Automatic Algorithm Multi-Configuration Applied to an Optimization Algorithm

Weerapan Sae-Dan, Marie-Eléonore Kessaci, Nadarajen Veerapen, Laetitia Jourdan

Automatic Algorithm Multi-Configuration Applied to an Optimization Algorithm

Weerapan Sae-Dan, Marie-Eléonore Kessaci[0000−0002−4372−5162], Nadarajen Veerapen[0000−0003−3699−1080], and Laetitia Jourdan[0000−0002−4170−6830]

Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL 59000, Lille, France
{weerapan.saedan.etu, mkessaci, nadarajen.veerapen, laetitia.jourdan}@univ-lille.fr

Abstract. Automatic algorithm configuration is concerned with finding the best hyper-parameter values for some specific algorithm. These values are then fixed throughout the execution of the algorithm. Another approach is parameter control where the values adaptively change during execution. In this work, we explore the hybrid concept of multi-configurations where values are still optimized before-hand, but as different sets of configurations that are then used one at a time during execution. In particular we explore a number of strategies based on fixed sequences of configurations and roulette wheel selection, and compare them to some baselines. We evaluate the strategies in the context of Iterated Local Search on the Permutation Flowshop Problem. Results show that both fixed and roulette strategies are better than the baselines, but also that roulette outperforms the fixed approach when hyper-parameters are optimized on more diverse sets of instances. We observe that the chosen values are not necessarily ones that would be considered the best in the literature because they are used as part of the multi-configurations.

Keywords: Local Search · Automatic Algorithm Configuration · Parameter Tuning

1 Introduction

Metaheuristics are used to solve difficult optimization problems because they usually provide a good compromise between solution quality and execution time. Like many AI approaches, they frequently have several hyper-parameters that impact their performance. Here onwards, we shorten hyper-parameter to parameter for convenience. For each instance of a problem tackled by some metaheuristic, there exists a best configuration, or set of parameter values, that achieves the best performance. Finding such a set of parameter values may be handled via parameter tuning and parameter control [2].

Tuning the parameters of an algorithm is an offline process where the parameter values are optimized before running the algorithm to obtain a final solution. In contrast, controlling the parameters is an online process where the parameter values are adjusted during the execution of the algorithm. In this

paper, we consider parameter tuning. This involves using a configurator to evaluate the performance of configurations for some algorithm on training instances and then choosing the best configuration. The literature includes a number of configurators, such as irace [5] and ParamILS [3].

Configurators usually provide a single configuration that is used during the entirety of the execution of an algorithm. Yet, it may be beneficial to alter the parameter values during execution. To bridge the gap between parameter tuning and parameter control, in this paper, we use the irace configurator, without loss of generality, to generate multiple configurations in an offline process. These are used at different points during execution according to predetermined strategies. We explore the benefits, or lack thereof, of using this hybrid approach compared to standard parameter tuning, building on our previous preliminary work [8].

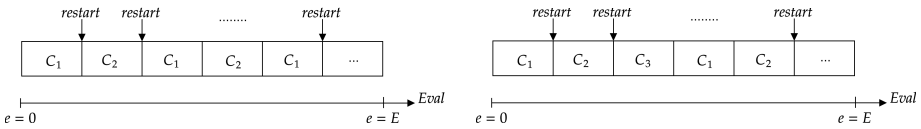
In order to test our idea, we consider an Iterated Local Search (ILS) [4] with restarts. ILS is simple yet is well-known for solving combinatorial optimization problems. It features a number of parameters that are not only numerical but also represent some very important algorithmic design choices. They are considered as categorical parameters. The restarts are parameters too. In particular, we compare three main scenarios with restarts: *fixed* multi-configuration, *roulette* multi-configuration and *random* multi-configuration.

Our results, on Permutation Flowshop Scheduling Problem instances, show that the fixed and roulette scenarios perform statistically better across various instance sizes. We observe that the roulette model is more robust than the fixed model when the configurator is asked to optimize across multiple instance sizes (i.e. provides a more general (multi-)configuration). We also note that some components that are generally considered less efficient are still used in the configurations returned by the configurator. This indicates that they can be useful in combination with other components. It is therefore not necessarily wise to only provide the "best" components to the configurator, because then it could not extract some less intuitive parameter combinations.

