this second integer. The second adaptation is that, for each task, [3] compute a penalty cost of doing the task inside each slot-agent, and order slot-agents by this cost for each task. We don't have for the moment a good metric to order our slot-agents, so we will just add to that ordered list (noted wsl in the other paper) for each task the slot-agents which are able to do task (these which respects the time window and the skill needed).

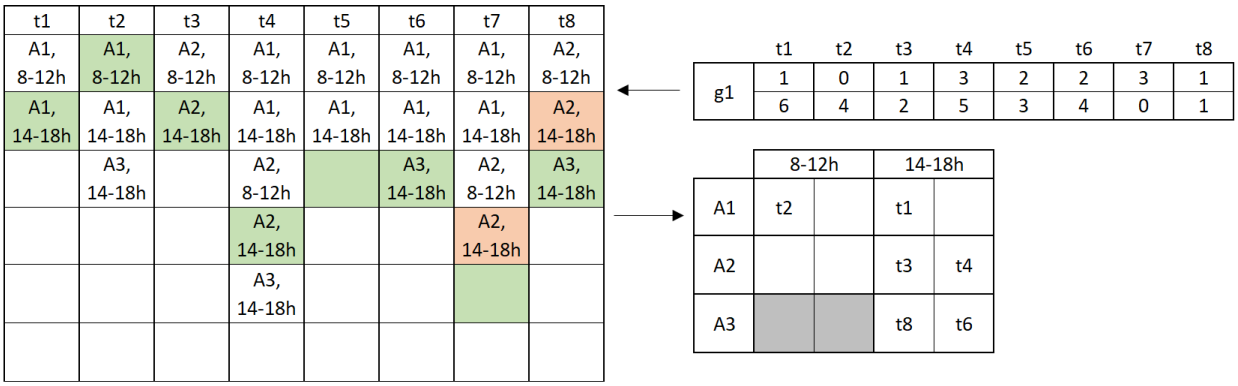


Fig. 1: Example of an indirect encoding

The figure 1 shows an example of indirect encoding. There is above left a gene $g1$, and the example will show how this gene could be decoded. We have three agents, with A_3 working only on afternoon. The left table shows for each task which agent can do them, that is wsl. We will assume for this example that each slot-agent can do exactly two tasks. Green cells indicate time windows retained, while orange ones indicate non-feasible solutions. That's why the cell just under is either orange or green (that is the test of the next solution). Index begin at 0, so the "1" for t_1 means the second line of the table.

We see that the first number in the gene impact the placement in the table. For t_7 and t_8, the first proposition was already occupied, so we take the next one. For t_5 and t_7, no proposition is selected, that means they are not affected. The planning proposed by this gene is the left-below table. With this, we can calculate the score.

The worst-case complexity of this algorithm is in $O(nk*c*n)$, where a is the number of workers, c is the number of time windows and n the number of tasks. But, the choice of time window given by the gene is often the good one, so it becames often a complexity in $O(n)$.

Crossover operator. Indirect encoding doesn't use ordering of tasks. Therefore, we can use more generic crossover algorithms. The one we used here is the Uniform Crossover (UX). Since the indirect encoding preserves the feasibility, UX is better at generating new solutions than one-point or two-points crossover, that are more adapted at keeping a sequence of the gene.

In UX, each gene of the offspring is inherited randomly from either parent with a fixed probability. This means that for each gene position in the offspring, there is a certain probability that the gene will be copied from the corresponding position of the first parent, and a complementary probability that it will be copied from the second parent.

Chromosome evaluation. A decoding procedure is necessary to extract solutions from chromosomes, and to evaluate it, we use the decoding method presented above. Then, we establish the distance between tasks for each route and we sum it. We add the penalty for each undone task and we obtain the total cost of the solution. The complexity is held by the decoding algorithm (see 3.1), so the complexity is generally in $O(t)$.

3.2 Genetic Algorithm with Splitting (GA-S)

In the following, we will present the chromosome encoding within the framework of the GA-S schema, the crossover operator used, and we will focus on the description of the optimal split method that we have adapted to the constraints of our problem.

Giant tour chromosome representation. Most genetic algorithms for routing problems use quasi-direct representations of solutions, as sequences of tasks. In our GA-S, a chromosome g simply is a sequence of n tasks, without trip delimiters, and with implicit shortest paths between consecutive tasks. g can be viewed as a giant tour ignoring the constraints of our problem.

Crossover operator. Our giant tour representation without trip delimiters can undergo classical crossovers for permutation chromosomes. The resulting children can be immediately evaluated with split. Because the chromosome before splitting may be viewed as a circular object (giant trip), we have decided to use a classical crossover, named Order Crossover (OX), which maintains the circular nature of the giant trip.

Given two parents P_1 and P_2 with n tasks, OX draws two cutting sites p and q with $1 \leq p \leq q \leq n$. To get the first child C_1, OX copies $P_1(p), ..., P_1(q)$ into $C_1(p), ..., C_1(q)$. P_2 is then scanned from $P_2(q+1), ..., P_2(n)$, then from $P_2(1), ..., P_2(p-1)$, with restriction that tasks from P_2 are taken only if missing in C_1. However, C_1 is interpreted as a circular list and the result stored such that $C_1(p) = P_1(p)$. The other child C_2 is obtained by exchanging the roles of P_1 and P_2. An example is shown in figure 2.

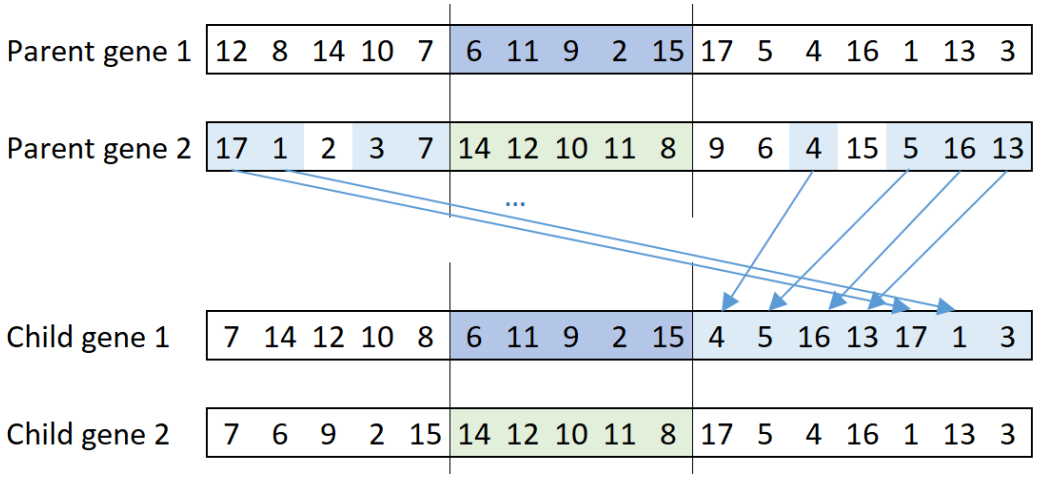


Fig. 2: Example of OX crossover

Optimal splitting. The main idea behind splitting is that the VRP problem has two parts: the clustering of tasks into multiple routes, and the scheduling of tasks into those routes. If the two aspects cannot be entirely uncorrelated, the splitting will participate in reinforcing the clustering side of the problem, with an order-first split-second approach [16]. The goal of the split procedure is to cut the giant tour given by the chromosome into routes for the technicians. it keeps the order given by the giant tour, and extract feasible routes and unrouted tasks. That's why we are in an order-first split second approach.

In this paper, we propose to extend the split procedure for heterogeneous fleet [23] to deal with the constraints of the studied TRSP . For a given visiting sequence $(t_1, \ldots..t_n)$ of n customers let t_i be the customer in position i. An auxiliary graph H=(V, U) is defined, V includes $n+1$ nodes indexed from 0 to n. Each subsequence $(t_{i+1}, \ldots, t_j)$ corresponding to a feasible trip is modeled in A by an arc (i, j), This trip is starting from the depot, visiting customers t_{i+1} to t_j, and coming back to the depot. Each arc (i, j) is evaluated by a cost that represents the sum of the total tour distance $(0, t_{i+1}, ...t_j, 0)$ and a penalty calculated as $P \times na$ (na denotes number of tasks not achieved, and P is a penalty of one unrouted task). Each task can be skipped in the sequence, with a penalty linked to its omission. We have to select one compatible resource for each arc in U, in such a way that each resource will be used at most one in the final path. We have to remember for each node j in U the resources consumption of the partial path to know if it can be extended. We should store at each node j the labels of all incoming paths because it is not certain that the cheapest path can be extended to reach the final node. We propose to use several vectors of multi-labels per node j. Each vector represents the consumption and the cost of one feasible partial path arriving to j. In each vector, labels design first the resources used for the considered partial path extended from i, then the incoming node i, and finally the total cost. We also consider the case of skipping all tasks from the depot to node j, by creating an arc with cost corresponding to the total penalties related to the number of skipped tasks. In practice, we use a dominance rule to discard many labels, and only non-dominated labels are stored on each node. The dominance rule helps to eliminate useless cases, and consists in two subrules :

- If a label is a subpart of another label and has a worst score, then it is removed
- If a label has a worst score than another label considering all tasks left as undone, then it is removed

The complexity is in $O(f*n)$ with f the number of vectors (depending on the number of agents) and n the number of tasks, as long as the number of vectors does not go wild. That's why we need dominance rules to keep this number not so high

Figure 3 and 4 illustrates the Split algorithm on a sequence of 8 customers t_1 to t_8. The first graph of figure 4 shows the giant tour and the indicated values represent distance of each arc. We consider the penalty parameter P=200, and the following data:

We have three agents (table 1) and eight tasks (table 2). Tasks are localized according to figure 3. We will admit that the numbers indicated is the travel time (in minutes) and also the road distance (in km), to simplify the example. For example, if "5" is indicated, that means it takes 5 min to travel this 5-km long.

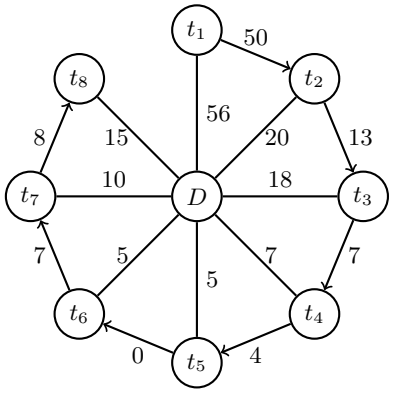
Name	Availability	Skills
A_1	8-12h and 14-18h	V and B
A_2	8-10h and 14-18h	V and R
A_3	14-18h	V and R

Table 1: Three agents

Name	Time window	Duration	Skills needed
t_1	8-12h	60min	V
t_2	14-18h	10min	V
t_3	8-12h	20min	R
t_4	8-12h	30min	V
t_5	14-18h	30min	B
t_6	14-18h	60min	V
t_7	8-12h	60min	V
t_8	14-18h	90min	R

Table 2: Eight tasks

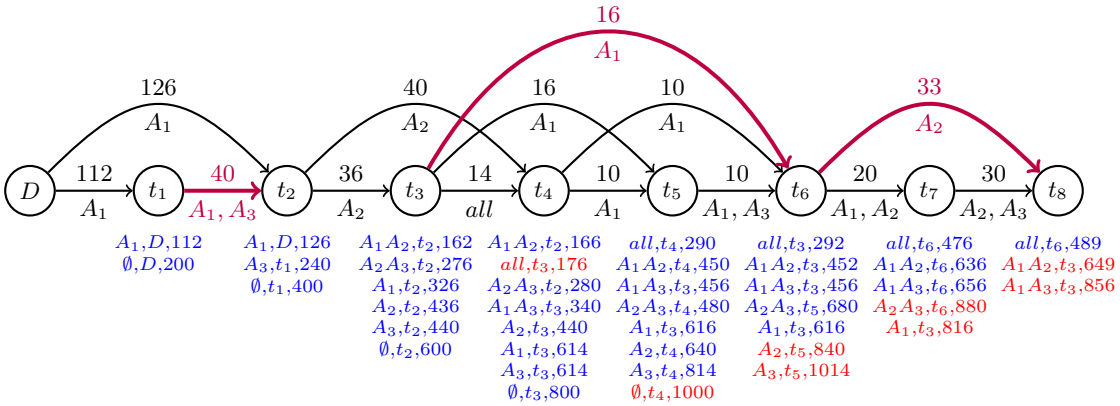
Fig. 3: Giant tour representation



The first step is to create the auxiliary graph. We represent only feasible trips, and notice only resources that can perform the task (the constraints of agent availability, task time window, and compatibility are all satisfied). The cost is based on the road distance. Note that the travel time and the task duration time are only used to determine if all these tasks can be done by agents. With this method, we obtain the auxiliary graph given by the figure 4.

Then, we have to find the optimal path in the graph. For this, we propose to use a vector of multi-labels on each node i (the blue and red text on the figure), to note the previous agents used until i, the incoming node to i to indicate the previous step, and the cost of the partial path using the incoming node until i. Also, we have to take into account the possibility to skip a task, and get the penalty P. The $\emptyset$ in the label of node ti, indicates that all tasks in the sequence $(D, ...t_i)$

Fig. 4: Auxiliary graph for the extended optimal split algorithm



are skipped. The multi-label system is mandatory to find the best path. We need to use each agent only once, so we have to track which agents we used on each step. Moreover, for each step, we don't know what path will be part of the optimal solution. By using a vector with labels that indicate the agents already used and the node of the previous step, we can address these problems. At each step, we have all possible paths that could be part of the optimal. The node of the previous step in the label is used to backtrack and to have the final resource affectation once the best path is found.

Let's give an example with the label development at node t_4. First, we will take paths from t_3. We take labels from t_3, add an agent which can do the transition (that means any agent not already used here). We obtain :

- $A_1A_2 + A_3 \implies all, t_3, 176$
- $A_2A_3 + A_1 \implies all, t_3, 290$ strictly worse than another solution, removed
- $A_1 + A_2 \implies A_1A_2, t_3, 340$
- $A_1 + A_3 \implies A_1A_3, t_3, 340$
- $A_2 + A_1 \implies A_1A_2, t_3, 450$ strictly worse than another solution, removed
- $A_2 + A_3 \implies A_2A_3, t_3, 450$
- $A_3 + A_1 \implies A_1A_3, t_3, 454$ strictly worse than another solution, removed
- $A_3 + A_2 \implies A_2A_3, t_3, 454$ strictly worse than another solution, removed
- $\emptyset + A_1 \implies A_1, t_3, 614$ strictly worse than another solution, removed
- $\emptyset + A_2 \implies A_2, t_3, 614$
- $\emptyset + A_3 \implies A_3, t_3, 614$
- $A_1 + \emptyset \implies A_1, t_3, 526$
- $A_2 + \emptyset \implies A_2, t_3, 636$ strictly worse than another solution, removed
- $A_3 + \emptyset \implies A_3, t_3, 640$ strictly worse than another solution, removed
- $\emptyset \implies 800$

We keep only the best options for each combination of agents. Then, we do the same for the other transitions. Here, we have only one other transition, and it is only with t_2 :

- $A_1 + A_2 => A_1A_2, t_2, 166$ better than the existant
- $A_3 + A_2 => A_2A_3, t_2, 280$ better than the existant
- $\emptyset + A_2 => A_2, t_2, 440$ better than the existant

Finally, all the computed labels for t_4 are dominated by the label $(A_1, A_2, t_2, 166)$ because of a better solution with less agents, so we remove the *all* other labels of t_4,

Note that we only need the best vector for each combination of agents.If a vector uses more agents, and give less cost (for example, the red vector at t_4 in figure 4) we can remove it. Finally, if a vector is worse than the actual best vector considering all penalties it would get to reach the end of the graph (that's the red multi-labels vectors in $t_5 - 8$ in figure 4), it will never be the best solution, and thus, be removed.

Finally, we take the best cost at the end node of the graph, and we can backtrack to obtain the best solution, in purple (t_1 and t_3 are skipped, giving a 400km penalty). With this, the tasks are affected as follows :

- A_1 does the tasks t_4, t_5, and t_6.
- A_2 does the task t_7, and t_8.
- A_3 does only t_2
- t_1 and t_3 are not affected (no purple arrow reach them)

With that algorithm, we can respect the constraints of our problem, while having access to the split benefits. However, the number of multi-labels vectors really increase with the number of agents.

3.3 Other genetic algorithm components

In this section, we present the common components of the GA-I and GA-S methods. These are the selection and replacement strategy, as well as the mutation operators. We have proposed two classical mutation operators, and one mutation operator based on a local search.

Selection and replacement. The selection by tournament has demonstrated it effectiveness for several transportation problems [1]. The principle is to choose a subset of s individuals from the population (called the tournament size), and then select the individual in the group which has the best total cost value. This process is repeated until the number of required individuals is reached. Initially, the population of N chromosomes is generated completely at random, and the binary tournament selection procedure ($s = 2$) is used to ensure the choice of $N/2$ pairs of parent candidates for the crossover and mutation process. After the crossover and mutation process, we have N new offspring. The binary tournament strategy is used again to choose among the $2N$ chromosomes. We chose two chromosomes randomly and remove the worst one until the new population of size N is formed. The best individual will always be retained.

Mutation operators.

- Relocate operator. For the giant tour encoding representation, move one customer i chosen randomly from its original position to another position selected randomly in the permutation. For the indirect encoding representation, randomly select a task, randomly assign it to another interval, and randomly give it a weight to determine its order within the selected interval. An unrouted task can become feasible, and inversely
- Swap operator.It consists of exchanging the position of two customers i and j in the permutation, or in exchanging the interval and the weight of I and j in the indirect encoding representation. An unrouted task can become feasible, and inversely
- Local search. The mutation is replaced here by a local search procedure, called with probability pm, which works on the individual solution instead of the chromosome encoding representation. Regarding our problem, many local search techniques cannot be applied, because of the time-window constraints. Some local search algorithms will create non-viable solutions. That's why we chose to use extra 2-OPT. An improvement is applied as soon as it has been found. Each iteration scans all feasible moves between two distinct trips and executes the first improving move detected. The whole local search stops when no more improvement can be found.The idea of this local search move is to switch a part of the routes between two workers. An example is shown in the figure 5. This 2-OPT is thought to not disturb the direction of each subroutes. Here, we see that u, v and x, y arrows become u, y, and x, v without disturbing other arrows. The solution obtained can be unsolvable, but the risk is greatly reduced here. Thus, we can obtain more interesting solutions with this local search. Of course, other methods could be more effective, but for the scope of this study, we will only test this one.

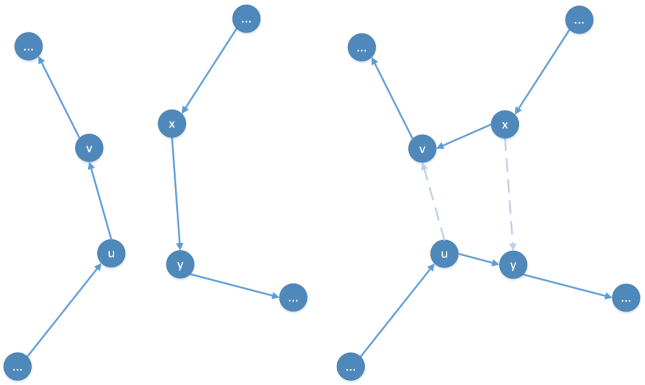


Fig. 5: Example of 2-OPT

4 Experimentation

In this section, we report the results of a comparison of 6 Genetic Algorithm implementations on industrial instances. Let GA-I1 and GA-S1 be the version of the proposed genetic algorithm using a mutation based on a classical operator (relocate or swap), and GA-I2, GA-S2 use a mutation based on local search . In order to evaluate the impact of the extended Split, we implemented a version of GA-S, named GA-SS where the split procedure has been replaced by a sequential splitting . Two versions of GA-SS are also proposed: GA-SS1 (with classical mutation operator) and GA-SS2 (with RL-based mutation operator). The sequential splitting involves scanning the giant tour sequence from left to right, and building a solution with tour delimiters. The sequence is simply scanned from left to right, an agent is randomly selected from the list of agents, and tasks are assigned to it in the order of the sequence until the agent is no longer available, or a task cannot be assigned to it. This process is repeated with a second agent, and so on until all tasks are assigned or no agent is available. Unassigned tasks in the sequence are considered unfeasible and will be penalized. All the proposed algorithms been programmed in Python and tested on a Windows computer, Intel core i9-9880H and 32 Go of RAM. In the following, the instances are first described, then the obtained results are discussed.

4.1 Presentation of the industrial instances

To test our algorithms, we have access to the réséda's database, which is the DSO of the UEM group, from which is also efluid. It contains all tasks done by the company since 2015, including all data needed to reproduce work days. All industrial instances revolves around the Metz Metropole, a town in the east of France. All interventions are located on three major areas : Metz, the Moselle, which forms a 20 km line going south from Metz, and the communal group of Faulquemont, 40km east of Metz. The depot is located at the réséda's headquarters, in Metz. The agents will have to travel to Faulquemont or across the Moselle to realize tasks located there.The first area concentrates around 50% of interventions, the second one 30% and the third one 5%. Others are placed around and between these three areas. Tasks are either on the morning time window (8-11h) or the afternoon one (13-16h), and last between 5 and 35 minutes, with half of them between 10 and 12 minutes. They are nos skills for this test.

Table 3 shows 8 industrial cases instances, based on real intervention routes. S1 and S2 corresponds to low-demand routes. S3, S4, S6 and S7 are more aligned to standard demand, while S5 and S8 is are saturated, with a non-sufficient number of agents. We must expect a lot of tasks to not be fulfilled.

4.2 Numerical results.

We performed several preliminary tests to set the parameters of our algorithms. The selected mutation operator is relocate for GA-I1, and swap for GA-S1, and GA-SS1, with a mutation probability $pm = 0.7$. Versions GA-I2, GA-S2, and GA-SS2 use the local search-based mutation

operator (2-opt) with probability $P'm = 0.2$.The population size is set to 1600 for GA-I1 and GA-I2, and 160 for others. The stopping criterion is a time limit that has been set to 5 minutes. It corresponds to the standard industrial use. Table 3 shows the average results obtained for each algorithm on 10 runs. The value in the columns 4, 5, and 6 indicate the total cost (score) and the number of non-assigned (noted na) tasks. In practice, each non-assigned task costs 200. That means that the "Direct" score for S4 is $255 + 63.8 \times 200 = 13015$. This very high-cost will ensure that the algorithm will construct routes with the higher number of tasks possible, even if the resulting routes is less efficient.

Results show that for all tested GA schemas, the GA with local search of each algorithm outperforms the version without local search. This is in line with the results of the literature, which claims the effectiveness of memetic algorithms (GA with RL-based mutation) compared to classical versions of GA without RL.We can note that the GA with indirect encoding really benefits from it (between 15 and 40% better). Our interpretation is that local search helps to allocate unnecessary unassigned tasks. The best results are obtained by GA-S2, which uses the split procedure that we developed specifically for the problem, and the local search as mutation operator. The impact of split can be measured by comparing GA-S2 with GA-SS2 (improvement from 3 to 7%), and by comparing GA-S1 with GA-SS1(from 3 to 20 %). For small instances (S1 & S2), the difference is low (indirect encoding is already efficient on them). But for higher instances (S3, S4, S6 & S7), the difference widens, and more for saturated instances (S5 & S8). If we use the total cost, we get 7% of improvement for S4, and 9% for S3. This high improvement is justified by the non-assigned tasks difference. The results clearly show that GA with indirect coding of [3] reports the worst results, and affirms the effectiveness of the giant tour encoding representation combined with the split method. We can explain this by the difficulty that the GAs algorithm with the indirect encoding have in concentrating good chromosomes. Indeed, indirect encoding and the associated crossover do not preserve information on task order per agent, so the GA will have difficulty in preserving good sequences.

	Interventions	Agents	GA-SS1	GA-I1	GA-S1
S1	75	4	69 , 0,0 na	92 , 0,0 na	70 , 0,0 na
S2	109	7	168 , 0,0 na	192 , 0,0 na	167 , 0,0 na
S3	121	8	188, 0,0 na	215 , 0,1 na	188 , 0,0 na
S4	145	10	281 , 0,3 na	403 , 0.9 na	242 , 0,1 na
S5	148	9	1472 , 6,0 na	4349 , 20,3 na	1192 , 4,1 na
S6	152	10	608 , 1,9 na	1047 , 4.1 na	589 , 1.8 na
S7	154	10	788 , 2.8 na	1588 , 6.8 na	769 , 2,7 na
S8	192	8	13015 , 63,8 na	16877 , 83,2 na	12140 , 59,4 na
	Interventions	Agents	GA-SS2	GA-I2	GA-S2
S1	75	4	67 , 0,0 na	76 , 0,0 na	66 , 0,0 na
S2	109	7	165 , 0,0 na	177 , 0,0 na	158 , 0,0 na
S3	121	8	186 , 0,0 na	191 , 0,0 na	180 , 0,0 na
S4	145	10	240 , 0,1 na	305 , 0.4 na	222 , 0,0 na
S5	148	9	1135 , 4,2 na	3694 , 17,1 na	1035 , 3,7 na
S6	152	10	567 , 1,7 na	868 , 3.2 na	530 , 1.6 na
S7	154	10	749 , 2.6 na	1170 , 4.7 na	711 , 2,4 na
S8	192	8	12119 , 59,3 na	14642 , 72,0 na	11281 , 55,1 na

Table 3: Comparison of different implementations of GA

5 Conclusion

In this paper, we presented a concrete industrial case of the technician routing problem, raised by efluid, a company specialized in the management of an energy distribution network. We proposed a memetic algorithm, named GA-S2, using a giant tour representation of the solution, a local search as mutation operator, and adapted the split procedure to handle the specificities of our problem. This

type of approach which associates a memetic algorithm using a representation of the solution as a giant tour, with an optimal procedure for splitting this giant tour into feasible trips, also known as order-first split-second MA based approach, has already proved its effectiveness on several variants of VRP. The aim of this paper was to confirm this statement on real instances in the context of energy distribution system operator. We also compared the GA with giant tour representation with another indirect coding representation proposed by [3], which also proved its effectiveness on real WRSP instances, a problem very close to our case study The results showed that the indirect coding versions of GA were not competitive with GA using a giant tour representation enconding, even with the use of a very simple sequential split method. This confirms that the association of optimal split with MA is very successful for TRSP, as shown in the literature on several variants of VRP. We are currently working on further enhancing the GA-S2 schema (the GA based on optimal split and local search), by integrating several neighborhoods in the local search, as well as various diversification techniques. Our aim is to compare our method with literature approaches developed for problems similar to our industrial case, more specifically we want to obtain the instances of [3] to extend the comparative analysis.

References

1. F. AA. *Data mining and knowledge discovery with evolutionary algorithms*. Springer Science Business Media, 2013.
2. U. Aickelin and D. K. A. An indirect genetic algorithm for a nurse-scheduling problem. *Computers & operations research*, 31(5):761–778, 2004.
3. H. Algethami, R. L. Pinheiro, and D. Landa-Silva. A genetic algorithm for a workforce scheduling and routing problem. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 927–934. IEEE, 2016.
4. C. Archetti, M. Savelsbergh, and M. G. Speranza. The vehicle routing problem with occasional drivers. *European Journal of Operational Research*, 254(2):472–480, 2016.
5. H. Bouly, D. Dang, and A. Moukrim. A memetic algorithm for the team orienteering problem. *4OR*, 8(1):49–70, 2010.
6. O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.
7. J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 239:39–67, 2016.
8. R. B. Damm, M. G. Resende, and D. P. Ronconi. A biased random key genetic algorithm for the field technician scheduling problem. *Computers & Operations Research*, 75:49–63, 2016.
9. R. d. B. Damm and D. P. Ronconi. A multi-objective biased random-key genetic algorithm for service technician routing and scheduling problem. In *Computational Logistics: 12th International Conference, ICCL 2021, Enschede, The Netherlands, September 27–29, 2021, Proceedings*, pages 471–486. Springer, 2021.
10. A. Goel and F. Meisel. Workforce routing and scheduling for electricity network maintenance with downtime minimization. *European Journal of Operational Research*, 231(1):210–228, 2013.
11. G. Guastaroba, J.-F. Côté, and L. Coelho. The multi-period workforce scheduling and routing problem. *Omega*, 102:102302, 2021.
12. M. Günther. Application of particle swarm optimization to the british telecom workforce scheduling problem. 2013.
13. S. Jbili, A. Chelbi, M. Radhoui, and M. Kessentini. Integrated strategy of vehicle routing and maintenance. *Reliability Engineering & System Safety*, 170:202–214, 2018.
14. A. Khalfay, A. Crispin, and K. Crockett. A review of technician and task scheduling problems, datasets and solution approaches. In *2017 intelligent systems conference (IntelliSys)*, pages 288–296. IEEE, 2017.
15. A. A. Kovacs, S. N. Parragh, K. F. Doerner, and R. F. Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15:579–600, 2012.
16. P. Lacomme, C. Prins, and W. Ramdane-Cherif-Khettaf. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131:159–185, 2004.
17. I. Mathlouthi, M. Gendreau, and J.-Y. Potvin. A metaheuristic based on tabu search for solving a technician routing and scheduling problem. *Computers & Operations Research*, 125:105079, 2021.
18. E. Pekel. Solving technician routing and scheduling problem using improved particle swarm optimization. *Soft Computing*, 24(24):19007–19015, 2020.
19. E. Pekel. A simple solution to technician routing and scheduling problem using improved genetic algorithm. *Soft Computing*, 26(14):6739–6748, 2022.

20. D. L. Pereira, J. C. Alves, and M. C. de Oliveira Moreira. A multiperiod workforce scheduling and routing problem with dependent tasks. *Computers & Operations Research*, 118:104930, 2020.
21. R. L. Pinheiro, D. Landa-Silva, and J. Atkin. A variable neighbourhood search for the workforce scheduling and routing problem, 2015.
22. E. Pourjavad and E. Almehdawe. Optimization of the technician routing and scheduling problem for a telecommunication industry. *Annals of Operations Research*, 315(1):371–395, 2022.
23. C. Prins. Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*, 22(6):916–928, 2009.
24. C. Prins, P. Lacomme, and C. Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.
25. A. Srinivasan and R. Sangeetha. A review of static, dynamic and stochastic vehicle routing problems in home healthcare. *European Journal of Molecular & Clinical Medicine*, 7(3):5037–5046, 2020.

Hyperheuristic as Tuning Tool of Generalized Swap Strategy

Marek Kvet[1] and Jaroslav Janáček[1]

University of Žilina
, Faculty of Management Science and Informatics
Univerzitná 8215/1
010 26 Žilina
Slovakia
{marek.kvet, jaroslav.janacek}@fri.uniza.sk

Abstract. This paper introduces a hyperheuristic, which is supposed to be used as a tuning tool of generalized swap strategy suggested for Pareto front approximation. Pareto front as a small subset of all feasible solutions is searched for whenever the solved location problem combines two contradictory criteria. The quality of obtained results will be evaluated based on comparing the complete Pareto front to its approximation. The used problem instances correspond to real systems established in self-governing regions of Slovakia.

1 Introduction

Almost each researcher who deals with heuristic implementations has to face the problem of finding proper values of the heuristic parameters. The process of heuristic tuning is said to demand for the same time as the heuristic development itself [7]. In this contribution, we focus on an elementary neighborhood search heuristic used as a substantial part of sophisticated methods for location problem solving. A solution of the location problem is given by a list of facility locations selected from a set of all possible facility locations [1, 2, 3, 5, 6, 8, 10, 17, 21]. A neighborhood search heuristic inspects a neighborhood of a given current solution, where the neighborhood of the current solution is defined as a set of all solutions, which can be obtained by performing exactly one of the permitted operations with the current solution. The neighborhood inspection terminates either by move to a new current solution or with declaration that the process is finished. The neighborhood inspection usually follows one of the two strategies. The first admissible strategy moves to the first found solution, which is better than the current one. The best admissible strategy scans the entire neighborhood and moves to the best admissible solution found, if one exists.

In [12, 14, 15, 16], there was presented the generalized strategy with two parameters, which enable to cover a spectrum of strategies including the first and best admissible ones. In this contribution, we deal with a hyperheuristic, which minimizes an area under step-wise function determined by a set of non-dominated solutions of a location problem with two conflicting criteria. The hyperheuristic disposes with a set of subordinate routines. These routines are selected individually and applied to solve a sub-problem, and each routine's score is updated based on its performance. The higher the score of the routine, the more often it is chosen.

In the presented study, we use the suggested hyperheuristic for identification of the best setting of the parameters of the generalized neighborhood search algorithm. The associated computational study solves the problem of obtaining a good approximation of the Pareto front members, which are public service system designs evaluated according to two conflicting criteria.

2 Problem of non-dominated solution set optimization

Let $\boldsymbol{Y}$ be a finite set of all problem solutions and let f_1 and f_2 be objective functions, which describe two conflicting objectives to be minimized.

We say that solution $\boldsymbol{y} \in \boldsymbol{Y}$ dominates solution $\boldsymbol{x} \in \boldsymbol{Y}$ if $f_1(\boldsymbol{y}) \leq f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{y}) \leq f_2(\boldsymbol{x})$ hold. A subset $PF \subset \boldsymbol{Y}$ of $noPF$ mutually non-dominated solutions $\boldsymbol{p^1},\ldots,\boldsymbol{p^{noPF}}$ is called Pareto front, if each element of $\boldsymbol{Y}$ is dominated by at least one solution of PF. A subset $APF \subset \boldsymbol{Y}$ of $noAPF$ mutually non-dominated solutions $\boldsymbol{a^1},\ldots,\boldsymbol{a^{noAPF}}$ is denoted as a relevant approximation of the Pareto front if $\boldsymbol{a}^1 = \boldsymbol{p}^1$

and $\boldsymbol{a}^{noAPF} = \boldsymbol{p}^{noPF}$. In the further text, we assume that the elements of APF and PF are ordered increasingly according to the values $f_2(\boldsymbol{a}^k)$.

Coming out of a given approximation APF and objective functions f_1 and f_2 a function $q(t)$ can be defined on the interval $[f_2(\boldsymbol{a}^1), f_2(\boldsymbol{a}^{noAPF})]$ by the following prescription: $q(t) = f_1(\boldsymbol{a}^k) – f_1(\boldsymbol{a}^{noAPF})$ for $t \in [f_2(\boldsymbol{a}^k), f_2(\boldsymbol{a}^{k-1})]$ and $k = 1, \ldots, noAPF–1$.

Integral of the function over the interval $[f_2(\boldsymbol{a}^1), f_2(\boldsymbol{a}^{noAPF})]$ will be denoted by $Q(APF)$ and it can be computed according to (**??**). It corresponds to the hatched area in Fig. 1.

$$Q(APF) = \int_{f_2(\boldsymbol{a}^1)}^{f_2(\boldsymbol{a}^{NOAPF})} q(t)dt = \sum_{k=1}^{NOAPF-1} \left(f_1\left(\boldsymbol{a}^{\boldsymbol{k}}\right) - f_1\left(\boldsymbol{a}^{\boldsymbol{noAPF}}\right)\right)\left(f_2\left(\boldsymbol{a}^{\boldsymbol{k+1}}\right) - f_2\left(\boldsymbol{a}^{\boldsymbol{k}}\right)\right) \quad (1)$$

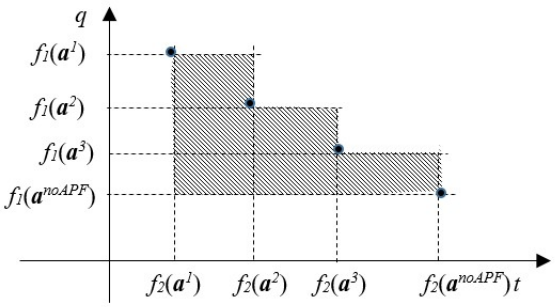


Fig. 1. Pareto front and its area

The problem of finding such APF which minimizes $Q(APF)$ will be called APF optimization problem. It is obvious that Pareto front is the optimal solution of the problem. Nevertheless, determination of the Pareto front asks for usage of exact tools of the mathematical programming, what is connected with enormous demand for computational time. That is why we accept a heuristic approach to the APF optimization under assumption that the approximate solution will be obtained in acceptable time and the difference $Q(APF)–Q(PF)$ will be sufficiently small.

The suggesting ordering of APF solutions enables fast decision on a candidate $\boldsymbol{x} \in \boldsymbol{Y}$ whether it can or cannot decrease the current $Q(APF)$. Thus a procedure $Improvement(APF,\boldsymbol{x})$ can be easily implemented. The procedure returns a positive value of $Q(APF)$ decrement if the candidate $\boldsymbol{x}$ is not dominated by any solution of the current APF and it returns the value of zero in the opposite case. The procedure also updates APF.

3 Hyperheuristic and subordinate routines

The presented neighborhood search algorithm with generalized strategy inspects neighborhood $N(\boldsymbol{y})$ of a current solution $\boldsymbol{y}$ consists of all feasible solutions, which can be obtained by performing one swap operation with the current solution $\boldsymbol{y}$. The swap operation performed with a solution of the location problem given by a set of selected locations replaces one of the selected locations by one location, which does not belong to the selected ones.

The algorithm processes the inputs, which consists of an initial solution $\boldsymbol{y}$ and an initial APF, which is improved during the algorithm run by the procedure $Improvement(APF,\boldsymbol{x})$. The next inputs are parameters $Threshold$ and $MaxNo$, where the value of $Threshold$ represents a limit, which must be exceeded by APF improvement to be considered admissible. The integer $MaxNo$ gives the number of step-by-step found admissible improvements, from which the best one is selected.

The algorithm description contains the following procedures.

Procedure $Improvement(APF,\boldsymbol{x})$ returns the value of $Q(APF)$ decrement and updates APF.

Function $getSol(NN(\boldsymbol{x}))$ withdraws one solution from the set $NN(\boldsymbol{y})$ of solutions neighboring the solution $\boldsymbol{y}$.

$GeneralizedNS(\boldsymbol{y}, APF, Threshold, MaxNo)$

1. Construct $NN(\boldsymbol{y})$ as a set of all non-fathomed neighboring solutions of $\boldsymbol{y}$. Set $NoAdm = 0$ and $BestImp = 0$.
2. If $NoAdm = MaxNo$ or $NN(\boldsymbol{y})$ is empty, then continue with 4, otherwise determine $\boldsymbol{x} = geSol(NN(\boldsymbol{y}))$, set $Imp = Improvement(APF, \boldsymbol{x})$ and continue with 3.
3. If $Imp > Threshold$, then perform: $NoAdm = NoAdm + 1$ and if $Imp > BestImp$, then $BestImp = Imp$ and $\boldsymbol{x}^{\boldsymbol{best}} = \boldsymbol{x}$. Continue with 2.
4. If $Threshold < BestImp$, then redefine $\boldsymbol{y} = \boldsymbol{x}^{\boldsymbol{best}}$ and go to 1, otherwise terminate.

Performance of algorithm *GeneralizedNS* depends on parameters $Threshold$ and $MaxNo$. If parameter $Threshold$ is set at the value of zero and parameter $MaxNo$ equals to one, the algorithm will perform according to the strategy "first admissible". If parameter $MaxNo$ will be near to cardinality of the neighborhood, then strategy "the best admissible" will be applied. Each setting of the pair of parameters causes different performance of the neighborhood search algorithm.

The proper setting the algorithm parameters is matter of this paper. This fact found by preliminary experiments inspired the idea of using a selective hyperheuristic as a teaching tool for this purpose.

The suggested selective hyperheuristic disposes with a set R of routines. A routine $r \in R$ has its own score denoted $Score(r)$. The score is initialized by a small positive value at the beginning of the hyperheuristic and also $InitialQ$ is determined as $Q(APF_0)$, where APF_0 consists only of two members $\boldsymbol{p}^1$ and $\boldsymbol{p}^{\boldsymbol{noPF}}$. The algorithm *GeneralizedNS* with setting $Threshold(r)$ and $MaxNo(r)$ represents the routine r. Let $\boldsymbol{y}$ be an input solution and $r(\boldsymbol{y})$ be the resulting solution of the routine applied to $\boldsymbol{y}$. Let $f(r(\boldsymbol{y}))$ denote the difference of $O(APF)$ before and after performance of the routine r. The following steps can briefly describe the core of the suggested selective hyperheuristic.

1. Transform *Score* values to probabilities assigned to the individual routines according to (2). Compute cumulated probabilities using the formula (3). Then, generate a random number rn from the interval [0, 1] and determine minimal r_0 that $rn \leq CumProb(r_0)$.
2. Determine $Q_0 = Q(APF)$. Perform $GeneralizedNS(\boldsymbol{y}, APF, Threshold(r_0), MaxNo(r_0))$ and compute $Q_1 = Q(APF)$.
3. Update $Score(r-0)4accordingtoScore(r_0) = Score(r_0) + (Q_0 – Q_1)/InitialQ$.

$$Probability(r) = \frac{Score(r)}{\sum_{k \in R} Score(k)} \tag{2}$$

$$CumProb(r) = \sum_{k=1}^{r} Probability(k) \tag{3}$$

The above hyperheuristic includes a state of learning represented by the structure *Score*. This suggested hyperheuristic core has to be embedded into the process described in the next section and, in addition, it must enable transition of experience quantified by *Score* to the next runs.

4 Process of gradual refinement

The construction of a Pareto front or at least of its good approximation may be performed in many different ways, which can be based also on the decrementing neighborhood search algorithm. One of possible strategies applicable in the *NDSS* creation process is the schema of its gradual refinement [15, 19, 20]. The process starts with two-element initial *NDSS* consisting of the most left and the most right bordering solutions of the Pareto front. These bordering solutions can be computed easily and their obtaining does not usually take too long time. Mentioned refinement process is performed repeatedly. It means that processing of one round brings such a *NDSS* set, which can be used as in input set for the following inspection process. Since the embedded decrementing algorithm may use random activities, the results of a process executed multiple times may vary. Therefore, the inner cycle is nested into time-controlled cycle, which repeats the inner cycle up to the moment, when the given time limit elapses. In other words, the refinement of *NDSS* repeats until given times restriction stops the algorithm performance.

The gradual refinement procedure processes the input *NDSS* solution by solution following the mentioned order $\boldsymbol{y^1}, \ldots, \boldsymbol{y^{noNDSS}}$ of *NDSS* solutions. If $\boldsymbol{y^k}$ is processed, *DecrementingNeighborhoodSearch* algorithm is applied in the studied case. As *NDSS* can change during one run of the algorithm, the solution corresponding with the k-th position may also change. If it happens, the algorithm is applied once again to this new solution $\boldsymbol{y^k}$, otherwise the following solution $\boldsymbol{y^k} + 1$ is processed. If $k = noNDSS - 1$, the basic refinement process terminates.

5 Computational study

5.1 Used tools and benchmarks

This section is devoted to a little computational study, in which real-world data were used. The main goal of the numerical experiments is to study the accuracy of suggested heuristics for Pareto front approximation. First, let us specify software and hardware tools and used dataset.

As the main software tool, the NetBeans IDE 8.2 was applied. This common development kit enables to write algorithms and application in Java language. As far as hardware is concerned, a common PC with the Intel® Core™ i7 4790 CPU@3.60 GHz processor and 8 GB RAM was used.

The dataset of problem instances was taken from our previous research, the results of which can be found except many other papers in [9, 11, 12, 13, 14, 15, 16, 19, 20]. In all of them, two criteria are denoted as so-called system and fair objectives respectively. While the system criterion f_1 minimizes the distance from an average user to the nearest available source of service, the fair objective f_2 minimizes the number of those, whose distance from the nearest located facility exceeds given limit. More details about the concrete forms of used optimization criteria can be found in the list of referenced papers, mainly in [4, 11, 13, 18]. Since the same dataset was used by our colleagues to verify the exact approach for Pareto front determination published in [9, 11], the quality of suggested heuristic approaches to Pareto front approximation can be easily evaluated.

