Time-Dependent Automatic Parameter Configuration of a Local Search Algorithm

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

Time-Dependent Automatic Parameter Configuration of a Local Search Algorithm

Weerapan Sae-Dan
Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL
F-59000 Lille, France
weerapan.saedan@univ-lille.fr

Marie-Eléonore Kessaci
Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL
F-59000 Lille, France
mkessaci@univ-lille.fr

Nadarajen Veerapen
Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL
F-59000 Lille, France
nadarajen.veerapen@univ-lille.fr

Laetitia Jourdan
Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL
F-59000 Lille, France
laetitia.jourdan@univ-lille.fr

ABSTRACT

In combinatorial optimization, where problems are often NP-hard, metaheuristics and other approximation algorithms frequently have many parameters in order to adapt to a wide range of scenarios. Very often, obtaining good values for these parameters is a long and tedious manual task but automatic algorithm configuration has been shown to overcome this issue. At the same time, it may also be useful for parameter values to change during the search in order to fine-tune the search process. These parameters include low-level heuristic components. In this article, we propose to use automatic parameter configuration coupled with a control mechanism that switches between parameter configurations at specific times during the search, as an in-between classic parameter tuning and selection hyperheuristics. We test this idea on a local search algorithm, whose parameters allow for selecting different design components, and three combinatorial problems: the Permutation Flowshop Problem, the Traveling Salesman Problem, and the Quadratic Assignment Problem. In comparisons with traditional static automatic parameter configuration, the proposed approach time-dependent is shown to perform better. Additionally, better-performing local search component configurations are identified and discussed.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; *Randomized local search*;

KEYWORDS

Combinatorial Optimization; Local Search; Automatic Algorithm Configuration

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07…$15.00
https://doi.org/10.1145/3377929.3398107

ACM Reference Format:
Weerapan Sae-Dan, Marie-Eléonore Kessaci, Nadarajen Veerapen, and Laetitia Jourdan. 2020. Time-Dependent Automatic Parameter Configuration of a Local Search Algorithm. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3377929.3398107

1 INTRODUCTION

Difficult optimisation problems are often solved by metaheuristics and hyperheuristics. These frequently have many parameters that alter their behavior. There exists a set of parameter values that corresponds to the best configuration for each instance of a problem. The problem of finding parameter values to achieve the best performance can be approached through parameter tuning and parameter control [5]. Parameter tuning of an algorithm involves optimizing the parameter values before running the algorithm, while parameter control, or dynamic configuration, is about adjusting the parameter values during execution. This paper focuses on parameter tuning, where we use a configurator that tests several instances and configurations, measures the performance of the algorithm across these instances and configurations and then chooses the best configuration. A number of configurators have been proposed, including irace [12] which uses statistical racing, ParamILS [9, 10] which is based on iterative search, and SPOT [1] which sequentially builds one or several meta-models and allows for interactive tuning. In this paper, we have chosen to use irace for automatic configuration.

Parameter control, or dynamic configuration, is aimed at adjusting running algorithm parameter values during execution. When design components, or low-level building blocks, of the algorithm are included as parameters, dynamic configuration is akin to a selection hyperheuristic. In time-dependent control, the parameter values can be changed using certain deterministic rules, such as using some rules when a set number of generations has elapsed. An alternative example is to set parameter values to use at evenly spaced intervals of evaluations [6]. Here, we explore the benefits of integrating some amount of time-dependent control within parameter tuning by proposing a tuning framework that switches between different configurations at deterministically chosen times during execution. In order to test this idea in a reasonably simple, yet challenging context, we consider an Iterated Local Search (ILS) with a basic hill-climber where the different algorithmic components (such as initial solution generation, local search, perturbation

neighborhoods and acceptance criterion) are considered and can be configured as required.

We evaluate the idea on three combinatorial problems: the Permutation Flowshop Scheduling Problem (PFSP), the Traveling Salesman Problem (TSP), and the Quadratic Assignment Problem (QAP).