The rest of the paper is arranged as follows. In Section 2 we provide the details of our multi-configuration scenarios. In Section 3 we present the permutation flowshop problem. Section 4 describes the ILS used, including its algorithmic components. Section 5 details the experimental protocol, and Section 6 presents the experimental results. Section 7 contains our conclusions and perspectives.

2 A Multi-Configuration Model

In this work we consider scenarios where offline tuning not only determines which algorithm configuration to apply initially, but also after a restart of some algorithm. Here, we instantiate this multi-configuration model on an Iterated Local Search [4] whose components are detailed in Section 4. A restart provides a fairly intuitive point at which it makes sense to potentially switch configuration. The type of restart operator is a parameter in itself, as is the condition that triggers the restart. The deciding criterion for a restart is often a number of non-improving steps, which we use here. The other parameters to consider are for



(a) Two-configuration model (b) Three-configuration model

Fig. 1: Fixed Model

the ILS, its building blocks, and the initialization mechanism. These are changed according to three types of scenarios that will be detailed further: *fixed* multi-configuration, *roulette* multi-configuration and *random* multi-configuration.

While automatic algorithm configuration is very powerful, increasing the number of parameters requires additional tuning time. For this reason, as well as for the sake of simplicity, we restrict ourselves to a maximum of three different configurations that can be returned by the configurator.

Fixed Multi-Configuration. In fixed multi-configuration the configurations are applied in a predefined sequence that loops around until the total execution budget is used up as illustrated in Figure 1. There can be a sequence of two or three configurations.

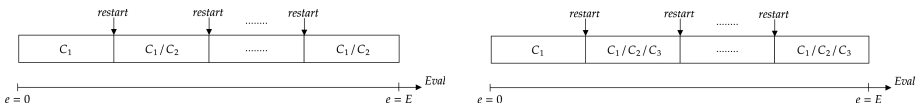
The trivial single-configuration variant is also used to serve as a baseline and the same configuration (c_1) is used after each restart until the stopping criterion is reached. Our experimental protocol (Section 5) includes a special version of this single-configuration model, where no configurator is used but exhaustive exploration of a restricted set of parameter values is carried out.

In the two- and three-configuration versions, we start with some initial configuration (c_1), then switch to c_2 after the restart. These two configurations are then used sequentially until the stopping criterion is met for the two-configuration version (Figure 1a). The three-configuration version naturally employs a third configuration (Figure 1b).

Roulette Multi-Configuration. For roulette multi-configuration, we consider two and three configurations with roulette wheel selection, and dispense from using a single trivial configuration as this is equivalent to the fixed single-configuration.

In this setting, the probability of applying each configuration is a parameter optimized by the configurator. The application of each configuration is illustrated in Figure 2. We always start with configuration (c_1). Then, after each restart, each configuration can be chosen with some fixed probability. Configuration c_1 is assigned some percentage value p_1 that will trivially correspond to probability P_1. In the two-configuration case, c_2 then automatically has probability $P_2 = 1 - P_1$. In the three-configuration case, c_2 is assigned some percentage value p_2 of the remaining $1 - P_1$, which gives a probability $P_2 = p_2 \times (1 - P_1)$ for c_2. Finally c_3 automatically has probability $P_3 = 1 - (P_1 + P_2)$.

Random Multi-Configuration. As a baseline and sanity check, the initial configuration, together with the new configuration after each restart are selected at



(a) Two-configuration model (b) Three-configuration model

Fig. 2: Roulette Model

random. The restart mechanism and stagnation period are also changed each time. No parameter tuning is used.

3 Permutation Flowshop Scheduling Problem

The flowshop scheduling problem (FSP) is a classical NP-hard combinatorial optimisation problem where a set of N jobs have to be processed on a set of M machines. Each machine can process only one job at a time. A job has to be fully processed at once on each machine and can be processed on only one machine at a time. The sequence of machines is fixed, as is each job. In the permutation variant (PFSP), each job is executed on the machine in the same order. The PFSP is widely used in the literature to evaluate the performance of metaheuristics. In this work, we consider the makespan minimization criterion where all jobs have to be processed on all machines with the minimum of time.