Table 1 summarizes the basic benchmarks properties. For each region denoted by its abbreviation, we report the cardinality of the possible service center candidates set denoted by $|I|$ and the number p of facilities to be located. The right part of the table brings the cardinality of the complete Pareto fronts in the column denoted by $|PF|$ and the corresponding value of $Q(PF)$.

Table 1. Benchmarks sizes and the exact Pareto fronts characteristics

Region	$\|I\|$	p	$\|PF\|$	$Q(PF)$
BA	87	14	34	569039
BB	515	36	229	1002681
KE	460	32	262	1295594
NR	350	27	106	736846
PO	664	32	271	956103
TN	276	21	98	829155
TT	249	18	64	814351
ZA	315	29	97	407293

5.2 Numerical experiments

This subsection is used to present the results of numerical experiments, which were aimed at studying the accuracy of Pareto front approximation by the suggested hyperheuristic. It must be noted that the mentioned hyperheuristic covers several subroutines, which are chosen by their score and probability distribution. At the beginning, the score of all subroutines was set to one. It means, that the probability of each subroutine is the same and it is distributed uniformly among all strategies. The obtained results are summarized in Table 2, which takes the following structure. Each row of the table corresponds to one solved instance corresponding to real Emergency Medical Service system in a given region. The regions are identified by the abbreviation of their name. Such a notation was used also in our previous case studies. Despite the fact that the computational process of suggested hyperheuristic was restricted to five minutes, we report also the computational time in seconds in the column denoted by CT. The

column denoted by $|APF|$ brings the information about the cardinality of the resulting APF set of non-dominated system designs. In other words, it is the number of found solutions approximating the original Pareto front. The last two columns are the most interesting from the viewpoint of the results accuracy. $Q(APF)$ represents the absolute size of the polygon formed by the members of APF. For more details, see Fig. 1. Since the absolute and big values are not suitable for comfortable comparison, the quality of obtained results is demonstrated also by the value of so-called *gap*. *Gap* is defined in percentage and it can be evaluated by the expression (4).

$$gap = 100 * \frac{Q(APF) - Q(PF)}{Q(PF)} \tag{4}$$

Since the suggested hyperheuristics is based on a selection of a partial subroutine to be applied, randomness and probability distribution play an important role. That is why we run the solving algorithm ten times for each problem instance. Then, the following Table 2 contains the average values of all studied characteristics.

Table 2. Results of numerical experiments

Region	CT [s]	$\|APF\|$	$Q(APF)$	*Gap* [%]
BA	300.04	33.00	577393.00	1.47
BB	317.98	209.20	1012436.80	0.97
KE	316.95	247.80	1332685.20	2.86
NR	304.26	100.90	743297.10	0.88
PO	335.48	269.00	983641.30	2.88
TN	301.46	92.00	841167.40	1.45
TT	300.76	62.00	817379.50	0.37
ZA	302.49	93.80	407981.40	0.17

Presented results of numerical experiments have proved the suitability of the hyperheuristic for practical usage. As we can see in the last column of Table 2, the average values of gap do not reach high values, on the contrary, they indicate, that the approximation of the complete Pareto front is very good. From the practical point of view, the hyperheurisitc is able to bring a satisfactorily accurate result in five minutes.

When studying particular subroutines and their probability distribution, one can not choose the best one, because all of them have showed usefulness. On the other hand, presented method deserves future deep analysis to accelerate the computational process or to improve the algorithm to bring better results.

6 Conclusions

Location science covers many subfields, in which plenty of different combinatorial problems are solved by means of mathematical programming. This contribution was aimed at a specific class of discrete network location problems with two conflicting criteria. When there are more than one objective to be optimized, a Pareto front seems to be a sufficient output for public authorities to make the final decision.

Obtaining the complete Pareto front proved to be a time consuming challenge and that is why many scientists and other experts in computation or algorithms development pay attention to searching for effective heuristic methods.

In this paper, we focused on a hyperheuristic, which is supposed to be used as a tuning tool of generalized swap strategy suggested for Pareto front approximation. Presented results of performed numerical experiments have shown that the hyperheuristic is able to approximate the original Pareto front in very satisfactory way, which means that the average gap did not exceed one percent in most cases.

Development in Applied Informatics and Operations research is a never-ending story. Achieving one research goal usually brings new challenges. Therefore, future research direction in the field of bi-criteria location science could be aimed at the development of various algorithms for Pareto front approximation, which could be based on machine learning and artificial intelligence.

Acknowledgements

This work was supported by the research grants VEGA 1/0216/21 "Design of emergency systems with conflicting criteria using artificial intelligence tools", VEGA 1/0077/22 "Innovative prediction methods for optimization of public service systems" and VEGA 1/0654/22 "Cost-effective design of combined charging infrastructure and efficient operation of electric vehicles in public transport in sustainable cities and regions". This work was also supported by the Slovak Research and Development Agency under the Contract no. APVV-19-0441.

References

1. Ahmadi-Javid, A., Seyedi, P. et al. (2017). A survey of healthcare facility location, Computers & Operations Research, 79, pp. 223-263.
2. Avella, P., Sassano, A., Vasil'ev, I. (2007). Computational study of large scale p-median problems. Mathematical Programming 109, pp. 89-114.
3. Brotcorne, L, Laporte, G, Semet, F. (2003). Ambulance location and relocation models. Eur. Journal of Oper.Research, 147, pp. 451-463.
4. Buzna, Ľ., Koháni, M., Janáček, J. (2013). Proportionally Fairer Public Service Systems Design. In: Communications - Scientific Letters of the University of Žilina 15(1), pp. 14-18.
5. Current, J., Daskin, M. and Schilling, D. (2002). Discrete network location models, Drezner Z. et al. (ed) Facility location: Applications and theory, Springer, pp. 81-118.
6. Doerner, K. F., Gutjahr, W. J., Hartl, R. F., Karall, M. and Reimann, M. (2005). Heuristic Solution of an Extended Double-Coverage Ambulance Location Problem for Austria. Central European Journal of Operations Research, 13(4), pp. 325-340.
7. Gendreau, M. and Potvin, J. (2010). Handbook of Metaheuristics, Springer Science & Business Media.
8. Gopal, G. (2013). Hybridization in Genetic Algorithms. International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, pp. 403–409.
9. Grygar, D., Fabricius, R. (2019). An efficient adjustment of genetic algorithm for Pareto front determination. In: TRANSCOM 2019: conference proceedings, Amsterdam: Elsevier Science, pp. 1335-1342.
10. Ingolfsson, A., Budge, S., Erkut, E. (2008). Optimal ambulance location with random delays and travel times. Health care management science, 11(3), pp. 262-274.
11. Janáček, J., Fabricius, R. (2021). Public service system design with conflicting criteria. In: IEEE Access: practical innovations, open solutions, ISSN 2169-3536, Vol. 9, pp. 130665-130679.
12. Janáček, J., Kvet, M. (2021). Swap Heuristics for Emergency System Design with Multiple Facility Location. In: Proceedings of the 39th International Conference on Mathematical Methods in Economics, 2021, pp. 226-231.
13. Janáček, J., Kvet, M. (2021). Emergency Medical System under Conflicting Criteria. In: SOR 2021 Proceedings, pp. 629-635.
14. Janáček, J., Kvet, M. (2022). Adaptive swap algorithm for Pareto front approximation. In: ICCC 2022: 23rd International Carpathian Control conference, Sinaia, Romania, Danvers: IEEE, 2022, pp. 261-265.
15. Janáček, J., Kvet, M.. (2022). Repeated Refinement Approach to Bi-objective p-Location Problems. In: INES 2022: Proccedings of the IEEE 26th International Conference on Intelligent Engineering Systems 2022, pp. 41-45.
16. Janáček, J., Kvet, M. (2022). Pareto Front Approximation using Restricted Neighborhood Search. In: Proceedings of the 40th International Conference on Mathematical Methods in Economics, 2022, Jihlava, pp. 141-147.
17. Jánošíková, Ľ. and Žarnay, M. (2014). Location of emergency stations as the capacitated p-median problem. In: Quantitative Methods in Economics (Multiple Criteria Decision Making XVII). pp. 117-123.
18. Kvet, M. (2014). Computational Study of Radial Approach to Public Service System Design with Generalized Utility. In: Proceedings of International Conference DT 2014, Žilina, Slovakia, pp. 198-208.
19. Kvet, M., Janáček, J. (2021). Incrementing Heuristic for Non-Dominated Designs of Emergency Medical System. In: SOR 2021 Proceedings, pp. 429-474.
20. Kvet, M., Janáček, J. (2022). Directed Search for Pareto Front Approximation with Path-relinking Method. In: Proceedings of the 40th International Conference on Mathematical Methods in Economics, 2022, Jihlava, pp. 212-217.
21. Marianov, V. and Serra, D. (2002). Location problems in the public sector, Facility location - Applications and theory (Z. Drezner ed.), Berlin, Springer, pp 119-150.

Artificial Intelligence for Metaheuristic Parameter Setting

Jaroslav Janáček[1] and Marek Kvet[1]

University of Žilina
, Faculty of Management Science and Informatics
Univerzitná 8215/1
010 26 Žilina
Slovakia
{jaroslav.janacek, marek.kvet}@fri.uniza.sk

Abstract. The scientific content of this paper focuses on service system optimization. The main goal of presented research consists in extending current portfolio of solving approaches for such discrete location problems, in which two contradictory objectives are to be optimized. Due to different optimization criteria, a small subset of feasible solutions respecting a specific non-dominance property needs to be searched for. Obtaining the complete Pareto front is a time-consuming challenge. This phenomenon has led to the development of various metaheuristic approaches, which are able to bring an approximation of the original Pareto front. Mentioned approximate approaches may be sensitive to different parameters. That is why we pay attention to artificial intelligence for metaheuristic parameter settings. Suggested algorithm was tested on middle-sized benchmarks derived from real world. The obtained Pareto front approximations are compared to the complete sets of non-dominated solutions, which are available.

1 Introduction

Metaheuristics are often used to solve complex optimization problems, where application of exact methods of mathematical programming is either impossible or asks for inacceptable amount of computational time. If exactly optimal results are not required, the strong side of metaheuristic approaches is presented by flexibility concerning model of the solved problems. Metaheuristic approaches can accept non-linear models and also absence of function continuity or existence of derivations in model formulae. Besides of impossibility to reach exactly optimal solution, the weak side of a bit more complex metaheuristic is dependence of their efficiency on metaheuristic parameter setting. The process of finding suitable values of metaheuristic parameters is generally called tuning of the metaheuristic and it is generally assumed that it asks approximately for the same time as the metaheuristic construction [8, 9]. The search for a good approximation of Pareto front of public service system designs represents a very computational time demanding task as shown in [10, 12, 14].

The public service system design problem [1, 2, 3, 5, 6, 7, 11, 19, 25] is a typical problem, quality of which is often assessed from several points of view. The often used couple of conflicting criteria are so called system and fair criteria. The first criterion is formulated as an average response time of the system to a user's demand for service and the second one reflects minority calls for fair access to service. Due to incomparability of the two criteria, it is impossible to transform the pair of criteria to an optimization problem with one objective function and thus a meaningful result of the problem solving can be Pareto front or a good approximation of it. Such a result can help to a system administrator to finish a negotiation with public representatives and determine a final design of the implemented public service system.

Within this paper, we focus our efforts on a special metaheuristic minimizing a difference between the exact Pareto front and a Pareto front approximation. Instead of tuning process of the metaheuristic parameters, we develop and test an adaptive process of parameter setting and implement it as a metaheuristic enlargement.

The remainder of this paper is organized into the following five sections. Section 2 aims at approximation of the Pareto front of public service system designs, where two conflicting criteria are taken into account. Section 3 focuses on a method of improving Pareto front approximation quality. In Section 4, we concentrate on description of adaptive algorithm for setting parameter of metaheuristic optimizing the Pareto front approximation. The findings from actual numerical experiments are presented in Section 5. The conclusions of the work, which are described in Section 6, contain the acquired results and recommendations for further study.

2 Quality measure of Pareto front approximation

We consider a mathematical programming problem with two objectives f_1 and f_2 to be minimized. The Pareto front of the problem solutions is a set of mutually non-dominated solutions, which satisfy the cause that if an arbitrary feasible solution $\boldsymbol{x}$ is chosen, then at least one element $\boldsymbol{y}$ of the Pareto front exists that inequalities $f_1(\boldsymbol{y}) \leq f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{y}) \leq f_2(\boldsymbol{x})$ hold. In other words, we say that $\boldsymbol{y}$ dominates $\boldsymbol{x}$.

If he set $\boldsymbol{Y}$ of all feasible solutions of the considered problem is finite, then the Pareto front must be also finite and it contains one element with minimal value of the objective function f_2 and the element, which minimizes the objective function f_1. These two elements of the Pareto front can be called the most left and most right solutions respectively.

As the Pareto front determination is a hard computational problem, we concentrate our effort on establishing a good approximation of it. The approximating set of non-dominated solutions (*NDSS*) will be represented by a sequence of *noNDSS* solutions $\boldsymbol{y^1},\ldots,\boldsymbol{y^{noNDSS}}$ ordered according to increasing values of f_2. To obtain a relevant approximation, the bordering solutions $\boldsymbol{y^1}$ and $\boldsymbol{y^{noNDSS}}$ must be determined to be very close to the most left and most right solutions of the Pareto front as concerns the values of f_1 and f_2. Under these assumptions, the quality of the approximation *NDSS* can be measured by so called *NDSS_Area* computed according to the expression (1).

$$\sum_{k=1}^{noNDSS-1} \left(f_1\left(\boldsymbol{y^k}\right) - f_1\left(\boldsymbol{y^{noNDSS}}\right)\right)\left(f_2\left(\boldsymbol{y^{k+1}}\right) - f_2\left(\boldsymbol{y^k}\right)\right) \tag{1}$$

The identical formula can be used to compute the area of the Pareto front (*PF_Area*) and it holds that the *PF_Area* is a lower bound of *NDSS_Area*.

Considering the order of *NDSS* solutions, it can be easily implemented an algorithm, which decides in *noNDSS* steps whether arbitrary feasible solution $\boldsymbol{x}$ is dominated by an element of the current *NDSS* or if it can be included into the *NDSS* improving the associated *NDSS_Area*. Such algorithm can be used for step-by-step updating of an initial *NDSS* whenever a source of candidate solutions is at disposal.

3 Decrementing algorithm for NDSS_Area minimization

The suggested decrementing algorithm is based on a neighborhood search, when the inspected neighborhood $N(\boldsymbol{y})$ of a current solution $\boldsymbol{y}$ consists of all feasible solutions which can be obtained by performing a permitted operation with the current solution $\boldsymbol{y}$.

The further described decrementing algorithm uses several auxiliary functions and procedures described below.

Function $getArea(NDSS)$ returns the area of the input *NDSS* computed according to (1).

Procedure $Update(NDSS,\boldsymbol{x})$ decides if the solution $\boldsymbol{x}$ will or will not improve the current *NDSS* and, in the positive case, the *NDSS* is updated.

Function $WithdrawFrom(NN(\boldsymbol{y}))$ chooses a solution of the input set $NN(\boldsymbol{y})$ of non-fathomed solutions and withdraws the chosen solution from $NN(\boldsymbol{y})$.

$DecrementingNeighborhoodSearch(\boldsymbol{y},NDSS,MinDec,T)$

1. Construct $NN(\boldsymbol{y})$ as a set of all non-fathomed solutions of $N(\boldsymbol{y})$.
2. If $NN(\boldsymbol{y})$ is empty, then terminate, otherwise determine $\boldsymbol{x} = WithdrawFrom(NN(\boldsymbol{y}))$, set $Area0 = getArea(NDSS)$ and continue with 3.
3. Perform $Update(NDSS,\boldsymbol{x})$. If the update is not successful, go to step 2. In the opposite case define $Area1 = getArea(NDSS)$ and continue with 4.
4. Generate a random number *rn* from the interval [0, 1] and if $rn \leq exp((Area0–Area1–MinDec)/T)$, then redefine $\boldsymbol{y} = \boldsymbol{x}$ and go to 1, otherwise go to 2.

The above algorithm starts from an initial solution $\boldsymbol{y}$ and an initial *NDSS*, which is step-by-step updated during the run of the algorithm. This way the associated *NDSS_Area* is gradually minimized. Input parameters are a threshold *MinDec* and parameter T known as a temperature in simulated annealing metaheuristics.

4 Adaptive algorithm for Pareto front approximation

The decrementing neighborhood search algorithm can be easily embedded into a more complex schema of *NDSS* improving. One of such schemes is known as the schema of gradual refinement [15, 16, 17, 23, 24]. The process which follows the schema starts with the two-element initial *NDSS* consisting of the most left and most right bordering solutions of the Pareto front. The stepwise refinement process itself is repeated so that the *NDSS* that is the result of the performance of one process is used as the input *NDSS* for the next application of the process. Because the embedded decrementing algorithm may use random activities, the results of a process executed multiple times may vary. Thus, the inner cycle is nested into time-controlled cycle, which repeats the inner cycle up to the moment, when the given time of the process elapses.

The gradual refinement procedure processes the input *NDSS* solution by solution following the mentioned order $\boldsymbol{y^1},\ldots,\boldsymbol{y^{noNDSS}}$. If $\boldsymbol{y^k}$ is processed, *DecrementingNeighborhoodSearch* algorithm is applied in the studied case. As *NDSS* can change during one run of the algorithm, the solution corresponding with the k-th position may also change. If it happens, the algorithm is applied once more to this new $\boldsymbol{y^k}$, otherwise solution $\boldsymbol{y^{k+1}}$ is processed. If $k = noNDSS - 1$, the basic refinement process terminates.

The algorithm *DecrementingNeighborhoodSearch* depends on parameters *MinDec* and T. According to similar processes, the parameter T is periodically changed (heated or cooled) depending on the number of steps taken. The proper setting of the threshold is matter of our study. The best setting for any solved problem instance can be hardly found. This fact following from the preliminary experiments evokes the further presented idea of enlarging the repeated gradual refinement process by an adapting process, which will be setting the value of *MinDec* dynamically with respect to experience obtained during the previous runs of *DecrementingNeighborhoodSearch*.

Mentioned experience of increase or decrease of the parameter *MinDec* will be quantified by the parameter *State*, which will be initialized by zero value before the start of the time-controlled cycle. Based on the value of the state, probability P of increasing *MinDec* by the increment *Delt* is determined.

At the beginning of the process, the initial value of probability P is set at 0.5. Before each run of *DecrementingNeighborhoodSearch* algorithm a random trial in favor of *MinDec* increase is performed with probability P. If the trial succeeds, then *MinDec* is increased by *Delt*, else *MinDec* is decreased by the same value. The value of *MinDec* before the change is saved as *MinDec*0. The value of the difference between *NDSS_Areas* before and after the run of *DecrementingNeighborhoodSearch* is computed and denoted by *DifArea*. The value of *State* is updated according to (2).

$$State = \alpha State + \beta sign\left((DifArea - MinDec0)\,(MinDec - MinDec0)\right) \tag{2}$$

The probability value P is updated according to (3).

$$\begin{aligned} IfState < -1 thenP = 0 \\ IfState > 1 thenP = 1 \\ otherwiseP = (1 + State)/2 \end{aligned} \tag{3}$$

5 Numerical experiments

5.1 Description of the solved problem

The discrete location problem, which we are interested in, follows from the weighted p-median formulation. To describe the problem by means of linear mathematical programming, let I represent the finite set of candidates for locating a facility or any other source, from which the associated service could be provided. The optimization problem consists in searching for the optimal selection I_1 of exactly p elements from I so that the given objective $f(\boldsymbol{y})$ takes the best possible value. The decision about locating a service center at any location i from I is modelled by a binary variable y_i. In case that the location i is selected for establishing a service center, the variable takes the value of one. Otherwise, it equals zero. The general formulation of the problem may take the form of (4).

$$min\,\{f(\boldsymbol{y}) : I_1 \subset I, |I_1| = p\} \tag{4}$$

The set I_1 may be implemented as a list of indexes corresponding to the original set I or it can be described by a vector $\boldsymbol{y}$ of location variables y_i for each i from I.

Complexity and solvability of the problem depends on the accuracy, with which the objective function $f(\boldsymbol{y})$ of any solution $\boldsymbol{y}$ describes the relations in real systems. In this paper, we assume that the modelled system yields emergency service to all inhabitants of a served region and these inhabitants live in communities, which form the set J. Obviously, the set J can equal the previously introduced set I. Each element j of J is connected with a weight coefficient b_j. This coefficient may take several different meanings, i.e., the number of inhabitants, expected frequency of randomly occurring demands, etc. The response time for satisfying the demand at the point j is computed as an expected traversing time from the assigned center to the location j. As the nearest service center may be currently unavailable due earlier assigned demands, the probabilities $q_1, \ldots, q_r$ are introduced to describe the situations that the nearest service center is the closest available one (q_1) or that the second nearest center is the closest available one (q_2) and so on up to the r-th situation [18, 20, 21]. If t_{ij} denotes the traversing time from a candidate location i to a community location j and if the result of the operator $min_k\{t_{ij} : i \in I_1\}$ returns the k-th minimal value of t_{ij} for $i \in I_1$, then the average response time is proportional to the value of the expression (5) for the set I_1 of chosen service center locations.

$$f_1(I_1) = \sum_{j \in J} b_j \sum_{k=1}^{r} q_k min_k \left\{ t_{ij} : i \in I_1 \right\} \tag{5}$$

Presented function $f_1(\boldsymbol{y})$ is often denoted as so-called system criterion of the design.

When designing a service system for a longer period and when the strategic decisions about new center locations are to be made, then not only the system criterion play a role. Except minimization of the average response time, fairness should be also taken into account.

Fair criteria evaluate the disutility perceived by the worst placed minority of the population provided with the associated service [4, 22]. Within this study, the fair criterion will be computed as the number of clients' demands, for which the response time from the nearest service center is higher than a given threshold T_{max}. The associated objective function $f_2(I_1)$ can be expressed by (6).

$$f_2(I_1) = \sum_{j \in J} b_j max\left\{0, sign\left(min\left\{t_{ij} : i \in I_1\right\} - T_{max}\right)\right\} \tag{6}$$

Mentioned criteria f1 and f2 are in conflict as discussed in [10, 12, 14]. Therefore, a Pareto front of solutions needs to be produced instead on one resulting system design.

5.2 Benchmarks and solving tools

As far as the technical support like hardware and software tools are concerned, we used the programming language Java within the NetBeans IDE 8.2 environment. The experiments were run on a common PC equipped with the Intel® Core™ i7 11700KF CPU@3.60 GHz processor and 16 GB RAM.

The dataset of problem instances was taken from our previous research, the results of which are available in [13, 14, 15, 16, 17, 23, 24]. In the used dataset, the road network of Slovak self-governing regions was taken as a source of input data for the studied mathematical models. It must be noted that all network nodes represent both the set of candidates for service center locating and the set of inhabitants being provided with service. As the objective function f_1 follows from the concept of so-called generalized disutility, the parameter r was set to 3. The coefficients q_k were set so that $q_1 = 77.063$, $q_2 = 16.476$ and $q_3 = 100 - q_1 - q_2$. These values were obtained from a simulation model of the Emergency Medical Service system in Slovakia [18]. Parameter T_{max} used in the fair objective function described by the formula (6) was set to the value of 10 minutes [15, 16, 17, 23, 24].

The properties of the benchmarks are outlined in Table 1 below. The problem size for each region is determined by the cardinality of the set of potential service center locations denoted by the symbol $|I|$ and by the number of located centers denoted by p. The right part of the table contains the basic characteristics of the complete Pareto fronts. Symbol NoS denotes the number of non-dominated elements. The last column denoted by $Area$ contains the value of PF_Area.

5.3 Results of experiments

This subsection is devoted to the results of numerical experiments, in which the suggested heuristic approach combined with artificial intelligence will be studied.

Table 1. Benchmarks sizes and the exact Pareto fronts characteristics

Region	$\|I\|$	p	NoS	$Area$
BA	87	14	34	569039
BB	515	36	229	1002681
KE	460	32	262	1295594
NR	350	27	106	736846
PO	664	32	271	956103
TN	276	21	98	829155
TT	249	18	64	814351
ZA	315	29	97	407293

An individual experiment was performed in such a way that for each problem instance, ten runs of the algorithm were performed. The following tables contain the average results of ten algorithm runs. While Table 2 contains the first half of results, i.e. average values for the self-governing regions of Bratislava, Banská Bystrica, Košice and Nitra, Table 3 summarizes the results for the remaining regions, i.e., Prešov, Trenčín, Trnava and Žilina.

The structure of both tables takes the following form. Each column of the table corresponds to one benchmark denoted by its abbreviation. Each row of the table contains one studied characteristic. It must be noted that the algorithm was run for $base = 1$, which means no heating. The parameter α used in the expression (2) was set to the value of 0.8. Initial temperature T took the value of 1000. Obviously, we have run the algorithms also with different values of suggested parameters, but the obtained results did not differ significantly. That is why we report only the results for mentioned settings of the parameters. On the other hand, performance of the algorithm and the impact of different parameters on the result accuracy deserves future research and deep analysis.

Let us return now to the results reported in Table 2 and Table 3. The resulting characteristic interesting for future analysis are the following: The row denoted by *MinDecrem* contains the resulting value of minimal decrement, which was originally set to 0.1 percent of the *NDSS_Area*. The second line is dedicated to the resulting values of *state* used in the expression (3). Even if one algorithm run was limited to 5 minutes of computation, we report the computational time in seconds in the row denoted by *CT*. Symbol *noNDSS* denotes the number of non-dominated solution found to approximate the original Pareto front. It can be understood also as the cardinality of the resulting *NDSS*. The row denoted by *noTimeRuns* brings the number of algorithms runs within given time limit. Each run means processing the current set of found non-dominated solutions. The last row of each table represents the most important result. Instead of comparing absolute values of areas formed by the complete Pareto front and its approximation by *NDSS*, we evaluated the accuracy by so-called *gap*. This value expresses the relative difference in percentage, in which the area of the complete Pareto front is taken as the base. Its mathematical formulation takes the form of (7).

$$gap = 100 * \frac{NDSS\_Area - PF\_Area}{PF\_Area} \tag{7}$$

Table 2. Results of numerical experiments for Bratislava, Banská Bystrica, Košice and Nitra

	BA	BB	KE	NR
MinDecrem :	1.447	0.372	0.433	0.331
State :	0.059	-0.051	0.026	0.051
CT [s]:	300.047	312.857	304.589	302.067
noNDSS:	32	219	251	102
noTimeRuns:	2472.1	4	5	36.2
gap [%]:	4.10	0.65	2.78	6.07

6 Conclusions

This research paper was aimed at the development of such heuristic approach to Pareto front approximation, in which elements of artificial intelligence are incorporated. Pareto front approximation methods

Table 3. Results of numerical experiments for Prešov, Trenčín, Trnava and Žilina

	PO	TN	TT	ZA
MinDecrem :	0.259	1.089	0.935	0.511
State :	-0.068	-0.116	-0.05	0.052
CT [s]:	369.516	301.225	300.815	304.246
noNDSS :	264	84	62	91
noTimeRuns :	4	66.1	191.3	45.4
gap [%]:	1.23	0.85	0.65	0.22

are necessary whenever there are more contradictory objectives to be optimized at the same time. This way, we have tried to extend the state-of-the-art solving approaches to bi-criteria location problems.

Based on the achieved results of performed numerical experiments, we have found that the heuristic supported by artificial intelligence is able to bring a good approximation of the complete Pareto front in given time limit of five minutes. Even if the heating process was not activated, the obtained results are of a very satisfactory accuracy. The average gap was smaller than one percent in most cases, but there were some exceptions which should be analyzed more deeply to find possible ways for improvement of the associated solving algorithm. On the other hand, the main goal of this study has been fulfilled and the proposed heuristic can be easily applied to such location problems, in which two conflicting criteria need to be met simultaneously.

Future research in this location field could address the development of different advanced approaches suitable for bi- or even multi-criteria decision problems. Special attention should be paid to the deep analysis of the proposed method in order to study possible impact of various parameter settings on the quality of obtained results measured by their accuracy.

Acknowledgements

This work was supported by the research grants VEGA 1/0216/21 ”Design of emergency systems with conflicting criteria using artificial intelligence tools”, VEGA 1/0077/22 ”Innovative prediction methods for optimization of public service systems” and VEGA 1/0654/22 ”Cost-effective design of combined charging infrastructure and efficient operation of electric vehicles in public transport in sustainable cities and regions”. This work was also supported by the Slovak Research and Development Agency under the Contract no. APVV-19-0441.

References

1. Ahmadi-Javid, A., Seyedi, P. et al. (2017). A survey of healthcare facility location, Computers & Operations Research, 79, pp. 223-263.
2. Avella, P., Sassano, A., Vasil’ev, I. (2007). Computational study of large scale p-median problems. Mathematical Programming 109, pp. 89-114.
3. Brotcorne, L, Laporte, G, Semet, F. (2003). Ambulance location and relocation models. Eur. Journal of Oper.Research, 147, pp. 451-463.
4. Buzna, Ľ., Kohániˇ, M., Janáček, J. (2013). Proportionally Fairer Public Service Systems Design. In: Communications - Scientific Letters of the University of Žilina 15(1), pp. 14-18.
5. Current, J., Daskin, M. and Schilling, D. (2002). Discrete network location models, Drezner Z. et al. (ed) Facility location: Applications and theory, Springer, pp. 81-118.
6. Doerner, K. F., Gutjahr, W. J., Hartl, R. F., Karall, M. and Reimann, M. (2005). Heuristic Solution of an Extended Double-Coverage Ambulance Location Problem for Austria. Central European Journal of Operations Research, 13(4), pp. 325-340.
7. Drezner, T., Drezner, Z. (2007). The gravity p-median model. European Journal of Operational Research 179, pp. 1239-1251.
8. Gendreau, M. and Potvin, J. (2010). Handbook of Metaheuristics, Springer Science & Business Media.
9. Gopal, G. (2013). Hybridization in Genetic Algorithms. International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, pp. 403–409.
10. Grygar, D., Fabricius, R. (2019). An efficient adjustment of genetic algorithm for Pareto front determination. In: TRANSCOM 2019: conference proceedings, Amsterdam: Elsevier Science, pp. 1335-1342.

11. Ingolfsson, A., Budge, S., Erkut, E. (2008). Optimal ambulance location with random delays and travel times. Health care management science, 11(3), pp. 262-274.
12. Janáček, J., Fabricius, R. (2021). Public service system design with conflicting criteria. In: IEEE Access: practical innovations, open solutions, ISSN 2169-3536, Vol. 9, pp. 130665-130679.
13. Janáček, J., Kvet, M. (2021). Swap Heuristics for Emergency System Design with Multiple Facility Location. In: Proceedings of the 39th International Conference on Mathematical Methods in Economics, 2021, pp. 226-231.
14. Janáček, J., Kvet, M. (2021). Emergency Medical System under Conflicting Criteria. In: SOR 2021 Proceedings, pp. 629-635.
15. Janáček, J., Kvet, M. (2022). Adaptive swap algorithm for Pareto front approximation. In: ICCC 2022: 23rd International Carpathian Control conference, Sinaia, Romania, Danvers: IEEE, 2022, pp. 261-265.
16. Janáček, J., Kvet, M.. (2022). Repeated Refinement Approach to Bi-objective p-Location Problems. In: INES 2022: Proccedings of the IEEE 26th International Conference on Intelligent Engineering Systems 2022, pp. 41-45.
17. Janáček, J., Kvet, M. (2022). Pareto Front Approximation using Restricted Neighborhood Search. In: Proceedings of the 40th International Conference on Mathematical Methods in Economics, 2022, Jihlava, pp. 141-147.
18. Jankovič, P. (2016). Calculating Reduction Coefficients for Optimization of Emergency Service System Using Microscopic Simulation Model. In: 17th International Symposium on Computational Intelligence and Informatics, pp. 163-167.
19. Jánošíková, Ľ. and Žarnay, M. (2014). Location of emergency stations as the capacitated p-median problem. In: Quantitative Methods in Economics (Multiple Criteria Decision Making XVII). pp. 117-123.
20. Kvet, M. (2014). Computational Study of Radial Approach to Public Service System Design with Generalized Utility. In: Proceedings of International Conference DT 2014, Žilina, Slovakia, pp. 198-208.
21. Kvet, M. (2018). Advanced radial approach to resource location problems. In: Developments and advances in intelligent systems and applications. Cham: Springer International Publishing, 2018, Studies in computational intelligence, 718, pp. 29-48.
22. Kvet, M. (2021). Impact of Fairness Constraints on Average Service Accessibility in Emergency Medical System. In: Information and Digital Technologies 2021, pp. 11-18.
23. Kvet, M., Janáček, J. (2021). Incrementing Heuristic for Non-Dominated Designs of Emergency Medical System. In: SOR 2021 Proceedings, pp. 429-474.
24. Kvet, M., Janáček, J. (2022). Directed Search for Pareto Front Approximation with Path-relinking Method. In: Proceedings of the 40th International Conference on Mathematical Methods in Economics, 2022, Jihlava, pp. 212-217.
25. Marianov, V. and Serra, D. (2002). Location problems in the public sector, Facility location - Applications and theory (Z. Drezner ed.), Berlin, Springer, pp 119-150.

A Honey Bee Mating Optimization HyperHeuristic for Patient Admission Scheduling Problem

Imen Oueslati[1,2*], Moez Hammami[1], Issam Nouaouri[2], Ameni Azzouz[1], Lamjed Ben Said[1], and Hamid Allaoui[2]

[1] Université de Tunis, Institut Supérieur de Gestion de Tunis, Smart-Lab, Tunisia.
[2] Université d'Artois, Faculté des Sciences Appliquées, LGI2A, F-62400 Béthune, France
imen_oueslati@hotmail.com

Abstract

Hyperheuristics represent a generic method that provides a high-level of abstraction so as to be able to deal with several problem domains. This category of methods consists in managing a set of heuristics and tries to find the best sequence that gives good quality results. This paper proposes a hyperheuristic simulating the honey bees mating behavior to solve the Patient Admission Scheduling Problem (PASP). The PASP is an NP-hard problem that represents an important field in the health care discipline. To perceive the influence of the heuristics on solving the problem, we implemented two versions of hyperheuristics, each one is working on a different set of heuristics. The results show that one of the versions generates better results than the other which reveals the important role of the quality of basic heuristics to enhance the hyperheuristic performance.

1 Introduction

Over the last decades, computational optimization has became the point of interest for a huge number of researchers since it arises in various domains such as health care, industry, data science, finance, software engineering... Several problems are resolved by exact methods. However, most of them require an unreasonable computing time. Therefore, their resolution might be impossible. It is for this reason that heuristics emerged. They become one of the best alternatives to deal with hard computational problems although they do not guarantee optimality [14].

As heuristics are problem-specific, metaheuristics were created to gain genericity and be able to deal with a wider range of problems. Most of these metaheuristics are inspired by natural phenomena, to reach a bit abstract level. They are applied to different problems, but, they require several parameters settings to move from one problem to another.

Hyperheuristics are created to form a new concept, that raises the level of generality as search techniques to solve computational search problems. The motivation behind these methods resides in the capacity to operate on the search space of heuristics rather than the search space of solutions [14]. This advantage allows the hyperheuristics to be independent of the problem domain specifications. The hyperheuristics consist in managing a set of constructive and/or perturbative low level heuristics in order to automatically find the adequate heuristics sequence or methods to solve a particular problem. These approaches are divided into two classes: the first class is the selection hyperheuristics that aims to find the best sequence of existing heuristics. The second class consists in generating new heuristics using the components of existing heuristics [12]. In our work, we are interested by the selection hypeheuristics.

Selection hyperheuristics represent an important field, which is still under evolution. This class of hyperheuristics aims to find the best sequence of a set of basic heuristics specific to the problem treated.

Recently, an important number of papers used the selection hyperheuristics to solve many problems from different disciplines. Mentioning the scheduling problems, the selection hyperheuristics are used to solve different problems in this domain. For example, Flow shop and it's variants [29][27], Jobs shop and its variants [15][34], Resource constraints problems [23]. Moreover, we can find the selection hyperheuristics handling with problems from other domains, like Cloud Computing [30], Health Care [25][6][22], Aircraft [4], Timetabling [21], Gaming [28], eternity puzzling [32], Winner Determination Problem [24], Cutting Stock Problem [20], Knapsack [13] and MAX-SAT [26].

Not only that, but the Multi-objective problems are also explored by the selection hyperheuristics [33][10].

In our paper, we choose to deal with a health care problem which is the Patient Admission Scheduling Problem (PASP). Nowadays, the PASP represents a critical problem as it treats a real issue that the hospitals is confronting every day. To solve this problem, we choose to work with a selection hyperheuristic inherited from a metaheuristic that simulates the real mating honey bees behavior [17]. In the literature, only one work used this type of hyperheuristic and the paper treats the MAX-SAT and the Bin-Packing problems [26].

This paper is organized as follows: the second section contains a review of the PAS problem, in which, we represent the different works that treated the considered problem and we describe the problem with its mathematical formulation. The next section details the proposed hyperheuristic and its steps clearly. The fourth section summarizes the experimentation part covering the parameter setting and the obtained results. We finish the paper with a conclusion and some future perspectives.

2 Patient admission scheduling problem description

In this section, we represent a state of the art of the PAS problem. Moreover, we define the considered problem with its mathematical formulation.

2.1 A review on the PAS problem

The PASP is a health-care problem which is treated by an important number of papers. The PASP consists in assigning patients to beds in a way to find the suitable bed for each patient, in other words, it aims to maximizing the satisfaction of patients needs and preferences.

A first introduction was elaborated by Deemester et al [11]. Then, the problem was tackled by different researchers using different categories of methods. We can classify them into four classes: Exact methods, Heuristics, Metaheuristics and Hyperheuristics.

For the first class, we find a work that used a Mixed integer Program (MIP) to formulate the PASP [5]. As an exact method, This work has obtained the best known values in 9 instances from 13 instances. Using the same formulation, Turhan et al. [31] solves the PASP with a method based on heuristics named Fix-and-Relax (F&R) and Fix-and-Optimize (F&O). This paper obtained good results in a less time than the compared papers.

Dealing with the problem using heuristics was the main objective of Borchani et al. work, who investigated two versions of heuristics based on the Hungarian method [8]. This method consists in transferring a cost matrix into a sequence of beds based on the constraints violation penalties. The first heuristic is applied on the problem while relaxing several constraints. However, the second heuristic considers all the constraints.

The metaheuristics are the most used method to treat the PAS problem. We find Ceshia et al. [9] who used the multi-neighborhood local search procedure based on two metaheuristics: Tabu Search and Simulated Annealing to obtain two competitive versions. A Biogeography-Based Optimization approach was proposed in [18]. This approach is inspired from the idea of species migration between different habitats. An improvement of the method was proposed in [19]. Moreover, Guido et al. proposed a metaheuristics-framework to solve the PASP [16]. This method is inspired from the rescheduling approaches and classified as cooperative between a metaheuristic and an exact method. In addition, Late acceptance hill climbing algorithm was proposed as an iterative local search procedure that is inspired by the simple hill climbing optimization algorithm [7]. This method presents a new strategy that tries to escape from the local optimum until reaching a better candidate solution. Another paper proposed a Harmony search algorithm to solve the PASP [3]. The proposed approach is based on three operators to generate new solutions and a memory operator shared by all the operators in order to combine the features of the existing solutions in the population. A new metaheuristic was proposed by abdelkareem et al. based on the discrete flower pollination inspired from the flower pollination algorithm which is dedicated for continuous problems [2]. The method is composed by a fundamental step called "discretization procedure" which consists in converting the problem to a discrete space to switch between continuous and combinatorial space. After the conversion, the algorithm applied two neighborhood operators according a probability.

Reaching the hyperheuristic class, as this method is not much used, we find two papers used the selection hyperheuristics [6][25]. The hyperheuritic used in these papers is composed by 3 fundamental steps: heuristic selection strategy, monitoring and move acceptance phase. The only difference between the two works resides in the operators used to perform the 3 steps. For example, the first paper used 3 operators as a heuristic selection strategy, the tournament selection for the monitoring and great deluge, Simulated Annealing, improving or equal and only improving for the move acceptance phase. The second paper used two versions of tournament selection for the monitoring and he adds the adaptive iteration limited list-based threshold accepting for the move acceptance phase.

2.2 Problem definition

The PASP belongs to the hospitalization domain that considers, for a given planning horizon, a set of patients that need to be assigned to a set of beds for each night of their stay. The static PASP is considered in this paper, in other words, the stay length of each patient is contiguous, given from the beginning and cannot be changed.

The PASP has the following basic features:

- The patient is characterized by age, gender, specialism needed, room preferences and a necessary and preferred properties.
- The hospital is composed by departments that are characterized by a minimum and maximum age limits. Each department is composed by rooms that offer several specialisms with a level of expertise according to the department to which they belong. Each room is characterized by a gender policy: **M**: accepts only male patients, **F**: accepts only female patients, **D**: accepts both genders but in a day, it accepts only one gender, **N**: accepts the both genders. Each room has different properties that the patients can need. The room is composed by beds that define its capacity, it can contain 1, 2 or 4 beds.

Several constraints are applied when processing the problem. We can find two types of constraints:

- Hard constraint: the violation of this constraint engenders an unfeasible solution which is represented in this problem by the capacity of the rooms.
- Soft constraints: the violation of these constraints generates a penalty in the objective function in a way that enhances its value . The soft constraints are:
  - The gender of the patient should respect the gender policy of the room.
  - The age of a patient should not exceed the age limits of the department of the room where he is assigned.
  - The specialism that the patient needs should be offered with a high level of expertise by the department of the room where he is assigned.
  - The needed and the preferred properties should exist in the room where he is assigned.
  - The room capacity should be equal or less than the room preferences of the patient.
  - A patient should not be transferred from one room to another room during his stay period.

We note that specialisms, age limits, properties features of the room, and room preferences constraints are merged into one constraint called **PRC**, the gender constraint is called **RG** and the transfer constraint is called **Tr**.

The objective is to minimize the patients assignment cost by respecting the hard constraint and minimizing the total sum of soft constraints violation penalties.

Mathematical formulation:

The mathematical model was proposed by [11], it includes the objective function (equation 1) and the constraints (equations 2, 3, 4, 5, 6 and 7). The definition of the terms is as follows:

D: The set of days including Dp representing the set of days in which the patient p is present in the hospital.
R: The set of rooms.

P: The set of patients including P_F (female patients) and P_M (male patients).
$C_{p,r}$**:** Penalty of assigning patient p to room r.
Cap_r**:** Number of beds available in a room r.
W_{RG}**:** Weight of a room gender policy constraint.
W_{Tr}**:** Weight of transfer constraint.

Decision variables

$x_{p,r,d}$**:** 1 if patient p is assigned to a bed in room r in day d, 0 otherwise.
$t_{p,r,d}$**:** 1 if patient p is transferred from room r in day d, 0 otherwise.
$m_{r,d}$**:** 1 if there is at least one male patient in room r in day d, 0 otherwise.
$f_{r,d}$**:** 1 if there is at least one female patient in room r in day d, 0 otherwise.
$b_{r,d}$**:** 1 if there are both male and female patients in room r in day d, 0 otherwise.

– **Objective function:**

$$\sum_{p\in P, r\in R, d\in D} C_{p,r} * x_{p,r,d} + \sum_{r\in R, d\in D} W_{RG} * b_{r,d} + \sum_{p\in P, r\in R, d\in D} W_{Tr} * t_{p,r,d} \quad (1)$$

The tree components of the equation refer to PRC, RG ans Tr.