The rest of the paper is structured as follows. First, in Section 2, we present parameter tuning and parameter control, and introduce our time-dependent automatic algorithm configuration framework. Then, in Section 3, we present the local search algorithm and the different design components. Sections 4, 5, and 6 present the results of our experiments on the PFSP, TSP and QAP, respectively. Finally, Section 8 presents our conclusions and aspects of forthcoming work.

2 AUTOMATIC DESIGN OF A TIME-DEPENDENT ALGORITHM

In the taxonomy for parameter setting proposed by Eiben et al. [5], there are two categories: parameter tuning and parameter control.

Parameter tuning can be considered as an optimisation problem. The objective is to identify the best configuration out of a set of possible configurations that are assessed on a set of training instances. Parameter tuning, as known as automatic algorithm configuration, is an offline process, with the best configuration then being used to run the algorithm on new instances of interest.

Parameter control is aimed at adjusting parameter values during execution instead of only using the initial values that will otherwise remain fixed throughout the algorithm's execution. It was observed that there is no reason why the entire behavior of an optimization method should be and remain ideal for a certain parameter configuration for the entire duration of a run. Eiben et al. [5] describe three categories of algorithms: (1) deterministic, where the parameter value changes using certain deterministic laws; (2) adaptive, where the parameter value is modified by some form of optimization feedback; and (3) self-adaptive, where the parameter value is encoded in a new genotype and evolves during the optimization process. Although, it was initially applied only to evolutionary algorithms, now it is used in a broader sense.

Over the 20 years following Eiben et al.'s classification in 1999 [5], new ideas and methodologies have been proposed across the three categories of parameter control that had been identified. Doerr and Doerr [2] have observed that adaptation strategies can be split into subcategories: (1) time-dependent, (2) fitness-dependent, (3) rank-dependent, and (4) diversity-dependent.

2.1 A Time-Dependent Algorithm Configuration Framework

Our contribution proposes an improved method to parameterize an algorithm with time-dependent parameter configurations. It runs some algorithm (A) with some initial configuration (θ), the latter being allotted a specific time budget. Following this, the algorithm is switched to a new configuration, carrying on its execution from its current state but without the algorithm being restarted whenever the configuration is changed. A number of further configuration changes may occur. This behavior was described as a dynamic framework by Pageau et al. [14] even though the configurations were precomputed. Here, we choose to simply define it as time-dependent – instead of dynamic – given that the configuration changes are determined offline via parameter tuning. Figure 1 illustrates two instantiations of the time-dependent framework F and F' where F uses three different configurations and F' uses four different configurations, and where both instantiations of the framework used a priori fixed time periods. One of the contributions of this paper is to allow for time periods of varying lengths.

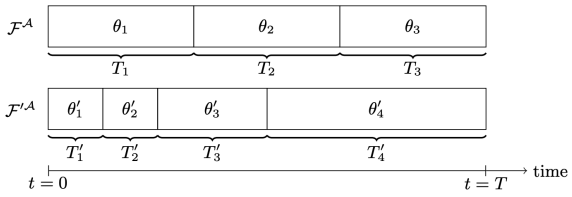

Figure 1: Examples of time-dependent parameter configuration [14].

2.2 Automatic Configuration with our Framework

To date, the majority of the automatic configurators in the literature (irace [12], SMAC [8], ParamILS [9]) focuses on optimizing the performance of an algorithm against a single criterion. We use irace, an iterated racing method. irace is an automatic configurator based on racing approaches and statistical tests. In practice, irace starts by evaluating a certain number of possible configurations of the algorithm on some training instances. Then, a statistical test is performed to discard the worst configurations and additional configurations are evaluated. These last configurations are chosen to be close to the best configurations already found. This process is iterated until a time budget is reached where the best performing configurations (for the training set of instances) are returned.