The Taillard instances [10] are widely used in the literature to assess the performance of algorithms. Different sizes are available depending on the number of jobs and machines. For each size, ten instances are provided. In this study, to assess the performance, we keep only larger instances with $\{50, 100, 200\}$ jobs and $\{10, 20\}$ machines as follows: 50×20, 100×10, 100×20, 200×10, and 200×20. In addition, we generate 100 training instances per instance size for the automatic configuration phase. This prevents overfitting parameter values and mimics real-life situations where one would want to apply the configuration step infrequently but use the configuration multiple times on new instances.

4 Multi-Configuration ILS

Hill-climbing is fast and easy to use, but it rarely yields the best results because it stops at the first local optimum found. There are strategies that allow the search to continue even after an optimal solution has been discovered. Iterated Local Search (ILS) is among those. From the local optimum found, the following steps are taken: (i) alter the current solution with a perturbation (ii) run this solution via a hill-climbing algorithm, or some other local search algorithm, (iii) select if the new optimum becomes the current solution if it is better than the previous one, then revert to (i) until some stopping criterion is reached.

The perturbation in ILS is meant to allow the algorithm to escape from the current basin of attraction, while still preserving most of the solution, in

the hope of ending up in a better basin. When this strategy fails, restarts can be used to start afresh from a new solution. Stagnation (stg) is the number of non-improving iterations that triggers a restart.

As with any local search, some components, such as the initialization and neighborhood are problem-specific, in our case PFSP-specific. The methods to perturb or diversify the search also have their own numerical parameters. We present the components, i.e. categorical parameters, and numerical parameters in the ILS with restarts for the PFSP.

Initialization heuristic. ILS requires a method to build an initial solution. While the PFSP is represented by a simple permutation, generating a random order of jobs is not an efficient way to obtain a good initial solution. We use the NEH heuristic [6], which is probably the best-known greedy heuristic for PFSP.

Neighborhood Operator. Hill Climbing and its variants are neighborhood-based algorithms and require an operator to generate neighbor solutions from the current solution. For permutation problems, such as the permutation flowshop problem, two operators are widely used: `shift` and `swap` [9]. The shift operator is known to be more adapted to the PFSP, but in a context of multi-configuration, it appears to be interesting to see if it can be useful at some stage of the search.

Neighborhood Order. A neighborhood may be explored in different orders. One option is to explore neighbors uniformly at random (`rnd`). Another is to always parse the neighborhood in some predefined order (`ord`).

Exploration Strategy. The hill-climber, or local search, chooses at each step a neighbor that improves the objective function. The two most commonly used acceptance strategies for candidate solutions are first (`IHCfirst`) and best (`IHCbest`) improvement. In the first case, exploration stops as soon as an improving neighbor is found, in the second the whole neighborhood is explored and the best improvement is selected. Worst improvement has also been shown to be beneficial in some cases [11] (`IHCworst`). Late Acceptance Hill-Climbing (LAHC) [1] (`IHClahc`) employs yet another acceptance strategy that considers fitness values encountered earlier in the run.

Perturbation. An ILS uses the perturbation to escape from a local optimum. The deconstruction-reconstruction perturbation, denoted IG-D/R, proposed by Ruiz and Stützle for the iterated greedy algorithm proved its efficacy [7], the most efficient sizes being $\{2, 3, 4\}$.

Diversification. When an ILS stagnates, it is necessary to perform a jump in the search space to find a better region to explore. This is achieved via a full or sometimes a partial restart. We retain two different strategies to make a large step possible: either the initialization heuristic NEH is performed (equivalent to a full restart) or a (fairly large) kick is applied on the current solution. This last strategy consists in applying the swap operator $k \in \{3, 4, 5, 10\}$ times.

5 Experimental Protocol

In this study, we propose three multi-configuration models using sequentially several values of the parameters/components. In order to assess the perfor-

mance of the multi-configuration models, we propose to compare them to a single-configuration model where only one fixed configuration is performed during the whole run. Actually, the single-configuration model is a degenerate variant of the fixed multi-configuration model where only one configuration is applied sequentially. An exhaustive exploration will be carried out for this single-configuration model as a baseline of our experiments to discuss the interest of a multi-configuration model.