– **Constraints:**

Hard Constraints:

- Each patient should be assigned to a single room.

$$\sum_{r\in R} x_{p,r,d} = 1, \forall p \in P, d \in D_p \quad (2)$$

- The number of assigned patients to a room should not exceed the number of beds available.

$$\sum_{p\in P} x_{p,r,d} <= Cap_r, \forall r \in R, d \in D \quad (3)$$

Soft Constraints:

- Every day, the gender of patients in a room should comply with the gender policy of the room.

$$x_{p,r,d} <= f_{r,d}, \forall p \in P_F, r \in R, d \in D \quad (4)$$

$$x_{p,r,d} <= m_{r,d}, \forall p \in P_M, r \in R, d \in D \quad (5)$$

$$f_{r,d} + m_{r,d} - 1 <= b_{r,d} \forall r \in R, d \in D \quad (6)$$

- Patients should be kept in the same room and not transferred to another one.

$$x_{p,r,d} - x_{p,r,d+1} <= t_{p,r,d}, \forall p \in P, r \in R, d \in D_p \quad (7)$$

3 Honey Bee Mating Optimization

The Honey Bee Mating Optimization (HBMO) metaheuristic was introduced, for the first time, by Haddad et al. [17]. This algorithm is an improvement of the Marriage Bee Optimization algorithm (MBO) proposed by Abbass et al. [1]. This method is an evolutionary algorithm and represents a hybridization between the Simulated Annealing (SA), and the Genetic Algorithm (GA).

The HBMO approach consists in simulating the marriage behavior in the bees' colony. Applying the metaheuristic method, the algorithm practices the different steps of the marriage process directly on the solution's components.

However, A hyperheuristic based on the marriage bees behavior uses the same strategy as the metaheuristics. But, it employs the changes on the heuristics sequences. In other words, the algorithm works on a vector of heuristics instead of a problem's solution, meanwhile, it explores the heuristic search space rather than the solutions search space.

The HBMO HyperHeuristic ($HBMOH^2$) works on solutions represented by a sequence of low level heuristics in a vector or chromosome. Aiming to obtain a feasible solution with an optimized strategy, each bee in the $HBMOH^2$ looks for a maximum exploration of the search space.
The $HBMOH^2$ algorithm starts by generating an initial population of vectors. While applying the vectors of heuristics on a problem solution, the best chromosome is assigned to the queen, and the others are assigned to the drones. Then, the queen begins the mating flights which are composed by 2 phases: Filling the spermatheca and reproduction. Then a step of replacement will be applied. Finally, the last generation is updated and a new cycle is started (see figure 1).

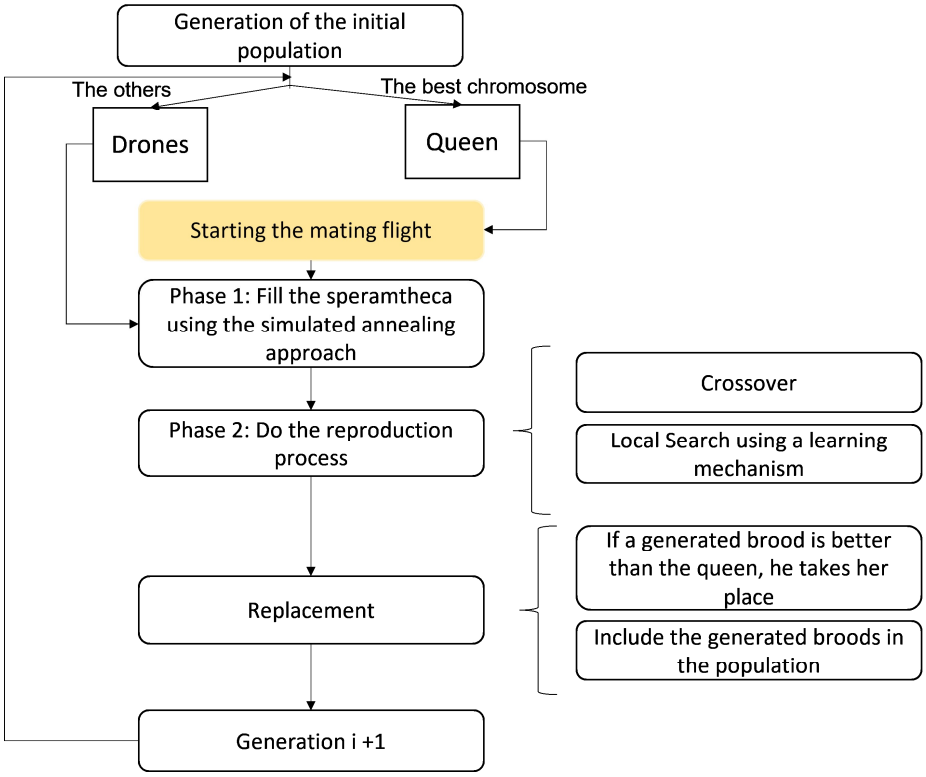


Fig. 1. $HBMOH^2$ process

3.1 Generation of the initial population

The most important component in the $HBMOH^2$ algorithm is the honey bee which represents the individual or the chromosome. Therefore, we represent it as a vector of low-level heuristics. The application of the vector's heuristics follows the same order as the vector. We note that a heuristic can be applied more than once and it can be not applied at all.

To constitute our hive, we generate the initial population of the honey bees which is composed by a predefined number of individuals. The construction of the vectors is made randomly, for each gene, we choose a heuristic from the heuristics set. This process allows the repetition of the same heuristic.

After that, we apply each vector on a problem's solution, the vector giving the best fitness will be assigned to the queen, whereas, the others will be assigned to the drones.

3.2 Mating flight

As our approach is inspired from real bees' behavior, only one queen is allowed to participate to each mating flight. The mating flight is the most important step in the $HBMOH^2$ algorithm which aims to reproduce a brood that can be the novel queen in the next cycle. After the generation of

the initial population, a predefined number of mating flights will be undertaken by the queen. Each mating flight is composed by two fundamental phases, the first phase consists in filling the queen spermatheca and the second phase represents the reproduction process.

The first phase:

This phase consists in filling the spermatheca by the drones' solution using the SA approach. For each iteration, the queen chooses randomly a drone and accepts it according to a probability calculated using the metropolis criteria (equation 8). When the drone is accepted, its vector will be added to the queen spermatheca. The drones in the spermatheca will form candidate individuals to mate with the queen.

As a projection of the SA on the $HBMOH^2$ algorithm, the temperature in our algorithm represents the speed of the queen which controls the probability to add a drone to the spermatheca. Not only speed, but the queen has also an energy that can not be reduced to a value near to zero.

To summarize, the mating flight starts with an initial value of energy which is a parameter of the hyperheuristic, an initial value of speed calculated as mentioned in the equation (9), and an empty spermatheca. The queen navigates around the different drones to fill its spermatheca using a probabilistic equation. For each iteration, the speed and the energy values will be reduced using respectively the equations 10, 11. This process ends when the energy is near zero or the spermatheca is full. We note that a drone can be chosen more than once and it may not be chosen at all.

$$Prob(Q, D) = \exp(-\delta / speed(t)) \tag{8}$$

- The metropolis equation of the simulated annealing approach that represents the move acceptance of adding a drone to the spermatheca.

- δ is the absolute difference between the fitness of the drone and the fitness of the queen.

$$initial\_speed = \delta E / \log(\theta) \tag{9}$$

- δ E is the average of the fitness value variations for 100 perturbations.
- θ is the initial acceptance rate.

$$speed(t+1) = speed(t) * \alpha \tag{10}$$

- speed(t) represents the flight speed of the queen at time t.

$$energy(t+1) = energy(t) * \alpha \tag{11}$$

- energy(t) represents the flight energy of the queen at time t.

- In the first iterations of the mating flight, the speed takes a high value, so the probability to add a drone is high leading to an intensification of the search.

- After each iteration in the space, the speed and the energy decrease (equations 10 and 11). Therefore, the probability to add the sperm of a drone decreases.

The second phase:

After filling the spermatheca, the queen starts the reproduction phase. It randomly chooses, for each iteration, a drone from the spermatheca and it does the crossover. The crossover is applied between the queen's vector and the chosen drone's vector to obtain the vector of the brood. After that, the bee workers will do the local search on the brood solution in order to improve it.

Crossover: The crossover operator applied is the 2-point crossover. It consists in choosing two points to divide the solutions parents to fragments and constructing the child solution by swapping the fragments of the parents vectors.

Local search using a learning mechanism: In order to diversify, we perform a learning mechanism based on a frequency matrix. The matrix is composed by n lines and n rows which n represent the number of heuristics. The cell i,j represents the frequency of the application of the heuristic i followed by the heuristic j. The frequency matrix represents a medium-term memory that saves the sequence of the heuristics and helps us to learn how to sequence it. The update of the frequency matrix is made at the end of each mating flight. In fact, we take the solutions of the best broods and we back up their solutions in the frequency matrix by incrementing, for each pair of successive heuristics in the solution, the corresponding cell (see figure 2).

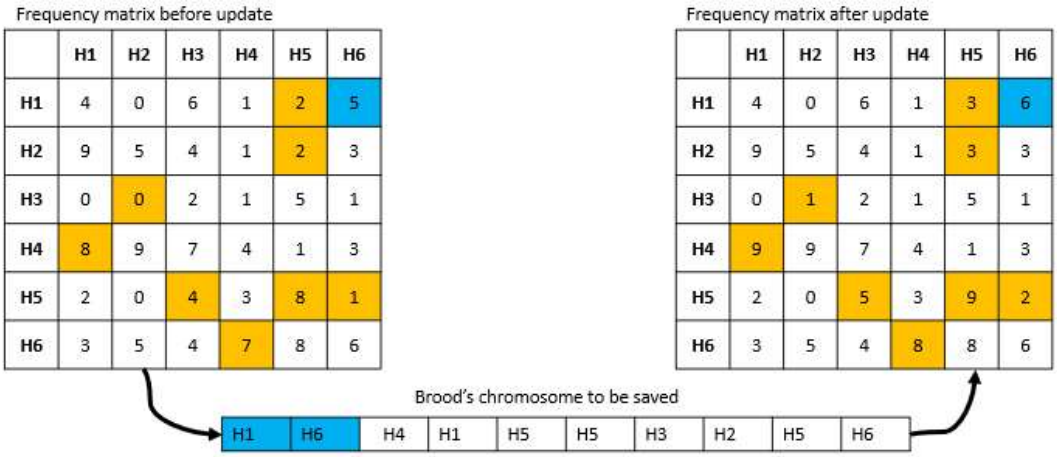


Fig. 2. The frequency matrix update

The local search step consists in discovering the neighbors of the solution's brood. Using a probability, for each gene of the brood chromosome starting by the second gene, we modify the gene, which represent a heuristic, by another one. The selection of the novel heuristic is based on a learning mechanism. In fact, in order to diversify, we choose the less used heuristic after applying the heuristic of the previous gene (see figure 3).

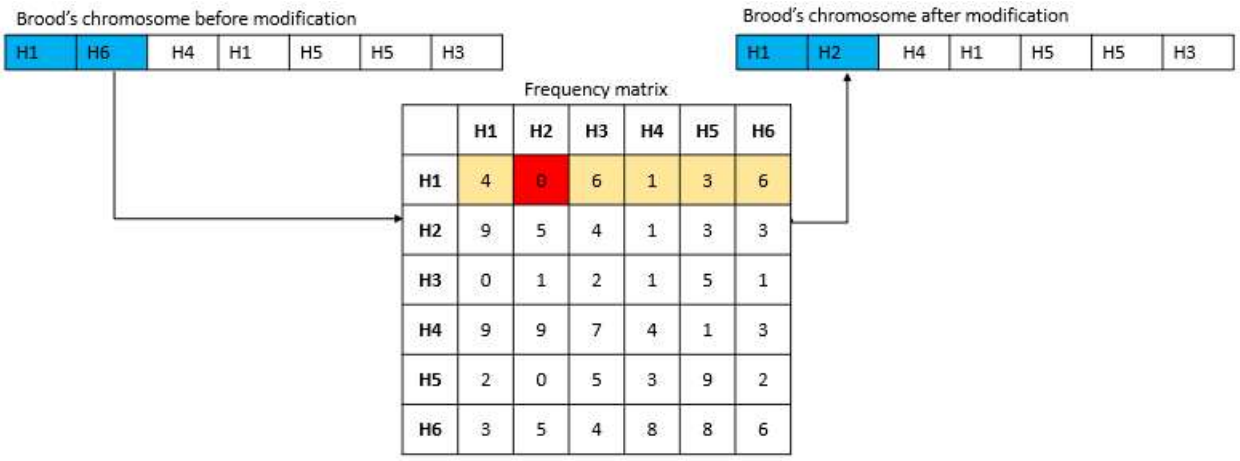


Fig. 3. An example of the local search process using the frequency matrix

4 Experimental study

In this section, we represent the different heuristics used to perform the hyperheuristic. Moreover, we detail how we apply the heuristics vector on the problem search space. After that, we report the benchmark instances used to test our hyperheuristic and the parameter tuning of the algorithm. Finally, we represent and discuss the experimental results.

4.1 Low level heuristics

As the solution of our algorithm is represented by a vector of heuristics, we can affirm that the quality of the heuristics has a direct influence on the performance of the algorithm. Therefore, we decide to create two versions of our hyperheuristic. Each version works with a set of heuristics different from the other version's set.

The first hyperheuristic's version, called $HBMOH^2$ V1, works with heuristics specific to the problem. The set is composed by 7 heuristics that can be summarized in Table 1.

Table 1. The used heuristics for the $HBMOH^2$ V1

Heuristic	Description
H1	Selects randomly two patients and swaps their bed assignments.
H2	Swaps all the bed assignments of two randomly selected rooms.
H3	Switches the bed assignments two by two of three randomly selected rooms.
H4	Swaps the bed assignments of a randomly selected patient to an empty bed.
H5	Swaps the bed assignments of a randomly selected patient to an empty bed while respecting the room properties.
H6	Swaps the bed assignments of a randomly selected patient to an empty bed while respecting the room preferences of the corresponding patient.
H7	Swaps the bed assignments of a randomly selected patient to an empty bed while respecting the room specialisms.

The second hyperheuristic, called $HBMOH^2$ V2, works with 2 types of heuristics: destructive heuristics consist in deleting components from the solution. We note that the destructive heuristics give always unfeasible solutions, and reparative heuristics consist in fixing the solution and converting it to a feasible solution. Table 2 describes the different heuristics used.

Table 2. The used heuristics for the $HBMOH^2$ V2

Destructive heuristics	
H1	Consists in randomly choosing a patient and deletes it from the solution. In another words, all the assignments related to this patient are deleted.
H2	Selects the patient that have the minimum cost and deletes it from the solution. Besides, the selected patient is the patient who has the least negative effect on the total cost of the solution.
H3	Selects the patient that have the maximum cost and deletes it from the solution. Besides, the selected patient is the patient who has the most negative effect on the total cost of the solution.
Reparative heuristics	
H4	Searches all the unassigned patients and it assigns them randomly to beds.
H5	Searches all the unassigned patients and, for each patient, it assigns him to the most suitable bed. In other words, it assigns each patient to the bed in a way that the assignment cost is minimal.
H6	Searches all the unassigned patients and, for each patient, it assigns him to the least suitable bed. In other words, it assigns each patient to the bed in a way the assignment cost is maximal.

We note that the heuristics vectors, used by the $HBMOH^2$ V2, are randomly filled by the mentioned heuristics. However, to obtain a feasible solution, the heuristics vectors must necessarily contain a reparative heuristic at the end, in a way that the last heuristic to be applied fixes the solution.

4.2 Application of the heuristics vectors on the problem search space

As we work with a hyperheuristic, we are managing the search space of the heuristics. Therefore, we need to project the algorithm's solution to the problem's solutions. In other words, we need to apply the vector of the heuristics generated on an existing solution.

For this reason, we decide to generate a population of solutions instead of a single solution in order to more explore the space. The population of the solutions is composed by 20 solutions, and then it is reduced to 5 by keeping the best ones.

In order to have a population with a medium quality, each solution is generated while respecting some soft constraints of the PAS problem and neglecting the others. Two important features we worked on to maintain them in the population: the first feature ensures that all the generated solutions are feasible ones, so they respect the hard constraints of the problem, and the second feature ensures that the generated solutions do not have the same heuristics sequences and they are well dispersed on the search space.

4.3 Parameter setting

To test our algorithm, we used the well known Benchmark performed by Deemester et al. [11]. The dataset is composed by 13 instances. In this work, we employed the first 6 instances. The characteristics of these instances vary according to the number of patients, rooms and beds. Moreover, the planning horizon is between 1 and 14 days. Table 3 shows the different features of the dataset.

Table 3. Description of the instances of the Benchmark

Instance	Patients	Beds	Rooms	Departments
1	286	652	98	4
2	465	755	151	6
3	395	708	131	5
4	471	746	155	6
5	325	587	102	4
6	313	685	104	4

To perform a first parameters calibration of the hyperheuristic, we used the trial and error method. This method consists in fixing, in each execution, all the parameters and varying the value of one parameter. Table 4 summarizes the fixed values of each parameter.

Table 4. Parameters Setting

Paramter	Range of values	Fixed value
Population size	[10, 20, 50, 100, 200, 500]	100
Vector size	[5, 10, 15, 20]	10
Number of generations	[10, 20, 30, 40, 50]	20
Local search probability	[0.01, 0.02, 0.05, 0.1, 0.15, 0.2]	0.15

4.4 Computational results

As mentioned before, we implemented two versions of $HBMOH^2$. Each version is based on a set of heuristics (see section 4.1). Table 5 shows the results of the two versions in the 6 benchmark instances. In order to compare and to analyze the gap between the results, we used the RPD metric which is calculated as mentioned in the equation 12.

– RPD: relative percentage difference

$$RPD_i = \frac{f_{1,i} - f_{2,i}}{f_{2,i}} \tag{12}$$

- Where $f_{1,i}$ and $f_{2,i}$ are the costs obtained respectively by the $HBMOH^2$ V1 and $HBMOH^2$ V2 in the instance i.

The results show that the second version gives the lowest cost in all the instances. Moreover, we find the largest RPD in the fifth instance which is the smallest one in number of patients, beds and rooms.

We can explain the performance of the second version by the quality of the heuristics. In fact, the heuristics that are used to perform it are destructive and reparative heuristics and they are divided between heuristics that intensify the search and heuristics that diversify it. This feature leads to a calibration between the intensification and diversification and helps to escape from the local optimum. But, the weakness of this version resides in the high time cost that it needs to obtain a result. From the other side, we find a version of hyperheuristic that works with heuristics specific to the problem. These heuristics bring close solutions that have several common features. Therefore, the algorithm is limited in a part of the search space and it can easily fall in an local optimum. But, this version costs less time to give a result.

Table 5. Computational Results of the two versions of the $HBMOH^2$

Instance	HBMOH V1		HBMOH V2		RPD
	Cost	Time (s)	Cost	Time (s)	
1	2015.6	43.4	1980	237.3	0.02
2	4148	130.5	3901	525.6	0.06
3	3156.6	93.6	3025.8	198.5	0.04
4	4139.2	105.5	4005.8	144.7	0.03
5	1610.4	55.1	1492.2	198.2	0.08
6	2392.6	62.4	2271.8	285	0.05

5 Conclusion

In this study, we implemented a selection hyperheuristic, called $HBMOH^2$, to handle with the Patient Admission Scheduling Problem. The problem belongs to the health care domain and represents a challenge for the researchers due to its complexity. The idea of this paper is to develop a hyperheuristic that simulates the honey bees mating behavior. In order to improve it, we focused on the heuristics that will be managed by the hyperheuristic. Therefore, we created two versions of our hyperheuristic that use two different sets of heuristics: the first set represents heuristics specific to the problem and the second set contains reparative and destructive heuristics. The results showed that the second version has a better performance than the first version but it costs more time. We conclude that the choice of heuristics in a hyperheuristic is very important and it has a loud effect on the performance of the hyperheuristic. As future work, we will use the same hyperheuristic to treat other instances from the same problem domain. Moreover, the multi-objective version of the PASP will be introduced and tackled by a variant of the honey bee mating hyperheuristic.

References

1. Abbass, H. A., : MBO: marriage in honey bees optimization-a Haplometrosis polygynous swarming approach. IEEE Congress on Evolutionary Computation (2001).
2. Abdalkareem, Z. A., Al-Betar, M. A., Amir, A., Ehkan, P., Hammouri, A. I., and Salman, O. H.: Discrete flower pollination algorithm for patient admission scheduling problem. Computers in biology and medicine 141: 105007 (2022).
3. Abu Doush, I., Al-Betar, M. A., Awadallah, M. A., Hammouri, A. I., Al-Khatib, R. E. M., ElMustafa, S., and ALkhraisat, H.: Harmony search algorithm for patient admission scheduling problem. Journal of Intelligent Systems 29.1: 540-553 (2018).
4. Allen, J. G., Coates, G., and Trevelyan, J.: A hyper-heuristic approach to aircraft structural design optimization. Structural and Multidisciplinary Optimization, 48, 807-819 (2013).
5. Bastos, L. S., Marchesi, J. F., Hamacher, S., and Fleck, J. L.: A mixed integer programming approach to the patient admission scheduling problem. European Journal of Operational Research 273.3: 831-840 (2019).
6. Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., and Vanden Berghe, G.: One hyper-heuristic approach to two timetabling problems in health care. Journal of Heuristics 18: 401-434 (2012).
7. Bolaji, A. L. A., Bamigbola, A. F., and Shola, P. B.: Late acceptance hill climbing algorithm for solving patient admission scheduling problem. Knowledge-Based Systems 145: 197-206 (2018).
8. Borchani, R., Masmoudi, M., Jarboui, B., and Siarry, P.: Heuristics-based on the Hungarian Method for the Patient Admission Scheduling Problem. Operations Research and Simulation in Healthcare: 33-62 (2021).
9. Ceschia, S., and Schaerf, A.: Local search and lower bounds for the patient admission scheduling problem. Computers & Operations Research 38.10: 1452-1463 (2011).
10. Cheng, L., Tang, Q., Zhang, L., and Zhang, Z.: Multi-objective Q-learning-based hyper-heuristic with Bi-criteria selection for energy-aware mixed shop scheduling. Swarm and Evolutionary Computation, 69, 100985 (2022).
11. Demeester, P., Souffriau, W., De Causmaecker, P., and Berghe, G. V: A hybrid tabu search algorithm for automatically assigning patients to beds. Artificial Intelligence in Medicine 48.1: 61-70 (2010).
12. Drake, J. H., Kheiri, A., Özcan, E., and Burke, E. K.: Recent advances in selection hyper-heuristics. European Journal of Operational Research, 285(2), 405-428 (2020).
13. Drake, J. H., Özcan, E., and Burke, E. K.: Modified choice function heuristic selection for the multidimensional knapsack problem. In Genetic and Evolutionary Computing: Proceeding of the Eighth International Conference on Genetic and Evolutionary Computing, October 18-20, 2014, Nanchang, China 225-234. Springer International Publishing (2015).
14. Epitropakis, M. G., and Burke, E. K.: Hyper-heuristics. In Handbook of Heuristics, 489-545. Springer, Cham (2018).
15. Garza-Santisteban, F., Sánchez-Pámanes, R., Puente-Rodríguez, L. A., Amaya, I., Ortiz-Bayliss, J. C., Conant-Pablos, S., and Terashima-Marin, H.: A simulated annealing hyper-heuristic for job shop scheduling problems. In 2019 IEEE congress on evolutionary computation (CEC) (pp. 57-64). IEEE (2019).
16. Guido, R., Groccia, M. C., and Conforti, D.: An efficient matheuristic for offline patient-to-bed assignment problems. European Journal of Operational Research 268.2: 486-503 (2018).
17. Haddad, O. B., and Afshar, A., and Mariño, M. A.: Honey-Bees Mating Optimization (HBMO) Algorithm: A New Heuristic Approach for Water Resources Optimization. Water Resources Management, 661–680 (2006).
18. Hammouri, A. I., and Alrifai, B.: Investigating biogeography-based optimisation for patient admission scheduling problems. Journal of Theoretical & Applied Information Technology 70.3 (2014).
19. Hammouri, Abdelaziz I.: A modified biogeography-based optimization algorithm with guided bed selection mechanism for patient admission scheduling problems. Journal of King Saud University-Computer and Information Sciences 34.3: 871-879 (2022).
20. Khamassi, I., Hammami, M., and Ghédira, K.: Ant-q hyper-heuristic approach for solving 2-dimensional cutting stock problem. In 2011 IEEE symposium on swarm intelligence (pp. 1-7). IEEE (2011).
21. Kheiri, A., Özcan, E., and Parkes, A. J.: A stochastic local search algorithm with adaptive acceptance for high-school timetabling. Annals of Operations Research, 239, 135-151 (2016).
22. Kheiri, A., Gretsista, A., Keedwell, E., Lulli, G., Epitropakis, M. G., and Burke, E. K.: A hyper-heuristic approach based upon a hidden Markov model for the multi-stage nurse rostering problem. Computers & Operations Research, 130, 105221 (2021).
23. Koulinas, G., Kotsikas, L., and Anagnostopoulos, K.: A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. Information Sciences, 277, 680-693 (2014).

24. Lassouaoui, M., and Boughaci, D.: A choice function hyper-heuristic for the winner determination problem. Nature Inspired Cooperative Strategies for Optimization (NICSO 2013) Learning, Optimization and Interdisciplinary Applications, 303-314 (2014).
25. Mısır, M., Verbeeck, K., De Causmaecker, P., and Berghe, G. V.: An investigation on the generality level of selection hyper-heuristics under different empirical conditions. Applied Soft Computing 13.7: 3335-3353 (2013).
26. Oueslati, I., and Hammami, M.: Honey Bee Cooperative HyperHeuristic. Procedia Computer Science, 192, 2871-2880 (2021)
27. Rodríguez, J. V., Petrovic, S., and Salhi, A.: A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications. MISTA: Paris, France,506-513 (2007).
28. Salcedo-Sanz, S., Jiménez-Fernández, S., Matías-Román, J. M., and Portilla-Figueras, J. A.: An educational software tool to teach hyper-heuristics to engineering students based on the bubble breaker puzzle. Computer Applications in Engineering Education, 23(2), 277-285 (2015).
29. Shao, Z., Shao, W., and Pi, D.: LS-HH: a learning-based selection hyper-heuristic for distributed heterogeneous hybrid blocking flow-shop scheduling. IEEE Transactions on Emerging Topics in Computational Intelligence (2022).
30. Tsai, C. W., Huang, W. C., Chiang, M. H., Chiang, M. C., and Yang, C. S.: A hyper-heuristic scheduling algorithm for cloud. IEEE Transactions on Cloud Computing, 2(2), 236-250 (2014).
31. Turhan, A. M., and Bilgen, B.: Mixed integer programming based heuristics for the patient admission scheduling problem. Computers & Operations Research 80: 38-49 (2018).
32. Vancroonenburg, W., Wauters, T., and Vanden Berghe, G.: A two phase hyper-heuristic approach for solving the eternity ii puzzle. In Proceedings of the International Conference on Metaheuristics and Nature Inspired Computing (2010).
33. Walker, D. J., and Keedwell, E.: Multi-objective optimisation with a sequence-based selection hyper-heuristic. In Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, 81-82 (2016).
34. Zhang, F., Mei, Y., and Zhang, M.: A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling. In Proceedings of the Genetic and Evolutionary Computation Conference, 347-355 (2019).

High Performance Algorithms for the Unrelated Parallel Machines Scheduling Problem with a Common Server and Job-Sequence Dependent Setup Times

Youssouf Hadhbi[1], Laurent Deroussi[1], Nathalie Grangeon[1], Sylvie Norre[1], and Christophe Blanchon[2]

[1] Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Étienne, LIMOS, 63000 Clermont-Ferrand, France
{youssouf.hadhbi, laurent.deroussi, nathalie.grangeon, sylvie.norre}@uca.fr
[2] Inoprod, 63800 Cournon-d'Auvergne, France blanchon@inoprod.com

1 Introduction

This paper deals with a new variant of the *non-preemptive unrelated parallel machines scheduling problem with setup times* (UPMS-SDST). In this context, the setup processing is not operated automatically such that a common server is used as an extra shared resource to operate the setup processing between each two jobs assigned to the same machine. For this, we consider the so-called job-sequence dependent setup times which means that the setup times depend on only the sequence of jobs. This problem is known under the name of *unrelated parallel machines scheduling problem with a common server and job-sequence dependent setup times* (UPMS-CS-SDST) [27]. It can be formally defined as follows. We consider a set of unrelated parallel machines M. There does not exist a dependency between these machines such that each machine $i \in M$ has its proper characteristics as speed, configuration, energy consumption and quality of work [27]. Notice that the machines M are available along a set of T_{max} consecutive periods denoted by $T = \{1, 2, ..., T_{max} - 1, T_{max}\}$. Let N be a set of n jobs numbered from 1 to n such that each job k is characterized by

- a subset of qualified machines $M_k \subseteq M$ that are capable of processing the job k,
- a processing-time $p_i^k \in \mathbb{N}$ on each machine $i \in M$ such that $p_i^k = +\infty$ for each non qualified machine $i \in M \setminus M_k$.
- a priority factor $w_k \in \mathbb{R}^*$,
- a set of setup-times $s_k^j \in \mathbb{N}$ of job k after job $j \in N_0 \setminus \{k\}$ if the two jobs k and j are assigned to the same machine that is to say the job-sequence dependent setup times, where N_0 denotes the set of jobs in N with an additional dummy-job 0 (i.e., $N_0 = N \cup \{0\}$) such that the dummy-job 0 precedes each first job assigned to each machine.

As mentioned before, a common server is used to manage the setup operations between each pair of consecutive jobs (j, k) assigned to the same machine. Moreover, the common server can be unavailable at some periods in T. For this, we consider a binary parameter a_t which equals to 1 if the server is available at period t, and 0 if not. However, the jobs N are available in all periods of T. The problem consists in assigning each job k to one of its qualified machine in M_k while satisfying the following technological constraints

- each job k must be processed only one time by one of its qualified machines $i \in M_k$. Moreover, p_i^k consecutive periods are assigned to each job k if it is assigned to machine i (*processing-time*). As a consequence, each job k has one completion period of processing in T (*non-premption of processing*),
- s_k^j consecutive periods are assigned to each job k if it is processed immediately after a job j over the same machine (*setup-time*),
- the common server cannot start the setup processing for a job if it hasn't finished the setup for another job yet (*non-premption of setup*),
- each machine can handle at most one job at each period such that two jobs cannot be processed at the same period on the same machine (*non-overlapping of processing*),
- the setup operation cannot be ensured at period t if the common server is not available at period t. Moreover, the common server can ensure the setup operation of at most one job at period t (*non-overlapping of setup*),

– the common server interrupts the setup processing for a job when it becomes unavailable. However, it can resume the setup processing of this job when it becomes available (*server availability*).

The objective is to minimize the total weighted completion time of processing for the n jobs.
From a practical point of view, the UPMS-CS-SDST problem arises when planning production and scheduling for some industrial flexible manufacturing systems. It appears in some applications related to the production of some mechanical parts of automobiles, hydraulic and electrical sectors [27] in the context of Industry 4.0. In this context, optimization of these real modern systems is a real challenge such that effective scheduling is a key issue to well manage the different resources and improve productivity of manufacturing systems.
The UPMS-CS-SDST problem is NP-hard in the strong sense [27]. The main contributions of this paper is as follows.
We first formulate the problem as a mixed integer linear program (MILP), and further devise an exact algorithm based on a branch-and-cut (B&C) algorithm to solve the problem. Due to the complexity of the problem, we propose a metaheuristic based on an iterated local search (ILS) algorithm [21] for solving the problem. Using this, we provide several matheuristics based on a two stage algorithm: ILS first and MILP last. We also carry out a comparative study between these methods and show the effectiveness of our approach using small-sized instances and large-sized instances.
The rest of this paper is organized as follows. In Section 2, we present some related works that have been well studied in the literature. In Section 3, we present a MILP for the problem. We introduce the ILS in Section 4. Matheuristics are presented in Section 5. We then present an extensive computational study using different classes of instances. Finally, we summarize our results and future outlook in Section 7.

2 Related Works

In the most general statement, the classical *unrelated parallel machines scheduling problem* (UPMS) has been well studied in the literature. The setup times are not considered in this case. It has been shown to be NP-hard [19]. Several exact algorithms have been proposed to solve the problem [1][8][23][22][33]. Despite the NP-hardness of the problem, heuristics and metaheuristics have also been required to solve the problem [14][33]. On the other hand, some approximate algorithms with good performance guarantee have been developed for solving the problem [16][19][24][32].
Notice that for some works, the setup-time has been considered as a part of processing-time. Here, we focus on the works that considered the processing and the setup operation separately. This is related to the UPMS-SDST problem. For this, we consider sequence-dependent setup times such that a setup operation is needed after each completion of processing of a job and before each starting of processing of another job assigned to the same machine. In this context, each machine is regulated to process the next job. This needs a setup time which depends only on the pair of consecutive jobs who share the same machine. This has first been studied by Allahverdi et al. [2]. Several exact algorithms have been proposed to solve the UPMS-SDST problem. They are based on branch-and-bound algorithm [29], branch-and-check algorithm [13], branch-and-price algorithm [25] and Bender decomposition [30]. However, these approaches have been shown to be less efficient when using large-sized instances of the same problem.
For this, heuristics [5][20][26] and metaheuristics have been used to solve the UPMS-SDST problem. Rabadi et al. [26] presented a greedy randomized adaptive search procedure to solve the problem. A simulated annealing has been used by Radhakrishnan and Ventura to solve the same problem [28]. Helal et al. [17] presented also a tabu search algorithm for the problem. On the other hand, some population-based metaheuristics have been introduced to solve the problem. Vallada and Ruiz [31] proposed a genetic algorithm for the problem. An ant colony optimization algorithm has been developed by Arnaout et al. [3] for solving the problem. Recently, Arnaout et al. [4] developed a worm optimization algorithm and compared it with some known metaheuristics for the same problem.
Hybrid methods have also been developed to solve the UPMS-SDST problem. Fang et al. [12] developed an hybridization of adaptive large neighborhood search algorithm with a tabu search

algorithm. Zeidi and Hosseini [34] proposed a two-stage algorithm which combines a genetic algorithm with a simulated annealing algorithm. Behnamian et al. [6] presented an hybridization of ant colony optimization, simulated annealing and variable neighborhood search.
Notice that the setup-processing is operated automatically in these previous works. This means that they did not consider the existence of a single or multiple servers to manage the setup process. Moreover, this resource is shared by all jobs N and machines M.
There exist a few works that have taken into account these additional resources (servers) and setup constraints such that the UPMS-CS-SDST problem is more less studied than the UPMS-SDST problem. A new integer linear programming formulation has been proposed by Bektur et al. [7] for solving the UPMS-CS-SDST problem without taking into account the unavailability of the common server in some periods of T. Moreover, the same authors developed some metaheuristics based on a simulated annealing and a tabu search algorithm for solving the problem. Elidrissi et al. [11] introduced a mixed integer programming formulation for a similar problem with two commun servers. They also developed two greedy heuristics and a general variable neighborhood search algorithm. The results showed that this latter outperformed the two other approaches.
To the best of our knowledge, the work done by Raboudi et al. [27] represents the initial study of the UPMS-CS-SDST problem taking into account the technological constraints and problem characteristics that have been considered in our study. Their mixed integer linear programming formulation has shown some limits such that it has been shown to be not able to solve small instances to optimality with a number of jobs up to 7 and 6 machines.

3 Mixed Integer Linear Programming Formulation

In what follows, we present a mixed integer linear programming formulation [15] for solving the UPMS-CS-SDST problem based on the following variables

- for each job $k \in N$ and machine $i \in M$, let u_i^k be a binary variables which equals to 1 if job k is assigned to machine i, and 0 if not,
- for each machine $i \in M$, job $j \in N_0$ and job $k \in N$, we denote by $x_{j,k}^i$ a binary variables which is related to the sequence-dependent setup times such that it takes 1 if jobs j is processed immediately before job k on machine i, and 0 if not,
- for each job $j \in N$ and period $t \in T$, variable y_t^k equals to 1 if the setup operation of job k is performed at period t, and 0 if not,
- for each two distinct jobs $j, k \in N$, we consider the variables q_k^j which takes 1 if the job k is processed after job j even if they are not assigned to the same machine, and 0 if not. This means that the starting period of the setup operation for job k is performed after the ending period of the setup operation of job j,
- we denote by $b^k \in \mathbb{R}^+$ (resp. $e^k \in \mathbb{R}^+$) the starting period (resp. the ending period) of the setup operation for job k,
- the completion period of processing for each job $k \in N$ is denoted by $c_k \in \mathbb{R}^+$.

The UPMS-CS-SDST problem is then equivalent to the following MILP

$$\min \sum_{k \in N} w_k c_k, \tag{1}$$

subject to

$$\sum_{i \in M_k} u_i^k = 1, \forall k \in N, \tag{2}$$

$$\sum_{i \in M \setminus M_k} u_i^k = 0, \forall k \in N, \tag{3}$$

$$\sum_{j \in N_0 \setminus \{k\}} x_{j,k}^i = u_i^k, \forall k \in N \text{ and } i \in M \tag{4}$$

$$\sum_{j \in N \setminus \{k\}} x_{k,j}^i \leq u_i^k, \forall k \in N \text{ and } i \in M, \tag{5}$$

$$\sum_{k \in N} x_{0,k}^i \leq 1, \forall i \in M, \tag{6}$$

$$\sum_{k\in N} y_t^k \leq a_t, \forall t \in T, \sum_{t\in T} y_t^k = \sum_{j\in N_0\setminus\{k\}} \sum_{i\in M} s_k^j x_{j,k}^i, \forall k \in N, \tag{7}$$

$$b^k \leq t y_t^k + B(1 - y_t^k), \forall k \in N \text{ and } t \in T, \tag{8}$$

$$t y_t^k - B \sum_{t'=1}^{t-1} y_{t'}^k \leq b_k, \forall k \in N \text{ and } t \in T, \tag{9}$$

$$\sum_{j\in N\setminus\{k\}} \sum_{i\in M} p_j^i x_{j,k}^i \leq b^k, \forall k \in N, \tag{10}$$

$$(t+1) y_t^k \leq e^k, \forall k \in N \text{ and } t \in \{1, ..., T_{max} - 1\}, \tag{11}$$

$$c_j + B(\sum_{i\in M} x_{j,k}^i - 1) \leq b^k, \forall k \in N \text{ and } j \in N \setminus \{k\}, \tag{12}$$

$$e^k - b^k \geq \sum_{j\in N_0\setminus\{k\}} \sum_{i\in M} s_k^j x_{j,k}^i, \forall k \in N, \tag{13}$$

$$e^k \leq b^j + B q_k^j, \forall k \in N \text{ and } j \in N \setminus \{k\}, \tag{14}$$

$$e^j \leq b^k + B(1 - q_k^j), \forall k \in N \text{ and } j \in N \setminus \{k\}, \tag{15}$$

$$c_k = e^k + \sum_{i\in M_k} p_k^i u_i^k, \forall k \in N, \tag{16}$$

$$c_k \leq T_{max}, \forall k \in N, \tag{17}$$

$$q_k^j \leq 1, \forall j \in K \text{ and } k \in N \setminus \{j\}, \tag{18}$$

$$u_i^k, x_{j,k}^i, y_t^k, q_k^j, b^k, e^k, c_k \geq 0, \tag{19}$$

$$u_i^k, x_{j,k}^i, y_t^k, q_k^j \in \{0, 1\}, \tag{20}$$

where $T' = \{1, ..., T_{max} - 1\}$, and B a large integer number (eg., $B = T_{max}$ is feasible).
The objective function (1) consists in minimizing the total weighted completion time of processing for the different jobs in N. Equations (2) express the fact that each job $k \in N$ is handled by only one machine of its qualified machine $i \in M_k$. Equations (3) show that each job $k \in N$ cannot be assigned to a non qualified machine $i \in M \setminus M_k$. Equations (4) ensure that a job k is preceded by one job $j \in N_0 \setminus \{k\}$ on a machine $i \in M$ if and only if job k is assigned to machine i. Moreover, constraints (5) shows that a job k can be the predecessor of at most one job $j \in N \setminus \{k\}$ on machine i if and only if it is assigned to machine i. The dummy-job can precede at most one job on each machine $i \in M$ as shown by constraints (6). The common server can handle at most one job at each period $t \in T$ if and only it is available at period t as noticed in constraints (**??**). However, variables y_t^k are forced to be equal to 0 for each $k \in N$ when the common server is not available at period t. Equations (7) show that the number of periods in which the setup is performed for job k must be equal to its setup-time. Constraints (8) and (9) ensure that a period t can be a starting period of the setup operation of jobs k if the setup of job k is performed at period t and there does not exist a period $t' \in \{1, ..., t-1\}$ in which the setup of job k is performed. Constraints (10) ensure that the starting period of setup for a job k is forced to be greater than the processing time of a job j which is processed immediately before job k on a shared qualified machine in $M_k \cap M_j$. In a similar way, we ensure in constraints (11) that the setup operation of each job $k \in N$ is accomplished after the last period t in which the setup is performed for job k. Constraints (12) ensure the non-overlapping of the processing operation with the setup operation for two distinct jobs that are processed one after the other and immediately. The gap between the ending period and the starting period of setup for job j is greater than its setup time as shown in constraints (13). Constraints (14) and (15) ensure the non-preemption of setup constraints. The completion period for each job $k \in K$ is computed as shown in constraints (16). Constraints (17) impose that the completion period should be smaller than T_{max}. Inequalities (18) and (19) are the trivial inequalities, and constraints (20) are the integrality constraints.
Using this formulation, we devise a branch-and-cut algorithm to solve the UPMS-CS-SDST problem [15] by combining a *branch-and-bound* algorithm with a *cutting plane* algorithm.

4 Iterated Local Search

In this section, we give a detailed description of the iterated local search algorithm [21] used to solve the UPMS-CS-SDST problem. We discuss the importance of its different procedures. This algorithm aims at building iteratively a sequence of solutions generated by an improvement heuristic based on the so-called local search (LS) algorithm. The ILS is based on the following procedures

- *Representation of a solution:* in this work, a solution S of the UPMS-CS-SDST problem is considered as a sequence of jobs for the setup-processing. It can be represented as vector $\{[1], [2], ..., [n]\}$ where $[k]$ denotes the job that is placed in position k in S.
- *Evaluation procedure:* for this, we first consider a solution S of the problem. This solution is then evaluated by using a greedy algorithm as follows. At each iteration, we select the first job k from the solution S that is not yet assigned to a machine $i \in M_k$. After this, for each qualified machine $i \in M_k$ of job k, we compute the starting period of setup, the ending period of setup, and the completion time of the processing for job k while satisfying all constraints of the problem with the set of jobs that are already proceeded (i.e., the set of jobs that precede k in the solution S). Then, the selected job k is assigned to the qualified machine $i \in M_k$ that offers the minimum completion time C_k. The algorithm stops when all jobs are assigned, or when the completion time C_k of the current job k exceeds T_{max} which means that solution S is infeasible. The output of this algorithm is given by a quadruple $(f(S), G, C, R)$ where
  - $f(S)$ denotes the total weighted completion time of processing for the different jobs in S,
  - G is a matrix of $m * T_{max}$ dimension such that each element G_t^i denotes the index of the job assigned at period t of the machine i, and equals to 0 if no job is assigned to machine i at period t. This can be used to draw a Gantt diagram.
  - C is a vector of size n which presents the completion time of processing for the set of jobs such that each C_k represents the completion time of job k as mentioned before.
  - R is a vector of size T_{max} such that each element R_t stores the index of the job for which the setup processing is done at period t by the common server, and 0 if no setup is done at period t
- *Initial solution:* an initial solution for the UPMS-CS-SDST problem can be seen as a starting point in a search area of the solutions. For this, one can randomly generate an initial solution S_0 for the problem. We then use an evaluation procedure described above to evaluate this solution and further show if it is feasible or not for the problem. Generally, this starting solution S_0 does not give a good quality solution. This step must not be neglected such that starting with a good solution improves the quality of the algorithm and allows achieving high quality solutions as fast as possible and especially the computation time is very short.
- *Neighborhood procedure:* this aims at exploring the neighborhood area of a solution S in particular and the search area of all solutions of the problem in general. This procedure generates a new solution S' for the problem, called neighbor of S (denoted by $S' = Neighbor(S)$) such that some positions of certain jobs of S are randomly changed in S'. This new solution is then evaluated to be compared with solution S. This procedure should be executed many times to explore the neighborhood space of a solution and extend the search space. For this, we distinguish several neighborhood strategies that can be used to explore the solution space
  - *Exchange:* we need to select randomly two jobs j and k and change their positions in the sequence J.
  - *Insertion:* it consists in moving randomly a job from a position a and inserting it in another position b in sequence J.
  - *Inversion:* we first select randomly two positions a and b. Then, we reverse the sub-sequence of jobs situated between a and b.
- *Acceptance Criterion:* choosing the ideal acceptance criterion is very important such that it aims at determining the rules for the acceptance of updating the current best solution and replacing it by an iteration solution. For this, we use the so-called "Better" criterion proposed by Lourenço et al. [21] such that a solution S' can replace a solution S at each iteration of the algorithm if and only if the quality of the new solution S' is better or equal to the quality of solution S.
- *Stopping criterion:* in our study, the algorithm terminates when we exceed a limited number of iterations or a maximum CPU time.