In this paper, the framework we propose uses automatic algorithm configuration instead of the parameter control mechanisms to switch between different parameter settings across different time periods. We denote the total time budget by T and the number of time splits by K. The duration of each time split (T_k) is chosen as a percentage (t_k) of the remaining time budget, in the list $\{10, 25, 50, 75, 90\}$. Each time split is calculated as follows: $T_1 = (T \times t_1)/100$; $T_2 = ((T - T_1) \times t_2)/100$; more generally, for $k < K$, $T_k = ((T - \sum_{i=1}^{k-1} T_i) \times t_k)/100$. The final time split, $T_K = T - \sum_{i=1}^{K-1} T_i$, uses up the remaining time budget. For K time splits, we therefore have $\{T_1, T_2, \ldots, T_K\}$ associated time budgets. We experiment with three scenarios where K equals 1, 2, or 3, corresponding to the different time splits $\{T\}$, $\{T_1, T_2\}$ and $\{T_1, T_2, T_3\}$ respectively.

3 LOCAL SEARCH

3.1 Local Search (LS)

Hill-climbing methods are well-known and fast. Given a solution space Ω and considering a minimization problem, without loss of generality, hill-climbing consists in (i) starting with an initial solution s from Ω and (ii) choosing a neighbor solution s' of s such that $f(s') < f(s)$, then replacing s by s' and repeating (ii) until

there are no improving neighbors. s is then a local optimum. There are several procedures for choosing a better neighbor, two of which are widely used. First improvement involves choosing the first neighbor encountered that has better quality (partial exploration of the neighborhood). Best improvement chooses the neighbor that most improves the objective function [13]. Tari et al. [16] studied a different strategy, the worst improving neighbor, which selects from all improving neighbors the one that improves the least the objective function. For each of the strategies, the stopping criterion is reached when a local optimum is met.

The major disadvantage of hill-climbing is that the search stops as soon as a local optimum is reached. To avoid this, there are different strategies to continue exploring the search space, for instance iterated local research.

3.2 Iterated Local Search (ILS)

Hill-climbing algorithms are quick and easy to implement but generally do not lead to the best optima because they stop as soon as a local optimum is found. To avoid getting stuck on this local optimum, the ILS framework has been proposed [11] where different perturbation strategies allow for further exploring the search space. As these strategies allow the search to continue after having found an optimum, it is then necessary to define a stopping criterion. The common stopping criteria include the execution time, the number of iterations, the total number of evaluations.

In an Iterated Local Search, the following steps are performed from the optimum found: (i) apply a perturbation to the current solution, (ii) apply a local search such as a hill-climbing algorithm to this solution, (iii) choose via an acceptance criterion if the new optimum becomes the current solution and loop back to (i) until the stopping criterion is reached. The perturbation may consist in restarting from a solution randomly taken from the research space, or choosing a solution in a region of the search space far from the optimum, or still choosing a neighbor of equivalent fitness to the optimum.

3.3 The Iterated Local Search Components

In this paper, four of the components of our ILS algorithm can be parameterized, and are highlighted in bold font:

(1) Initialization: to generate the starting solution.
(2) Local Search: we apply a hill-climbing algorithm as Local Search which uses the three following components.
   (a) **Neighborhood Order**: the neighborhood is either explored randomly or in order.
   (b) **Neighborhood Operator**: we are considering four neighborhood operators. The *shift* operator consists in changing the position of an element by shifting it into another position in the solution. The *swap* operator consists in exchanging the positions of two elements of the solution. The *k-opt* operator (*2-opt* and *3-opt*) consists in breaking and reconnecting k edges in a Hamiltonian cycle.
   (c) **Exploration Strategy**: the neighborhood of each chosen solution is searched and an archive of nominee solutions is generated with some of the neighbors visited. The procedure for choosing an improving neighbor can be either *first improvement*, *best improvement* or *worst improvement*.
(3) **Perturbation**: a perturbation can consist in restarting a solution randomly in the search space or applying a less drastic perturbation to the optimum. The perturbation strategies used here are a complete *restart* or partially modifying the solution, i.e., a *kick*.
(4) Acceptance Criterion: only accepts strictly improving solutions.