Table 1 reports all the search components/parameters and their respective values presented in the Sections above. It leads to a configuration space of 1200 different configurations. This configuration space is used by the random multi-configuration model and the single-configuration model. With 1200 configurations, the size of the space is already too large to exhaustively test all of them within a reasonable time. Therefore, we propose to fix $d = 4$ and $k = 3$, being the best parameter values found in the literature respectively and to fix $stg = 100$, being the middle value in that parameter's domain. It drastically reduces the configuration space size to 32 and allows us to perform the exhaustive analysis.

The roulette multi-configuration model uses a probability P to select the next configuration to run. We set selection probability $P = \{25, 50, 75\}$ and allow only three values to avoid a combinatorial explosion of the configuration space. We use the irace package [5] to automatically configure both fixed and roulette multi-configuration models. We give a budget of 5000 runs and we bound the number of different configurations used by the multi-configuration models to three. This leads to more than 10^{+7} and 10^{+8} different configurations respectively.

The termination criterion of the algorithm is defined as a maximum number of evaluations. Instance sizes 50×20, 100×10, 100×20, 200×10 and 200×20 are allotted a maximum budget of 40×10^6, 60×10^6, 100×10^6, 200×10^6 and 400×10^6 evaluations respectively. These numbers has been experimentally determined in order to encounter several stagnation periods and therefore trigger several restarts. As metaheuristics are stochastic algorithms, we perform 30 runs for each instance for each algorithm model.

In order to compare the models on different instances, we propose to compute the relative percentage deviation (RPD) from the best-known value. Then, the Friedman test is used to test the statistical equality of each model considering all instances or each size separately.

Table 1: Configuration Space.

Component/Parameter	**Possible Values**	**Component/Parameter**	**Possible Values**
Neighborhood Operator	shift, swap	Diversification Algorithm	init, kick
Neighborhood Order	rnd, ord	Diversification Strength k	3*, 4, 5, 10
Exploration Strategy	IHCfirst, IHCbest, IHCworst, IHClahc	Stagnation stg	0, 50, 100*, 150, 200
Perturbation Algorithm	IG-D/R	Perturbation Strength d	2, 3, 4*

Numerical values with a star (*) have been selected for the exhaustive analysis.

Table 2: Best configurations of the single-configuration model.

Instance Size	ILS	Diversification	Stagnation
50×20	shift, rnd, IHCfirst, IG-D/R(4)	kick(3)	100
100×20	shift, rnd, IHClahc, IG-D/R(4)	kick(3)	100
200×20			
100×10	shift, rnd, IHCfirst, IG-D/R(4)	kick(3)	100
200×10	shift, rnd, IHClahc, IG-D/R(4)	kick(3)	100
	shift, rnd, IHClahc, IG-D/R(4)	init	100
All Instances	shift, rnd, IHCfirst, IG-D/R(4)	kick(3)	100
	shift, rnd, IHClahc, IG-D/R(4)	kick(3)	100

6 Experimental Results

In this section, we first present the results of the exhaustive analysis for the single-configuration model. Then, we present the multi-configuration ILS returned by irace for the fixed and the roulette models. We finish with a comparison between the proposed models and the baselines.

Exhaustive Analysis of the Single-Configuration Model. Table 2 reports, for each size of instance, the best configurations of the single-ILS where the strength of the perturbation and the strength of the diversification have been set to 4 and 3 respectively and the stagnation criterion to 100 in order to enable an exhaustive exploration. The best configurations shown are the ones that are statistically better among the 32 available ones.

Not surprisingly, the shift operator is preferred to the swap operator to generate the neighbors of the current solutions and the random exploration of the neighborhood is chosen in each single-ILS. In these experiments, the hill climbing algorithms based on the first-improvement strategy or the late acceptance strategy applied with a kick diversification lead to the best performance for all instances. For instances with 10 machines, the init diversification with the late acceptance strategy are also equivalent to the latter. These results are those expected based on the literature. This shows that our protocol is well adjusted to assess the performance of our models to solve the Taillard PFSP instances.

Automatic Multi-configuration Models. Let us now focus on the fixed and roulette multi-configuration models. First we present the best configurations returned by irace among configuration spaces larger than 10^{+7}. Then, the performance of the best configurations are compared on Taillard instances. We recall that up to three ILS can be configured to be used sequentially in the multi-configuration model. A configuration is then composed of a maximum of three differently tuned ILS. Table 3 first reports the three best configurations returned by irace for the fixed multi-configuration model, and then the four best configurations returned by irace for the roulette multi-configuration model.