– *Improvement procedure:* this is based on a local search algorithm. Consider a solution $S^{'}$, the local search algorithm aims at finding a nearby solution $S^{'*}$ with better quality than $S^{'}$. This algorithm needs as input the maximum number of iterations without improvement, neighborhood method, a maximum CPU time, and an initial solution $S^{'}$. We then explore the neighborhood space of solution $S^{'}$ given at each iteration of the ILS which is considered as the initial best solution of the LS (denoted by $S^{*\prime}$). At each iteration of the local search algorithm, we generate a new solution $S^{"}$ that can be seen as a neighbor of the current best solution $S^{*\prime}$ and then evaluate it using the greedy-algorithm. This will then be compared with the current best solution $S^{*\prime}$. We update the current best solution if it satisfies the acceptance criterion. The algorithm stops when the stopping criterion is verified. Algorithm 1 summarizes the different steps of the local search algorithm.

Algorithm 1 Local Search Algorithm

Require: a maximum number of iterations without improvement denoted by $Max_{it} \geq 0$, a maximum CPU time denoted by $Max_{Cpu} \geq 0$, perturbation method denoted by $Neighbor$, evaluation procedure, initial solution denoted by S'.
$i \leftarrow 0$ and $S^{*\prime} \leftarrow S'$
while $i \leq Max_{it}$ and Max_{Cpu} is not exceeded **do**
  $S" \leftarrow Neighbor(S^{*\prime})$ and $i \leftarrow i + 1$
  if $f(S") \leq f(S^{*\prime})$ **then**
    if $f(S") < f(S^{*\prime})$ **then**
      $i \leftarrow 0$
    end if
    $S^{*\prime} \leftarrow S"$
  end if
end while
return $S^{*\prime}$

To summarize, the ILS can be seen as an iterative improvement technique. Here, the goal is to find the best sequencing of jobs which gives a high quality solution for the problem. For this, we first use a construction method to generate an initial feasible solution S_0 for the problem that will be improved by the LS to provide an initial best solution S^* for the problem. After this, and at each iteration, we apply multiple perturbations on the current solution S^* using a neighborhood method. As a result, a new solution S' is found for the problem. We then use the local search algorithm to explore the neighborhood space of solution S' and return a local optimum $S^{*\prime}$ of the LS. The resulting solution will then be compared with the current best solution of the ILS and become the new best solution if it satisfies the acceptance criterion. The algorithm is stopped if one of the stopping criteria is verified.
All these steps are summarized in Algorithm 2.

Algorithm 2 Iterated Local Search Algorithm

Require: a maximum number of iterations for the ILS denoted by $Max_{it}^{ILS} \geq 0$, a maximum CPU time for the ILS denoted by $Max_{Cpu}^{ILS} \geq 0$, number of perturbations for the ILS denoted by b, perturbation method for the ILS denoted by $Neighbor_{ILS}$, LS's parameters, evaluation procedure, initial solution denoted by S_0.
$S^* \leftarrow LS(Max_{it}^{LS}, Max_{Cpu}^{LS}, Neighbor_{LS}, f, S_0)$ and $i \leftarrow 0$
while $i \leq Max_{it}^{ILS}$ and Max_{Cpu}^{ILS} is not exceeded **do**
  $S' \leftarrow S^*$, $a = 0$ and $i \leftarrow i + 1$
  while $a \leq b$ **do**
    $S' \leftarrow Neighbor(S')$ and $a \leftarrow a + 1$ *//Perturbation(S')*
  end while
  $S^{*\prime} \leftarrow LS(Max_{it}^{LS}, Max_{Cpu}^{LS}, Neighbor_{LS}, f, S')$
  if $f(S^{*\prime}) \leq f(S^*)$ **then** *//AcceptanceCriterion($S^*, S^{*\prime}$)*
    $S^* \leftarrow S^{*\prime}$
  end if
end while
return S^*

5 Matheuristics

In what follows, we introduce three matheuristics for solving the UPMS-CS-SDST problem. They can be considered as a two stage algorithm combined an iterated local search algorithm with a modified version of our MILP formulation [15]. Notice that these approaches provide approximate solutions for the problem without guarantee of optimality.
Throughout the following sections, our MILP formulation already presented in Section 3, will be considered as the basic MILP formulation for the problem.

5.1 Matheuristic I: Machine-Job-Sequencing Fixing

For this, we first use an ILS to solve the problem. Consider the resulting solution S. We then use an updated formulation of the basic MILP with some additional constraints. This matheuristic aims at producing an optimal solution for the resulting sequencing of jobs provided by S while respecting some additional precedence constraints required by S. This means that each job $j \in N$ should be preceded by each job k having a smallest index in S compared with the index of j in S if and only if the two jobs j and k are assigned to the same machine. Otherwise, the precedence constraint between these two jobs is not considered. For this, we consider a matrix P of $n * n$ dimension such that $P_{j,k} = 1$ if job j is preceded by job k in S, and 0 if not. After this, we use a MILP formulation to solve the problem such that we keep the same variables and constraints used in the basic MILP. Moreover, we add the following precedence constraints

$$c_k + B(u_i^j + u_i^k - 2) \leq b_j, \forall k \in N, \forall j \in N \setminus \{k\}, \forall i \in M \text{ with } P_{j,k} = 1. \tag{21}$$

Inequalities (21) ensure that if job j and k are assigned to the same machine, and $P_{j,k} = 1$ then the starting period of setup for job j must be greater than the completion period of job k.
Notice that this modified MILP can also be used as an exact method to solve a new variant of the UPMS-CS-SDST problem that can be called *unrelated parallel machines scheduling problem with a common server, job-sequence dependent setup times and precedence constraints*.

5.2 Matheuristic II: Machine-Job-Assignment Fixing

In this case, we aim at identifying the optimal solution of the problem with pre-assignment of machines. For this, we make the set of qualified machines $M_k = \{i\}$ if job k is assigned to machine i in solution S of ILS. Based on this, we introduce a new MILP to solve the problem taking into account the pre-assignment of machines as additional constraints. It's based on the same variable of the basic MILP without taking into account the variables u_i^k given that the jobs are already assigned to machines. We also modify the definition of variables $x_{j,k}^i$ such that we consider a new variable $x_{j,k}$ which equals to 1 if job k is processed immediately after job j and they should be already assigned to the same machine, and 0 if not. As a consequence, the number of variables is decreased and becomes less compared with the basic MILP. Moreover, the constraints that are related to the machine assignment should be deleted, and constraints (4), (5), (7), (10), (12), (13) and (16) should be modified as follows

$$\sum_{j \in N_k \cup \{0\}} x_{j,k} = 1, \forall k \in N, \tag{22}$$

$$\sum_{j \in N_k} x_{k,j} \leq 1, \forall k \in N, \tag{23}$$

$$\sum_{t \in T} y_t^k = \sum_{j \in N_k \cup \{0\}} s_k^j x_{j,k}, \forall k \in N, \tag{24}$$

$$\sum_{j \in N_k} p_j^i x_{j,k} \leq b^k, \forall k \in N \text{ with } \{i\} = M_k, \tag{25}$$

$$c_j + B(x_{j,k} - 1) \leq b^k, \forall k \in N, \forall j \in N_k, \tag{26}$$

$$e^k - b^k \geq \sum_{j \in N_k \cup \{0\}} s_k^j x_{j,k}, \forall k \in N, \tag{27}$$

$$c_k = e^k + p_k^i, \forall k \in N \text{ with } \{i\} = M_k, \tag{28}$$

where N_k denotes the set of jobs that are assigned to the same machine with job k in solution S. As can be noticed, the number of variables and constraints is largely decreased due to these modifications compared with the basic MILP.

5.3 Matheuristic III: Setup-Job-Sequencing Fixing

The third matheuristic consists in determining the optimal solution for the problem with some additional constraints such that the predecessor of each job is known in advance using the solution S of the ILS. As a consequence, the setup time of each job is known in advance. For this, we denote by j_k the index of job that has been processed immediately before job k and assigned to the same machine in solution S. Using this, we introduce another MILP based on the original MILP with some modifications. We keep the same variables of the original MILP without considering the setup variables $x^i_{j,k}$. Some constraints should be removed from the model (e.g., constraints (4), (5), (6)), and other ones as (7), (10), (12), (13), should be modified as follows

$$\sum_{t \in T} y^k_t = s^{j_k}_k, \forall k \in N, \tag{29}$$

$$\sum_{i \in M} p^i_{j_k} \leq b^k, \forall k \in N, \tag{30}$$

$$c_{j_k} \leq b^k, \forall k \in N, \tag{31}$$

$$e^k - b^k \geq s^{j_k}_k, \forall k \in N. \tag{32}$$

Moreover, each job k and its predecessor j_k should be assigned to the same machine. For this, we add the following constraints to the new MILP formulation

$$u^{j_k}_i = u^k_i, \forall k \in N, \forall i \in M. \tag{33}$$

6 Computational Study

The different approaches described above have been implemented in Java and run on high performance computing servers with 64 GB as a memory limit. For each MILP, we devise a Branch-and-Cut algorithm to solve the problem. We have also implemented the MILP of Raboudi et al. [27]. For this, we use CPLEX [10] to solve the MILP formulations provided in previous sections. We also use its proper cuts to obtain tighter bounds for the linear relaxation and boost the performance of the Branch-and-Cut algorithm. We consider 900 sec as a CPU time limit for the ILS and 3600 sec for the B&C algorithm. The maximum number of iterations for the ILS is limited to 200000 such that the number of perturbations of the ILS equals to 4 at each iteration. The common server is available for 8 contiguous periods and then unavailable for 8 contiguous periods in T. All the approaches have been tested using three families of instances described as follows.

Instances / Characteristics	$\|M\|$	$\|N\|$	w_k	p^i_k	s^j_k	$\|M_k\|$	T_{max}
Instances I [27]	{2,3,4}	{4,5,6,7, 8, 10, 20, 24}	[1; 5]	[10; 350]	[1; 10]	≥ 1	[200; 1000]
Instances II	{2,3,4}	{4,5,6,7, 8, 10, 20, 24}	[1; 5]	[10; 350]	[1; 10]	≥ 2	[200; 1000]
Instances III	{2,5,10, 15, 20}	{5,10, 20, 30, 40}	[1; 6]	[1; 16]	[1; 6]	≥ 1	[300; 4000]

Table 1. Instances characteristics.

Concerning the ILS, for each instance and each neighborhood procedure, we compute the average value of the objective function (denoted by *avg*) and the objective function for the best solution (denoted by *min*) of 10 independent ILS solutions starting from the same initial solution that has been generated randomly.
On the other hand, for the different matheuristics, and for each instance, we use the best solution (*min*) found by 10 independent ILS solutions that will provide some additional constraints for the second stage of matheuristic as already described in Section 5.
The main objective of this study is to show the effectiveness of our approaches using different instances (30 instances of type I, 32 instances of type II and 131 instances of type III). For this, we address a comparison study between the different approaches in tables 2, 3, 4 and 5.
We consider the following indicators:

- % opt (avg): percentage of instances that are solved to optimality at each replication by the chosen algorithm (10 replications for the ILS and one replication when using a Branch-and-Cut algorithm for each instance).
- % opt (min): percentage of instances that are solved to optimality in at least one replication by the chosen algorithm.
- % best sol (avg): percentage of instances for which the chosen algorithm provides the best *avg*.
- % best sol (min): percentage of instances for which the chosen algorithm provides the best *min*.
- % low standard deviation (SD): percentage of instances for which the chosen algorithm has the best SD.
- % best sol the ILS is improved or equal: percentage of instances for which the chosen algorithm found a solution value which is equal or strictly better than the ILS's solution value.
- % best sol the ILS is improved: percentage of instances for which the chosen algorithm found a solution value which is strictly better than the ILS's solution value.

Notice that the results of our B&C algorithm (using our MILP) will be taken as a benchmark in our computational study given that it has been shown to be the best model in our previous study [15] compared with the MILP of Raboudi et al. [27], and other ones that we already proposed in previous studies.
First, results show that for the two first classes of instances I and II respectively, the B&C algorithm is able to solve to optimality 70% and 53,13% of instances respectively. Moreover, the B&C is shown to be able to beat the existing one MILP of Raboudi et al. [27] which suffers more from a tractability point of view even for small-sized instances. The latter solves to optimality 10%, 3,13% and 2,29% of instances of type I, II and III respectively. However, the B&C algorithm becomes less performant for the large-sized instances (Instances III) such that 6,11% of these instances are solved to optimality. As a consequence, the B&C becomes more less efficient when using large-sized instances and even for the medium ones. For this reason, and as mentioned before, we use the ILS and some matheuristics to solve the UPMS-CS-SDST problem.
We first show in Table 2 the effectiveness of the ILS using different neighborhood procedures denoted respectively by ILS_Exc , ILS_Inv and ILS_Ins when using respectively exchange, inversion and insertion as neighborhood procedure while considering the instances that are solved to optimality by the B&C. For this, we consider the two indicators % opt (avg) and % opt (min) to compare between these algorithms using the instances that are solved to optimality by the B&C.

		% opt (avg)	% opt (min)
Instances_I	**Raboudi et al.**	14,29	14,29
	ILS_Exc	90,48	90,48
	ILS_Inv	90,48	90,48
	ILS_Ins	90,48	90,48
	Math_MJSF	**95,24**	**95,24**
	Math_MJAF	85,71	85,71
	Math_SJSF	76,19	76,19
Instances_II	**Raboudi et al.**	5,88	5,88
	ILS_Exc	70,59	70,59
	ILS_Inv	70,59	70,59
	ILS_Ins	70,59	70,59
	Math_MJSF	**88,24**	**88,24**
	Math_MJAF	82,35	82,35
	Math_SJSF	70,59	70,59
Instances_III	**Raboudi et al.**	37,50	37,50
	ILS_Exc	75,00	75,00
	ILS_Inv	75,00	75,00
	ILS_Ins	75,00	75,00
	Math_MJSF	62,50	62,50
	Math_MJAF	**87,50**	**87,50**
	Math_SJSF	62,50	62,50

Table 2. Comparison between different algorithms based on the instances that are solved to optimality by the B&C.

The ILS has been shown to be very performant for the different instances such that it is able to solve several instances to optimality and even for the instances that are not solved to optimality by the MILP of Raboudi et al. [27]. The ILS also allows solving several instances that are not solved

by the B&C but without guarantee of optimality.
Moreover, we show in Table 3 the efficiency of each algorithm based on two indicators: % best sol (avg) and % best sol (min) while using all instances.

		% best sol (avg)	% best sol (min)
Instances_I	**Raboudi et al.**	46,67	46,67
	B&C	76,67	76,67
	ILS_Exc	**83,33**	**86,67**
	ILS_Inv	80,00	80,00
	ILS_Ins	**83,33**	**86,67**
	Math_MJSF	76,67	76,67
	Math_MJAF	73,33	73,33
	Math_SJSF	70,00	70,00
Instances_II	**Raboudi et al.**	40,63	40,63
	B&C	75,00	75,00
	ILS_Exc	**84,38**	**78,13**
	ILS_Inv	71,88	68,75
	ILS_Ins	75,00	**78,13**
	Math_MJSF	78,13	78,13
	Math_MJAF	75,00	75,00
	Math_SJSF	71,88	71,88
Instances_III	**Raboudi et al.**	7,63	7,63
	B&C	9,16	9,16
	ILS_Exc	**61,07**	64,12
	ILS_Inv	18,32	32,82
	ILS_Ins	59,54	**81,00**
	Math_MJSF	16,03	16,03
	Math_MJAF	9,92	9,92
	Math_SJSF	12,98	12,98

Table 3. Comparison between different algorithms based on all instances.

We notice in Table 3 that the ILS improves the quality of certain solutions proposed by the B&C and is able to find better solutions (based on *average* and *min*) than those found by the B&C or the MILP of Raboudi et al. [27]. These results are still stable for the different neighborhood procedures such that the exchange neighborhood procedure is shown to be advantageous in some instances compared with the other ones and also having the low SD (see Table 4) for the different instances of type I, II and III. Moreover, and using large-sized instances of type III, the ILS with its different neighborhood procedures, is able to solve some instances to optimality. However, for the other ones, the ILS cannot guarantee the optimality. For these same instances, the ILS gives several high quality solutions compared with the B&C.

	% low SD by ILS_Exc	% low SD by ILS_Inv	% low SD by ILS_Ins
Instances_I	**86,67**	83,33	**86,67**
Instances_II	**100,00**	87,5	90,63
Instances_III	**61,83**	27,48	59,54

Table 4. Comparison between different neighborhood procedures using standard deviation indicator.

Next, we combine the ILS with a B&C algorithm to devise a matheuristic for solving the UPMS-CS-SDST problem as already described in Section 5. For this, we assess the performance of three matheuristics denoted respectively by Math_MJSF, Math_MJAF and Math_SJSF. Table 2 shows that that they are also able to solve several instances to optimality and showed to be better than the ILS while using Math_MJSF for instances of type I and II. This latter solves some instances to optimality that are not solved to optimality by the ILS and the MILP of Raboudi et al. [27]. Table 3 also shows that the Math_MJSF is advantageous in some instances compared with the other approaches. However, when the optimality is not guaranteed or when using large-sized instances, the ILS proposes better solutions than the different matheuristics due to the usage of the MILP and the B&C in the second stage such that each MILP becomes more less tractable when using large-sized instances.
On the other hand, we aim to evaluate the impact of using the best solution of the ILS to add some additional constraints to the MILP in order to devise the different matheuristic, and also evaluate the impact of using this solution as a warm-start for these matheuristics. For this, Table 5 shows initially that the different matheuristics are able to produce several solutions with quality better or equal than those provided by the ILS (see column 1 in Table 5), and strictly better than those

of ILS for other ones (see column 2 in Table 5). Moreover, the warm-start technique is shown to be very efficient when using large-sized instances such that this technique is capable of boosting these matheuristics, and further allows obtaining strictly better solutions for more instances compared with when it's not used.

		Without Warm-Start		With Warm-Start
		% best sol the ILS is improved or equal	% best sol of the ILS is improved	% best sol of the ILS is improved
Instances_I	**Math_MJSF**	86,67	10,00	10,00
	Math_MJAF	83,33	10,00	10,00
	Math_SJSF	83,33	13,33	13,33
Instances_II	**Math_MJSF**	75,00	18,75	18,75
	Math_MJAF	78,13	31,25	31,25
	Math_SJSF	78,13	31,25	34,38
Instances_III	**Math_MJSF**	17,56	6,11	28,24
	Math_MJAF	12,21	4,58	29,01
	Math_SJSF	16,03	4,58	32,82

Table 5. Effectiveness of Matheuristics.

As a consequence, all these previous results prove the high quality performance of our approaches for solving the UPMS-CS-SDST problem.

7 Conclusion

In this paper, we have addressed the non-preemptive unrelated parallel machines scheduling problem with a common server and job-sequence dependent setup times. First, we have presented some related works considering the setup processing. We have proposed a mixed integer program to formulate the problem, and have devised an exact algorithm based on a branch-and-cut algorithm for solving the problem. Despite the NP-hardness of the problem, we have developed a metaheuristic based on an iterated local search algorithm to solve the problem. Using these results, we have presented different matheuristics that can be seen as post-optimization algorithms. The results have shown the effectiveness of these approaches and the advantages of certain ones.
Finally, it would be interesting to further develop some adaptive and hybrid metaheuristics for solving the problem. We also plan to study new realistic or real variants of the problem while considering some additional technological constraints (machine availability, delay for jobs) and ressources (multiple servers).

Acknowledgment

This work was supported by the French Government, France-Relance Project in collaboration with InoProd (https://www.inoprod.com/).

References

1. E. Åblad, A. Strömberg, and D. Spensieri, "Exact makespan minimization of unrelated parallel machines". Open Journal of Mathematical Optimization, 2, 2021, pp. 1-15.
2. A. Allahverdi, J. N. D Gupta, and T. Aldowaisan, "A review of scheduling research involving setup considerations". OMEGA International Journal of Management Science, 27, 1999, pp. 219–239.
3. J.P. Arnaout, G. Rabadi, and R. Musa, "A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines—part II: enhancements and experimentations". Journal of Intelligent Manufacturing, 25, 2014, pp. 43–53.
4. J.P. Arnaout, "A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times". Annals of Operations Research, 285, 2020, pp. 273-293.
5. A. Al-Salem, "Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times". Engineering Journal of University of Qatar, 17, 2004, pp. 177-187.
6. J. Behnamian,M. Zandieh, and S. M. T. Fatemi Ghomi, "Parallel-machine Scheduling Problems with Sequence-dependent Setup times Using an ACO, SA and VNS Hybrid Algorithm". Expert Systems with Applications, 2009, 36, pp. 9637–9644.
7. G. Bektur, and T. Saraç, "A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server". Journal of Computers and Operations Research, 103, 2019, pp. 46-63.

8. Z. Chen, and W. B. Powell, "Solving parallel machine scheduling problems by column generation". INFORMS Journal on Computing, 11, 1999, pp. 78–94.
9. L. P. Cota, F. G. Guimarães, F. B. de Oliveira, and M. J. Freitas Souza, "An Adaptive Large Neighborhood Search with Learning Automata for the Unrelated Parallel Machine Scheduling Problem". IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 185-192.
10. I.I. Cplex: V12. 9, "User's Manual for Cplex". IBM, 46(53), pp. 157.
11. A. Elidrissi, R. Benmansour, and A. Sifaleras, "General variable neighborhood search for the parallel machine scheduling problem with two common servers". Optimization Letters, 2022, pp. 1-31.
12. W. Fang, H. Zhu, and Y. Mei, "Hybrid meta-heuristics for the unrelated parallel machine scheduling problem with setup times". Knowledge-Based Systems Journal, 241, 2022, 108193 p.
13. L. Fanjul-Peyro, R. Ruiz, and F. Perea, "Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times". Computers and Operations Research Journal, 101, 2019, pp. 173-182.
14. C.A. Glass, C.N. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search". Mathematical and Computer Modelling Journal, 20, 1994, pp. 41–52.
15. Y. Hadhbi, L. Deroussi, N. Grangeon, and S. Norre, "Improved Formulations and Branch-and-Cut Algorithm for the Unrelated Parallel Machines Scheduling Problem with a Common Server and Job-Sequence Dependent Setup Times". 9th International Conference on Control, Decision and Information Technologies (CoDIT), 2023, Rome, Italy, pp. 1-6.
16. L.A. Hall, "Approximation algorithms for scheduling". Book of Approximation Algorithms for NP-Hard Problems, D.S. Hochbaum, editor, PWS Publishing, Boston, MA, 1997, pp. 1–45.
17. M. Helal, G. Rabadi, A. Al-Salem, "A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times". IJOR, 3, 2006, 182-192.
18. A.A. Juan, H. R. Lourenço, M. Mateo, R. Luo, Q. and Castella, "Using iterated local search for solving the flow-shop problem: Parallelization, parametrization, and randomization issues". International Transactions in Operational Research, 21, 2014, pp. 103-126.
19. J.K. Lenstra, B.S. David, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines". Mathematical Programming, 46, 1990, pp. 259–271.
20. S.W. Lin, C.C. Lu, and K.C. Ying, "Minimization of total tardiness on unrelated parallel machines with sequence- and machine-dependent setup times under due date constraints". International Journal of Advanced Manufacturing Technology, 53, 2011, pp. 353–361.
21. H.R. Lourenço, O.C. Martin, T. Stützle, "Iterated Local Search". Glover, F., Kochenberger, G.A. (eds) Handbook of Metaheuristics, International Series. Operations Research & Management Science, vol 57. Springer, Boston, MA, 2003.
22. E. Mokotoff, and P. Chrétienne, "A cutting plane algorithm for the unrelated parallel machine scheduling problem". European Journal of Operational Research, 141, 2002, pp. 515–525.
23. S. Martello, F. Soumis, and P. Toth, "Exact and approximation algorithms for makespan minimization on unrelated parallel machines". Discrete Applied Mathematics Journal, 75, 1997, pp. 169-188.
24. Z. Pei, M. Wan, and Z. Wang, "A new approximation algorithm for unrelated parallel machine scheduling with release dates". Annals of Operations Research, 285, 2020, pp. 397–425.
25. M.J. Pereira Lopes, and J.M.V. De-Carvalho, "A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times". EJOR, 176, 2007, pp. 1508–1527.
26. G. Rabadi, R. Moraga, and A. Al-Salem, "Heuristics for the unrelated parallel machine scheduling problem with setup times". Journal of Intelligent Manufacturing, 17, 2006, pp. 85-97.
27. H. Raboudi, G. Alpan, F. Mangione, G. Tissot, and F. Noels, "Scheduling unrelated parallel machines with a common server and sequence dependent setup times". 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022, 55, 2022, pp. 2179-2184.
28. S. Radhakrishnan, and J. A. Ventura, "Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent setup times". International Journal of Production Research, 38, 2000, pp. 2233–2252.
29. P. Rocha, M. Ravetti, G. Mateus, and P. Pardalos, "Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times". Computers and Operations Research Journal, 35, 2008, pp. 1250-1264.
30. T.T. Tran, A. Araujo, and J.C. Beck, "Decomposition methods for the parallel machine scheduling problem with setups". INFORMS Journal on Computing, 28, 2016, pp. 83-95.
31. E. Vallada, and R. Ruiz, "A genetic algorithm for the unrelated parallel machine scheduling problem with sequence-dependent setup times". EJOR, 211, 2011, pp. 612–622.
32. V. V. Vazirani, "Scheduling on Unrelated Parallel Machines". Approximation Algorithms, Springer, Berlin, Heidelberg, 2003.
33. A. Wotzlaw, "Scheduling Unrelated Parallel Machines—Algorithms, Complexity, and Performance". VDM Verlag Dr. Mueller e.K., 2007.
34. J. R. Zeidi, and Sa. M. Hosseini, "Scheduling unrelated parallel machines with sequence-dependent setup times". The International Journal of Advanced Manufacturing Technology, 81, 2015, pp. 1487–1496.

A Simulated Annealing Based Approach for the Roman Domination Problem

Jakob Greilhuber*, Sophia Schober*, Enrico Iurlano, and Günther R. Raidl

Algorithms and Complexity Group, TU Wien, Favoritenstr. 9/1921, 1040 Vienna, Austria
jakob.greilhuber@student.tuwien.ac.at, sophia.schober@tuwien.ac.at
{eiurlano|raidl}@ac.tuwien.ac.at

Abstract. The Roman Domination Problem is an NP-hard combinatorial optimization problem on an undirected simple graph. It represents scenarios where a resource shall be economically distributed over its vertices while guaranteeing that each vertex has either a resource itself or at least one neighbor with a sharable surplus resource. We propose several (meta-)heuristic approaches for solving this problem. First, a greedy construction heuristic for quickly generating feasible solutions is introduced. A special feature of this heuristic is an optional advanced tiebreaker. This construction heuristic is then randomized and combined with a local search procedure to obtain a greedy randomized adaptive search procedure (GRASP). As an alternative, we further propose a simulated annealing (SA) algorithm to improve the solutions returned by the construction heuristic. As we observe different pros and cons for the GRASP and the SA, we finally combine them into a simulated annealing hybrid, which interleaves phases of greedy randomized construction and phases of simulated annealing. All algorithms are empirically evaluated on a large set of benchmark instances from the literature. We compare to an exact mixed integer linear programming model that is solved by Gurobi as well as to a variable neighborhood search from the literature. In particular the simulated annealing hybrid turns out to yield on average the best results, making it a new state-of-the-art method for the Roman domination problem.

Keywords: Roman Domination Problem · Metaheuristics · GRASP · Simulated Annealing

1 Introduction

The *Roman Domination Problem (RDP)* is a combinatorial optimization problem on graphs, formally introduced in ReVelle and Rosing [25]. It is related to the classical dominating set problem and originates from the following scenario: a Roman emperor might wonder how many legions it takes to ensure that all provinces of the empire can be defended against a single attack, without leaving any province vulnerable. A province in the empire is considered defended if a legion is stationed in it or if there is a neighboring province with two stationed legions, as such a neighbor can send one of its legions to help the attacked province. More formally, the problem is defined as follows. Given an undirected simple graph $G = (V, E)$ with vertex set V (corresponding to the provinces) and edge set E (representing the neighborhoods), a labeling function $f : V \to \{0, 1, 2\}$ assigns each vertex a label (the number of stationed legions). If this is done in such a way that every vertex with label 0 has at least one neighbor labeled 2, this function is called a *Roman Domination Function (RDF)* [4]. The weight of this function is given by $|f| = \sum_{v \in V} f(v)$, and the lowest weight of any RDF of G is the *Roman domination number* of G. The objective of the RDP is to find an RDF of lowest weight. Beyond the historical background in military strategy planning, practical applications can occasionally be found more generally when an area represented as a graph shall be covered with a minimum amount of some resource and neighboring vertices may share units of the resource. For instance, Pagourtzis et al. [23] analyze different problem formulations concerning optimal server placement and highlight that one of these corresponds to the RDP. Similarly, Ghaffari et al. [10] describe how the RDP can be used in the deployment of wireless sensor networks.

In terms of complexity, the RDP is known to be NP-hard [6]. Thus, optimally solving the problem is in general not possible in polynomial time, unless $P = NP$, which creates a desire for heuristic approaches that can produce reasonably good solutions in a short amount of time also

* The first two authors contributed equally.

for large instances. We first introduce a greedy construction heuristic, which is then randomized and extended to a *greedy randomized adaptive search procedure (GRASP)*. Moreover, we propose a *simulated annealing (SA)* algorithm, as well as a *simulated annealing hybrid (SAH)* that combines the randomized construction heuristic and simulated annealing approaches. All these algorithms are experimentally evaluated and compared to a former *variable neighborhood search (VNS)* from Ivanović and Urošević [15] on benchmark instances from the literature. Results indicate that SAH performs best in our tests.

The next Section 2 surveys related work. In Sections 3 to 5, we introduce the construction heuristic, the GRASP, the SA, and the SAH approaches, respectively. Results are discussed in Section 6. Finally, we give concluding remarks in Section 7.

2 Related Work

The RDP is inspired by the strategy that the Roman emperor Constantine proposed to defend the Roman empire, which is discussed in an article by Stewart [27] from 1999. ReVelle and Rosing [25] formally define the problem and propose the first *binary linear programming* formulation for it. Cockayne et al. [4] introduce the term Roman domination, and give a variety of theoretical results for the problem, such as the observation that the Roman domination number is at least the size of the domination number (i.e., the cardinality of the smallest dominating set) and at most twice this number for any graph. They also characterize the graphs with a Roman domination number that exceeds the domination number by at most two. Xing et al. [29], on the other hand, provide a characterization of graphs for which the difference between Roman domination number and domination number equals a fixed constant not smaller than two.

Dreyer [6] dedicates a chapter of their doctoral thesis to theoretical findings regarding the problem, also providing a proof for the NP-hardness of the problem. Favaron et al. [7] provide an optimal upper bound for the Roman domination number of connected graphs, as well as bounds for the number of vertices labeled with 0, 1, or 2 in an RDF of minimal weight. Shang and Hu [26] consider the problem on unit disk graphs. They provide an approximation algorithm and a polynomial-time approximation scheme for graphs of this class. Moreover, they show that the RDP is NP-hard on unit disk graphs. Bounds and exact results for the Roman domination number on cardinal products of paths, cycles, and general graphs are given by Klobučar and Puljić [18, 19]. Peng and Tsai [24] prove that the problem can be solved in linear time on graphs of bounded treewidth. Liedloff et al. [20, 21] show that the RDP can be solved in linear time on interval graphs and cographs, among other algorithmic results. Currò [5] dedicates their doctoral thesis to the RDP on grid graphs, proving lower and upper bounds on the Roman domination number of such graphs.

A wide variety of related domination problems have also been investigated by researchers over the past decades, including the *Weak Roman Domination Problem (WRDP)* devised by Henning and Hedetniemi [12], the *Signed Roman Domination Problem* introduced by Abdollahzadeh Ahangar et al. [1], the *Signed Total Roman Domination Problem* proposed by Volkmann [28] and the *Double Roman Domination Problem* [2].

Burger et al. [3] propose another binary programming formulation for the RDP. The formulations by ReVelle and Rosing [25] and Burger et al. [3] are examined by Ivanović [13], who further provides improved versions of both formulations.

The first heuristic approaches for the RDP appear to stem from Nolassi [22] and Currò [5]. In their doctoral theses, multiple construction heuristics are proposed and evaluated. Furthermore, Currò [5] presents genetic algorithms for the problem. Later, Ivanović and Urošević [15] propose a VNS algorithm for the RDP and the WRDP, and report mostly superior results on many instances in comparison to the earlier solution approaches. Moreover, Ghaffari et al. [10] describe two simple construction heuristics. The only other heuristic approach for the RDP we found in the literature is a genetic algorithm by Khandelwal et al. [16]. These authors, however, test their genetic algorithm on a different instance set than previous publications and do not reference earlier heuristic approaches. Therefore, we compare primarily to an exact mixed binary linear programming approach as well as to the VNS from [15]. Filipović et al. [9] provide heuristic and exact solution methods for the signed and signed total RDP variants.

Algorithm 1: Construction heuristic

```
   Input: Graph G = (V, E)
 1 d_u(v) ← |N[v]| ∀v ∈ V
 2 if using tiebreaker then
 3 |  d²_u(v) ← number of vertices with distance exactly 2 from v ∀v ∈ V
 4 end
 5 f(v) ← unlabeled ∀v ∈ V
 6 S ← V
 7 while ∃v ∈ S d_u(v) ≥ 2 do
 8 |  v ← vertex of S with largest d_u value (prefer vertices with lower d²_u value if tiebreaker is used)
 9 |  forall n ∈ N(v) do
10 |  |  if n is unlabeled then
11 |  |  |  f(n) ← 0
12 |  |  end
13 |  end
14 |  f(v) ← 2
15 |  S ← S \ {v}
16 end
17 label all unlabeled vertices 1
   // Post-processing: local improvement
18 forall v ∈ V with f(v) = 2 do
19 |  f(v) ← 0
20 |  if RDF condition violated for any u ∈ N(v) then
21 |  |  f(v) ← 2
22 |  else if RDF condition violated for v then
23 |  |  f(v) ← 1
24 |  end
25 end
26 return f
```

In general, the literature has been mostly focusing on theoretical results thus far. This lack of focus on heuristic algorithms for the RDP in the literature indicates that the potential of such algorithms may not have been exhausted yet and promising approaches might be left to discover, which motivates our work.

3 Greedy Construction Algorithm

We now present our (deterministic) greedy construction heuristic. It is inspired by the algorithms described by Ghaffari et al. [10]. We later realized that its core principle is shared by the "GainFactor" heuristic described by Currò [5], however, our algorithm differs from the latter by including a tiebreaker, using the unlabeled degree instead of the GainFactor, incorporating a label-reduction step in the end, and other smaller differences. The general idea of our construction heuristic is that assigning the label 2 to vertices of high degree is frequently a good decision. For a vertex $v \in V$, $N(v) = \{u \in V | uv \in E\}$ denotes the open neighborhood of v, while we write $N[v] = N(v) \cup \{v\}$ for the closed neighborhood. Vertex w dominates vertex v if $w \in N[v]$ and w is labeled 2 or $w = v$ and w is labeled 1. A vertex v is dominated if it is dominated by some vertex w.

Algorithm 1 shows the heuristic in pseudocode. The procedure takes a graph G as input. For a vertex $v \in V$, $d_u(v)$ represents the number of unlabeled vertices in $N[v]$. This number is updated accordingly whenever changes to the labeling function are made. Intuitively, $d_u(v)$ is the number of vertices that are not yet dominated in the current partial solution, but that would be dominated when assigning the label 2 to v. Our algorithm can be used with an optional tiebreaker, which will be described in Section 3.1. If this tiebreaker is used, one must also manage the d^2_u values, for each vertex $v \in V$, where $d^2_u(v)$ is the number of unlabeled vertices with distance exactly two from v. In other words, $d^2_u(v)$ is the number of unlabeled vertices that are reachable in two hops from v, but that are not reachable with less than two hops. After initializing d_u and possibly d^2_u, the label $f(v)$ of each node $v \in V$ is set to *unlabeled* and a set S is initialized with all nodes to be

processed. As long as there exists some vertex in S, such that labeling it with 2 would dominate at least two vertices that are undominated so far, the following is iterated. A vertex with largest d_{u} is selected (potentially breaking ties by preferring vertices of lower d_{u}^2-value), labeled with 2, and all its still unlabeled neighbors are labeled with 0. Vertices with label 2 are removed from the set of vertices S. Once no vertex v with $d_{\mathrm{u}}(v) \geq 2$ exists anymore, labeling further vertices with 2 is not beneficial anymore. Therefore, the loop is finished and the algorithm proceeds by labeling all remaining unlabeled vertices with 1. Thereafter, the resulting labeling f is already an RDF, but some vertices may have a higher label than necessary. As a post-processing step, the algorithm therefore performs a local improvement by attempting to reduce the label of each vertex so far labeled with 2. Finally, the obtained locally optimal labeling function f is returned as a solution. To improve the efficiency of our post-processing step, we keep track of the number of vertices dominating every vertex. Using a heap data structure, the algorithm can be implemented to run in time $O(\Delta|V|\log(|V|)) = O(|V|^2\log(|V|))$ without the tiebreaker, where Δ is the maximum degree of the graph (we assume $\Delta > 0$).

3.1 Tiebreaker

Since many vertices with the same d_{u} values can exist, one may want to employ a meaningful tiebreaker. Our tiebreaker is inspired by the one used by Ghaffari et al. [10] in their second greedy algorithm. We also attempted to implement their approach, however, we found it difficult to do so efficiently in conjunction with a heap. In case of a tie, we select a vertex that has the lowest number of unlabeled vertices with distance exactly two from it. The intuition behind this tiebreaker is that a vertex v with a high d_{u} value and a low d_{u}^2 value may have many neighbors that cannot be dominated well by vertices other than v. These values adapt to changes made by the greedy algorithm and the required bookkeeping can easily be implemented in a performant way. Efficiently computing the vertices with distance exactly two for every vertex can initially be done via breadth-first-search or by squaring the adjacency matrix of the graph. We opted for the first approach. This computation generally dominates the run-time of the algorithm, but the construction heuristic still terminates within seconds on all test instances in our computational experiments discussed in Section 6.

4 GRASP

GRASP is a prominent metaheuristic consisting of a construction and a local improvement phase [8]. These two phases are executed repeatedly until some stopping criterion is met, and the best found solution is returned. Our GRASP approach utilizes a randomized version of the above greedy heuristic and a k-flip neighborhood local search. The k-flip neighborhood is inspired by an abstract view of the RDP that is used by Currò [5] to encode individuals in their genetic algorithms. In this view, solutions are encoded as binary strings where a bit i is set to one iff vertex i has label 2, and set to zero otherwise. To obtain feasible solutions from such encodings, all vertices with their corresponding bit set to one receive the label 2, all vertices with their corresponding bit set to zero that are adjacent to a vertex labeled 2 receive the label 0 and all remaining vertices receive the label 1. Flipping a bit therefore causes a vertex to be relabeled either from 2 to a lower label, or the other way around, implicitly also adjusting the labels of affected neighbors with labels different from 2. In our randomized version of Algorithm 1, we do not always select the vertex of S with the highest d_{u} value, but use a threshold based approach instead. This selection procedure is shown in Algorithm 2. We first determine the maximum unlabeled degree of any vertex of the set S, d_{u}^*, and then form a *restricted candidate list (RCL)*, consisting of all vertices $v \in S$ with $d_{\mathrm{u}}(v) \geq \tau d_{\mathrm{u}}^*$, where $\tau \in [0, 1]$ is a strategy parameter. In the end, a vertex is selected uniformly at random from the *RCL* and returned. If the returned vertex v has $d_{\mathrm{u}}(v) < 2$, it is skipped, since labeling such a vertex 2 would not be worth-while. Note that the randomized version of the construction algorithm is performed without the tiebreaker.

5 Simulated Annealing

Simulated Annealing (SA) is another widely used metaheuristic, introduced under its current name by Kirkpatrick et al. [17]. Our SA approach computes an initial solution using our greedy construction heuristic with the tiebreaker, and then tries to refine it by using the k-flip neighborhood

Algorithm 2: GRASP vertex selection

Input: Vertex set S, unlabeled vertex degrees d_{u}
$d^*_{\mathrm{u}} = \max_{v \in S}(d_{\mathrm{u}}(v))$
$RCL = \{v \in S \mid d_{\mathrm{u}}(v) \geq \tau d^*_{\mathrm{u}}\}$
return *vertex from RCL selected uniformly at random*

Algorithm 3: Simulated Annealing

Input: Graph $G = (V, E)$
$x_{\mathrm{best}} \leftarrow$ deterministic construction heuristic with tiebreaker (G)
$x \leftarrow x_{\mathrm{best}}$ // `current solution`
$T \leftarrow T_{\mathrm{init}}$ // `current temperature`
while *time limit not exceeded* **do**
  flipVertices $\leftarrow k$ vertices selected uniformly at random with replacement
  remove duplicates in *flipVertices*
  $x' \leftarrow$ solution resulting from flipping the *flipVertices* in x
  if $|x'| < |x|$ **then**
    $x \leftarrow x'$
    if $|x'| < |x_{best}|$ **then**
      $x_{\mathrm{best}} \leftarrow x'$
    end
  else
    $d \leftarrow |x'| - |x|$
    if (*random value* $\in [0, 1)) < e^{-d/T}$ **then**
      $x \leftarrow x'$
    end
  end
  if T *has not changed in* $|V|^2$ *iterations* **then**
    $T \leftarrow \alpha \cdot T$
  end
  if $e^{-2/T} < \beta$ *and* $|x|$ *not improved in last* $k \cdot \phi$ *iterations* **then**
    $T \leftarrow \frac{-2k}{\ln \gamma}$ // `reheating`
  end
end
return x_{best}

structure in the usual SA fashion. This procedure is shown in Algorithm 3. It uses the following additional strategy parameters: an initial temperature T_{init}, a geometric cooling factor α, a threshold β for the minimum probability of accepting worse moves, a parameter γ controlling the probability of accepting worse moves after reheating, a number ϕ of steps without improvement that is used to determine when reheating should occur, and a time limit for termination.