Some of the component choices are problem specific. In this paper we consider the Permutation Flowshop Problem Problem (PFSP, Section 4), the Traveling Salesman Problem (TSP, Section 5) and the Quadratic Assignment Problem (QAP, Section 6). The respective component choices are summarized in Table 1. They represent 24 (2x2x3x2), 48 (4x2x3x2) and 24 potential different configurations of the ILS, respectively. These numbers being fairly low, this allows us to exhaustively test all configurations and identify the best, thus providing a baseline against which to compare the proposed time-dependent approach.

Table 1: Search Components

Parameter	Value
Neighborhood for PFSP / QAP	{swap, shift}
Neighborhood for TSP	{swap, shift, 2-opt, 3-opt}
Neighborhood Order	{order, random}
Exploration Strategy	{firstimp, bestimp, worstimp}
Perturbation	{restart, kick}

4 PERMUTATION FLOWSHOP PROBLEM

The Permutation Flowshop Problem (PFSP) consists in scheduling a set of N jobs on a set of M machines. Machines are so-called critical resources because at most one task can be executed at the same time on a machine. Job J_i is composed of M consecutive tasks to be performed in order on the M machines. Each task has a specific execution time on each machine. We are trying to minimize the makespan of the latest scheduled task.

We use the `Taillard` instances [15]. We use different problem sizes of N jobs, M machines: 50×20, 100×10, 100×20, 200×10, and 200×20. For each size, 10 instances are used.

For the PFSP, we consider four experimental conditions that correspond to apply automatic algorithm configuration to different combinations of instance sizes in order to observe whether and/or how the set of training instances has an impact on the chosen parameter settings. The experimental conditions are summarized in Table 2.

Table 2: Design of experiments for PFSP instances

Scenario	Instances ($N \times M$)
S_{all}	all instances
$S_{N=100}$	$\{100 \times 10, 100 \times 20\}$
$S_{N=200}$	$\{200 \times 10, 200 \times 20\}$
$S_{M=20}$	$\{50 \times 20, 100 \times 20, 200 \times 20\}$

For the PFSP, we have 24 potential configurations (Table 1) of our algorithm per time split. For a single time split ($K = 1$), we therefore have the same 24 configurations and are not using the time-dependent nature of the framework. This is our baseline, which we call the *exhaustive* no control environment (or *ex*). For $K = 2$, or 2 time splits, then the fraction of the first time split will be chosen among the 5 different possibilities already presented ($|s| = |\{10,25,50,75,90\}| = 5$) and the second and final time split will take the remaining time budget. This amounts to approximately 2.9×10^3 different configurations ($24 + 5 \times 24^2$). When $K = 3$, or 3 time splits, then we have a total of approximately 3.5×10^5 different configurations ($24 + 5 \times 24^2 + 5^2 \times 24^3$). For $K = 2$ and $K = 3$, the values of the parameters are modified during execution, and we call this the *with control* (or *c*) environment.

We set the stopping criterion of the algorithm to $0.1 \times N^2 \times M$ milliseconds.

5 TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) involves finding the shortest Hamiltonian path between n cities. We use the `portgen` and `portcgen` generators of the 8th DIMACS Implementation Challenge to create two types of random instances respectively: random uniform Euclidean, S_u, and random 10-cluster Euclidean instances, S_c.

We use different values of n: 100, 200, and 400 cities. For each size, 10 instances are generated. They are solved to optimality with the Concorde TSP solver[1] in order to obtain the objective function value of the global optimum.

Since we consider the k-opt neighborhood operator, which is specific to the TSP, we have a different number of possible configurations for our ILS than for the PFSP or the QAP. In fact, we have 48 configurations, per time split, that can be generated from the combination of components given in Table 1.