Let us first consider the fixed scenario. The first multi-ILS (fix_1) is composed of two tuned ILS only, while the last two multi-ILS (fix_2, fix_3) are composed of

Table 3: Best configurations of the automatic multi-configuration found by irace.

Conf.	ILS_1	ILS_2	ILS_3	Divers.	stg	P_1	P_2
Fixed multi-configuration model							
fix_1	shift,rnd,IHCfirst,IG-D/R(4)	shift,rnd,IHClahc,IG-D/R(3)	–	kick(3)	200		
fix_2	shift,rnd,IHCfirst,IG-D/R(3)	shift,rnd,IHClahc,IG-D/R(4)	swap,ord,IHCbest,IG-D/R(3)	kick(3)	200		
fix_3	shift,rnd,IHClahc,IG-D/R(3)	shift,rnd,IHCfirst,IG-D/R(4)	swap,rnd,IHClahc,IG-D/R(4)	kick(5)	100		
Roulette multi-configuration model							
rlt_1	shift,rnd,IHClahc,IG-D/R(3)	shift,rnd,IHCbest,IG-D/R(4)	shift,rnd,IHCfirst,IG-D/R(4)	kick(5)	150	75	25
rlt_2	shift,rnd,IHClahc,IG-D/R(4)	shift,rnd,IHCfirst,IG-D/R(4)	swap,rnd,IHClahc,IG-D/R(3)	kick(3)	200	75	75
rlt_3	shift,rnd,IHClahc,IG-D/R(4)	shift,rnd,IHCfirst,IG-D/R(2)	shift,rnd,IHCworst,IG-D/R(2)	kick(3)	150	75	75
rlt_4	shift,rnd,IHCfirst,IG-D/R(3)	shift,rnd,IHCbest,IG-D/R(4)	shift,rnd,IHCworst,IG-D/R(3)	kick(5)	150	75	25

three tuned ILS. All three contain the best single-configuration presented in the previous section except for the strength of the perturbation algorithms and the stagnation criterion. We recall that all the numerical values were not available for the exhaustive exploration and maybe increasing the possible values would have produced some different results. However, the fixed model configuration with only one ILS was in the configuration space of irace and no such configuration has been returned. This shows that having at least two ILS seems to lead to better performance. Surprisingly, in both fix_2 and fix_3, the ILS_3 uses the swap operator. The neighbors generated with the swap operators are different from ones generated with the shift. This choice enables new connections between solutions as it changes the search landscape.

The best-improvement strategy is selected in fix_2 for ILS_3 with the ordered neighborhood exploration. This is logical: this strategy evaluates all the current solution's neighbors to select the best among them. Moreover, only the kick is used to diversify the search for each (partial) restart. It thus seems to be better to jump to closer regions of the search space rather than starting anew.

Let us now consider the roulette scenario. Contrary to the fixed model, the four multi-ILS (rlt_1, rlt_2, rlt_3 and rlt_4) are always composed of the maximum number of ILS allowed. Even if they are all composed in part of the best single-ILS seen in the previous section, the best-improvement and the worst-improvement strategies are more represented but at the same position. Indeed, the best-improvement strategy is set in ILS_2 while the worst-improvement is preferred for ILS_3. Moreover, the shift operator is always selected except in ILS_3 of rlt_2. For the roulette model, the stagnation criterion is at least set to 150. This implies that the configurations leave time to the perturbation IG-D/R to find a better adjacent region before restarting. In this multi-configuration model, the probability of choosing among the three ILS is also tuned automatically. The four best configurations give 75% of chance to select ILS_1. It seems that ILS_2 and ILS_3 aim at introducing diversity in the search as well in changing the exploration strategy as modifying the definition of the neighborhood.

In order to compare our automatic multi-configuration models, we run the seven configurations of multi-ILS presented above on the Taillard instances. We average the 30 RPDs per instance and compute the Friedman test using the 10 in-

Table 4: Statistical comparison of fixed and roulette multi-configuration models.