In each iteration, our procedure samples k vertices uniformly at random, and then proceeds to remove duplicate entries from the sampled vertices. We sample with replacement so that also moves flipping less than k vertices are possible. These up to k vertices are then flipped in the current solution x to generate a new neighboring solution x'. This solution is accepted if it is better, or, in traditional simulated annealing fashion, with random probability depending on the solution quality as well as the current temperature T if it is worse. The temperature is cooled down by geometric cooling with a factor of α every $|V|^2$ iterations. Once the temperature has fallen to a point where the probability of accepting a move that increases the objective value by two is smaller than β, and the objective value of the current solution has not improved in the last $k \cdot \phi$ iterations, a reheating is performed such that worse moves are accepted again with a higher probability. As an overall stopping criterion, we use a maximum time limit, but also other stopping conditions, e.g., based on convergence, are conceivable.

As will be shown in more detail in Section 6, this algorithm performed well in our experiments, especially on graphs with substantial structure, like grid and net graphs. However, even though the SA outperformed Gurobi on large instances with less structure, i.e., randomly constructed graphs,

the GRASP from the last section yielded in general slightly better solutions on these instances. This inspired us to investigate a combination we call *Simulated Annealing Hybrid* (SAH) to possibly get the best of both worlds: We decided to split the algorithm into phases of randomized greedy construction and phases of refining the best found greedy solution through simulated annealing. These two phases are executed in an alternating fashion as shown in Algorithm 4. In the randomized greedy phase, we first generate an initial solution by running our deterministic greedy construction using the tiebreaker, and then repeatedly generate solutions using the randomized greedy construction with a threshold of τ. Since the randomized greedy phase will usually only be run a handful of times, running the greedy algorithm using the tiebreaker once per phase is not an issue in terms of run-time. The best solution found in the randomized greedy construction phase is then used as the initial solution for the SA phase. This approach may improve upon the previous ones by combining algorithms that are strong on different instance types, and by potentially generating more diverse initial solutions for the simulated annealing phase. Both phases are executed multiple times, effectively making this a multi-start approach. Even though we did not do this in our experiments, the algorithm can easily be parallelized by running each pair of random construction and simulated annealing phases on different cores.

6 Experimental Evaluation

In this section, we present the results of computational experiments that we conducted with the proposed algorithms. All methods were implemented in Julia 1.8.5 and performed on a cluster with Intel Xeon E5540 quad-core CPUs with 2.53GHz and 24 GB RAM. Our benchmark instance set consists of instances already used by Currò [5], Ivanović [14], Ivanović and Urošević [15], and Filipović et al. [9]. Thus, these instances appear to be the somewhat standard instances for the empirical evaluation of algorithms for Roman domination problems. The instance set includes graphs of different classes and sizes, with the largest graph having 1,000 vertices and around 450,000 edges. More specifically, there are six different types of graphs: grid, random, bipartite, net, planar, and recursive. We evaluate the performance of our greedy construction heuristic when used with and without the tiebreaker, as well as our GRASP, SA, and SAH approaches. Moreover, we compare to the VNS from Ivanović and Urošević [15] and the *mixed binary integer linear programming (MBIP)* formulation $\mathcal{BVV}_{\text{Imp2}}$ by Ivanović [13] solved with the Gurobi 10.0.0 mixed integer linear programming solver [11]. The lower bounds found by Gurobi are used to express the qualities of solutions obtained by the heuristic approaches in terms of percentage gaps. If z denotes the value of a heuristic solution and z_{lb} the corresponding lower bound obtained from Gurobi, then the percentage gap is calculated as $100\% \cdot \frac{z - z_{lb}}{z}$.

Time-gap cumulative distribution plots are used for comparison, in which the y-axis indicates how many solutions were solved, whereas the time it took to solve instances to optimality and —if not possible within the time limit— the relative optimality gap are depicted on the x-axis. This way, one can observe how many solutions were solved to proven optimality within the time-limit, as well as the quality of all obtained solutions when compared to the lower bound obtained via the MBIP model. The results for the VNS algorithm are taken from the respective paper [15], and, thus, runtimes cannot directly be compared. Full results of the experiments and the problem instances can be found online[1].

6.1 Tiebreaker

In order to examine the performance of the (deterministic) greedy construction heuristic with and without the tiebreaker, we performed one run of each variant on all instances. Note that, even when using the tiebreaker, any remaining ties are broken arbitrarily in a deterministic fashion. Figure 1 displays obtained results in a time-gap plot. Moreover, Figure 2 plots runtimes with the tiebreaker (Greedy+TB) against the runtimes without the tiebreaker (Greedy) for each instance. A summary of the results for each type of benchmark instance is given in Table 1. We show these data in the same fashion as Filipović et al. [9] display their results on (a subset of) the instances. The table lists the number of instances that were solved to proven optimality (#opt), the number

[1] https://www.ac.tuwien.ac.at/research/problem-instances/#Roman_Domination_Problem

Algorithm 4: Simulated Annealing Hybrid

Input: Graph $G = (V, E)$
$x_{\text{best}} \leftarrow$ *nothing*
while *time limit not exceeded* **do**
  $x_{\text{best\_greedy}} \leftarrow$ deterministic construction heuristic with tiebreaker (G)
  if x_{best} = *nothing* **then**
    $x_{\text{best}} \leftarrow x_{\text{best\_greedy}}$
  end
  while *randomized greedy construction phase* **do**
    $x' \leftarrow$ randomized construction heuristic using threshold τ (G)
    if $|x'| < |x_{\text{best\_greedy}}|$ **then**
      $x_{\text{best\_greedy}} \leftarrow x'$
    end
    if $|x'| < |x_{\text{best}}|$ **then**
      $x_{\text{best}} \leftarrow x'$
    end
  end
  $x \leftarrow x_{\text{best\_greedy}}$
  $T \leftarrow T_{\text{init}}$ // `current temperature`
  while *simulated annealing phase* **do**
    flipVertices $\leftarrow k$ vertices selected uniformly at random with replacement
    remove duplicates in *flipVertices*
    $x' \leftarrow$ solution resulting from flipping the *flipVertices* in x
    if $|x'| < |x|$ **then**
      $x \leftarrow x'$
      if $|x'| < |x_{best}|$ **then**
        $x_{\text{best}} \leftarrow x'$
      end
    else
      $d \leftarrow |x'| - |x|$
      if (*random value* $\in [0, 1)) < e^{-d/T}$ **then**
        $x \leftarrow x'$
      end
    end
    if T *has not changed in* $|V|^2$ *iterations* **then**
      $T \leftarrow \alpha \cdot T$
    end
    if $e^{-2/T} < \beta$ *and* $|x|$ *not improved in last* $k \cdot \phi$ *iterations* **then**
      $T \leftarrow \frac{-2k}{\ln \gamma}$ // `reheating`
    end
  end
end
return x_{best}

of times the approach achieved the best solution of the displayed approaches (#best), the mean objective value, the mean time to the best solution found by the approach, as well as the mean percentage gap. We remark that values listed under #opt do not necessarily represent the numbers of instances for which optimal solutions were found, but only the numbers of instances for which optimality could be proven by means of the lower bounds from Gurobi.

Running the construction heuristic with the tiebreaker increases the run-time on average by a factor of about ten, but significantly more instances were solved to proven optimality when using it. Moreover, out of the 623 total instances of the benchmark set, the algorithm with the tiebreaker managed to find better solutions for 211 instances, while reporting slightly worse solutions for only 132 instances and yielding equally good results for the remaining 280 instances. Obtained average gaps were about 0.5% lower when using the tiebreaker. The largest difference in solution quality can be observed on the four tested net graphs. Here, the construction heuristic with the tiebreaker manages to always find the optimal solution, while the basic version obtained significantly worse

Table 1. Computational results of the greedy construction heuristic with and without tiebreaker.

Method	Measure	Grid (172 inst.)	Bipartite (135 inst.)	Net (4 inst.)	Planar (17 inst.)	Random (288 inst.)	Recursive (7 inst.)	All (623 inst.)
Greedy+TB	#opt	13	19	4	3	61	7	107
	#best	119	97	4	15	249	7	491
	mean value	46.60	66.63	80.50	20.65	35.17	56.43	45.28
	mean time (ms)	0.72	44.73	3.22	63.74	227.83	52.83	117.57
	mean gap (%)	10.35	17.75	0.00	24.38	22.87	0.00	17.94
Greedy	#opt	5	15	0	3	58	7	88
	#best	88	85	0	14	218	7	412
	mean value	47.02	66.99	92.50	20.71	35.47	56.43	45.69
	mean time (ms)	0.50	5.64	1.18	5.10	24.01	2.33	12.63
	mean gap (%)	11.37	17.69	12.19	24.48	23.17	0.00	18.43

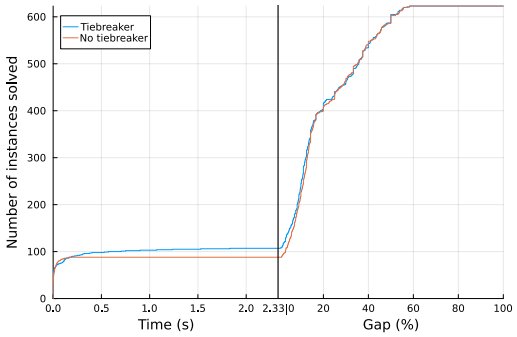


Fig. 1. Time-gap plot of the greedy construction heuristic with and without the tiebreaker.

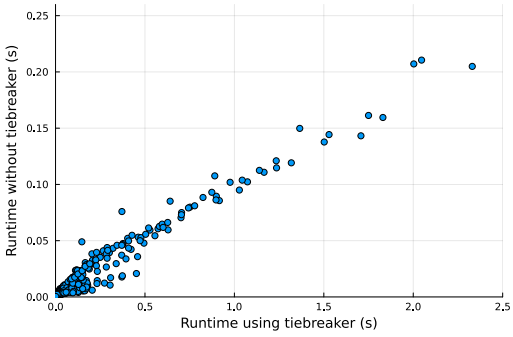


Fig. 2. Scatter plot for the run-times of the greedy construction heuristic with and without tiebreaker.

results. We conclude that using the tiebreaker within the construction heuristic can make sense, especially if it is used on graphs with a similar structure as the tested net graphs or if it is used sparingly, such that the added runtime has no large impact overall. On the other hand, it may not be well suited for approaches in which the greedy-algorithm is run many times, as one may add significant run-time for only small improvements.

6.2 Metaheuristic Approaches

The performance of our GRASP, SA, and SAH was evaluated with a two-hour time limit on each of our 623 problem instances. According to preliminary tests we found the following parameter settings to be robust choices, which we employed in all successive tests discussed here. The GRASP algorithm was run with a threshold value of $\tau = 0.9$ and uses the 1-flip neighborhood structure in conjunction with the next-improvement step-function for the improvement phase. For SA, we chose the parameter values $k = 2$, $\alpha = 0.95$, $\beta = 10^{-6}$, $\gamma = 10^{-4}$, and $\phi = 2 \cdot 10^4$. The initial temperature was set to $\frac{-2}{\ln 0.03}$. The same parameter values were chosen for the SAH algorithm, and we ran the randomized greedy phase for 2 minutes followed by 8 minutes of the simulated annealing phase over 12 full rounds, for a total run-time of 2 hours. In the randomized greedy phase, we used a threshold value of $\tau = 0.9$ for the randomization of the construction heuristic.

The main results of the experiments are summarized in Table 2. Here included are also the results of Gurobi on the MBIP model. Note that for this latter approach, we do not display the average time it took to find the best solution, but the average time reported by Gurobi (for a given instance, this is either the time it took to find and prove optimality, or the time limit). Thus, these times cannot be directly compared to the times of the heuristic approaches. Moreover, the table includes the results of Greedy+TB again for a direct comparison. We further display the results of GRASP, SA, SAH, and MBIP also in the form of time-gap plots for all instances as well as only selected subsets in Figure 3.

Table 2. Computational results of the MBIP solved by Gurobi, the greedy construction heuristic with tiebreaker and our metaheuristics GRASP, SA, and SAH.

Method	Measure	Grid (172 inst.)	Bipartite (135 inst.)	Net (4 inst.)	Planar (17 inst.)	Random (288 inst.)	Recursive (7 inst.)	All (623 inst.)
MBIP	#opt	172	68	4	8	123	7	382
	#best	172	84	4	10	162	7	439
	mean value	41.30	63.08	80.50	19.53	33.72	56.43	42.34
	mean time (s)	4.57	3745.79	0.01	4041.80	4356.68	0.03	2937.25
	mean gap (%)	0.00	12.92	0.00	18.33	19.72	0.00	12.42
Greedy+TB	#opt	13	19	4	3	61	7	107
	#best	13	21	4	3	91	7	139
	mean value	46.60	66.63	80.50	20.65	35.17	56.43	45.28
	mean time (s)	<0.01	0.04	<0.01	0.06	0.23	0.05	0.12
	mean gap (%)	10.35	17.75	0.00	24.38	22.87	0.00	17.94
GRASP	#opt	151	53	2	8	111	7	332
	#best	151	102	2	17	259	7	538
	mean value	41.72	62.72	85.25	18.71	32.85	56.43	41.99
	mean time (s)	292.02	889.67	755.81	508.21	434.10	<0.01	492.80
	mean gap (%)	0.38	12.14	3.21	16.67	17.82	0.00	11.45
SA	#opt	172	66	4	8	117	7	374
	#best	172	102	4	12	193	7	490
	mean value	41.30	62.59	80.50	19.24	33.27	56.43	42.02
	mean time (s)	2.81	787.28	<0.01	998.69	431.47	0.05	398.09
	mean gap (%)	0.00	12.72	0.00	17.69	19.49	0.00	12.25
SAH	#opt	172	66	4	8	123	7	380
	#best	172	124	4	17	262	7	586
	mean value	41.30	62.27	80.50	18.71	32.80	56.43	41.72
	mean time (s)	36.16	540.57	<0.01	715.74	470.69	0.06	364.24
	mean gap (%)	0.00	11.96	0.00	16.67	17.91	0.00	11.33

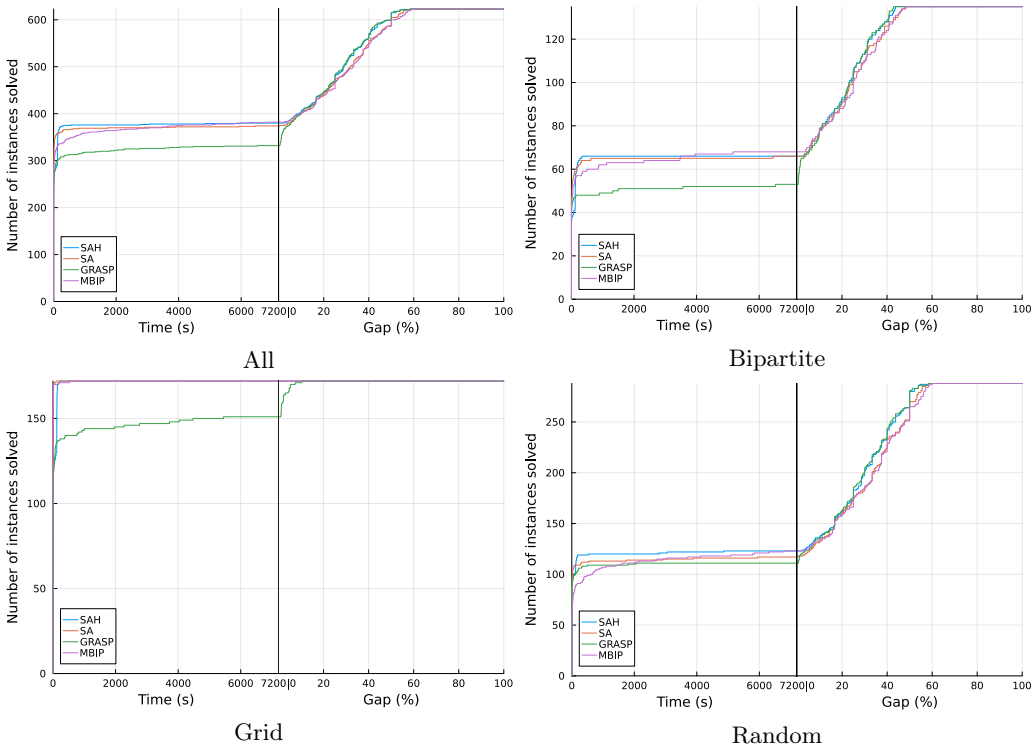


Fig. 3. Time-gap plots comparing the approaches on different graph classes.

We can observe that the metaheuristic approaches obtained better results than the greedy-heuristic, achieving much higher scores in the #opt and #best metrics. The only exceptions to this are the

Table 3. Comparison with the VNS from Ivanović and Urošević [15].

Algorithm	#opt	mean value	mean time (s)	mean gap (%)
VNS	218	41.004	113.171	0.163
GRASP	203	41.182	252.933	0.380
SA	230	40.771	55.077	0.048
SAH	231	40.766	33.238	0.000

net and recursive graphs, on which the greedy algorithm already obtains optimal solutions. The SA approach is strong on grid graphs, achieving the optimal solution on all of them, whereas GRASP only managed to optimally solve 151 out of 172 instances. Even compared to MBIP, the SA approach showed promising performance on this graph class. On the random graph class GRASP achieves the best solutions on 259 instances, and SA on only 193, making GRASP the better choice for these instances. SAH is the most prospective approach in terms of solution quality, finding the best solution on 586 instances, better than the MBIP approach that manages 439, GRASP that manages 538, and SA that manages 490. Furthermore, there were only two instances which the MBIP approach could solve to proven optimality, and on which SAH was not able to find an optimum. The mean time to converge to the best solution of a run is lower for SAH than for all the other metaheuristic approaches on average over all instances, indicating that SAH can in general obtain better solutions on the tested instances without needing (much) more time. When an algorithm for a specific graph class, e.g., grid graphs is needed, other tested approaches (in the case of grid graphs, the basic simulated annealing algorithm) may however obtain solutions of similar quality in less time.

Finally, we compare our algorithms also to the VNS from Ivanović and Urošević [15] in Table 3. In their article, only results for instances for which the optimal solution is known are reported, and thus we also had to restrict the comparison to the respective subset benchmark instances. Note that in contrast to our other displayed results, the lower bounds used now do not stem from our MBIP results, but from [15]. Ivanović and Urošević [15] employ a time limit of two hours together with an early stopping criterion, and report the time it took to obtain the best solution found as well as the obtained solution value. Their CPU is slightly slower than the one we used, having a frequency of 2.4GHz compared to ours with 2.53GHz. Unlike for our experiments, Ivanović and Urošević [15] report results for the best out of 20 independent runs per instance. Nevertheless, our SA and SAH show clearly better performance on these instances, solving more of them to optimality in a much shorter mean time. Especially SAH performed particularly well, solving all instances optimally in the shortest mean time of all tested approaches.

Overall, SAH successfully manages to combine the strengths of the simulated annealing and the randomized construction heuristic, and is an algorithm with excellent practical performance on all considered instances.

7 Conclusion

We investigated several heuristic approaches to tackle the Roman Domination Problem and also compared them to a mixed integer linear programming model solved by Gurobi as well as the VNS by Ivanović and Urošević [15]. We observed that simulated annealing and GRASP can be applied as efficient metaheuristics for the problem, as they provided stronger results over our benchmark instances than the exact and greedy algorithms. The simulated annealing approach showed especially good results on grid graphs, whereas GRASP prevailed on a class of randomly generated graphs. Motivated by the individual strengths of these two, we came up with a simulated annealing hybrid, that interleaves phases of randomized solution construction with solution refinement according to simulated annealing. In our experiments, this hybrid manages to achieve the overall best results out of all considered approaches. When comparing our SAH algorithm to the VNS from Ivanović and Urošević [15], our hybrid managed to solve all considered instances to proven optimality, and this on average in less time.

Given the impressive results obtained by the MBIP, an interesting direction for future work is to investigate hybrid metaheuristics that make use of a mixed integer linear programming solver

for solving smaller subproblems, such as in large neighborhood search. This may be particularly appealing to address huge instances. Moreover, there are many variants of the basic Roman domination problem, such as the signed, double, or weak variants. Adapting our approaches for these seems to be partly easy, but there are also some more challenging questions that would need to be solved. Finally, evaluating the application of the proposed algorithms on an actual use case from practice would be interesting.

Acknowledgments. We acknowledge the financial support of this project by Austria's Agency for Education and Internationalisation under grant BA05/2023. This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF), grant W1260-N35.

References

1. Abdollahzadeh Ahangar, H., Henning, M.A., Löwenstein, C., Zhao, Y., Samodivkin, V.: Signed Roman domination in graphs. Journal of Combinatorial Optimization **27** (2014) 241–255
2. Beeler, R.A., Haynes, T.W., Hedetniemi, S.T.: Double Roman domination. Discrete Applied Mathematics **211** (2016) 23–29
3. Burger, A.P., De Villiers, A.P., Van Vuuren, J.H.: A binary programming approach towards achieving effective graph protection. In: Proceedings of the 2013 ORSSA Annual Conference, ORSSA. (2013) 19–30
4. Cockayne, E.J., Dreyer, P.A., Hedetniemi, S.M., Hedetniemi, S.T.: Roman domination in graphs. Discrete Mathematics **278** (2004) 11–22
5. Currò, V.: The Roman domination problem on grid graphs. PhD thesis, Università degli Studi di Catania (2014)
6. Dreyer, P.: Applications and variations of domination in graphs. PhD thesis, Rutgers, The State University of New Jersey (2000)
7. Favaron, O., Karami, H., Khoeilar, R., Sheikholeslami, S.M.: On the Roman domination number of a graph. Discrete Mathematics **309** (2009) 3447–3451
8. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. Journal of Global Optimization **6** (1995) 109–133
9. Filipović, V., Matić, D., Kartelj, A.: Solving the signed Roman domination and signed total Roman domination problems with exact and heuristic methods (2022) arXiv:2201.00394.
10. Ghaffari, F., Bahrak, B., Shariatpanahi, S.P.: A novel approach to partial coverage in wireless sensor networks via the Roman dominating set. IET Networks **11** (2022) 58–69
11. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023)
12. Henning, M.A., Hedetniemi, S.T.: Defending the Roman empire–a new strategy. Discrete Mathematics **266** (2003) 239–251
13. Ivanović, M.: Improved mixed integer linear programing formulations for Roman domination problem. Publications de l'Institut Mathematique **99** (2016) 51–58
14. Ivanović, M.: Improved integer linear programming formulation for weak Roman domination problem. Soft Computing **22** (2018) 6583–6593
15. Ivanović, M., Urošević, D.: Variable neighborhood search approach for solving Roman and weak Roman domination problems on graphs. Computing and Informatics **38** (2019) 57–84
16. Khandelwal, A., Srivastava, K., Saran, G.: On Roman domination of graphs using a genetic algorithm. In Singh, V., Asari, V.K., Kumar, S., Patel, R.B., eds.: Computational Methods and Data Engineering. Volume 1227 of Advances in Intelligent Systems and Computing., Springer (2021) 133–147
17. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220** (1983) 671–680
18. Klobučar, A., Puljić, I.: Some results for Roman domination number on cardinal product of paths and cycles. Kragujevac Journal of Mathematics **38** (2014) 83–94
19. Klobučar, A., Puljić, I.: Roman domination number on cardinal product of paths and cycles. Croatian Operational Research Review **6** (2015) 71–78
20. Liedloff, M., Kloks, T., Liu, J., Peng, S.L.: Efficient algorithms for Roman domination on some classes of graphs. Discrete Applied Mathematics **156** (2008) 3400–3415
21. Liedloff, M., Kloks, T., Liu, J., Peng, S.L.: Roman domination over some graph classes. In Kratsch, D., ed.: Graph-Theoretic Concepts in Computer Science. Volume 3787 of Lecture Notes in Computer Science., Springer (2005) 103–114
22. Nolassi, S.M.: Algoritmi euristici per il problema della dominazione romana. PhD thesis, Università degli Studi di Catania (2013)

23. Pagourtzis, A., Penna, P., Schlude, K., Steinhöfel, K., Taylor, D.S., Widmayer, P.: Server placements, Roman domination and other dominating set variants. In: Foundations of Information Technology in the Era of Network and Mobile Computing: IFIP 17th World Computer Congress—TC1 Stream/2 nd IFIP International Conference on Theoretical Computer Science (TCS 2002) August 25–30, 2002, Montréal, Québec, Canada, Springer (2002) 280–291
24. Peng, S.L., Tsai, Y.H.: Roman domination on graphs of bounded treewidth. In: Proceedings of the 24th Workshop on Combinatorial Mathematics and Computation Theory. (2007) 128–131
25. ReVelle, C.S., Rosing, K.E.: Defendens imperium Romanum: A classical problem in military strategy. The American Mathematical Monthly **107** (2000) 585–594
26. Shang, W., Hu, X.: The Roman domination problem in unit disk graphs. In Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A., eds.: Computational Science – ICCS 2007. Volume 4489 of Lecture Notes in Computer Science., Springer (2007) 305–312
27. Stewart, I.: Defend the Roman empire! Scientific American **281** (1999) 136–139
28. Volkmann, L.: Signed total Roman domination in graphs. Journal of Combinatorial Optimization **32** (2016) 855–871
29. Xing, H.M., Chen, X., Chen, X.G.: A note on Roman domination in graphs. Discrete Mathematics **306** (2006) 3338–3340

Feature Selection Problem: A Short Systematic Literature Review about its formulation

José Barrera[1] 0009-0007-9934-5250, Broderick Crawford[1] 0000-0001-5500-0188, Felipe Cisternas-Caneo[1] 0000-0001-7723-7012, Ricardo Soto[1] 0000-0002-5755-6929, and Giovanni Giachetti[2] 0000-0003-2809-5120

[1] Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile
{jose.becerra,broderick.crawford,ricardo.soto}@pucv.cl
felipe.cisternas.c@mail.pucv.cl
[2] Universidad Andres Bello, Chile
giovanni.giachetti@unab.cl

Abstract. This extended abstract presents the methodology employed in a Systematic Literature Review (SLR) that aims to investigate Feature Selection Problem (FSP) and Optimization Techniques. The study focuses on understanding the objective function, performance evaluation metrics, optimization techniques, and practical applications of FSP. The methodology section provides an overview of the systematic approach adopted for conducting the literature review.

Keywords: Short Systematic Literature Review · Feature Selection Problem · Objective Function

1 Methodology

Research Questions The research questions guide the Systematic Literature Review (SLR) [7] and help structure the investigation [3]. The following research questions were formulated: How is the objective function of the Feature Selection Problem (FSP) [6] defined in the literature? What are the performance evaluation metrics used in solving the FSP? What optimization techniques are employed to solve the FSP? What are the practical applications of FSP across different domains?

Search Strategy and Study Selection For conduct a comprehensive literature review, a systematic search strategy was developed. The search covered major scholarly databases, including SCOPUS, Web of Sciences, IEEE Xplore, ScienceDirect by Elsevier, Wiley, and SpringerLink. The keyword "feature selection" was used to retrieve relevant articles. The search was limited to English-language articles published between 2019 and April 20, 2023.

The study selection process involved multiple stages to identify pertinent articles. In one stage, the database was further refined by selecting only those articles that explicitly mentioned the "Feature Selection Problem" in their abstracts. This filtering process resulted in a subset of articles, which underwent additional filtering in subsequent stages. In another stage, specific inclusion and exclusion criteria were applied to select primary studies specifically related to feature selection and optimization. The inclusion criteria encompassed studies that focused on feature selection and optimization, provided a clear definition and explanation of the feature selection problem and the objective function used, discussed metrics employed for evaluating feature selection performance, explored techniques or algorithms for solving the feature selection problem, and presented practical applications of feature selection in various fields. Conversely, exclusion criteria were applied to studies that did not pertain to feature selection and optimization, comprised surveys, systematic literature reviews, or reviews solely focused on the topic of feature selection. Initially, a total of 20,343 documents were found. After the rigorous selection process, a final set of 171 documents met the inclusion criteria and were included in the review.

2 Partial Finding

This extended abstract provides a condensed overview of the main findings and insights derived from specific sections of the article. It offers a glimpse into the ongoing investigation of feature

selection within the context of optimization techniques. Based on the current progress, the following conclusions can be made:

Objective Function The literature analysis revealed different approaches to representing the objective function in feature selection [6]. These approaches can be summarized as follows:

1. **Single-objective functions:** These functions focus on optimizing a single evaluation metric associated with the classifier [10]. However, they overlook the important aspect of feature reduction. For instance, using accuracy or error rate as the sole objective function does not address the need for reducing the original dataset. Thus, there is a requirement to treat the feature selection problem as a multi-objective problem.
2. **Multi-objective functions:** Multi-objective functions consider both the classifier evaluation metric and a feature evaluation metric. Two types of multi-objective functions were identified:
   - *Weighted multi-objective functions:* These functions are complex to solve as they involve balancing multiple objectives simultaneously. Different weights are assigned to each objective to indicate their importance in solving the problem. Researchers have proposed various weighted multi-objective functions, where the weights determine the trade-off between classification performance and feature size [8]. However, altering the weights can lead to varying results, making it challenging to find the optimal balance.
   - *Pure multi-objective functions:* Pure multi-objective functions treat each objective independently without assigning weights. Each objective function is optimized separately[12]. This approach offers simplicity and independence but presents computational challenges. Recent trends in the literature indicate a preference for pure multi-objective functions due to their independence and potentially better results. However, implementing pure multi-objective functions requires careful consideration of computational complexities.

In summary, single-objective functions have limitations, while multi-objective functions, particularly pure multi-objective functions, offer advantages in feature selection. Striking the right balance between classifier evaluation metrics and feature evaluation metrics is crucial. However, addressing the implementation and computational complexities associated with multi-objective functions is an ongoing challenge in this research area.

Evaluation Metrics The metric-based analysis of the Feature Selection Problem revealed three main types of metrics: Classifier-related metrics, Metaheuristics-related metrics, and Feature-related metrics.

Classifier-related metrics: These metrics relate to the performance of the classifier in using the subset of features issued by the metaheuristic. In the systematic literature review, 10 different evaluation metrics related to the classifier were detected. The 10 metrics are the following: Accuracy [10][2][8][9][5], Error rate [12], Precision [2][9], Sensitivity/Recall [2][5], F-measure/F-score [2][9][5], Specificity [2][5], Negative predictive value (NPV) [5], Matthew's correlation coefficient (MCC) [9], False positive rate (FPR) [2], False negative rate (FNR) [2].

Metaheuristics-related metrics: These metrics are related to the pure performance of the metaheuristics. In the systematic literature review, 5 different evaluation metrics related to the metaheuristics were detected this metrics are: Fitness [10][2][8], Computational time [11], Hypervolume metric (HV) [12][11], Two-Set Coverage (TC) [12], Inverted Generational Distance (IGD) [11].

Feature-related metrics: These metrics are purely related to the reduction of the features of the original dataset. In the systematic literature review, 3 different evaluation metrics related to features were detected and those are: Feature Selected [2][12], Cost [12], Number of features selected [10][8][5][9].

Classifier The selection and utilization of classifiers significantly impact the accuracy and reliability of feature selection results. The k-nearest neighbor (k-NN) algorithm is the most commonly used classifier, followed by Support Vector Machine (SVM). Other classifiers, such as Naive Bayes (NB), Decision Tree (C4.5 - DT), Random Forest (RF), and Artificial Neural Network (ANN), have also been explored, demonstrating the diversity of algorithms employed in addressing the feature selection problem.

Practical Applications Optimization techniques in feature selection find diverse applications across different domains. The reviewed articles utilize various datasets, including popular ones from repositories like the UCI Repository and the ASU scikit-feature feature selection repository (e.g., "Ionosphere" and "Connectionist Bench (Sonar, Mines vs. Rocks)"). These datasets are widely used, indicating their relevance and suitability for feature selection research in different domains.

3 Intended Contributions

In this work-in-progress systematic literature review (SLR) on feature selection and optimization techniques, we have made significant progress in understanding the objective function, evaluation metrics, classifiers, and practical applications of feature selection. However, further research is needed to fully address the challenges associated with multi-objective functions and explore novel optimization techniques. This ongoing SLR aims to provide valuable insights and contribute to the advancement of feature selection methodologies as other similar investigation did [1, 4].

4 Acknowledgments

Broderick Crawford, Ricardo Soto and Felipe Cisternas-Caneo are supported by Grant DI Investigación Asociativa Interdisciplinaria/VINCI/PUCV/039.347/2023.

Broderick Crawford and Ricardo Soto are supported by Grant ANID/ FONDECYT/REGULAR/ 1210810.

Felipe Cisternas-Caneo is supported by National Agency for Research and Development (ANID)/ Scholarship Program/DOCTORADO NACIONAL/2023-21230203.

References

1. Prachi Agrawal, Hattan F Abutarboush, Talari Ganesh, and Ali Wagdy Mohamed. Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019). *Ieee Access*, 9:26766–26791, 2021.
2. Alanoud Alsaleh and Wojdan Binsaeedan. The influence of salp swarm algorithm-based feature selection on network anomaly intrusion detection. *IEEE Access*, 9:112466–112477, 2021.
3. Marcelo Becerra-Rozas, José Lemus-Romani, Felipe Cisternas-Caneo, Broderick Crawford, Ricardo Soto, Gino Astorga, Carlos Castro, and José García. Continuous metaheuristics for binary optimization problems: an updated systematic literature review. *Mathematics*, 11(1):129, 2022.
4. Tansel Dokeroglu, Ayça Deniz, and Hakan Ezgi Kiziloz. A comprehensive survey on recent metaheuristics for feature selection. *Neurocomputing*, 494:269–296, 2022.
5. Ahmed A. Ewees, Mohamed A. El Aziz, and Aboul Ella Hassanien. Chaotic multi-verse optimizer-based feature selection. *Neural Computing and Applications*, 31(4):991–1006, 2019.
6. Anil Jain and Douglas Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE transactions on pattern analysis and machine intelligence*, 19(2):153–158, 1997.
7. Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.
8. Jingwei Too, Abdul Rahim Abdullah, and Norhashimah Mohd Saad. A new quadratic binary harris hawk optimization for feature selection. *Electronics*, 8(10), 2019.
9. Jingwei Too, Abdul Rahim Abdullah, Norhashimah Mohd Saad, and Weihown Tee. Emg feature selection and classification using a pbest-guide binary particle swarm optimization. *Computation*, 7(1), 2019.
10. Yu Xue, Bing Xue, and Mengjie Zhang. Self-adaptive particle swarm optimization for large-scale feature selection in classification. *ACM Trans. Knowl. Discov. Data*, 13(5), sep 2019.
11. Nian Zhang, Yue Li, Zhiheng Sun, Xin Liu, Wei-Tung Chen, Der-Juinn Horng, and Kuei-Kuei Lai. Feature selection based on a large-scale many-objective evolutionary algorithm. *Computational Intelligence and Neuroscience*, 2021:9961727, 2021.
12. Yong Zhang, Shi Cheng, Yuhui Shi, Dun wei Gong, and Xinchao Zhao. Cost-sensitive feature selection using two-archive multi-objective artificial bee colony algorithm. *Expert Systems with Applications*, 137:46–58, 2019.

Ergodic Annealing: intelligent optimization under uncertainty

Carlo Baldassi, Fabio Maccheroni, Massimo Marinacci, Marco Pirazzini

Artificial Intelligence Lab, Bocconi University

Abstract

NP-hard decision problems in which the cost function is *a priori* unknown to the Decision Maker arise naturally in many applications. We present a novel algorithm, inspired by Simulated Annealing, that exploits the structural randomness of Metropolis exploration in order to simultaneously find the optimal solution and learn its cost. As benchmark cases, we test our algorithm on the Directed Steiner Tree and Traveling Salesman problems. Our results suggest that problems for which Simulated Annealing works with known costs can be efficiently attacked with our algorithm when costs are unknown.

Index Terms

Simulated Annealing, stochastic optimization, reinforcement learning.

This paper is dedicated to the memory of Erio Castagnoli whose first job at the University of Parma was at the "Centro di Calcolo" of the Faculty of Economics.

The authors thank Simone Cerreia-Vioglio, Daniele Durante, Salvatore Greco, and Piero Veronese for very useful discussions as well as PRIN (grant 2017CY2NCA) for financial support. The authors have no conflicts of interest to declare.

1 Introduction

The recent years and events lead to a massive development of content-oriented cloud services. The most popular and voluminous content offered in today's networks are videos that must be efficiently delivered to end customers. The objective of the service provider (root) is to optimize the delivery of content to its costumers (terminals). In this optimization problem the cost is usually assumed to be known (Figure 1, left graph). Yet, in reality it is often unknown because it depends on many stochastic factors, such as the traffic on the network, the level of demand, and so on (Figure 1, right graph).

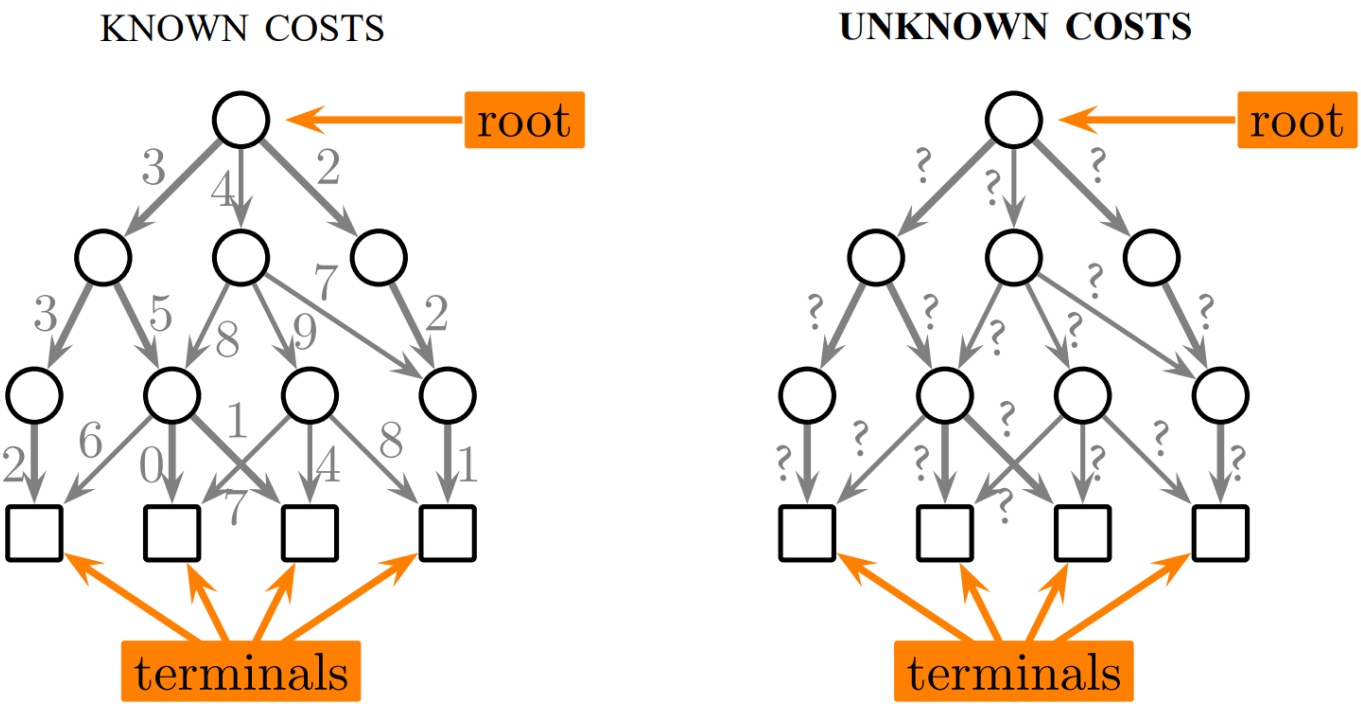


Fig. 1. Directed Steiner trees with known and unknown costs.

This is just an instance of a general decision problem in which, *ex ante*, the Decision Maker (DM) ignores the payoff of the available actions and has limited resources to discover it. In this note, we show that a natural modification of the Simulated Annealing algorithm of Kirkpatrick et al. (1983) permits to efficiently solve this conceptually non-trivial problem.

2 Known payoffs: Metropolis Algorithm and Simulated Annealing

Let A be a finite set of actions and $u : A \to \mathbb{R}$ a **known** objective function that the DM aims to maximize.[1] When the number of alternatives is small, the DM can just use a brute force comparison-and-elimination algorithm that, after $|A| - 1$ binary comparisons, finds the optimal alternative.

When the number of alternatives increases, one needs to go beyond this basic algorithm. In particular, one can rely upon the celebrated *Metropolis Algorithm* (Metropolis et al., 1953) and its evolution called *Simulated Annealing* (Kirkpatrick et al., 1983).

1. It goes without saying that for minimization problems the analysis is similar.

Metropolis Algorithm: Let $\beta > 0$.

Step 0**.** Choose $a \in A$ randomly.

Step $n+1$**.** Choose $b \in A$ randomly.

- If $u(b) \geq u(a)$, then set $a \doteqdot b$.
- If $u(b) < u(a)$, then set $a \doteqdot b$ with probability $e^{\beta[u(b)-u(a)]}$.

Specifically, the random choice at Step 0 is performed by drawing a from an initial distribution $\mu(\cdot)$ on A; the one at Step $n+1$ by drawing b from a Markovian distribution $Q(\cdot \mid a)$ which depends on the incumbent a.[2] The full name of the algorithm is thus *Metropolis Algorithm* with *inverse temperature* β, *initial distribution* μ and *proposal matrix* Q. Its key property is that, if the current state is a, the next state b is determined according to the following transition probabilities

$$P(b \mid a) = \begin{cases} Q(b \mid a) \min\left\{1, e^{\beta[u(b)-u(a)]}\right\} & \text{if } b \neq a \\ 1 - \sum\limits_{c \neq a} Q(c \mid a) \min\left\{1, e^{\beta[u(c)-u(a)]}\right\} & \text{else} \end{cases}$$

Thus, the algorithm realizes an aperiodic and irreducible Markov chain with stationary distribution

$$p_\beta(a) = \frac{e^{\beta u(a)}}{\sum_{b \in A} e^{\beta u(b)}} \qquad \forall a \in A$$

Therefore, the long-run frequency with which a is chosen from A is almost surely $p_\beta(a)$.

The idea of Simulated Annealing is to slowly increase β, while the Metropolis algorithm runs, with the objective of approaching the limit distribution

$$p_\infty(a) = \lim_{\beta \to \infty} p_\beta(a) = \begin{cases} \dfrac{1}{|\arg\max_A u|} & \text{if } u(a) = \max_A u \\ 0 & \text{else} \end{cases}$$

which concentrates on the maximizers of u on A. In the words of its creators: "At each temperature [here $1/\beta$], the simulation must proceed long enough for the system to reach a steady state." Thus, the constant β is a replaced with a sequence β_n of inverse temperatures such that $\beta_n \equiv \beta_{t_0}$ is maintained constant for $n = 0, ..., t_0$, with t_0 large enough to achieve the stable (empirical) frequency

$$\hat{p}_0(a) \approx \frac{e^{\beta_{t_0} u(a)}}{\sum_{b \in A} e^{\beta_{t_0} u(b)}} \qquad \forall a \in A$$

Subsequently, $\beta_n \equiv \beta_{t_1}$ is maintained constant for $n = t_0 + 1, ..., t_1$, with t_1 large enough to achieve the stable (empirical) frequency

$$\hat{p}_1(a) \approx \frac{e^{\beta_{t_1} u(a)}}{\sum_{b \in A} e^{\beta_{t_1} u(b)}} \qquad \forall a \in A$$

and so on, aiming at a long-run frequency

$$\lim_{k \to \infty} \hat{p}_k(a) \approx \lim_{k \to \infty} \frac{e^{\beta_{t_k} u(a)}}{\sum_{b \in A} e^{\beta_{t_k} u(b)}} = p_\infty(a) \qquad \forall a \in A$$

2. The distribution $Q(\cdot \mid a)$ corresponds to the a-th row of a symmetric and irreducible $A \times A$ stochastic matrix Q. This matrix describes the way in which the algorithm explores the landscape A. Irreducibility guarantees full exploration of A, symmetry is intuitive and can be dispensed with. See Hastings (1970) or Madras (2002) for a textbook treatment.

as β_n diverges (so that β_{t_k} diverges too). The sequence $\{\beta_n\}_{n\in\mathbb{N}}$ is called *annealing schedule*.[3]

Simulated Annealing: Let $\beta_n \to \infty$.