For a single time split ($K = 1$), we have 48 configurations exhaustively run and this will be our baseline (or *ex*). For $K = 2$ and $K = 3$, our *with control* (or *c*) environments, we have approximately 11.6×10^3 and 27.8×10^5 different configurations, respectively.

We set the stopping criterion of the algorithm to n^2 milliseconds.

6 QUADRATIC ASSIGNMENT PROBLEM

The Quadratic Assignment Problem (QAP) involves assigning n facilities to n locations according to both the distance between locations and the flow between facilities.

We use instances constructed using the generator proposed by Drugan [4]. The instances are constructed in such a way that the value of the objective function for the global optimum is known. We use different instance sizes n: 20, 25 and 30. For each size, 10 instances are generated. All instances are used in the experiments S.

The operators for the QAP being the same as those for the PFSP, we also have 24 configurations per time split and the calculations when different number of time splits considered are identical.

We set the stopping criterion of the algorithm to $10 \times n^2$ milliseconds.

[1] http://www.math.uwaterloo.ca/tsp/concorde.html

7 EXPERIMENTS

In this paper, we use irace (Section 2.2) to implement the automatic algorithm configuration and to find the configuration of our ILS (Section 3.3) and its components (*neighborhood operator*, *neighborhood order*, *exploration strategy*, and *perturbation*) best adapted to the instances of the problem at hand. The training instances and the validation instances are independent. Only the results on the validation instances are shown in the experimental results. Table 3 presents, for each problem or scenario, the total number of configurations tested and the distribution following the number of time splits. This table will help with the analysis of the results for each problem.

Table 3: Data provided by irace runs

Problem	Scenario	Number of Configurations Tested			
		Total	Number of Time Splits		
			$K = 1$	$K = 2$	$K = 3$
PFSP	S_{all}	646	24	139	483
	$S_{N=100}$	676	12	159	505
	$S_{N=200}$	531	12	167	352
	$S_{M=20}$	654	12	151	491
TSP	S_u	549	24	247	278
	S_c	717	24	191	502
QAP	S	528	12	146	370

Naturally, the simplicity of the ILS and its design components studied in this paper cannot match the performance of the state-of-the-art approaches for each of the three problems considered, and a comparison is beyond the scope of this paper. Nevertheless, the same simplicity allows us to observe which components have some effect on the algorithm and to assess the merits of our proposed time-dependent approach.

ILS is a stochastic method. Therefore, all parameter configurations of the algorithm are executed 30 times per instance, and Friedman and Wilcoxon statistical tests are performed to compare configurations.

For the three problems, we will first analyze the results of the exhaustive runs and then the ones obtained by irace according to the different scenarios. In each case, irace returns up to three best statistically significant configurations it found within an allotted budget of 5000 algorithm runs.

7.1 Experimental Results for the PFSP

Let us first consider the PFSP. Table 5 gives the description of the best *static* ILS configurations for each problem size tackled. Only two configurations are statistically better than the other ones and they share three parameters out of four, namely the neighborhood operator (`shift`), the neighborhood order (`random`) and the exploration strategy (`firstimp`). For 50×20, 100×10 and 200×10 instances, the two possible perturbation parameters give equivalent performance, whereas for the remaining two other types of instances, the `kick` perturbation gives better results. Obviously,

Table 4: Best time-dependent configurations returned by irace for the PFSP instances