Instance Size	Fixed			Roulette			
	fix_1^*	fix_2	fix_3	rlt_1^*	rlt_2	rlt_3	rlt_4
All instances							
S_{all}	+	+	−	+	+	+	+
Each size of instance							
50×20	+	+	+	+	+	+	−
100×10	+	+	−	+	+	+	+
100×20	+	+	−	+	+	+	−
200×10	+	+	+	+	−	+	+
200×20	+	+	−	+	+	+	+

stances per size to rank the seven configurations. Table 4 presents the result of the statistical comparison between the seven configurations. Configurations marked with '+' statistically outperform the ones with '-' for the considered instances. In the first line, we give the comparison considering all the instances and below the comparison per size of problem. Clearly, the roulette multi-configuration model is more robust than the fixed model. Indeed, if we focus on all instances – recall that the tuning has been performed considering all sizes – the four best configurations known for the roulette model are best ranked while fix_3 is less efficient. If we look at the results for each size, fix_1, fix_2, rlt_1 and $rlt3$ are always best ranked. In theses multi-ILS configurations, not only the best single-ILS configurations are represented. This shows the importance of putting in the configuration space used by the configurator all the possible components known for a problem with no *a priori* knowledge.

Comparison between Single- and Multi-configuration Models. We compare our two automatic multi-configuration models, namely fixed and roulette, with the random multi-configuration model and the single-configuration model on the Taillard instances. In all the scenarios (i.e. across all instances or when considering instances of the same size), the fixed and the roulette multi-ILS always outperform the random multi-ILS and the best configurations previously obtained in the exhaustive single-ILS analysis. If we focus on the multi-configuration models, these results show that using at least two different configurations during a same run should be done carefully. Indeed, the bad performance of the random model is explained by the online modification of too many parameters at the same time while the configurations obtained with automatic multi-configuration models (fixed and roulette) are well adapted to the instances tackled. Finally, the results between the automatic configured multi-ILS and single-ILS make the case for the use of many well tuned configurations.

7 Conclusion and Perspectives

In this paper, we propose two automatic multi-configuration models applied to a restart Iterated Local Search. Multiple ILS are automatically tuned during a

training phase and then are performed in different way according to the multi-configuration model. The fixed model iterates the tuned ILS in a predefined order while the roulette model selects, after each restart, one of the tuned ILS following a predefined probability. These models were tested on the permutation flowshop scheduling problem. Our experiments on the Taillard instances showed that automatic multi-configuration models give better performance than the classical restart ILS or a random multi-configuration model selecting randomly the configuration after each restart. Moreover, the roulette models appears more robust than the fixed model and should be preferred. Finally, we noticed that configurations *a priori* known as not effective are used in the multi-ILS and help to lead to better performance. This shows that the automatic design knows how to take advantage of all available components.

In future works, a deeper analysis of the behavior of the automatic multi-configurations model will be conducted. Moreover, the higher the number of ILS in the multi-configuration model, the larger the configuration space. The training phase will have to be reconsidered to increase the size of configuration space while remaining manageable.

References

1. Bazargani, M., Lobo, F.G.: Parameter-less late acceptance hill-climbing. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 219–226. GECCO '17, Association for Computing Machinery, New York, NY, USA (2017)
2. Eiben, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation **3**(2), 124–141 (1999)
3. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stuetzle, T.: ParamILS: An Automatic Algorithm Configuration Framework. Journal of Artificial Intelligence Research **36**, 267–306 (2009)
4. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated Local Search, pp. 320–353. Springer US, Boston, MA (2003)
5. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
6. Nawaz, M., Enscore, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983)
7. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. EJOR **177**(3), 2033–2049 (2007)
8. Sae-Dan, W., Kessaci, M.E., Veerapen, N., Jourdan, L.: Time-dependent automatic parameter configuration of a local search algorithm. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. p. 1898–1905. GECCO '20, Association for Computing Machinery, New York, NY, USA (2020)
9. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. Computers Operations Research **34**(10), 3143–3153 (2007)
10. Taillard, E.: Benchmarks for basic scheduling problems. EJOR **64**(2), 278–285 (1993)
11. Tari, S., Basseur, M., Goëffon, A.: Worst Improvement Based Iterated Local Search. In: Liefooghe, A., López-Ibáñez, M. (eds.) Evolutionary Computation in Combinatorial Optimization. pp. 50–66. LNCS, Springer International Publishing (2018)