Step 0**.** Choose $a \in A$ randomly.

Step $n+1$**.** Choose $b \in A$ randomly.

- If $u(b) \geq u(a)$, then set $a \doteqdot b$.
- If $u(b) < u(a)$, then set $a \doteqdot b$ with probability $e^{\beta_n[u(b)-u(a)]}$.

The algorithm runs until the system "freezes", that is, a stops changing for a fixed number of consecutive iterations (or until a given number N of iterations has been performed). The selected alternative is a candidate maximizer for the objective function u.

The grain of randomness injected in the last line of the pseudo-code allows the DM to escape from local maxima and explore the actions' landscape (especially at small values of β). In the next section, we use this explorative feature of the algorithm to obtain information about u and maximize it, even when u itself is initially unknown.

3 UNKNOWN PAYOFFS: ERGODIC ANNEALING

Now assume that the objective function u is **unknown** to the DM. In particular, consider the important case when $u(a)$ is the expectation of the random payoff $U(a)$ of alternative $a \in A$, which is unknown because the DM ignores the distribution $F_{U(a)}$ of the random variable $U(a)$ itself.

Example *A basic example is when each $U(a)$ has Bernoulli distribution with unknown probability of success $u(a)$.* □

The initial beliefs of the DM about u are represented by a pair $(u_0, C_0) \in \mathbb{R}^A \times \mathbb{N}^A$, where

$$u_0(a)$$

represents the *ex ante* expected payoff assigned to a by the DM and

$$C_0(a)$$

represents the DM's confidence in this evaluation. Intuitively, $C_0(a)$ is the number of "preliminary samples of a" on which the DM bases his expectations.

Example (continued) *Assume there are three alternatives a, b, c. The pair (u_0, C_0) given by*

$$u_0 = (1,0,1) \quad \textit{and} \quad C_0 = (1,1,1)$$

is interpreted as follows: the DM sampled once from each distribution $F_{U(a)}$, $F_{U(b)}$, $F_{U(c)}$ and observed the realizations

$$U(a) = 1 \quad U(b) = 0 \quad U(c) = 1$$

"Sampling" and "observing" can be factual or hypothetical.

3. Originally, $t_k = (k+1)L$ for some fixed "large" loop length $L \in \mathbb{N}$ and $\beta_{t_k} = (1+\rho)^k \beta_0$ for some "small" factor $\rho \in (0, \infty)$.

Analogously, the pair (u_0, C_0) *given by*

$$u_0 = (0.5, 0.7, 1) \quad \textit{and} \quad C_0 = (2, 10, 1)$$

is interpreted as follows: the DM took 2 *samples from* $F_{U(a)}$ *and observed an average value of* 0.5 *(that is, one success and one failure),* 10 *samples from* $F_{U(b)}$ *and observed an average value of* 0.7 *(that is seven successes and three failures), and* 1 *sample from* $F_{U(c)}$ *and observed a value of* 1 *(one success).*

Note that the same ex ante evaluation of two different alternatives may be based on different information. For instance, the pair (u_0, C_0) *given by*

$$u_0 = (1, 0.6, 0.6) \quad \textit{and} \quad C_0 = (1, 1000, 5)$$

is interpreted as follows: the DM took 1 *sample from* $F_{U(a)}$ *and observed one success —the DM has almost no information about* $u_0(a)$*,* 1000 *samples from* $F_{U(b)}$ *and observed six-hunded successes and four-hundred failures —the DM has a lot of information about* $u_0(b)$*,* 5 *samples from* $F_{U(c)}$ *and observed three successes and two failures —the DM has some information about* $u_0(c)$*.*[4] *It seems safe to assume that the DM has no confidence in* $u_0(a)$*, lots of confidence in* $u_0(b)$*, some confidence in* $u_0(c)$*. The fact that the observation pseudocount* $C_0(x)$ *represents a degree of confidence about the ex ante evaluation* $u_0(x)$ *of* $x \in \{a, b, c\}$ *can be formalized in a Bayesian way by means of a product of beta priors, but this goes beyond the scope of the present note.* □

Given (u_0, C_0), if an alternative a is selected again and another sample from $F_{U(a)}$ is taken, a new realization $v(a)$ of $U(a)$ is observed. The empirical evaluation of a becomes

$$u_1(a) = \frac{C_0(a)}{C_0(a)+1} u_0(a) + \frac{1}{C_0(a)+1} v(a)$$

and of course $C_1(a)$ is now $C_0(a)+1$. For all $b \neq a$ in A, **empirical evaluation** and **observation count** do not change, that is $u_1(b) = u_0(b)$ and $C_1(b) = C_0(b)$.

The more observations from a are taken, the better the estimate of the expectation of $U(a)$ becomes. This **ergodic property** is at the basis of our algorithm.

Ergodic Annealing: Let $\beta_n \to \infty$.

Initialize. Set $u \doteqdot u_0$ and $C \doteqdot C_0$.

Step 0**.** Choose $a \in A$ randomly.

Update. Observe $U(a)$

$$\text{set} \quad u(a) \doteqdot \frac{C(a)}{C(a)+1} u(a) + \frac{1}{C(a)+1} U(a) \quad \text{and} \quad C(a) \doteqdot C(a) + 1.$$

Step $n+1$**.** Choose $b \in A$ randomly.

- If $u(b) \geq u(a)$, then set $a \doteqdot b$.
- If $u(b) < u(a)$, then set $a \doteqdot b$ with probability $e^{\beta_n [u(b) - u(a)]}$.

Update. Observe $U(a)$

$$\text{set} \quad u(a) \doteqdot \frac{C(a)}{C(a)+1} u(a) + \frac{1}{C(a)+1} U(a) \quad \text{and} \quad C(a) \doteqdot C(a) + 1.$$

4. Again, this "information" can be factual or introspective.

The only difference with Simulated Annealing is the update routine that allows the algorithm to learn the values as it explores.

From a purely conceptual point of view one could think of first learning the values of alternatives, then performing a standard Simulated Annealing procedure. But this would be unfeasible even in cases with relatively few alternatives (say a Traveling Salesman Problem with 40 cities). In fact, just performing one draw from each distribution would be extremely costly ($40!$ draws, one for each possible tour), and performing sufficiently many draws until the empirical averages converge is utterly impossible.

Our approach learns the payoffs and optimizes it simultaneously. This speeds up the search because the DM is not interested in finding the true payoff of all alternatives, but only an alternative with highest true payoff. In the simulations below this corresponds to the fact that while the sequence u_n of Step n evaluations does not necessarily converge, the payoff of the chosen alternative converges to the true optimal payoff. Our agent is an empirical optimizer, not an empirical statistician.

4 SIMULATIONS

In this section we benchmark the Ergodic Annealing algorithm for two classical combinatorial problems: the Directed Steiner Tree problem on graphs (DST), our initial example, and the Traveling Salesman Problem (TSP), in which the randomness of traffic and viability between two nodes makes compelling the uncertainty in the cost function.

For the DST problem we adapted the Simulated Annealing algorithm of Osborne and Gillett (1991), while for the TSP we adapted the original Simulated Annealing algorithm of Kirkpatrick et al. (1983). As discussed above, by "adapting," we mean augmenting it with an ergodic updating procedure.

4.1 Directed Steiner Tree

The first motivating example for the Ergodic Annealing algorithm is the one mentioned in the introduction and pictured in Figure 1, which is formally known as the Directed Steiner Tree problem.

In a DST problem, a directed graph $G(V, E)$ with a non-negative cost $c(e)$ associated to each edge $e \in E$ is considered. The objective is sending a packet from a root node r to each of the terminal nodes R, at *minimum cost*. Each of the $|R|$ packets is allowed to travel through some intermediate nodes, called Steiner nodes. The cost of the whole operation is the sum of the costs of the edges used to send all the packets of information, and the goal is to minimize this quantity.[5] The subset of Steiner nodes used is a variable in this problem, and the optimal configuration coincides with the minimum spanning tree of the subgraph of G induced by the root r, the terminals R, and an optimal subset of intermediate nodes.

Coherently with Figure 1, we consider networks with a *layered structure*, meaning that the vertices can be divided in an ordered partition $\{V_l\}_{l=0}^{L}$, with the singleton $\{r\}$ making up the first layer V_0 and the set of terminal nodes R making up the last layer V_L. Every edge $e = (v, w)$ in the graph must be such that its vertices are in subsequent layers, that is, $v \in V_i$ and $w \in V_{i+1}$, for some $i \in \{0, ..., L-1\}$.[6]

This is a highly non-trivial combinatorial optimization problem (indeed, it is NP-hard), and Simulated Annealing is a very successful approximation scheme used to tackle it. Therefore a DST with unknown costs is a natural candidate to test the performance of Ergodic Annealing.

5. Note that we can use an edge as an intermediate channel to reach two different terminal nodes, but its cost will be counted only once. This can be interpreted as having only a fixed "opening" cost of the channel and no capacity constraints.

6. Since the graph is directed, the order of the vertices is important.

A key step required to run an annealing algorithm for this problem is the selection of feasible moves from one candidate solution to another.[7] There are different ways to do this. We follow the proposal of Osborne and Gillett (1991), which simply consists in allowing to move one potential Steiner node ($v \in \bigcup_{l=1}^{L-1} V_l$) from the set of used Steiner nodes to the set of unused nodes, and *vice versa*. Then one can easily compute the new Steiner tree by computing the minimum spanning tree on the resulting subgraph by using Edmonds' algorithm.[8]

To study and compare the performance of Ergodic Annealing with respect to Simulated Annealing we ran both algorithms on a test set of 1000 random graphs with the same true costs each time —**known** for the Simulated Annealing agent, **unknown** to the Ergodic Annealing one. Each graph $G = (V, E)$ in the test set has 13 layers (so 11 layers of potential Steiner nodes), with a maximum of 12 nodes for each non-root layer. The actual number of nodes for each layer is chosen uniformly at random from $\{2, ..., 12\}$. For each $v \in V$, a node from the previous layer is selected randomly and automatically connected, to guarantee feasibility. All other possible edges in the graph are present with probability $1/2$. The (true) arc costs are drawn uniformly from the interval $(0, 1)$, and the ex ante expected cost of each arc is initialized to $1/2$.

Since root and terminals are fixed in the optimal Steiner tree, we define the *size* of the graph as the number of potential Steiner nodes. The average size of a graph in the test was 71.3.

In these moderately large graphs, the two algorithms performed quite similarly. Indeed, they reached the same final configuration on 322 graphs, and the average absolute deviation with respect to the best configuration found[9] was 0.04664. In words, on average the two procedures found solutions whose costs differed by 4.66%, sometimes with Simulated Annealing being closer to the true optimal solution, sometimes with Ergodic Annealing performing better.

In Figure 2 we present two examples of Steiner trees found by Ergodic Annealing and Simulated Annealing on graphs from the test set.

Ergodic Annealing finds configurations of similar cost compared Simulated Annealing, even if the problem it faces is orders of magnitude harder. Simulated Annealing optimizes over a large but finite set of **known** configuration costs, while Ergodic Annealing searches for a minimum cost configuration on a space that is potentially infinite, because the true costs are **unknown** and are *learned* on a continuum space.

4.2 Traveling Salesman Problem

The second benchmark studied in this paper is the well-known Traveling Salesman Problem.

In the classical case a list of cities and distances between each pair of cities are given and known, and the objective is to find the shortest possible route that visits each city and returns to the starting point. Just like the DST, the TSP is an NP-hard problem in combinatorial optimization, important in theoretical computer science, operations research and economics.

In our variant, distances are replaced by average travel times, and the objective is finding the fastest route. The Simulated Annealing algorithm can solve this problem when these travel times are **known**, the Ergodic Annealing can solve it even when travel times are **unknown**.

We ran a simulation over 2000 random instances of TSP, with cities location chosen randomly from the unit square. The number of cities was selected at random between 30 and 90, with an average size of graphs in the simulation of 59.68. The

7. A move is considered feasible if it transforms a feasible solution into another feasible solution.

8. In this layered version computing the minimum arborescence is particularly efficient, because the graph is a DAG and there are no recursive calls in the algorithm.

9. The best configuration is the one of lower cost among the two final configurations found by the algorithms.

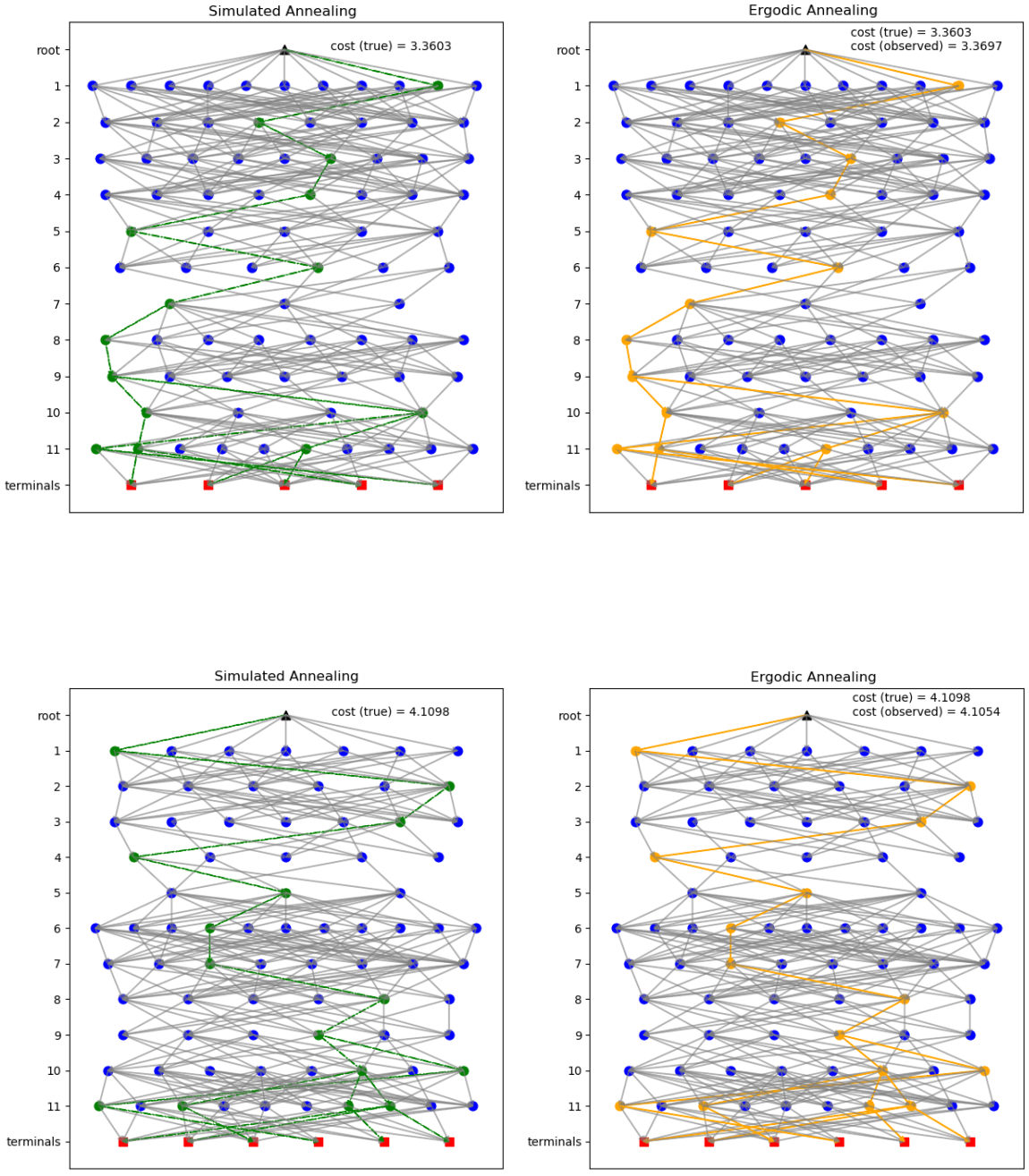


Fig. 2. Comparison of Ergodic Annealing and Simulated Annealing on two random DST instances.

performance of the two algorithms on the test was almost identical, with Ergodic Annealing performing at least as well as Simulated Annealing on 995 graphs. The average absolute deviation with respect to the best configuration found was 1.90%.

This simulation provides an even stronger evidence than the one found with the previous benchmark about the validity of Ergodic Annealing.

In Figures 3 and 4 we present two examples of optimal routes found by Ergodic Annealing and Simulated Annealing

on graphs from the test set.

5 Conclusion

For a given expected payoff u, Ergodic Annealing (implementable by a DM who ignores u and must learn it from the environment) performs almost as well as Simulated Annealing (which requires the DM to know u ex ante). Thus, Ergodic Annealing seems to be a promising extension of Simulated Annealing to decision making under uncertainty.

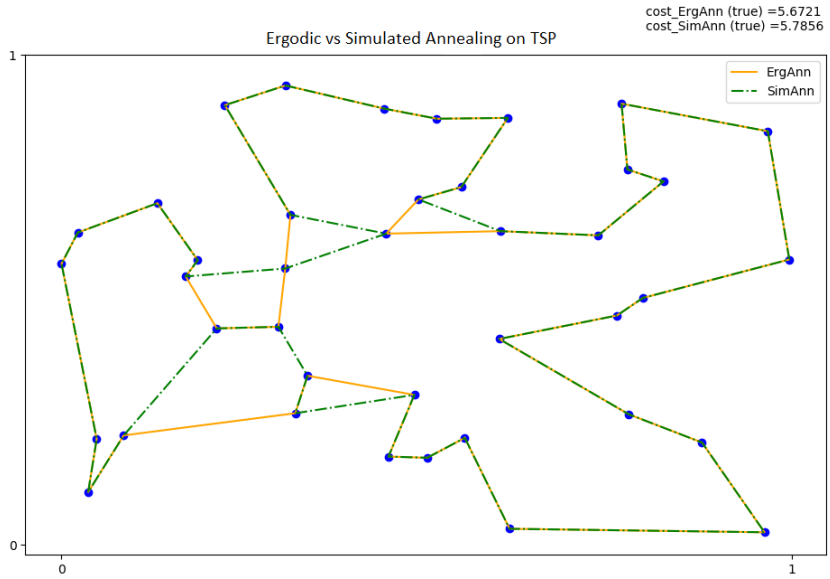


Fig. 3. Random TSP instance with 40 cities where Simulated Annealing finds a slightly suboptimal route compared with Ergodic Annealing.

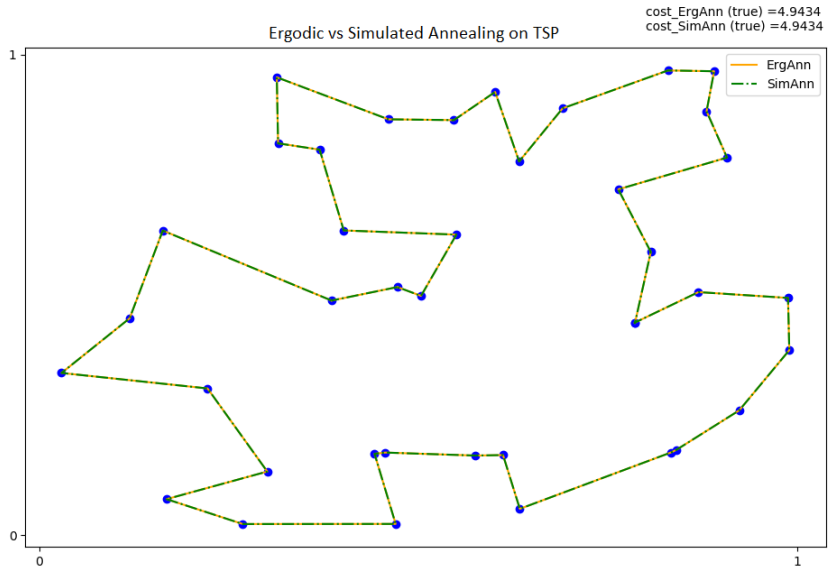


Fig. 4. Random TSP instance with 40 cities that produced the same final configuration with both algorithms

REFERENCES

[1] Edmonds, J. (1967). Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71.4, 233-240.

[2] Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57, 97-109.

[3] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220, 671-680.

[4] Madras, N. N. (2002). *Lectures on Monte Carlo methods*. American Mathematical Society.

[5] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1087-1092.

[6] Osborne, L. J., & Gillett, B. E. (1991). A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks. *ORSA Journal on Computing*, 3, 213-225.

Uniformly Deployed Sets in Computer Science and Optimisation

Peter Czimmermann[0000−0002−4661−1646]
Zuzana Borčinová[0000−0003−1768−6283]

University of Žilina, Slovakia
peter.czimmermann@fri.uniza.sk
zuzana.borcinova@fri.uniza.sk

Abstract. Uniformly deployed sets (UDS) are combinatorial objects that contain n-bit binary words of weight p as their elements and each pair of such words has at most t ones on the same positions (where t is a given natural number). In our contribution, we present a fast algorithm for the construction of UDS and we show some possibilities of their usage in various areas of computer science. For example, UDS can be used for increasing the effectiveness of heuristics in optimisation problems.

Keywords: Uniformly deployed set · Optimisation · Location problems.

1 Introduction and definitions

The concept of uniformly deployed sets was introduced and developed in works [1], [2], [3] and [4]. Our main motivation was an improvement of some meta-heuristics that are used in the optimisation of location problems.

Let S contain n-bit binary words of weight p (it means that each word has exactly p 1's and $n-p$ 0's). We say that S is a t-uniformly deployed set (where $0 < t < p$) if every pair of words $\vec{x}, \vec{y} \in S$ has at most t overlapping 1's.

A similar concept is known in statistics, where combinatorial structures called block designs are used in the planning of statistical experiments. A block design on n points with block-size p is an ordered pair (P, B), where P is a set of points ($|P| = n$) and B is a set of blocks ($|B| = b$). An incidence matrix of block design is $n \times b$ matrix with element $a_{ij} = 1$, if point $i \in P$ belongs to the block $j \in B$ and $a_{ij} = 0$ otherwise.

Regular uniform block design with parameters n, p and r is a special case of block design with n elements; each block of which contains p elements and each element is contained in exactly r blocks. The number of blocks is $b = nr/p$. Rows (similarly columns) of the incidence matrix of regular uniform block design form a t-uniformly deployed set with parameters n, p and appropriate $t \in \{1, \ldots, p-1\}$.

In combinatorics, there are known structures called difference sets [5] and abelian groups are needed to define them. A set X with operation $*$ is called an abelian group if the following properties hold

1. $\forall a, b \in X \ a * b \in X$,
2. $\forall a, b, c \in X \ (a * b) * c = a * (b * c)$,
3. $\forall a, b \in X \ a * b = b * a$,
4. $\exists e \in X \ \forall a \in X \ a * e = a$,
5. $\forall a \in X \ \exists a^{-1} \in X \ a * a^{-1} = e$.

The best-known abelian groups are cyclic groups Z_n with operation $\oplus_n$, where $Z_n = \{0, 1, \ldots, n-1\}$ and $\forall a, b \in Z_n \ a \oplus_n b$ is the remainder after dividing $a+b$ by n.

The difference set with parameters (n, p, t) is the p-element subset D of the n-element group X such that every non-identity element of X can be expressed as a result $a * b^{-1}$ of elements $a, b \in D$ in exactly t ways. Difference sets are interesting combinatorial objects with many useful properties but, unfortunately, their construction for given parameters is usually a hard combinatorial problem.

A well-known example of a difference set with parameters $(7, 3, 1)$ is set $D = \{1, 2, 4\} \subset Z_7$. If we consider all translates of D (sets $a \oplus_7 D$ for $a = 0, 1, \ldots, 6$), then we obtain the collection of subsets $\{1, 2, 4\}$, $\{2, 3, 5\}$, $\{3, 4, 6\}$, $\{4, 5, 0\}$, $\{5, 6, 1\}$, $\{6, 0, 2\}$, $\{0, 1, 3\}$. These subsets form the well-known combinatorial object called Fano plane (Figure 1). It is easy to check that Fano plane is a

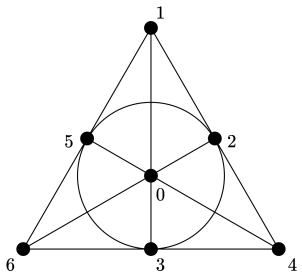


Fig. 1. Diagram of Fano plane.

regular uniform block design with parameters $n = 7$, $p = 3$, $r = 3$ and $b = 7$. Rows of its incidence matrix form 1-uniformly deployed set with parameters $n = 7$ and $p = 3$.

$$
\begin{array}{c}
0\,1\,1\,0\,1\,0\,0 \\
0\,0\,1\,1\,0\,1\,0 \\
0\,0\,0\,1\,1\,0\,1 \\
1\,0\,0\,0\,1\,1\,0 \\
0\,1\,0\,0\,0\,1\,1 \\
1\,0\,1\,0\,0\,0\,1 \\
1\,1\,0\,1\,0\,0\,0
\end{array}
$$

Similarly, we can obtain from an arbitrary difference set with parameters (n, p, t) a t-uniformly deployed set with parameters n and p such that each pair of its words have exactly t overlapping 1's.

2 Construction of UDS

The construction of t-UDS with given parameters is described in [3] and [4]. This construction uses voltage and Caley graphs and our method works with the group Z_n and operation $\oplus_n$. In this approach, a crucial task is the construction of p-element subset $D \subseteq Z_n$. All translates of D (as in the example with the Fano plane) form a system of n subsets of Z_n. Rows of its incidence matrix form a t-UDS with parameters n, p and appropriate t. Our latest version of the algorithm (in Python programming language) for the construction of the set D (less cumbersome than versions described in [3] and [4]) can be seen below:

```
def nptset(n,p,t):
    a = [0,1]
    k = len(a)
    f = lambda u, v:(u-v) % n

    def check(x):
        ax = a + [x]
        Amn = [f(ax[i],ax[j]) -->
-->for i in range(len(ax)) for j in range(len(ax)) if i!=j]
        freq = max([Amn.count(i) for i in range(n)])
        return freq <= t

    x = 2
    while x < n and k < p:
        if check(x):
            a += [x]
            k += 1
        x += 1

    if k == p:
        return a
    else:
        return 'no solution'
```

The computational complexity of the algorithm is $O(np)$ for parameters n, p and t. If we try to find a minimal value t such that t-UDS with parameters n and p exists, then we need to repeat the algorithm at most p times.

Example 1. We show the construction of t-UDS with parameters $n = 7$, $p = 4$ and t minimal possible:
For $t = 1$, we obtain the answer that there is 'no solution'.

For $t = 2$, we obtain the output $D = \{0, 1, 2, 4\}$. All translates $a \oplus_7 D$ (for $a = 0, 1, \ldots, 6$) are $\{0, 1, 2, 4\}$, $\{1, 2, 3, 5\}$, $\{2, 3, 4, 6\}$, $\{3, 4, 5, 0\}$, $\{4, 5, 6, 1\}$, $\{5, 6, 0, 2\}$ and $\{6, 0, 1, 3\}$. The incidence matrix of this system of subsets is

$$\begin{array}{c} 1\,1\,1\,0\,1\,0\,0 \\ 0\,1\,1\,1\,0\,1\,0 \\ 0\,0\,1\,1\,1\,0\,1 \\ 1\,0\,0\,1\,1\,1\,0 \\ 0\,1\,0\,0\,1\,1\,1 \\ 1\,0\,1\,0\,0\,1\,1 \\ 1\,1\,0\,1\,0\,0\,1 \end{array}$$

and it is easy to consider that the rows of this matrix form 2-UDS for $n = 7$ and $p = 4$.

3 UDS and the location problems

The concept of t-Uniformly Deployed Sets was primarily suggested for location problems (typically the weighted p-median and the weighted p-center problem). Each feasible solution of these problems can be represented as an n-bit word of weight p. We test t-Uniformly Deployed Sets for the weighted p-median problem. In this section, we show that t-UDS allow us to go deeper in the space of feasible solutions than randomly chosen sets for the neighbourhood search heuristic algorithm with multiple starts (we denote it by NS-MS). This heuristic can be described as follows:
A network $G = (V, H, d, w)$ (where $|V| = n$) and a set M of n-bit binary words with weight p be given.

$M_{hist} = M$, $M_{end} = \emptyset$;
WHILE $(M \neq \emptyset)$
    FOR $(w \in M)$
        $w' = ba(w)$;
        IF $w' \neq e$ THEN $M = (M - w) \cup w'$, $M_{hist} = M_{hist} \cup w'$;
                ELSE $M = M - w$, $M_{end} = M_{end} \cup w$;
bestsolution=ARGMAX$\{f(w), w \in M_{end}\}$;

Set M_{hist} contains all n-bit binary words of weight p which occurred in M during the computation. The objective function (value of weighted p-median) is denoted by $f()$. The function $ba(w)$ represents the neighbourhood search heuristic which starts with the word w and the strategy: choose the best admissible neighbour. (The neighbour of w is a result of simple interchange.) The output of this heuristic is a word w' such that w' is a neighbour of w, $f(w') > f(w)$ and $w' \notin M_{hist}$, or conversely w' is the empty word (denoted by e) if there is no neighbour of w with such properties. Set M_{hist} contains end words obtained during computation. Where the word w is called an end word if $ba(w)$ is an empty word.

A chain of length k is an ordered k-tuple $(w_1, w_2, \ldots, w_k)$ of n-bit binary words of weight p such that w_{i+1} is the neighbour of w_i and $f(w_{i+1}) < f(w_i)$ for $i = 1, 2, \ldots, k-1$.

In our computational experiments, we used graphs which serve for weighted p-median testing. For each such graph, a t-UDS with the lowest possible value t was constructed and a random set of n binary n-bit words of weight p was generated. Both sets were used as starting set M in the NS-MS heuristic. For each word w from these sets, we computed the length of the chain

$$(w, ba(w), ba(ba(w)), \ldots, w^*),$$

where w^* is an end word. For each set, we can compute the average length of the chain. Random permutations of the vertex set allow us to repeat computations several times for the same graph and testing sets. The results of the tests can be seen in Figure 2. Blue columns represent the average length of the chains for graphs denoted by numbers $1, \ldots, 11$ which are obtained from t-UDS and orange columns represent the average length of chains obtained from the randomly generated set of starting solutions. The parameters of the tested graphs can be seen in Table 1. All tested graphs are from [6].

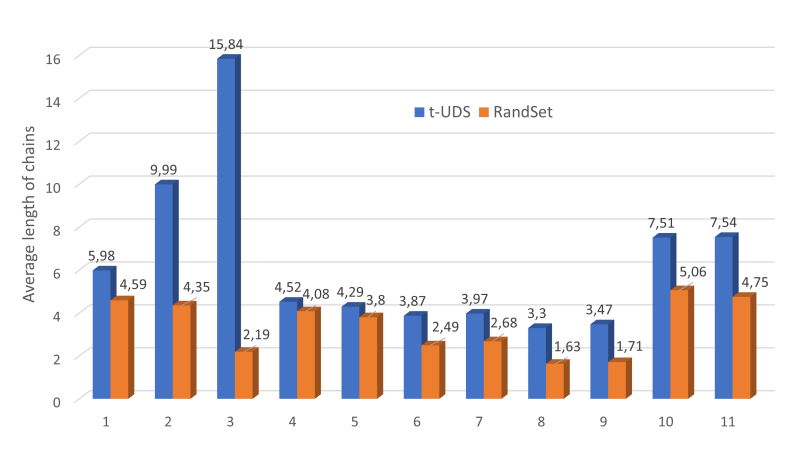


Fig. 2. Average length of chains.

graph number	1	2	3	4	5	6	7	8	9	10	11
n	100	100	100	100	100	100	100	200	200	200	200
p	5	10	20	10	10	20	20	20	20	40	40

4 Collections of UDS

Our next goal is to develop an algorithm for the construction of a collection of UDS with parameters n and p, where $p = 1, \ldots, n-1$. Such collections can be

used in problems, in which a set of feasible solutions can be represented by all n-bit binary words. For example, the covering problem is a good candidate for using a collection of UDS. In coding theory, we have the constructions of binary linear (n, k) codes for appropriate k, in which each pair of coding words has the maximal possible Hamming distance. This construction does not allow to control a number of words of the given weight. Another possibility is the collection of UDS:

$$\{U_1, U_2, \ldots, U_{n-1}\},$$

where U_p is t_p-UDS with parameters n, p and appropriate t_p (for $p = 1, 2, \ldots, n-1$). We must note that the maximal possible Hamming distance (minimal possible t) is provided by this approach only for the pairs of words from the same set U_p. A disadvantage of this approach is that the numbers of words of weights i and j are the same. Various (but determined) frequencies of words of given weights are guaranteed if we use a collection of sets:

$$\{U_{1,1}, U_{2,1}, \ldots, U_{n-1,1}\}.$$

It means that we construct p sets $U_{p,1}, \ldots, U_{p,p}$ of words of weight p, if $p \leq n/2$. For $n/2 < p > n$, we have $n - p$ sets of weight p. The cardinality of each set is n. Hence we have pn ($(n-p)n$ respectively) words of weight p in the collection.

Example 2. For $n = 8$, we have $1, 2, 3, 4, 3, 2, 1$ sets of weights $1, \ldots, 7$. It means that in the collection we have $8, 16, 24, 32, 24, 16, 8$ words of given weights. For $p = 4$, we have 4 sets, which can be constructed from $D_1 = \{0, 1, 2, 4\}$, $D_2 = \{0, 1, 2, 5\}$, $D_3 = \{0, 1, 2, 6\}$, $D_4 = \{0, 1, 3, 4\}$ and their translates $a \oplus_8 D_i$.

In general, we can start with a set D_1 of voltages. From D_1 we obtain D_2, from D_2 we have D_3, etc. Algorithmic construction of D_{i+1} from D_i is as follows:
Let n and $D_i = \{a_1, a_2, \ldots, a_p\}$ be given. We try to replace a_p by values $a_p + i < n$ and we check if we obtain t-UDS for given t. If not, we continue with replacing of pair (a_{p-1}, a_p) by pairs $(a_{p-1} + i, a_p + j)$ (where $a_{p-1} + i < a_p + j < n$). If it is necessary, we continue with triples, quadruples etc. If we do not find t-UDS, then we increase the value t and repeat the previous algorithm.

5 Conclusions

In our contribution, we study the usage of t-Uniformly Deployed Sets in location problems. We show that the neighbourhood search algorithm is able to search a larger part of the space of feasible solutions if we start with t-UDS. We also show the construction of a collection of UDS. Our future research plan is to use these collections in reliability analysis, artificial neural networks (computation of significance of neurons, estimation of the number of neurons in layers, etc.) and in studying of properties of boolean functions. We also prepare a new method for the construction of uniformly deployed sets of permutations which we would like to use in the heuristics for the travelling salesman problem.

Acknowledgements

This work was supported by the research grants VEGA 1/0776/20 "Vehicle routing and scheduling in uncertain conditions", VEGA 1/0216/21 "Designing of emergency systems with conflicting criteria using tools of artificial intelligence", and VEGA 1/0077/22 "Innovative prediction methods for optimization of public service systems". This work was supported by the Slovak Research and Development Agency under Contract No. APVV-19-0441 "Allocation of limited resources to public service systems with conflicting quality criteria".

References

1. J. Janacek, M. Kvet, "Usage of Uniformly Deployed Set for p-Location Min-Sum Problem with Generalized Disutility," SOR 2019: Proceedings of the 15th International Symposium on Operational Research, ISBN 978-961-6165-55-6, pp. 494–499, 2019
2. M. Kvet, J. Janacek, "Population Diversity Maintenance using Uniformly Deployed Set of p-Location Problem Solutions," SOR 2019: Proceedings of the 15th International Symposium on Operational Research, ISBN 978-961-616555-6, pp. 354–359, 2019
3. P. Czimmermann, "The Construction of Uniformly Deployed Set of Feasible Solutions for the p-Location Problem," In Proc. META2021, Communications in Computer and Information Science, Springer Nature, ISBN 978-3-030-94215-1, pp. 80–89, 2022
4. P. Czimmermann, "Application of Uniformly Deployed Sets," In Proceedings of IEEE 26th International Conference on Intelligent Engineering Systems 2022 (INES 2022), 2022
5. J.H. van Lint, R.M. Wilson, "A Course in Combinatorics", Cambridge University Press, 2001
6. http://people.brunel.ac.uk/ mastjjb/jeb/orlib/files/

Multi-Objective Metaheuristic Solution Approach for the Crop Plant Scheduling Problem

M. Lalić[1], N. Obrenović[1], S. Ataç[2], S. Bortolomiol[3], M. Oskar[1], S. Brdar[1], V. Crnojević[1], I. Luković[4]

[1] Institute BioSense, University of Novi Sad, Serbia
{maksim.lalic, nikola.obrenovic, oskar.marko, sanja.brdar, crnojevic}@biosense.rs
[2] École Polytechnique Fédérale de Lausanne, Transport and Mobility Laboratory (TRANSP-OR), Switzerland, selin.atac@epfl.ch
[3] Optit Srl, Bologna, Italy, stefano.bortolomiol@optit.net
[4] Faculty of Organizational Sciences, University of Belgrade, Serbia, ivan.lukovic@fon.bg.ac.rs

1 Introduction

Most of the farm field operations are scheduled quite imprecisely or, even worse, determined ad hoc. Following intuitionally and empirically handcrafted schedules can lead to suboptimal harvest time. As a consequence, it can affect harvest effectiveness by reducing the total amount of yield and increasing the amount of unnecessary waste, as well as its efficiency by imposing a longer harvest horizon than required or by making harvest intensity and labor force engagement unevenly distributed over the horizon.

Results from our previous study [1] suggest that by proper planting and harvest scheduling, there is room for significant improvement in waste reduction and consistent harvest intensity. Our study was inspired by the Syngenta Crop Challenge in Analytics 2021 (the challenge) [2]. The aim of the challenge was to determine desired planting and harvest schedule for the farm field whose crop is divided into *populations*. Each population represents a management unit that will be treated individually and uniformly in any aspect. For each population, the allowed planting time window and expected yield are known. Afterward, based on the weather forecast, it is possible to derive time-window during which the population should be harvested. Finally, we have defined a limit on the amount of weekly harvested yield that can be adequately processed and stored while any surplus beyond that would be wasted.

Some other papers discuss slightly different solution approaches for a problem proposed at the Challenge [3–5]. However, all proposed decision frameworks have certain specificities. They all incorporate integer-linear problems (ILP) solved by exact off-the-shelf solvers and the multi-objective nature of the problem has been treated either by aggregation of multiple objectives as a weighted sum or by iterative or sequential exploitation of models as part of a broader decision framework.

Here, we simplify the optimization process and approach it with a single run of a multi-objective metaheuristic method. This enables us to produce and analyze Pareto fronts of solutions with various effectiveness and efficiency and to deal with instances unmanageable for exact solvers. The optimization problem has been approached with two, in some sense complementary, metaheuristics. We examined adaptive large neighborhood search (ALNS) [6] as a model-driven single-solution-based metaheuristic and non-dominated sorting genetic algorithm II (NSGA-II) [7] as a population-based data-driven metaheuristic. We provide our preliminary results, which show that ALNS consistently and significantly outperforms NGSA-II.

2 Methodology

Given a set of populations P and a set of weeks comprising harvesting horizon W, our problem formulation is as follows:

$$F_1 = \sum_{w \in W} |q_w - C| \tag{1}$$

$$F_2 = |\{w \in W : q_w > 0\}| \tag{2}$$

$$q_w = \sum_{p \in P} hq_p \cdot 1_w([t_p]), \qquad \forall w \in W \tag{3}$$

$$e_p \leq t_p \leq l_p, \qquad \forall p \in P \tag{4}$$

The first objective function $F1$ expresses the effectiveness of a schedule and it is calculated as the sum of deviations between weekly harvested quantities q_w and the stated storage capacity C. The second objective function $F2$ expresses the schedule efficiency and it is equal to the number of working weeks, i.e. the number of weeks with a positive harvested quantity.

Constraints (3) are used to determine weekly harvested quantities q_w, for each week $w \in W$, based on selected harvesting time t_p for a population $p \in P$ and its expected yield hq_p. Since t_p takes real values, we consider that a population p would be harvested in a week w if an indicator function $1_w([t_p])$ yields 1, i.e. if a rounded value of t_p equals to w. Constraints (4) ensure that each population $p \in P$ is harvested within its feasible time window $[e_p, l_p]$.

We solved the proposed problem using two well-known metaheuristics, ALNS and NSGA-II. For the ALNS application, we designed five domain-specific neighborhood operators, where each of them was tailored to favor a particular desirable characteristic of the schedule. Operator one is a rebalancing operator designed to reschedule populations from highly stressed weeks to least stressed weeks. Operator two is a stability operator designed to equalize the amount of harvested quantities between consecutive weeks. Operator three is an emptying operator which finds two consecutive weeks with the lowest total harvested quantity and empties the less stressed one into another one. Operator four is also an emptying operator. This operator will empty a week which is preceded by an already emptied one. Operator five is a capacity operator rescheduling populations for weeks that are loaded over storage capacity C.

As for the second metaheuristic, we used real-coded constrained NSGA-II without any modifications. We experimented with two different ways of initial population generation. By our first approach, called *random*, the initial population is created by uniformly sampling value for each gene t_p inside its allowed time window $[e_p, l_p]$. By our second approach, called *smart*, we select our initial population as a subset of 100 solutions out of 500 initially produced distinct solutions generated by the previous run of the ALNS for the same case.

3 Results

We synthesized 20 different case studies, which may have $n \in \{500, 1000, 2000, 3000, 4000\}$ populations and planning horizon $l \in \{26, 52, 78, 104\}$ weeks long. For each case study, we run each algorithm 25 times. For each case, we combined all solutions nondominated individually per optimization run and derive the overall Pareto front of all solutions on the case level.

Figures 1 and 2 show combined Pareto fronts for cases with the highest and the lowest ratio between the number of populations and the duration of the planning horizon. Solutions produced by ALNS are depicted as ▲, or ■ if they are overall nondominated. Individually nondominated solutions produced by NSGA-II are depicted as x for *random* initialization or ★ for *smart* initialization. Bounding boxes of overall nondominated solutions are presented as red rectangles. Figures emphasize a strong dominance of ALNS over the NSGA-II approach. All overall nondominated solutions are produced by ALNS. NSGA-II has never produced any solution with a relatively low $F2$ objective value and a significant portion of individually optimal solutions produced by NSGA-II turns out to be out of the bounding box surrounding overall nondominated solutions. On the other side, individually nondominated solutions produced by ALNS are close to overall nondominated solutions.

The presented results are consistent with all other our results. For each case, all overall nondominated solutions have been produced solely by ALNS. Pareto fronts generated by ALNS are firmly consistent from run to run and they converge relatively fast, in about 10000 to 15000 iterations. On the other side, all individually nondominated solutions produced by NSGA-II are from the first generation, which indicates that populations don't improve themselves by evolution at all. That also explains why *smart* initialization strategy provides better results than *random* strategy since all solutions from its first generation are actually previously produced by the ALNS.