Inst.	Conf.	Parameter Values			Time Split	
		T_1	T_2	T_3	t_1	t_2
S_{all}	${}_c\theta_1$	{shift, random, firstimp, restart}	{shift, order, firstimp, restart}	{shift, random, firstimp, kick}	50%	10%
	${}_c\theta_2$	{shift, random, firstimp, restart}	{shift, order, firstimp, kick}	{shift, random, firstimp, kick}	50%	10%
	${}_c\theta_3$	{swap, random, firstimp, restart}	{swap, random, worstimp, kick}	{shift, random, firstimp, kick}	50%	90%
$S_{N=100}$	${}_c\theta_1$	{swap, random, firstimp, kick}	{swap, random, firstimp, kick}	{shift, random, firstimp, kick}	10%	75%
	${}_c\theta_2$	{swap, random, firstimp, kick}	{shift, random, firstimp, kick}	{shift, random, firstimp, kick}	10%	75%
	${}_c\theta_3$	{swap, random, firstimp, restart}	{swap, random, firstimp, kick}	{shift, random, firstimp, kick}	10%	50%
$S_{N=200}$	${}_c\theta_1$	{swap, random, firstimp, kick}	{shift, order, firstimp, kick}	{shift, random, firstimp, kick}	25%	10%
	${}_c\theta_2$	{shift, random, firstimp, kick}	{swap, order, firstimp, kick}	{shift, random, firstimp, kick}	90%	75%
	${}_c\theta_3$	{swap, random, firstimp, restart}	{swap, order, firstimp, kick}	{shift, random, firstimp, kick}	25%	25%
$S_{M=20}$	${}_c\theta_1$	{swap, random, firstimp, restart}	{shift, random, bestimp, restart}	{shift, random, firstimp, kick}	10%	25%
	${}_c\theta_2$	{swap, random, firstimp, kick}	{swap, random, firstimp, kick}	{shift, random, firstimp, kick}	25%	10%
	${}_c\theta_3$	{swap, random, firstimp, restart}	{shift, random, bestimp, restart}	{shift, random, firstimp, kick}	25%	50%

Occurrences of the best static configuration ${}_{ex}\theta$ are highlighted in blue.

Table 5: Best configurations for PFSP for the static algorithm

Instances	Best Configurations
50×20	{shift, random, firstimp, kick} {shift, random, firstimp, restart}
100×10	{shift, random, firstimp, kick} {shift, random, firstimp, restart}
100×20	{shift, random, firstimp, kick}
200×10	{shift, random, firstimp, kick} {shift, random, firstimp, restart}
200×20	{shift, random, firstimp, kick}
All scenarios	{shift, random, firstimp, kick}

when we consider the best configuration across all 4 scenarios (defined in Table 2), only the configuration with the `kick` perturbation remains. Therefore, in the following, this static configuration is called ${}_{ex}\theta$.

Table 4 gives, for each scenario, the three configurations returned by irace (${}_c\theta_1$, ${}_c\theta_2$ and ${}_c\theta_3$), including the actual parameters chosen for each time split and the percentage length (with respect to the remaining time budget) of the first 2 time splits (the last one using up the remaining time). These *time-dependent* configurations all have three splits, meaning that three different ILS should be performed successively during one run. The ${}_{ex}\theta$ configuration is used by all the time-dependent configurations in the third phase. Surprisingly, the parameters of the first and the second phases are more varied, with the `swap` neighborhood operator, considered quite inefficient by PFSP specialists, being used here. Interestingly, the `worstimp` selection operator was found to be useful in one of the time periods (for S_{all}, in T_2 of ${}_c\theta_3$). This is in keeping with the literature [16] that shows that this unlikely operator can be useful in specific situations. Moreover, the values reported in Table 3 show that irace tested all (24) or half (12) of static configurations (column $K = 1$) and eliminated them. In addition, it seems that 3 time splits gives better results since more configurations have been tested.

Table 6 gives, for each scenario, the result of the statistical test between the static configuration (${}_{ex}\theta$) and the three time-dependent configurations returned by irace (${}_c\theta_1$, ${}_c\theta_2$ and ${}_c\theta_3$) on the validation instances. A plus (+) sign indicates a configuration significantly better than, or equivalent to, the other configurations under the considered scenario, while a minus sign (-) indicates a significantly worse configuration. For all scenarios, the three time-dependent configurations outperform the static configuration. These experiments show that it is important to consider different sets of parameters during a single run. Moreover, no parameters should be discarded: parameters that are considered inefficient when they are used for the whole run could be efficient during a phase of a time dependent ILS. This is indeed one of the key insights also observed with selection hyperheuristics [3].