ALNS dominance over the NSGA-II is particularly obvious regarding the $F2$ objective function, where NSGA-II performed poorly all the time. That is in accordance with our most recent experimental results, which are beyond the scope of this paper. Results indicate that offsprings' $F2$ objective values consistently degrade in comparison to parents' until they finally reach a certain poor quality standard where their values retain.

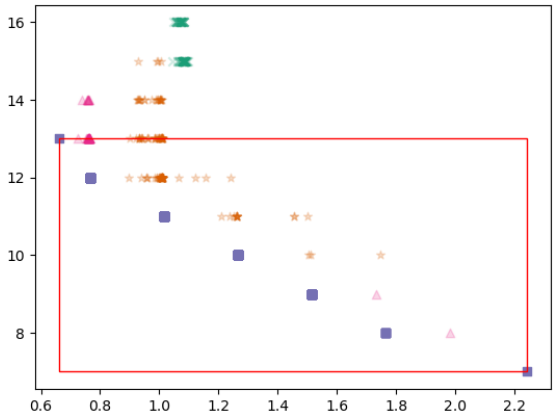


Fig. 1: Pareto front for $n = 4000$ and $l = 26$

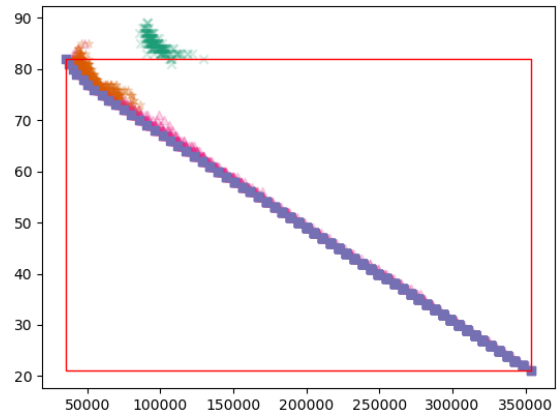


Fig. 2: Pareto front for $n = 500$ and $l = 104$

4 Conclusion

In this paper, we outlined a bi-objective formulation of the Crop Plant Scheduling Problem proposed in our previous work [1]. We adapted two metaheuristics to solve the problem. Our experimental results indicate a very strong dominance of ALNS over NSGA-II. Since those results are quite indicative, our future work shall be focused on gaining deeper insights into the reasons for such dominance. More precisely, we shall analyze the structure of the $F2$ objective function in relation to the operators integrated into the ALNS and NSGA-II and derive assumptions on what makes ALNS capable and NSGA-II incapable to provide good Pareto fronts for such a simple objective function.

5 Acknowledgements

This work was supported by the Provincial Secretariat for Higher Education and Scientific Research, Autonomous Province of Vojvodina (No. 142-451-2261/2022-01/01).

References

1. N. Obrenović, S. Ataç, S. Bortolomiol, S. Brdar, M. Oskar, V. Crnojević, "The Crop Plant Scheduling Problem", *International Conference on Optimization and Decision Science, ODS 2022*, Florence, Italy, 2022.
2. Syngenta Crop Challenge in Analytics 2021, url: https://www.ideaconnection.com/syngenta-crop-challenge/challenge.php, [Accessed: April, 2023.]
3. S. S. Sajid, G. Hu, "Optimizing Crop Planting Schedule Considering Planting Window and Storage capacity", *Frontiers in Plant Science*, 13:762446, 2022, doi: https://doi.org/10.3389/fpls.2022.762446
4. Z. Khalilzadeh, L. Wang, "An MILP Model for Corn Planting and Harvest Scheduling Considering Storage Capacity and Growing Degree Units", *bioRxiv*, 2021, doi: https://doi.org/10.1101/2021.02.06.430062
5. J. Ansarifar, F. Akhavizadegan, L. Wang, "Scheduling Planting time Through Developing an Optimization Model and Analysis of Time Series Growing Degree Units", *arXiv*, 2022, doi: https://doi.org/10.48550/ARXIV.2207.00745
6. S. Ropke, D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows", *Transportation Science*, vol. 40, issue. 4, pp. 455-472, 2006., doi: https://doi.org/10.1287/trsc.1050.0135
7. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, pp. 182-197, vol 6, no. 2, 2002, doi: https://doi.org/10.1109/4235.996017

Mixed Integer Linear Programming Based Large Neighborhood Search Approaches for the Directed Feedback Vertex Set Problem[0]

Maria Bresich[1], Johannes Varga[1], Günther R. Raidl[1], and Steffen Limmer[2]

[1] Institute of Logic and Computation, TU Wien, Austria
{mbresich,jvarga,raidl}@ac.tuwien.ac.at
[2] Honda Research Institute Europe, Germany
steffen.limmer@honda-ri.de

Abstract. A directed feedback vertex set (DFVS) of a directed graph is a subset of vertices whose removal makes the graph acyclic. Finding a DFVS of minimum cardinality is the goal of the directed feedback vertex set problem, an NP-hard combinatorial optimization problem. We first consider two mixed integer linear programming (MILP) models for this problem, which, when solved with Gurobi, are effective on graphs of small to medium complexity but do not scale well to large instances. Aiming at better scalability and higher robustness over a large variety of graphs, we investigate a large neighborhood search (LNS) in which a destroy operator removes randomly chosen nodes from an incumbent DFVS and one of the MILP models is used for repair. Regarding the destroy operator, finding a best degree of destruction is challenging. A main contribution lies in proposing several selection strategies for this parameter as well as a strategy for choosing the more promising MILP model for repair. We evaluate the performance of the MILP models and different LNS variants on benchmark instances and compare the approaches to each other as well as to state-of-the-art procedures. Results show that our LNS variants yield clearly better solutions on average than standalone MILP solving. Even though our approaches cannot outperform the state-of-the-art, we gain valuable insights on beneficially configuring such a MILP-based LNS.

1 Introduction

The directed feedback vertex set problem (DFVSP) is a classical combinatorial optimization problem in which the goal is to remove from a given directed graph as few vertices as possible in order to make it acyclic. More specifically, a *directed feedback vertex set* (DFVS) of a directed graph is a subset of vertices that contains at least one vertex of every simple cycle in the graph, and we are interested in a DFVS of smallest cardinality. The removal of such a DFVS from the graph in conjunction with the incident arcs results in a *directed acyclic graph* (DAG). This problem is one of the first problems shown to be NP-complete by Karp [1]. The DFVSP is a variant of the feedback vertex set problem (FVSP), which only considers undirected graphs. Exact algorithms for the (D)FVSP are limited in their practical applicability to instances of rather small and medium size due to the complexity of the problem. Research thus has also focused on approximation algorithms [2], parameterized algorithms [3,4], or graph classes for which efficient algorithms are possible [5,6]. Moreover, in the last decade, heuristics and especially (hybrid) metaheuristic procedures have also been proposed for addressing large (D)FVSP instances [7,8,9,10]. These approaches do not guarantee optimality but aim at providing good or near-optimal solutions in a reasonable time also for larger instances. Compared to exact methods, this often makes them more suitable for real-world applications, such as deadlock detection and recovery in operating systems, program verification, and VLSI chip design, where oftentimes a short runtime is more important than achieving optimality.

We follow this line of research and more specifically investigate diverse variants of large neighborhood search (LNS) in which partially destroyed solutions are repaired by means of mixed integer linear programming (MILP). Large neighborhood search [11,12] is a prominent metaheuristic that has already been successfully applied to a multitude of challenging combinatorial optimization problems including routing, scheduling, and location problems [11,12,13]. In our LNS, we apply a

[0]This project was financially supported by Honda Research Institute Europe GmbH.

neighborhood structure that is based on destroying a current solution by moving multiple vertices from the current DFVS back into the corresponding DAG, thus introducing cycles again. For making the solutions feasible again, i.e., repairing them, we propose two MILP formulations for the DFVSP, which we solve with a leading general purpose MILP solver. These MILP formulations are also evaluated as standalone solving procedures, and we will see that they perform reasonably well for small to medium sized instances but can hardly be applied to large instances. The LNS framework therefore is a natural and promising choice to exploit the power of the MILP approaches in order to also solve large instances heuristically. Our main goal is to gain insights on beneficial configurations and on the performance of such a hybrid LNS. We investigate various strategies for selecting the degree of destruction for the LNS and also a technique for dynamically choosing the MILP formulation for each instance. The resulting variants are tested on three sets of benchmark instances and compared to each other as well as to two state-of-the-art solving approaches from the literature. This work is based on the first author's master's thesis [14].

In the next section, we review related work before formally defining the DFVSP and introducing the two MILP formulations in Section 3. In Section 4, our graph reduction procedure is explained and Section 5 deals with the construction heuristic and local search. Our proposed LNS variants are presented in detail in Section 6, and Section 7 discusses the computational study and experimental results. Finally, we conclude this article and give an outlook on possible future work in Section 8.

2 Related Work

In the literature, various types of FVSPs are considered, which generally differ in two dimensions: graph orientation and involvement of weights. Oftentimes, the main concepts of solving approaches that were designed for one variant can be adapted and applied to the others. This is also true for the closely related feedback arc set (FAS) problem for directed graphs, whose goal is to find a subset of arcs of minimum cardinality that contains at least one arc of every cycle in the graph. This problem can be efficiently reduced to the DFVSP and vice versa. One approach to the FAS problem on unweighted graphs is given by Noughabi and Baghbani [15] who propose a genetic algorithm. They represent solutions as vertex orderings where all backward arcs constitute a FAS. This is possible due to the relationship of the FAS problem and the linear ordering problem because the latter can be reduced to the complement of the FAS problem, the maximum acyclic subgraph problem. Baharev et al. [16] also make use of this relationship and introduce an integer programming (IP) model for the FAS problem on edge-weighted graphs, which employs triangle inequalities to encode a minimum cost ordering of the vertices. The authors further propose two other IP models, which are both based on a minimum set cover formulation of the FAS problem, with the second model being an extension of the first one by adding lazy constraint generation.

For the undirected FVSP on vertex-weighted graphs, Cutello et al. [7] introduce a metaheuristic that combines an immune algorithm and a local search (LS) procedure and is specifically designed for large-scale instances. Melo et al. [9] tackle the same problem by solving its complement, which is the maximum weighted induced forest problem, and they introduce the first two compact MILP models for this problem. One formulation is flow-based and the other is based on an adaptation of a lifted version of the Miller-Tucker-Zemlin (MTZ) constraints. As both formulations are designed for connected and directed graphs, they are also applicable to the DFVS problem. Based on these MILP models, Melo et al. propose two variants of a hybrid metaheuristic that combines a multi-start iterated local search with a MILP-based local search procedure. In the next section, we will adapt the MTZ-based formulation to obtain one of our two MILP models for the unweighted DFVSP, which results in the first compact model for this problem to the best of our knowledge.

Considering heuristic approaches, there have not been many for the unweighted DFVSP until recently. The first ones were greedy heuristics and a greedy randomized adaptive search procedure (GRASP) by Pardalos et al. [17]. Galinier et al. [8] propose a local search procedure with an efficient neighborhood structure based on a new solution representation of the DFVS. Instead of minimizing the size of the DFVS, they focus on its complement and try to maximize the size of a contained DAG that is represented by a topological ordering. On top of the LS, the authors describe a simulated annealing (SA) algorithm called the SA-FVSP. This approach is extended by Tang et al. [10] with nonuniform neighborhood sampling (NNS) resulting in an algorithm called SA-FVSP-NNS. In this approach, new priority and sampling functions are applied to guide the search. The priority function

considers vertex degrees in a greedy manner similarly as in Cai et al. [18]. We will also utilize this concept in our procedures.

Recently in 2022, the development of new heuristic approaches for the unweighted DFVSP was sparked by the Parameterized Algorithms and Computational Experiments (PACE) challenge[3], which was one of the inspirations for our work. We also participated with an early version of our solving procedures, which, however, only ended up in the bottom third of the ranking. This paper describes improved variants, but as the challenge took place at the same time as the development of our solving approaches, we were not able to already learn from its results. Two of the three best performing heuristic solvers, one by Swat [19] and one by Du et al. [20], apply a transformation of the DFVSP into a vertex cover problem as well as diverse extensions of SA-FVSP. The solver by Bathie et al. [21] employs two greedy local search heuristics based on vertex swapping and perturbations of topological orderings.

What all of the finally leading solvers from the challenge have in common is the application of partly advanced graph reduction operations to decrease the size of the input graph and possibly partition it into unconnected components that can be independently addressed. Five popular reduction rules were already introduced in 1988 by Levy and Low [22]: IN0, OUT0, IN1, OUT1, and LOOP. Lin and Jou [23] proposed another three called PIE, CORE, and DOME. Six rules were also described by Fleischer et al. [4], but three of them were already covered by Levy and Low. An important property of all these operations is that they preserve optimal solutions. In their work on the FAS problem on unweighted graphs, Park and Akers [24] suggest four reduction operations, two of which also coincide with rules by Levy and Low.

3 Problem Formalization and MILP Models

The unweighted DFVSP we consider in this work is defined on a directed graph $G = (V, E)$ with vertex set V and arc set E. We only consider simple graphs that do not have parallel arcs or self-loops. The subgraph of G induced by a subset of vertices $V' \subseteq V$ is $G[V'] = (V', E')$ with $E' = E \cap (V' \times V')$ being the set of arcs from E whose endpoints are both contained in V'. A directed feedback vertex set (DFVS) is a subset $F \subseteq V$ of vertices that contains at least one vertex of every simple cycle in G. Let $\overline{F} = V \setminus F$ be the complement of F, then the induced subgraph $G[\overline{F}]$ is a DAG if and only if F is a valid DFVS of G. The goal of the DFVSP is to find a DFVS F^* of minimum cardinality, i.e., $|F^*| \leq |F|$ for every DFVS $F \subseteq V$. By these definitions, solving the DFVS problem is equivalent to finding a vertex set $\overline{F^*}$ of maximum cardinality such that the induced subgraph $G[\overline{F^*}]$ is acyclic.

We introduce two mixed integer linear programming formulations for the DFVSP. The first model is inspired by the subtour elimination constraints from Miller, Tucker, and Zemlin and is therefore called the *MTZ-based formulation*. The other formulation uses so-called *cycle elimination constraints* (CECs), which prevent cycles in the DAG corresponding to the DFVS.

3.1 MTZ-Based Formulation

This formulation is derived from the MILP model by Melo et al. [9] for the vertex-weighted, undirected FVSP. In order to consider the aspect that we are dealing with a directed graph as input, we add an artificial *source vertex* s to V and arcs from s to all other vertices. Thus, graph $G = (V, E)$ is augmented to the new graph $G_s = (V_s, E_s)$ with $V_s = V \cup \{s\}$ and $E_s = E \cup \{(s, v) \ : \ v \in V\}$. Our MTZ-based formulation actually models the complement of the DFVSP in which the aim is to find a maximum cardinality subset $\overline{F} \subseteq V$ whose induced subgraph is acyclic. We use binary decision variables y_v, $v \in V$, indicating with value one that $v \in \overline{F}$, i.e., the vertex is selected for the DAG, and consequently with value zero that v is part of the DFVS. Moreover, continuous variables $\Phi_v \in [0, |V_s| - 1]$ represent a *potential* for each vertex $v \in V$. In a valid solution, the potentials have

[3] `https://pacechallenge.org/`

to increase with the distance of v to s. The MTZ-based formulation of the DFVS problem is then:

$$\min \quad |V| - \sum_{v \in V} y_v \tag{1}$$

$$\text{s.t.} \quad \Phi_u - \Phi_v + |V_s| \cdot y_v \leq |V_s| - 1 \qquad \forall (u,v) \in E_s \tag{2}$$

$$y_v \leq \Phi_v \qquad \forall v \in V \tag{3}$$

$$y_s = 1 \tag{4}$$

$$\Phi_s = 0 \tag{5}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{6}$$

$$0 \leq \Phi_v \leq |V_s| - 1 \qquad \forall v \in V \tag{7}$$

The objective function (1) minimizes the cardinality of the DFVS, which corresponds to maximizing the number of vertices $v \in V$ with $y_v = 1$ constituting $\overline{F}$ and thus belonging to the DAG. Constraints (2) are the MTZ inequalities ensuring that for each arc (u, v) whose target vertex v is chosen for $\overline{F}$, the potential Φ_v of v must be larger by at least one than the potential Φ_u of vertex u. In this way, the potentials of the nodes on any path originating from s will always increase monotonically and cycles are prevented. The linear programming relaxation is strengthened by Inequalities (3), which explicitly enforce that the potential of selected vertices (except s) is at least one. Equations (4) and (5) define that the source node has to be connected and has potential zero. Lastly, (6) and (7) define the domains of the decision variables.

We remark that the original MILP formulation from Melo et al. [9] additionally uses binary variables for all arcs and the MTZ inequalities are based on these. We also considered such an approach in a corresponding model for the DFVSP, but computational experiments with Gurobi indicated that the more compact MTZ-based model presented above performs significantly better; see the first author's thesis [14] for details. Therefore, we only consider this variant in the following.

3.2 CEC-Based Formulation

The CEC-based formulation builds on cycle elimination constraints, which explicitly state for each cycle in the input graph G that at least one of its nodes must be removed to avoid this cycle. The model uses the same binary variables y_v, $v \in V$, as before, where $y_v = 1$ indicates that vertex $v \in \overline{F}$, i.e., is part of the DAG and, thus, not included in the DFVS, and is stated as follows:

$$\min |V| - \sum_{v \in V} y_v \tag{8}$$

$$\text{s.t.} \sum_{v \in C} y_v \leq |C| - 1 \qquad \forall C \in \mathcal{C} \tag{9}$$

$$\sum_{v \in K} y_v \leq 1 \qquad \forall K \in \mathcal{K} \tag{10}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{11}$$

The objective function (8) is the same as in the MTZ-based formulation and minimizes the number of vertices included in the DFVS. The cycle elimination constraints are given by Inequalities (9), where $\mathcal{C}$ denotes the set of all simple cycles in G, and $C \in \mathcal{C}$ is one of these cycles represented as the set of the involved vertices.

As there can be exponentially many cycles in an input graph, the model has exponentially many such constraints. For practically solving the model, we therefore rely on lazy constraint generation for dynamically adding the cycle elimination constraints during the solving process only for cycles actually occurring in intermediate integral solutions. Such cycles are identified by breadth-first search. To speed up the solving process, we statically include from the beginning constraints for *2-cycles*, i.e., cycles of length two. To do so, we derive a graph $G_{2\text{cyc}}$ that consists of the vertices involved in 2-cycles and has an undirected edge for each 2-cycle in the original graph. We then strengthen the initial model by greedily identifying a set $\mathcal{K}$ of cliques in this graph $G_{2\text{cyc}}$. From each such clique, only a single vertex can appear in any DAG, which is expressed in our model by Inequalities (10). Note that a DFVS in G has to be a vertex cover in $G_{2\text{cyc}}$ and adding constraints

for cliques is a common way to strengthen formulations for the vertex cover problem. For a detailed description of the greedy construction of cliques, we refer to [14]. The domain of the decision variables is given by (11).

4 Graph Reduction

Our solving procedure starts with the preprocessing of the input graph, where various measures for graph reduction are taken, with the main step being the repeated application of five reduction rules. We adopt all five rules by Levy and Low [22] as mentioned in Section 2, but we combine the rules IN0 and OUT0 into a single step called IN/OUT0. Our fifth reduction operation called SCC-reduction is inspired by the work of Park and Akers [24]. It is based on the partitioning of the graph into its strongly connected components (SCCs) and directly solving all subproblems with less than three vertices. This rule as well as the LOOP rule are able to identify a part of the vertices belonging to a minimum DFVS, which are then stored in a preliminary DFVS and added to the final solution in the end. As all these operations preserve optimal solutions, their application order can be arbitrary. Nevertheless, a certain order can be beneficial for performance reasons and is in our work as follows: IN/OUT0, IN1, OUT1, LOOP, and SCC.

After the exhaustive application of these reduction rules, the remaining graph is partitioned into its SCCs, which are then sorted according to non-decreasing size. If their number exceeds 1000, we successively merge SCCs in the obtained ordering into subgraphs of up to 100 vertices to reduce memory consumption. All resulting subproblems are processed separately and sequentially, each starting with another repeated application of the five reduction rules. Afterwards, components with exactly three remaining vertices are always cliques that are dealt with by randomly selecting two vertices that are also added to the preliminary DFVS. Subgraphs or SCCs with up to 100 vertices are also handled differently from bigger ones as they are directly solved with the MILP solver. This is also the reason why we chose the size of the merged subgraphs in case of many SCCs to be at most 100 as the small SCCs would have fallen into this category as well. The remaining bigger SCCs are then passed to the LNS.

5 Construction Heuristic and Local Search

An initial solution for the LNS is generated by the consecutive execution of a construction heuristic (CH) and a local search. The employed CH combines the concept of topological orderings with the greedy function by Cai et al. [18]:

$$h(v) = \deg^-(v) + \deg^+(v) - \lambda \cdot |\deg^-(v) - \deg^+(v)| \tag{12}$$

for vertices $v \in V$, where $\deg^-(v)$ and $\deg^+(v)$ denote v's out- and in-degrees, and we use the recommended balancing factor $\lambda = 0.3$. According to this heuristic, vertices with lower h-values are less likely to belong to a minimum DFVS, or in other words, they are more promising to be contained in a maximum topological ordering. Thus, CH sorts the vertices in non-decreasing h-value order and greedily builds a topological ordering of maximal cardinality, skipping all vertices that would introduce an arc to an already selected vertex. In the end, the vertices that are not contained in the topological ordering constitute a valid DFVS and provide an initial solution.

This solution is then possibly further improved by applying LS with the one-flip neighborhood structure, which is based on moving one vertex from the current DFVS to the corresponding DAG if no cycle is introduced. In this LS, the vertices in the DFVS are again considered in non-decreasing h-value ordering. When accepting a solution, we additionally store the position of the last selected vertex in the ordering to avoid repeated checks of already rejected candidate solutions in the following iterations of the LS. This enhancement does not only strengthen the guidance of the local search but also speeds up the search process.

6 Large Neighborhood Search

Large neighborhood search [11,12] is a prominent LS-based metaheuristics where the key idea is to apply pairs of *destroy* and *repair* methods for identifying promising candidate solutions. A

neighborhood is here implicitly defined by the applied destroy operator, which usually selects a subset of the problem's decision variables to be re-optimized while the remaining variables are kept as set in the current incumbent solution. In this way, a subproblem is induced, for which the repair method tries to find an optimal or promising heuristic solution. As in classical LS, we accept new solutions that are better than the incumbent while worse solutions are discarded. Besides, we use a combination of two termination criteria: a runtime limit and a limit on the number of consecutive iterations without improvement.

6.1 Enlarge-DAG Neighborhood Structure

In our *enlarge-DAG neighborhood*, the basic idea is to remove vertices from a current DFVS, which implies to add them and their original incident edges back into the corresponding DAG. The obtained graph is then larger but in general not acyclic anymore. A DFVS subproblem is induced, which is typically smaller than the original DFVSP and therefore should be easier to solve.

More formally, let F be a valid DFVS and the vertex set of the corresponding DAG be denoted by $D = V \setminus F$. The destroy method selects a k-element subset $A \subseteq F$ of the DFVS. The destroyed solution is then represented by $F' := F \setminus A$ and the DAG is augmented to graph $G' = G[\, D' \,]$ with $D' := D \cup A$. In this induced DFVS subproblem, the repair operator has to remove a set $F'' \subset D'$ of vertices to obtain a DAG again. The repair method first applies the graph reduction rules from Section 4 again to possibly reduce G'. As the MILP models we proposed in Section 3 can be highly effective on smaller instances, we ultimately apply them as repair methods.

Still, we need to clarify crucial details on how the selection of the nodes in the destroy operation is done and how the MILP solver actually is applied. In the following paragraphs, we propose different options for these, which we will later experimentally evaluate and compare.

6.2 Degree of Destruction

Parameter k specifies the number of vertices selected in the destroy operation and thus controls the *degree of destruction*. Selecting a suitable value is crucial for an LNS procedure to perform well. If k is too small, the neighborhood is too restricted and the benefit of applying a powerful repair is mitigated. On the other hand, a too large degree of destruction can result in too complex subproblems. The repair method may then either require too long for single iterations or may yield unsatisfactory results in limited time. Due to the importance of this parameter, we consider different techniques for choosing k as described in the following.

Fixed Degree. In the simplest case, we use a constant value k throughout all LNS iterations and for all DFVSP instances. We denote this strategy by *fixed_degree*$(\cdot)$ where $\cdot$ stands for the selected value.

Random Selection. Similar to Ropke and Pisinger [25], k is here randomly selected at each LNS iteration from a certain range. While not every choice of k will be ideal in any situation, the hope is that this random selection makes the LNS less sensitive to occasional bad choices and thus generally more robust.

Dynamic Selection. In this more advanced technique, we choose the degree of destruction for a DFVSP instance in dependence on properties of the instance graph and the MILP formulation used for repair, i.e., defining rules to predict suitable fixed values for each instance. To do so, we build on experience gained from earlier performed test runs with the fixed_degree strategy for a range of different values k. We refer to Section 7 and [14] for the details on how these experiments were performed. More specifically, we consider the following five strategies for the dynamic selection.

#2-cycles. In the results for the fixed_degree strategy, we observed that the number of 2-cycles in the input graphs has a substantial impact on how promising different values for k in general are. Making use of these results, we came up with a partitioning of the numbers of 2-cycles into seven consecutive ranges, for which we identified different values for k to yield the best average final solution values. Ties were broken in favor of the smallest k. As the most suitable values also depend significantly on whether the MTZ- or CEC-based model is used for repair, we also distinguish between these as second dimension.

best_triple. For this strategy, we preselect for each MILP formulation three values for k, called *best-mean*, *mode*, and *most-best*, which performed best over all fixed_degree strategy tests in different senses. Best-mean denotes the degree of destruction that achieves the best mean solution quality just as above but now over all test instances. The mode value was selected by identifying for each instance all k-values that find a solution of smallest known size, taking the smallest in each case, and then determining the mode of these values over all instances. Similarly, most-best is the smallest k for which in general the most solutions of smallest size, hence the best solutions, were found. The LNS then selects in each iteration one of these three values at random.

#2-cycles_best_triple. This variant combines the above two approaches. Thus, we reuse the classification of the graphs into one of the seven number of 2-cycle ranges and the distinction between the used MILP models, and utilize individually derived best-mean and most-best values for k, from which we again select randomly in each LNS iteration.

#2-cycles_regression. Here, we performed a regression to obtain a function yielding the degree of destruction in dependence on the number of 2-cycles. As the distribution of the number of 2-cycles of our benchmark instances as well as the distribution of the instance-best values for k are rather skewed, we found it most meaningful to perform a linear regression over the base-10 logarithms of the numbers of 2-cycles and the base-10 logarithms of the k-values that performed best for each instance. In simplified form, this resulted in

$$k = \begin{cases} 15.85 \cdot z^{0.365} & \text{for the MTZ model} \\ 20.14 \cdot z^{0.433} & \text{for the CEC model,} \end{cases} \tag{13}$$

where z is the the number of 2-cycles. The k-value determined by this function for a given DFVSP instance is then used in each LNS iteration again.

#vertices_regression. For this strategy, we performed a regression over the number of nodes $|V|$ of the graph and the best k-value for each instance. Again, we more specifically applied a linear regression over the base-10 logarithmic values of both. This led to

$$k = \begin{cases} 0.2917 \cdot |V|^{0.808} & \text{for the MTZ model} \\ 0.1159 \cdot |V|^{1.004} & \text{for the CEC model.} \end{cases} \tag{14}$$

6.3 Element Selection

Aside from the degree of destruction, it is also important how elements are chosen in the destroy method as this also impacts the speed of convergence, diversification, and intensification of the search. Remember that in our destroy method, we denoted by A the set of vertices that are selected for removal from the DFVS. Our selection again makes use of the heuristic function $h(v)$ given in Equation (12). However, as a purely greedy selection would lead to a too narrow search and stagnation, we choose each vertex for A by tournament selection. Preliminary experiments indicated that a tournament size of three is a robust choice, which we apply throughout this work.

6.4 Dynamic MILP Selection

We consider two ways of determining which MILP formulation is used in the repair operator of the LNS: preselecting one without any knowledge or choosing it in dependence on instance characteristics as described in the following. Most importantly, we observed differences in the average performance of the MILP formulations as repair operator for various instances, which seem to be linked to the product of the number of 2-cycles and the density of the graphs. Similarly to the #2-cycles strategy for the selection of the degree of destruction, we therefore determined ranges for the values of these products, in which one of the two MILP models performed better than the other in terms of final solution qualities. A general conclusion for the largest product values is that in the context of the LNS, the CEC-based model is usually the better choice, while for the standalone application the MTZ-based model is on average more beneficial. Detailed results can be found in Chapter 5.6.3 of [14].

7 Computational Study

All experiments were performed on single cores of a cluster with Intel Xeon E5540 processors with a memory limit of 23 GB. Our solving procedures were implemented in Julia 1.7.1, and we used Gurobi 9.5.1[4] as MILP solver in single-threaded mode with a memory limit of 20 GB. Each run had a general time limit of 550 s whereof at most 60 s are used for the LS to improve the initial solution.

We use three existing benchmark instances sets, which all consist of simple, directed graphs without any self-loops or multi-edges. Two of the data sets are taken from the heuristic track of the PACE 2022 challenge[5] and one set[6] comes from Pardalos et al. [17]. We will refer to the two sets from PACE 2022 as *pace-public* and *pace-private* and to the third as *fsp-data*. The sets pace-public and pace-private consist of 100 instances each, and the graphs have quite different sizes and structures with $|V| \in \{843, \dots, 2394385\}$ nodes and $|E| \in \{2103, \dots, 5105039\}$ arcs. These instances were mostly generated using KaGen[7] [26], a set of generators for network models, and five graph models, but also contain real-world instances. The fsp-data set consists of 40 smaller graphs which are divided into four subsets with 50, 100, 500, and 1000 vertices, respectively. Each subset contains ten randomly created instances of varying density.

Our main evaluation metric is the *solution quality*, which is calculated for a solution F of an instance with a best known solution F^* as $100\% \frac{|F^*|}{|F|}$. Besides, we employ the *geometric mean* for averaging the solution qualities over all instances of a benchmark set. Both is adopted from the ranking method used in the PACE 2022 challenge. For the pace-public instances, we utilize the smallest solutions recorded by the PACE 2022 challenge as best known solution values. As the submitted solutions for the pace-private instances are not accessible, we have to rely on our own results[8] for this data set. For the fsp-data benchmark set, the best known solution values[8] are derived from the results reported by Pardalos et al. [17], Galinier et al. [8], and Tang et al. [10].

7.1 Comparison of MILP Models in Standalone Solving

Before employing the proposed MILP models within the LNS, we have a look at their standalone solving capabilities for the pace-public instances. As for the LNS, we initially apply our graph reduction procedure to each input graph. In case that the reduction yields multiple subproblems, we split the allowed computation time equally among the subproblems and solve them sequentially. If a subproblem is solved to optimality before its assigned time is used up, the spare time is evenly split and distributed over the remaining subproblems. For the CEC-based formulation, we additionally use a time limit for the construction of the 2-cycle constraints as this part may already need too much time for the largest instances. Following preliminary tests, we decided on half of the total time allotted for the solving of a (sub-)instance. We observe that over the whole benchmark set, models MTZ and CEC show similar performances with MTZ being about 2.34 percentage points better in terms of the geometric mean but CEC finding more proven optimal solutions, as shown in Table 1.

As for the LNS an initial solution is always created by CH+LS, we also evaluate the impact of warm starting the standalone MILP solving with solutions from CH+LS. Table 1 and Figure 1 show that providing such initial solutions is clearly beneficial for the MILP solving. There is less variance in the results produced with the warm starts, no solution has less than 45% of the best known solution values, and the solutions exhibit an overall higher quality. The difference between the performances of the CEC- and MTZ-based formulations in terms of average solution quality decreases to less than one percentage point.

Formulation CEC achieves now on average 93.43% quality and outperforms MTZ with an average of 92.58%. The main reason for the significant improvement of especially CEC seems to be linked to the fact that this model performs particularly poorly on a small number of instances. There, the MILP solver returned the solutions from CH+LS but could not further improve on them.

[4] `https://www.gurobi.com`

[5] `https://pacechallenge.org/2022/tracks/#heuristic-track`

[6] `http://mauricio.resende.info/data/index.html` (feedback set problem)

[7] `https://github.com/sebalamm/KaGen`

[8] `https://www.ac.tuwien.ac.at/files/resources/results/DFVSP/DFVSP-benchmarks-results.csv`

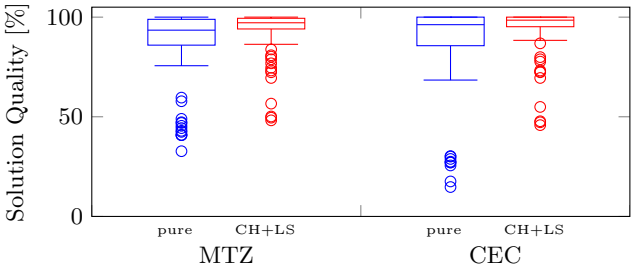


Fig. 1: Box plots showing the solution qualities of standalone MILP solving without (pure) and with warm starts (CH+LS) on pace-public instances.

Table 1: The geometric mean of the solution qualities of standalone MILP solving without (pure) and with warm starts (CH+LS).

Formulation	Avg. Solution Quality [%]	Best Known Solutions
MTZ_{pure}	84.93	18
MTZ_{CH+LS}	92.58	18
CEC_{pure}	82.59	38
CEC_{CH+LS}	93.43	38

7.2 Comparison of LNS Variants

Now, we evaluate various configurations of the proposed LNS with the enlarge-DAG neighborhood. We consider the different selection strategies for the degree of destruction and the two alternative MILP models for the repair. All experiments are first performed on the pace-public data set and in order to test the generalization capabilities of our dynamic selection strategies, we further apply the approaches to the pace-private data set. The overall time limit is again 550 s per instance. Graph reduction is applied, and if the graph is split into independent subgraphs, the time limit is again distributed as described above for the standalone MILP solving. Moreover, we terminate the LNS also after 50 consecutive unsuccessful LNS iterations. In this case, we expect no or only minor further improvements, and time is probably better spent for solving successive subproblems. Gurobi is now given up to 90 s for solving each model to repair a solution. If it fails to find an optimal solution within the first 60 s, the optimality gap is increased to 0.5% for the remaining optimization.

Regarding the degree of destruction, we started with investigating the fixed_degree and random selection approaches. For fixed_degree, we considered 15 different values from 25 up to 100000, and for random selection the range to randomly choose from was decided to be $\{5, \ldots, 1000\}$ following results of preliminary tests. We tested the resulting variants with both the MTZ- and the CEC-based formulations on the pace-public benchmark set and compared them in terms of the average solution quality. Figure 2 illustrates the results and Table 2 shows average solution qualities of some of the best performing approaches over the whole pace-public benchmark set. When using CEC, the highest average solution quality of 95.10% is measured for $k = 75$ whereas the MTZ formulation peaks at 93.42% with $k = 25$. The average performance of the random selection lies right in between as both combinations with the two MILP formulations achieve a higher mean than fixed_degree(25) with MTZ and a lower mean than fixed_degree(75) with CEC. This shows that fixing the value of k for all DFVSP instances to a generally best value is not always advantageous as different instances benefit from different degrees of destruction. Note that in comparison to the standalone MILP results, especially those without CH+LS, we can already see a significant advantage of the LNS variants. They achieve improvements of up to twelve percentage points compared with the MILP solving without warm starts and in the range of two to three percentage points compared with the MILP warm start approaches.

We now move on to our dynamic strategies for selecting the degree of destruction as presented in Section 6.2, where we make use of the results gained from the fixed_degree tests. Table 3 shows average solution qualities and how often the best known solutions were achieved for the LNS using the five different dynamic selection strategies in conjunction with either the CEC model, the MTZ model, or the dynamic model selection strategy.

It turns out that on average, the approaches employing the CEC-based model always achieve better results than the corresponding ones with the MTZ-based model. We also observe that not all combinations of the MILP models with the dynamic selection strategies for k perform better than the random or fixed_degree selection in terms of the average solution quality. This illustrates that even rather simple selection techniques can provide good solutions. However, the best average solution quality is found by the #2-cycles strategy for both MILP models, with a mean of 96.15% for CEC and 94.57% for MTZ. The best_triple strategy when employing the CEC-based formulation achieves 32 best known solutions, which is by far the most and may be linked to the possibility of

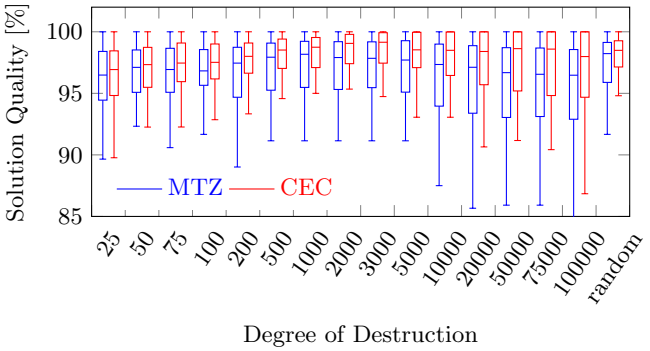


Fig. 2: Box plots showing the solution qualities of fixed_degree and random selection approaches, each with MTZ (blue) and CEC (red), on the pace-public instances.

Table 2: The geometric mean of the solution qualities for the pace-public instances.

Selection Strategy	MILP	Avg. Solution Quality [%]
fixed_degree(25)	MTZ	93.42
	CEC	94.20
fixed_degree(75)	MTZ	92.92
	CEC	95.10
random	MTZ	93.72
	CEC	94.87

using very different values for k up to 75000. Such a high value can result in total re-optimization done in the repair operator and this might also be the reason why this strategy does not perform as well on average because it leads to large subproblems that are frequently also too hard to solve with the MILP solver.

When combining the dynamic selection of the employed MILP model with three of the dynamic selection strategies for the degree of destruction, we are able to improve the average solution quality for two of them compared to the corresponding variants with a preselected formulation. The combination with the #2-cycles strategy shows the best performance out of the three with a mean solution quality of 96.21% and 15 best known solutions for the pace-public data set, as given in Table 3. Regarding the average solution quality, this is the overall best achieved result for this benchmark set, which validates the application of the dynamic selection of the MILP formulation.

In order to test the generalization capabilities of the dynamic selection strategies to other DFVSP instances, we have a look in particular at the pace-private data set. We again examine the geometric mean of the solution qualities and the number of found best known solutions, which are shown in Table 3 in the columns labeled pace-private. The observed mean values illustrate that the approaches using the MTZ-based formulation have a worse average performance than the others, with the exception of the best_triple combination where MTZ slightly outperforms CEC. However, the best_triple strategy achieves the lowest average solution quality of all CEC-based approaches, although it again finds the most best known solutions. In terms of mean values, two of the three variants employing the dynamic selection of the MILP formulation achieve improvements and the combination with the #2-cycles strategy again obtains the highest value with an average of 98.96%. Overall, we see similar results for the pace-public and pace-private data set, which confirms that our selection strategies generalize well.

This is also reflected in the observed results for the fsp-data set, as given in Table 3. The #2-cycles strategy in combination with the CEC-based model achieves the highest average solution quality and CEC-based approaches again achieve higher means than MTZ-based ones. The dynamic MILP selection is also superior to the MTZ model but it performs worse than solely using CEC models, which is in contrast to the results for the two pace data sets and indicates room for improvement of this selection strategy.

Lastly, we compare our LNS variants also with two state-of-the-art solving approaches for the DFVSP, which are the simulated annealing based methods SA-FVSP by Galinier et al. [8] and the newer SA-FVSP-NNS by Tang et al. [10]. As we do not have access to the respective implementations, we have to rely on the results reported in these works for the fsp-data benchmark set. Table 3 shows derived average solution qualities and numbers of obtained best solutions at the bottom. As Tang et al. [10] claim to have re-implemented the SA-FVSP from Galinier et al. and tested independently, the table also lists corresponding results in line SA-FVSP [10]. Unfortunately, there is a strong discrepancy in the results for SA-FVSP originally reported by Galinier et al. and later by Tang et al. for their re-implementation, for which no explanation is given. Thus, all these results need to be taken with care. In comparison to our LNS, the results reported by Galinier et al. are significantly better, while our results clearly dominate those reported by Tang et al. for both,

Table 3: Mean solution qualities and number of found best known solutions of the LNS variants for three benchmark sets and of the SA-based approaches from the literature.

MILP-based LNS Algorithms		Average Solution Quality [%]			Best Known Solutions		
Selection Strategy	Formulation	pace-public	pace-private	fsp-data	pace-public	pace-private	fsp-data
#2-cycles	MTZ	94.57	97.25	95.49	13	17	15
	CEC	96.15	98.83	**96.37**	12	26	18
	dynamic	**96.21**	**98.96**	96.01	15	26	15
best_triple	MTZ	93.68	96.48	93.38	13	15	15
	CEC	94.53	96.33	94.77	**32**	**39**	**19**
#2-cycles_best_triple	MTZ	94.34	97.10	95.54	13	17	15
	CEC	95.84	97.48	96.23	17	27	**19**
#2-cycles_regression	MTZ	93.89	95.83	94.80	8	12	15
	CEC	95.62	97.53	95.71	15	25	18
	dynamic	95.63	97.44	95.57	14	21	15
#vertices_regression	MTZ	93.62	95.81	94.53	6	10	15
	CEC	94.81	96.77	95.58	9	27	18
	dynamic	94.80	96.79	95.42	9	20	15
SA Algorithms							
SA-FVSP [8]				**99.77**			**27**
SA-FVSP [10]				63.24			1
SA-FVSP-NNS [10]				70.40			1

SA-FVSP as well as SA-FVSP-NNS. The performance gap between our approach and SA-FVSP [8] on the fsp-data set might be linked to our focus on the mostly substantially larger instances in pace-public and pace-private. Note also that it is unclear how the SA-based approaches scale to these larger instances and thus, more experiments with these approaches would be highly desirable.

8 Conclusion and Future Work

We proposed two MILP models for the DFVSP, one based on Miller-Tucker-Zemlin inequalities and one based on cycle elimination constraints. Both are effective for graphs up to a certain complexity but do not scale well to larger instances. Therefore, we investigated an LNS framework in which a MILP model is utilized for repairing solutions. Applying a suitable degree of destruction is crucial for the performance of the LNS, and consequently we proposed different selection strategies for this parameter. Among these are five dynamic strategies that exploit results obtained by tests with fixed values. Additionally, we also suggested a dynamic selection strategy for the MILP model to apply. Incorporating these strategies into the LNS framework resulted in multiple variants, which we evaluated on three benchmark sets.

Experiments with standalone MILP solving show that the MTZ model performs best in terms of average solution quality when no initial solution is provided, whereas the CEC model is superior in the case of warm starts with CH+LS. Using the CEC model within the repair operator of the LNS also leads to a higher mean quality than employing the MTZ model. The different LNS variants achieve average solution qualities that are up to 12 percentage points higher than those of standalone MILP solving. Regarding the selection strategies for the degree of destruction, even rather simple techniques such as random selection provide solutions of good quality but are clearly outperformed by some of our dynamic selection strategies, in particular #2-cycles and #2-cycles_best_triple. Combining the dynamic selection techniques with the dynamic selection of the MILP model leads to further improvements and to the overall best performing LNS variant for two benchmark sets.

There is still room left for improvement. With regard to the results of the PACE 2022 challenge, we assume that extending the graph reduction with even more sophisticated rules would be beneficial. Moreover, adopting ideas from the state-of-the-art simulated annealing based approaches, e.g., by trying to improve solutions from the MILP by a fast local search, appears promising. Last but not least, also more advanced machine learning approaches from the field of automated algorithm configuration can be applied.

References

1. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., Bohlinger, J.D., eds.: Complexity of Computer Computations, Springer (1972) 85–103
2. Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.M.: Approximation algorithms for the vertex feedback set problem with applications to constraint satisfaction and bayesian inference. In: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms. (1994) 344–354
3. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. Journal of the ACM **55** (2008) 1–19
4. Fleischer, R., Wu, X., Yuan, L.: Experimental study of FPT algorithms for the directed feedback vertex set problem. In: Algorithms – ESA 2009. Volume 5757 of LNCS., Springer (2009) 611–622
5. Carrabs, F., Cerulli, R., Gentili, M., Parlato, G.: Minimum weighted feedback vertex set on diamonds. Electronic Notes in Discrete Mathematics **17** (2004) 87–91
6. Carrabs, F., Cerulli, R., Gentili, M., Parlato, G.: A linear time algorithm for the minimum weighted feedback vertex set on diamonds. Information Processing Letters **94** (2005) 29–35
7. Cutello, V., Oliva, M., Pavone, M., Scollo, R.A.: An immune metaheuristics for large instances of the weighted feedback vertex set problem. In: 2019 IEEE Symposium Series on Computational Intelligence, IEEE (2019) 1928–1936
8. Galinier, P., Lemamou, E., Bouzidi, M.W.: Applying local search to the feedback vertex set problem. Journal of Heuristics **19** (2013) 797–818
9. Melo, R.A., Queiroz, M.F., Ribeiro, C.C.: Compact formulations and an iterated local search-based matheuristic for the minimum weighted feedback vertex set problem. European Journal of Operational Research **289** (2021) 75–92
10. Tang, Z., Feng, Q., Zhong, P.: Nonuniform neighborhood sampling based simulated annealing for the directed feedback vertex set problem. IEEE Access **5** (2017) 12353–12363
11. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming, Springer (1998) 417–431
12. Pisinger, D., Ropke, S.: Large neighborhood search. In Gendreau, M., ed.: Handbook of Metaheuristics. Springer (2010) 399–419
13. Jatschka, T., Rodemann, T., Raidl, G.R.: A Large Neighborhood Search for a Cooperative Optimization Approach to Distribute Service Points in Mobility Applications. In Dorronsoro, B., Yalaoui, F., Talbi, E.G., Danoy, G., eds.: Metaheuristics and Nature Inspired Computing, Springer (2022) 3–17
14. Bresich, M.: Hybrid metaheuristics based on large neighborhood search and mixed integer linear programming for the directed feedback vertex set problem. Master's thesis, TU Wien, Austria (2023)
15. Noughabi, H.A., Baghbani, F.G.: An efficient genetic algorithm for the feedback set problems. In: 2014 Iranian Conference on Intelligent Systems, IEEE (2014) 1–4
16. Baharev, A., Schichl, H., Neumaier, A., Achterberg, T.: An exact method for the minimum feedback arc set problem. ACM Journal of Experimental Algorithmics **26** (2021) 1–28
17. Pardalos, P.M., Qian, T., Resende, M.G.C.: A greedy randomized adaptive search procedure for the feedback vertex set problem. Journal of Combinatorial Optimization **2** (1998) 399–412
18. Cai, X., Huang, J., Jian, G.: Search algorithm for computing minimum feedback vertex set of a directed graph. Jisuanji Gongcheng/Computer Engineering **32** (2006) 67–69
19. Swat, S.: PACE Solver Description: DiVerSeS – A Heuristic Solver for the Directed Feedback Vertex Set Problem. In: 17th Int. Symposium on Parameterized and Exact Computation. Volume 249 of LIPIcs., Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022) 27:1–27:3
20. Du, Y., Zhang, Q., Xu, J., Zhang, S., Liao, C., Chen, Z., Sun, Z., Su, Z., Ding, J., Wu, C., Lu, P., Lv, Z.: PACE Solver Description: Hust-Solver – A Heuristic Algorithm of Directed Feedback Vertex Set Problem. In: 17th Int. Symposium on Parameterized and Exact Computation. Volume 249 of LIPIcs., Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022) 29:1–29:3
21. Bathie, G., Berthe, G., Coudert-Osmont, Y., Desobry, D., Reinald, A., Rocton, M.: PACE Solver Description: DreyFVS. In: 17th Int. Symposium on Parameterized and Exact Computation. Volume 249 of LIPIcs., Dagstuhl, Germany, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022) 31:1–31:4
22. Levy, H., Low, D.W.: A contraction algorithm for finding small cycle cutsets. Journal of Algorithms **9** (1988) 470–493
23. Lin, H.M., Jou, J.Y.: On computing the minimum feedback vertex set of a directed graph by contraction operations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **19** (2000) 295–307
24. Park, S., Akers, S.B.: An efficient method for finding a minimal feedback arc set in directed graphs. In: Proc. of the IEEE Int. Symposium on Circuits and Systems. Volume 4., IEEE (1992) 1863–1866
25. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science **40** (2006) 455–472
26. Funke, D., Lamm, S., Sanders, P., Schulz, C., Strash, D., von Looz, M.: Communication-free massively distributed graph generation. In: 2018 IEEE Int. Parallel and Distributed Processing Symposium, IEEE (2018) 336–347

Cluster images with AntClust: a clustering algorithm based on the chemical recognition system of ants

W. Oed[1] and P. Memarmoshrefi[2]

[1] Department of Computer Science
Georg August University of Goettingen, Germany
winfired.oed@stud.uni-goettingen.de
[2] Department of Computer Science
Georg August University of Goettingen, Germany
memarmoshrefi@informatik.uni-goettingen.de

Abstract. We implement *AntClust*, a clustering algorithm based on the chemical recognition system of ants and use it to cluster images of cars. We will give a short recap summary of the main working principles of the algorithm. Further, we will describe how to define a similarity function for images and how the implementation is used to cluster images of cars from the vehicle re-identification data set. We then test the clustering performance of *AntClust* against DBSCAN, HDBSCAN and OPTICS. Finally one of the core parts in *AntClust*, the rule set can be easily redefined with our implementation, enabling a way for other bio-inspired algorithms to find rules in an automated process. The implementation can be found on GitLab [3]

1 Introduction

In this work we describe the principle and usage of *AntClust*, a clustering algorithm based on the chemical recognition system of ants, which was developed by Nicolas Labroche, Nicolas Monmarch´e and Gilles Venturi [1].

The clustering algorithm's general idea is based on ant's chemical recognition system. Ants form colonies where many individuals help the colony to survive by maintaining it. To prevent harm from intruders the individual ants of a colony have developed a mechanism to recognize their nestmates. Labroche et al. refer to the sources [3, 4] for more information. Generally the nestmate recognition is based on odors. These are individual to every ant, meaning that every ant has its own odor. Additionally, every ant maintains an odor template which is generated based on its encounters with other ants. If two ants meet, they will recognize each other's odors and if the odors are similar enough - the ants "like" to smell each other - they will accept, know each of them belongs to the same colony. Rejection can happen if the odors do not match the template. As the odor, the template is individual to every ant. It is constantly updated during the encountering of other ants. Thus there is no global colony template and the whole identification process is decentralized. It is only defined in the sum of the different templates carried within every ant and is a dynamic process, subject to constant change.

The algorithmic idea from Labroche et al. is to create artificial ants which will be initialized with a certain genetic. This artificial genetic is a data tuple of a data set, e.g., an image. Based on this genetic, the ant will define its own template for recognizing potential nestmates and repel intruders. This will generate artificial ant colonies or differently framed, it will form clusters. We implement the described algorithm using Python and use our implementation to cluster images of cars, taken by public surveillance cameras. In this clustering task, the different images of a distinct car form one particular cluster. Images are taken from the *vehicle re-identification (VeRi)* data set [7]. Clustering images is not an easy task since clustering needs a similarity measure between its data tuples. For this we construct a concept for a similarity function that work on images and use it in our implementation.

[3] https://gitlab.com/Winnus/antclust

This paper is structured as follows. First, there will be a general description of the algorithm - which is a summary of the original paper - in section 2. In the following section 3, we will describe our approach to construct a similarity function that works on images. So then, the obtained results and the clustering performance will be shown in section 4. Finally, there is a general discussion about the algorithm, its implementation and possible improvements as well as thoughts on how to improve/combine it with other bio-inspired and evolutionary-based approaches in section 5.

2 AntClust Theoretical Framework

This section is a summary of the original *AntClust* paper [1]. It explains how *AntClust* works and how the algorithm is structured. The AntClust algorithm can be summarized into the sub actions in table 1.

Table 1. AntClust algorithmic steps

Phase	
1	initialize ants
2	initialize ant templates
3	randomly meet ants and apply meeting rules
4	shrink ant colonies
5	re-assign ants with no colony

These sub actions will now be described one after another.

For this section, it will be assumed that the data set to perform the clustering algorithm is a very simplistic one, having just one feature. This feature is just a real number between 0 and 1. Therefore one tuple from the set would be just a number. This simplifies in order to understand the working principle of *AntClust*.

Phase one is the initialisation of the ants. To represent the data set as an ant colony *AntClust* will create artificial ants where each ant will correspond to one distinct data tuple from the data set -meaning in our example, the data set [1,2,3] will be represented by three ants, each holds one number as their genetics-. Inside the computer these ants are program objects, an instance of a certain class. Every ant has their own attributes.

- Genetic of an ant is defined as one data tuple from the data set.
- Label of an ant indicates to which colony or cluster the ant belongs.
- Template is distinct to every ant. The template is a real number $0 \leq x \leq 1$ and is used to estimate whether another ant is accepted during a meeting or not. The template is dynamic and will evolve over time.
- Age is an indicator of how many meetings the ant has had with other ants during the meeting phase of the algorithm.
- M is an estimator, $0 \leq M \leq 1$, that reflects how successful the ant is during its meetings with other ants. If the ant is not very well accepted during its meetings, M will decrease
- M^+ is an estimator, $0 \leq M^+ \leq 1$, which reflects how well the ant is accepted inside its nest / colony. This estimator is therefore updated whenever the ant is meeting with another ant and is reset to zero if the ant loses its label.
- $Max(Sim(ant, \cdot))$ is the maximal similarity obtained during a meeting with another ant. Here "$\cdot$" means all other ants an ant has met.
- $\overline{Sim}(ant, \cdot)$ is the mean similarity between this ant and all the ants this ant has met.

To initialize an ant and finish phase 1, the parameters will be set as follows:

- Genetic $\longleftarrow i^{th}$ tuple of the data set.
- Label $\longleftarrow$ 0, Age $\longleftarrow$ 0, M $\longleftarrow$ 0, $M^+ \longleftarrow 0$.
- $Max(Sim(ant, \cdot)) \longleftarrow 0$, $\overline{Sim}(ant, \cdot) \longleftarrow 0$.
- Template $\longleftarrow$ 0, will be initialized directly in the next step.

Initialisation of the template is done in phase 2, where the template of an ant is defined by

$$template \longleftarrow \frac{\overline{Sim}(ant, \cdot) + Max(Sim(ant, \cdot))}{2}$$

The template serves the function of determining a distance in by which the ants should accept each other or not. It is a reference that estimates how far ants can be - in a distanced manner - away from each other but still belonging to the same colony. To initialize it, the ant will have meetings with randomly selected ants. The number of meetings can be freely defined, but Labroche et al. suggest that it is calculated [4]. After these meetings, the ants will have formed their template and the algorithm can now continue with phase 3.

In phase 3, two ants are randomly chosen and meet each other. Meeting in the sense of *AntClust* means that a rule set is applied. The rule set given by Labroche et al. is defined below. Given rules will be applied one after another. If no rule matches, then nothing will happen. It is assumed that there is an ant_i and an ant_j with their corresponding variables $x_{i,j}$.

- New colony creation rule (R1):
  $If(label_i = label_j)$ *and* $Acceptance(i, j)$ then a new colony will be created by defining a new label $label_{new}$ and this label will be assigned to the ants
  $label_{i,j} \leftarrow label_{new}$.
  If the ants do not accept each other, R6 is applied.

- Ant with no label is assigned to existing colony (R2):
  $If(label_i = 0 \wedge label_j \neq 0)$ *and* $Acceptance(i, j)$
  then $label_i \longleftarrow label_j$. This rule applies symmetrically if the condition is reversed.

- accepting of colony mates (R3):
  $If(label_i = label_j) \wedge (label_i \neq 0) \wedge (label_j \neq 0)$ *and* $Acceptance(i, j)$
  then increase M_i, M_j, M_i^+, M_j^+.
  Increasing means $x \leftarrow (1 - \alpha) \cdot x + \alpha$, where $\alpha = 0.2$.

- not accepting of two colony mates (R4):
  $if(label_i = label_j) \wedge (label_i \neq 0) \wedge (label_j \neq 0)$ *and* $Acceptance(i, j) = False$, increase M_i, M_j, decrease M_i^+, M_j^+.
  Decreasing means $x \leftarrow (1 - \alpha) \cdot x$, where $\alpha = 0.2$. Additionally, the ant with the smaller colony integration value - which is ant_i $if(M_i^+ < M_j^+)$ or ant_j if the condition is reversed - loses its label ($label \leftarrow 0$) and does not belong to a colony anymore.

- meeting of colony different ants (R5):
  $if(label_i \neq label_j) \wedge Acceptance(i, j)$ decrease the colony size estimator M_i, M_j. The ant with the lower estimator will change its label and belongs to the colony of the other ant.

- Default rule (R6):
  If none of the above rules apply, nothing will happen during the meeting of the two ants.

[4] The iteration amount can be computed by $0.5 \cdot \alpha \cdot N$, where N is the number of ants, i.e. data tuples inside the data set. $\alpha = 150$ was tested by Labroche et al. and found as the most general optimal value. For some data sets better results can be obtained by tuning the parameter.

The above rules give a working framework to form clusters as explained now. In the very beginning all ants are initialized with no label - i.e. $label \leftarrow 0$. Thus when two ants meet in the beginning, mostly R1 will be applied. This will create a lot of little clusters containing only two ants. Once more and more ants have got a label assigned through R1, it becomes more likely that R2 is applied and thus, the initial formed clusters begin to grow. If two ants belong to the same cluster, there are two cases: they accept each other or they do not. In the first case, R3 will be applied which will increase the cluster size estimator M and the colony integrity estimator M^+, which makes sense since it seems that the colony is quite big if two randomly chosen ants belong to the same colony, accepting each other means that the colony is still in a good integrity state. In the second case, R4 will be applied as the colony mates do not accept each other. This will increase the colony size estimator since the colony must be relatively big if two randomly chosen ants belong to the same colony and decrease the integrity estimator M^+ since the two ants did not accept each other but belong to the same colony, suggesting the integration of colony members is relatively low. If two ants meet that do not belong to the same colony but accept each other then R5 is applied which will, over time, lead to the ability that smaller colonies - where many of these are initially formed in the beginning via R1 - getting integrated into the bigger colonies. For all other cases, the default rule is applied and thus nothing happens.

One thing not explained until now is the acceptance function $Acceptance(ant_i, ant_j)$. It will return true or false based on the comparison of the ants templates against their similarity and is defined by

$$\begin{aligned} Acceptance(ant_i, ant_j) \leftrightarrow \\ (Sim(ant_i, ant_j) > template_i) \wedge \\ (Sim(ant_i, ant_j) > template_j) \end{aligned}$$

Where $Sim(ant_i, ant_j)$ is the similarity, $0 \leq similarity \leq 1$. A similarity of 1 means that the two ants are completely similar, a similarity of 0 means they are anti similar. The similarity function needs to be defined based on the data set *AntClust* is running on. Remember that each tuple of the data set defines one ant by defining the genetics of that ant, letting each ant represent one data tuple inside the set. If the data set is the simplistic one described above - just numbers between zero and one - a similarity function would simply be $Sim_{1d}(x, y) = 1 - |x - y|$. Therefore $Sim(ant_i, ant_j)$ would simply extract the genetics of the ants - which is then a single number for every ant representing one data point inside the data set - and put it into $Sim_{1d}()$ to compute the similarity. In a more sophisticated data set, each tuple of the set might contain more than one feature and these features might not be just numbers but vectors. For each feature, the user would need to define and tell *AntClust* which similarity function should be used. The mean similarity from all used similarity measures will thereafter be used to compute the total similarity of two ants.

$$Sim(ant_i, ant_j) = \frac{1}{n_{sim}} \cdot \sum_{w}^{n_{sim}} Sim_w(x_w, y_w)$$

Where n_{sim} is the number of similarity measures or features, $Sim_w()$ one specific similarity function for the data type w, x_w and y_w the w^{th} feature of the data tuple extracted from the ant genetic, where x_w is extracted from ant_i and y_w from ant_j.

With the above similarity definition it is possible to run the algorithm on any data set if a similarity function can be specified by the user.

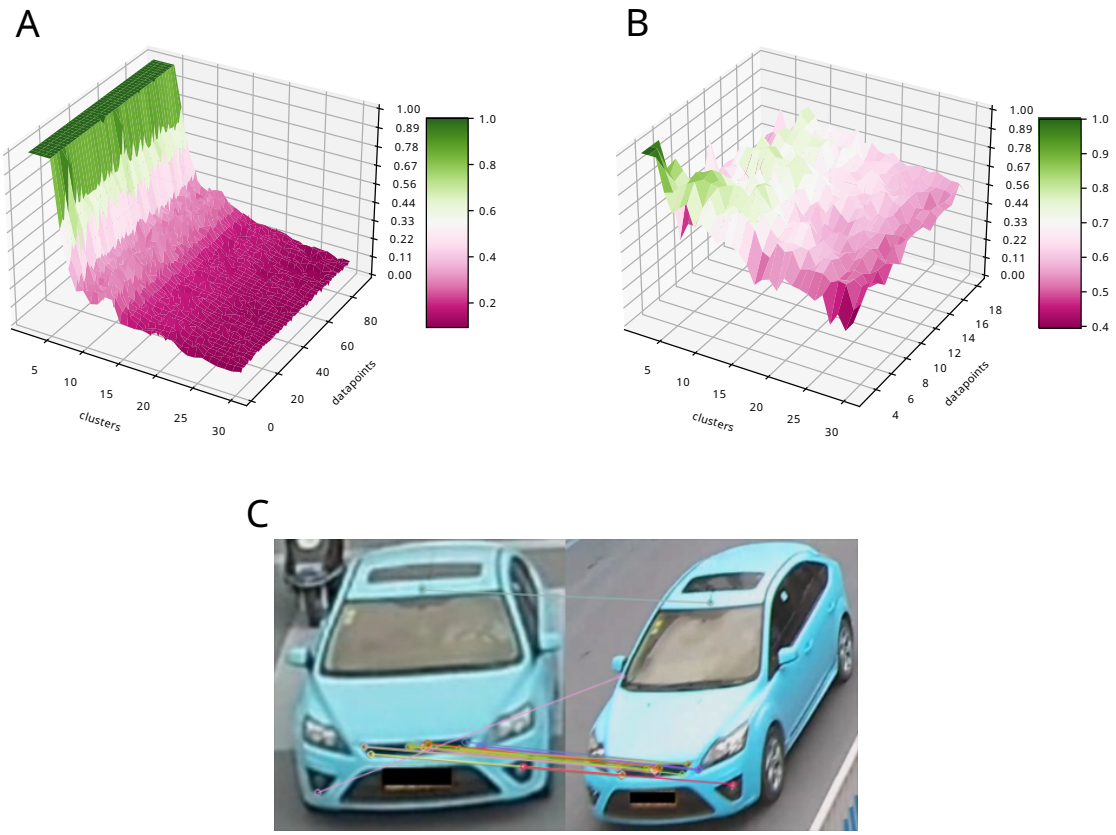


Fig. 1. Adjusted Rand Index (ARI) score for the self generated float data set (**A**) and image data set (**B**). The X-axis represent the number of clusters. The Y-axis shows the amount of data tuples in each cluster. This means that for each amount of clusters on the X- axis, all amounts of data tuples on the Y axis are tested in a clustering test. The ARI score is shown on the Z-axis. Here a score of 1 indicates a perfect clustering - found labels match exactly the ground truth labels - where a score of 0 indicates a random labeling of the data. (**C**) shows the 20 best matching features between two images of the same car from the VeRi data set.

3 Proposed Method

In most cases, clustering is performed on numerical data. However, we apply clustering on images [5]. The images we cluster are images of cars from the vehicle re-identification (VeRi) [6] data set. This set contains 50.000 images of 776 vehicles captured by 20 different cameras. A re-identification of a vehicle takes place if all images of one distinct vehicle are labeled into the same cluster. This is the task for the clustering algorithm. To achieve this task, a clustering algorithm uses a distance function or metric which tells how similar or anti-similar two data tuples are. For numerical values, there exist many metrics and the most common used are the Manhattan or the Euclidean distance. On images, distance metrics are not trivial to define. The similarity of an image might not be defined by comparing the pixel values of each pixel and calculating the difference. However, pictures usually contain certain points of interest, so-called features. For example, an image of the sky might mostly contain blue parts, which are not very interesting when comparing the picture. Yet an interesting part might be a bird flying through the sky. To detect such features there exist many different feature detection methods. One well-known feature detection method is the Harris Corner Detection algorithm which will detect the parts of the image where corners cross. Harris corner detection has problems when the scale of the image changes. There exit other more robust and faster methods like Scale-Invariant Feature Transform (SIFT) or Speeded-Up Robust Features (SURF). However, these methods are patented. A more performant and not patented method comes from the developers of the open source computer vision library *OpenCV* [7]. Their method is called Oriented FAST and Rotated BRIEF (ORB), where BRIEF stands for Binary Robust Independent Elementary Features [5]. The method is implemented in *OpenCV* and is free to use for everyone.

We use the *ORB* features to define a distance metric on images with it. Features of images are image specific and can be compared to each other. By comparing the features - which is usually done using the Manhattan norm - one retrieves a distance between features. Thus it is possible to define a similarity metric for images with them. *OpenCV* contains feature matchers which allow to compare the features of the images. A brute force matcher (BF) will compare each feature of the first image to all features of the second image by using a specified norm. There are more advanced methods for comparing image feature like the Fast Library for Approximate Nearest Neighbors (FLANN) matcher, which will utilize different algorithms to find the best matching descriptors. We used the brute force matcher - as it fast on multi core CPU's - in combination with the hamming norm to find distances between the features. These distances can then be used to evaluate the similarity between two pictures by comparing how many features from the first image can be found in the second image and how far the distance between the features is. We only used the distance of the best matching feature. This means that from each picture, only the distance from the best matching feature is used to define the distance metric of two images. We know that this is problematic as if, for some reason, the feature exists in the two images but these are generally not similar, e.g. the same street sign is present in both images but the images show two completely different cars. Using more features for distance calculation would need more tuning as distances between the features vary to a high degree and as such, it is not easy to normalize them correctly between zero and one - which is needed for *AntClust*. As such, by using more features there was a high mismatch in the similarity function, which did not always return 1 if the images showing the same car. The score was far below 1 as scaling did not work properly. This is a major problem and for gaining better results, it should be addressed as discussed in the final section 5.

[5] An image cluster library using traditional clustering methods is *clustimage*, "https://github.com/erdogant/clustimage

[6] https://vehiclereid.github.io/VeRi/

[7] https://opencv.org/

4 Experiments and Results

We examine the clustering performance of *AntClust* on an image data set. We compare *AntClust* to other cluster algorithms such as DBSCAN, HDBSCAN and OPTICS[8] for reference. Additionally, we perform testing on a self-defined artificial float data set to show that the clustering performance of *AntClust* depends on the used ruleset. To evaluate the clustering performance, we used the Adjusted Rand Index score. Adjusted Rand Index is defined as a similarity measure that computes the differences in samples from a ground truth labeling to the provided labeling by the clustering algorithm. The highest score is 1.0, which indicates a perfect labeling of the data. A score of 0.0 indicates a random labeling. If the score is negative, the labeling is even worse than a random labeling.

We analyze the clustering performance for images in the following way. From the VeRi data set, we select 30 distinct cars, where for each car we take 18 images showing the car from the front. The images are used as a base to construct different clustering tasks. We start with only taking the images of two cars to have a very simple clustering task of clustering the images into two different clusters. Thereafter, images of three different cars, then four, until images of thirteen different cars are used to generate the task of finding three, four and finally thirteen clusters. The clustering task is harder if more clusters need to be found. To variate the tasks we change the amount of images in each cluster. For each of the above tasks, we change the amount of images in each cluster from 3 to 18. Results of these tasks show that the clustering performance worsens when more clusters are present - result can be seen in Figure 1 (B). For up to 10 clusters, the performance stays relatively good. Having more clusters in the clustering performance drops significantly. It has no influence on how many images are inside each cluster. The mean clustering performance only changes if the numbers of clusters increases, not the number of images inside each cluster.

Declining in clustering performance for many clusters is expected due to the fact that there is only one rule in the Labroche rule set, which is creating new clusters - and this rule is only applied in the beginning of the clustering phases of *AntClust*. To test the hypothesis, we generated a simplistic, one-dimensional data set which contains only float numbers that belong to a cluster. Each cluster has an integer as a pivot element. The pivot elements are defined as $P = \{1, 2, ..., n\}$. Each data point in a cluster differs from this pivot element by a range $R = [-0.1, 0.1]$. The data set is then defined as $D = \{d_i | d_i = x + r, x \in P, r \in R\}$. This creates a very clear and easy clustering task that can easily be extended to many clusters or data tuples in each cluster. We generate N clusters, each cluster having the exact same amount of data tuples d in it. We start with $N = 2$ clusters, each with $d = 3$ tuples. Then increase the number of tuples up until $d = 90$ in each cluster. After that, we incremented the number of clusters by one and started again with $d = 3$ tuples in each cluster, increasing them to $d = 90$. We proceeded until we reached $N = 30$ clusters. Meaning in the last test, we have 30 clusters, each having 90 data tuples in it. The clustering result gets worse if more than 5 clusters are present in the artificially generated data - as seen in Figure 1 (A). This supports the finding that *AntClust* does not handle many clusters in a data set. However, the tests revealed only the performance on this particular artificial data set. Since *AntClust* only sees the data in the form of similarity, this should however show the performance trend, the direction in which the performance for many clusters will evolve.

As reference, we test *AntClust* against DBSCAN, HDBSCAN and OPTICS. The task for all algorithms is to cluster data sets containing images of cars. Each cluster contains 18 images of cars. The Number of cars - and thus the number of clusters to be found - is varied from 2 to 30. For each algorithm, the found clustering partition is evaluated using the ARI score. We pre-compute a distance matrix with our similarity function and provide it to the algorithms. Therefore the clustering task is exactly the same for all algorithms. Results show that it depends on the number of clusters which algorithm performs best. In this task, *AntClust* delivers the best clustering partitioning. All ARI scores for the respective algorithms and their mean score are shown in Table 2.

[8] we use the scikit-learn implementation with the default parameters except for DBSCAN where we set eps=0.33 and min-samples=2

AntClust can be a clustering algorithm of choice if not many clusters are expected in the data. A good reason for using *AntClust* is that it does not need to know how many clusters reside in the given data set. This is a significant improvement over clustering algorithms such as K-means which require the number of clusters to be found as an input parameter.

Table 2. ARI clustering score for different amounts of clusters inside the car image data set. In the case of two clusters, images from two cars were used, and images of four cars were used to have four clusters. Mean represents the mean ARI score for all cluster numbers.

Clusters	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	Mean
AntClust	0.31	0.74	0.66	0.78	0.66	0.69	0.69	0.66	0.63	0.65	0.59	0.61	0.6	0.57	0.53	0.62
DBSCAN	0.75	0.73	0.74	0.73	0.72	0.34	0.23	0.2	0.17	0.21	0.15	0.11	0.083	0.059	0.062	0.35
HDBSCAN	0.86	0.76	0.55	0.71	0.65	0.49	0.45	0.48	0.41	0.4	0.36	0.41	0.38	0.35	0.39	0.50
OTICS	0.45	0.63	0.29	0.45	0.39	0.27	0.28	0.24	0.16	0.17	0.12	0.1	0.094	0.076	0.086	0.25

5 Conclusion and Future Work

We implemented *AntClust* and used our implementation to cluster images of cars taken from the VeRi data set. We used ORB features extracted from the images to define the similarity between images and matched the feature distances. The results show that *AntClust* performs well on our chosen data set, outperforming DBSCAN, HDBSCAN or OPTICS. We found that clustering performance gets significantly worse for the artificially generated float assembly data set if many clusters are present.

An improvement factor in future work is the similarity function. We extract image features using ORB. Currently, the similarity is calculated using only the distance between the nearest features in all obtained features, meaning only one feature is used to obtain the similarity. As a future work, more features should be used to have a more robust match. Additionally, the current approach is not color-aware. It would be a benefit to include color-aware similarity measures. Generally - as done in many classification tasks - using an ensemble of image similarity methods to improve the overall performance of the similarity function would be necessary to test. We plan to test the performance of these new similarity functions on more than one image data set. Having more than one image data set will give a better generalized picture of the performance of the *AntClust* algorithm used on images.

AntClust has the advantage that it does not need to know the numbers of clusters in the data beforehand - such as k-means. Finding the number of clusters that best separates a given data set is not a trivial task. Our tests show that despite not knowing how many clusters are existing in the data, still, a good clustering result can be obtained. However, this is only true if the data does not contain to many clusters. The reason for this is the following. The rule set used by *AntClust* should be able to generate and change clusters dynamically. For example, the default Labroche rule set will create new clusters with its rule R1 and alters clusters with rule R5. After the meeting phase, *AntClust* will take care that only clusters with a high fitness exists by deleting clusters with lower fitness in the nest shrink step (see Table 1). In theory, this should lead to the best partitioning, i.e. the "right" amount of clusters. From the results obtained by Labroche et al., it can be seen that this works pretty well for data sets that have only up to four clusters [1]. As we showed, if more clusters are inside the data, *AntClust* struggles with generating enough clusters. As suggested by Labroche et al., this might be due to the fact that the only rule that can create new clusters is rule R1. The problem here is that R1 is only applied if, during a meeting, both ants do not have any labels. As such, in the beginning, R1 is applied very often, whereas in the end, R1 will never be applied. If a particular cluster is not formed in the beginning, or if it is deleted during runtime, it can not be recreated again. Therefore it would be necessary to alter the rule set in a way that

it allows for creating new clusters even in the later phases of the algorithm. This is, however, a challenging task. Our implementation opens the way for experimenting with different rule sets, makeing it easy to change and replace the rule set via an informal interface in *Python*.

Future work will be on an automated rule set generation approach. Neuronal networks could be trained and used as a rule set - and might be trained by neural evolution mechanisms such as NeuroEvolution of Augmenting Topologies (NEAT) [8]. Rule sets or rule mechanisms could be evolved using genetic algorithms or a gene expression programming (GEP) approach. This will result in a bio-inspired algorithm based on the chemical recognition system in ants and the evolutionary algorithm for more complete and accurate results. It enables bio-inspired algorithms to enhance each other and become better together.

References

1. Labroche, Nicolas & E, Nicolas & Venturini, Gilles. (2003). A New Clustering Algorithm Based on the Chemical Recognition System of Ants.
2. Labroche, Nicolas & Monmarché, Nicolas & Venturini, Gilles. (2003). AntClust: Ant Clustering and Web Usage Mining. 25-36. 10.1007/3-540-45105-6_3.
3. B. Hölldobler and E.O. Wilson, The Ants, chapter Colony odor and kin recognition, 197–208, Springer Verlag, Berlin, Germany, 1990.
4. D. Fresneau and C. Errard, 'L'identité coloniale et sa reprsentation chez les fourmis', Intellectica, 2, 91–115, (1994).
5. Rublee, Ethan & Rabaud, Vincent & Konolige, Kurt & Bradski, Gary. (2011). ORB: an efficient alternative to SIFT or SURF. Proceedings of the IEEE International Conference on Computer Vision. 2564-2571. 10.1109/ICCV.2011.6126544.
6. Implementation of the Algorithm "https://gitlab.com/Winnus/antclust
7. Xinchen Liu, Wu Liu, Tao Mei, Huadong Ma: PROVID: Progressive and Multimodal Vehicle Reidentification for Large-Scale Urban Surveillance. IEEE Trans. Multimedia 20(3): 645-658 (2018)
8. Kenneth O. Stanley, Risto Miikkulainen; Evolving Neural Networks through Augmenting Topologies. Evol Comput 2002; 10 (2): 99–127. doi: "https://doi.org/10.1162/106365602320169811

Noise Impact On Convolutional Neural Networks Ability to Generalize

Abdeldjallil AIDI[1], Karim MEZZOUG[2] and Abou EL Hassane BENYAMINA[3]

1. Ahmed Ben Bella University of Oran, Algeria
aidi.abdeldjallil@edu.univ-oran1.dz
2. Ibn Khaldoun University of Tiaret, Algeria
karim.mezzoug@univ-tiaret.dz
3. Ahmed Ben Bella University of Oran, Algeria
benyamina.hassen@univ-oran1.dz

Abstract. In this paper, we try to clarify the effects of noise contained in the images within image classification tasks by analyzing two different types of noise (Salt and Pepper, Gaussian) with five different levels on three Convolutional Neural Network (CNN) models (XceptionNet, GoogleNet, ResNet) using the same parameters (Dataset, noise, and level of noise), and how denoising methods can help to alleviate this problem, for the last matter, we consider two algorithms the median filter for the salt and pepper noise, and the non-local means (NLM) for the gaussian noise.
We perform our experiments with the Cifar-10 dataset, our results show that noise in images can hinder classification tasks and cause it a problem (make it harder to separate classes). Although images were denoised, we were unable to reach the results obtained in the noise-free scenarios.

Keywords: Classification, Noise, Denoising, Convolution, Neural Network

1 Introduction

According to literature, nearly 90% of the information received by humans is visual. Hence the production of quality images, as well as their digital (and if possible) automatic processing, is therefore of considerable importance, while most of computer vision and classification systems neglect pre-processing [08] and assume that images are given with high quality [04].

In the internet, in our phones and laptops, Millions of pictures ranging from biomedical images to the images of natural surroundings, such pictures might contain a lot of important information in diverse domains of application, which represent a primary source of information and/or visualization. The quality of the output image may be inferior to that of the original input picture when converted from one form to another by processes like imaging, scanning, or transmitting. Hence, there is a need to improve the quality of such images, so that the output image is better for human perception or machine analysis.

Nowadays Image classification is used in various applications, such as agricultural [01], educational and medical [05]. Our work is a part of the classification of images using deep learning which is a learning technique that enables a program, for example, to understand spoken language or recognize the content of an image allowing machines to learn and recognize objects, now back to our work, it is about of a set of comparisons between three architectures of convolutional neural networks. We aim to create a certain number of classifiers to images that contains noise and their restored versions, the results will be compared with each other to see the resilience of Convolutional Neural Networks (CNN) over distinct type of noise with different levels.

2 Related works

This is not the first time such experimentation is done to understand the impact of noisy images (in any way possible) in the performance of CNNs within image classification tasks. There are papers studying the effects of noise in the capability of CNNs to learn [02], [07].

Noise in images can hinder classification tasks; this knowledge is already discovered with systems that employ convolutional Neural Networks [06], and in systems that use handcrafted features. In [04] They evaluated 4 deep neural network architectures, they show that the performance of these networks is affected when classifying images with lower quality compared with the image quality in the training set, their experiments do not cover the presence of noisy images in the training set.

While Paranhos da Costa et al in [03] takes in consideration that noisy images might appear in the training set, they created noisy versions and generated their restored versions, they used hand-crafted features (LBP and HOG) and SVM classifiers were trained with each version of the training set. Their results show that classifiers suffer to generalize to different noisy data and image classification becomes harder.

Our work is an extension of [03]. As all we believe that noise in any way possible, makes classification more difficult, our experiments exploited state of the art deeper artificial neural network architectures such as RESNET and XCEPTIONET, second, we train all the architectures using the same dataset version for better comparison.

3 Results and Discussion

To evidently visualize the impact of noise and filtering methods in the image quality, the structural similarity index measure (SSIM) [09] and the peak signal-to-noise ratio (PSNR) [10] values are shown in **TABLE I**.

Smaller values in both the PSNR and SSIM indicate less similarity between the images of the original dataset version and the version compared to. By comparing the results while training and testing our chosen models with the same dataset (same noise and level) we can see how much harder separating between classes gets for these models. It has long been clear that increasing noise levels have undesired consequences. To better understand this, **TABLE II** shows the accuracy of our selected models trained on noisy datasets and their restored versions.

			SSIM		PSNR	
			noisy	filtered	noisy	filtered
Cifar-10 Data set	Salt and Pepper	P = 0.1	0.76	0.97	15.26	25.39
		P = 0.2	0.60	0.95	12.46	23.47
		P = 0.3	0.48	0.93	10.90	21.19
		P = 0.4	0.41	0.88	9.85	18.92
		P = 0.5	0.34	0.82	9.08	16.87
	Gaussian	σ = 10	0.87	0.87	16.92	17.17
		σ = 20	0.73	0.77	14.22	15.50
		σ = 30	0.58	0.67	12.45	15.29
		σ = 40	0.44	0.54	11.20	14.62
		σ = 50	0.31	0.45	10.28	13.77

TABLE I SSIM AND PSNR FOR EACH NOISE LEVEL

			XceptionNet		GoogleNet		ResNet	
			noisy	filtered	noisy	filtered	noisy	filtered
Cifar-10 Data set	original		85.89		84.96		77.46	
	Salt and Pepper	P = 0.1	77.12	81.15	77.98	80.12	66.07	73.04
		P = 0.2	74.47	78.50	72.11	77.84	62.89	72.38
		P = 0.3	66.26	76.14	67.59	75.61	58.22	69.38
		P = 0.4	62.77	72.88	62.48	74.80	51.43	67.33
		P = 0.5	57.02	74.43	57.06	73.30	49.04	64.94
	Gaussian	σ = 10	76.76	69.92	73.70	68.02	68.11	63.29
		σ = 20	75.24	70.45	73.82	71.42	68.21	63.61
		σ = 30	60.37	62.54	66.58	58.97	62.31	56.98
		σ = 40	52.33	55.69	51.44	44.99	47.32	52.15
		σ = 50	27.97	51.54	25.60	51.28	23.19	46.83

TABLE II PERCENTAGE ACCURACY OF EACH MODEL WHEN TRAINING AND TESTING USING THE SAME DATASET VERSION

our scrutiny is divided into two steps (in both steps we chose the middle level of noise to work with throughout this scrutiny):

A. Inter-models: The goal here, is to visualize which architecture is better resilient when dealing with noise in datasets, and see how much harder classifying these image datasets gets. Even so it seems that GoogleNet did the great work but in general, and accordingly to **TABLE II** the XceptionNet model did better than the other models when dealing with noisy dataset.

B. Intra-models: Here we compare the obtained accuracies using the same model with three datasets (noise-free, noisy, denoised) the goal is to see how the image quality affect the models, and how much the denoising methods are helping in the cases that of noisy images. To display the importance/effects of de-

noising methods, and how they are helping override this hurdle the results shown in **TABLE II** are plotted in **Fig**. 1 and **Fig**. 2.

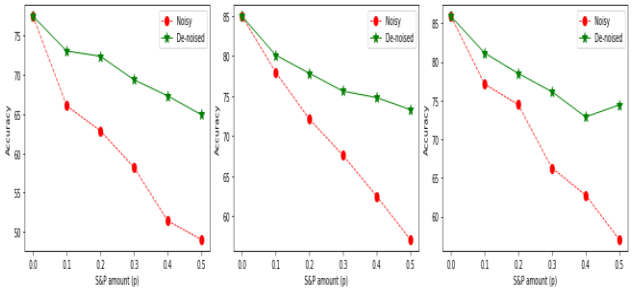


Fig. 1. Comparison of the accuracy of each network while training with Salt and Pepper noisy datasets and de-noised ones using median filter (a) results using ResNet architecture (b) results using GoogleNet architecture (c) results using XceptionNet architecture.

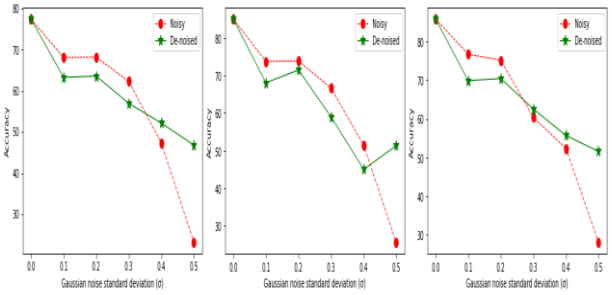


Fig. 2. Comparison of the accuracy of each network while training with Gaussian noisy datasets and de-noised ones using NLM filter (a) results using ResNet architecture (b) results using GoogleNet architecture (c) results using XceptionNet architecture.

4 Conclusion

Training deep convolutional neural networks with datasets after adding noise to their images at a known level has shown how hard classifying these images gets, our study covered two types of noise with five levels for each type, likewise you can test other types of noise with different models. Our results show that the CNN architectures are having a hard time classifying the images affected by noise in both training and test sets.

Regarding the noise reduction algorithms, the dataset restored by median filtering where able to improve accuracy and image quality compared to their Salt and Pepper noisy counterparts, but in the other hand, datasets versions that were hindered with Gaussian noise gave better results than their restored counterparts using NLM filter, that's because of the resulted blur in the image after using the NLM filter.

The final point of this paper is that the more noise you add to an image, the harder it is to classify.

References

[01] Gill, H. S., Khalaf, O. I., Alotaibi, Y. Alghamdi, S. & Alassery, F. (2022). Fruit Image Classification Using Deep Learning. Computers, Materials and Continua, 71(2), 5135–5150.

[02] AJ. Bekker and J.Goldberger(2016). Training deep neural networks based on unreliable labels.

[03] GB. P. da Costa, WA. Contato, TS. Nazare, JE. S. BNeto, and M. Ponti (2016). An empirical study on the effects of different types of noise in image classification tasks.

[04] S. Dodge and L. Karam (2016). Understanding how image quality affects deep neural networks.

[05] AH. Khan, S. Abbas,MA. Khan, U. Farooq, WA. Khan, SY. Siddiqui, and A. Ahmad (2022). Intelligent Model for Brain Tumor Identification Using Deep Learning. Applied Computational Intelligence and Soft Computing.

[06] M. Momeny, Ali M. Latif, M. Agha Sarram, R. Sheikhpour, YD. Zhang (2021), A noise robust convolutional neural network for image classification, Results in Engineering, Volume 10.

[07] DF. Nettleton, A. Orriols-Puig, and A. Fornells (2010). A study of the effect of different types of noise on the precision of supervised learning techniques. Artificial Intelligence Review, 33.

[08] M. Ponti, TS. Nazaré, and GS. Thumé (2016). Image quantization as a dimensionality reduction procedure in color and texture feature extraction. Neurocomputing, 173.

[09] De Rosal Igantius Moses Setiadi (2021). Psnr vs ssim: imperceptibility quality assessment for image steganography. Multimedia Tools and Applications, 80.

[10] Z. Wang, AC. Bovik, HR. Sheikh, and Eero P. Simoncelli (2004). Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing, 13.

Implementation of Metaheuristic Search for Finding Shortest Path in Tunnelled Network

Habiba Akter*[1]

[1]Queen Mary, University of London – United Kingdom

Abstract

This work investigates the implementation of an Evolutionary Algorithm (EA) as a metaheuristic search tool to design a path computation tool to help find the shortest path(s) from a source node to a destination node to send data over. The bespoke tool will also provide an additional benefit of loose source routing in a scenario where internet tunnels are present in a part of the network topology. However, the EA-generated optimal paths may or may not have tunnels present in them but the costs associated will be optimised using the evolutionary computation. The objective functions for optimisation will be designed based on the cost values associated with the end-to-end path.

*Speaker