Table 6: Statistical comparison of configurations for each scenario of PFSP

Scenario	${}_{ex}\theta$	${}_c\theta_1$	${}_c\theta_2$	${}_c\theta_3$
S_{all}	−	+	+	+
$S_{N=100}$	−	+	+	+
$S_{N=200}$	−	+	+	+
$S_{M=20}$	−	+	+	+

For this problem, we designed different experiment scenarios in order to see whether it is better to train irace on either all instances or instances sharing the same number of jobs or machines. Table 7 summarizes, for each size of PFSP, the results of the statistical tests

between the static configuration and the time-dependent configurations for the different scenario tested. Here, the equal sign (=) denotes the statistical tests that cannot discriminate between the configurations. It seems, based on these experiments, that the training instances can share either the same number of jobs or the same number number of machines (see results for 100×20 and 200×20). Regarding the results for 100×10 instances, irace gives better configurations when the training used *similar* instances (sharing the same number of jobs) than all the available instances. Indeed, based on these experiments, we would suggest to use instances with same number of jobs or machines to train irace rather than using all available instances.

Table 7: Statistical comparison of configuration performance for each size of PFSP

Instances	${}_{ex}\theta$	S_{all}			$S_{N=100}$			$S_{N=200}$			$S_{M=20}$		
		${}_{c}\theta_{1,2,3}$			${}_{c}\theta_{1,2,3}$			${}_{c}\theta_{1,2,3}$			${}_{c}\theta_{1,2,3}$		
50×20	−	+	+	−							+	+	+
100×10	−	−	+	−	+	+	+						
100×20	−	+	+	+	+	+	+				+	+	+
200×10	=	=	=	=				=	=	=			
200×20	−	−	+	+				+	+	+	+	+	+

7.2 Experimental Results for the TSP

For the TSP, two scenarios are considered: the first one with random uniform Euclidean instances (S_u) and the second one with random 10-cluster Euclidean instances (S_c) as detailed in Section 5.

Table 8 gives the description of the best *static* ILS configurations for the two scenarios. For the S_u scenario with random uniform Euclidean instances, three configurations are statistically better over the 48 tested. These three static configurations (${}_{ex}\theta_1$, ${}_{ex}\theta_2$ and ${}_{ex}\theta_3$) share two parameters namely the neighborhood operator (`2-opt`) and the exploration strategy (`bestimp`). Regarding the two other parameters, the neighborhood order (`random`) is associated with the two available perturbations (`restart` and `kick`) while the neighborhood order (`order`) is more efficient with the `kick` perturbation. For the S_c scenario with random 10-Cluster Euclidean instances, one single static configuration (${}_{ex}\theta$) is statistically better than the others. Contrary to S_u, the `3-opt` operator is preferred together with the `random` neighborhood order, the `firstimp` exploration strategy and the `restart` perturbation.

Table 9 gives, for each scenario (S_u and S_c), the time-dependent configurations returned by irace. For the S_u scenario, only two time-dependent configurations was returned and each one has only two time splits. Table 3 show that irace tested half (24) of static configurations (column $K = 1$) and eliminated them, and almost the same number of configurations with 2 or 3 splits. This is in contrast to the rest of the scenarios across the 3 problems, where the 3 splits configurations are explored to a greater extent. It seems that, here, only two splits are preferable for this scenario. The difference between these two configurations is the time allocated to t_1: 90% for ${}_{c}\theta_1$ and 50% for ${}_{c}\theta_2$. We also observe that, during the first phase T_1,

Table 8: Best configurations for TSP for the static algorithm

Instances	Best Configurations
Random Uniform Euclidean Instances	
S_u	{2-opt, order, bestimp, restart} {2-opt, random, bestimp, kick} {2-opt, random, bestimp, restart}
Random 10-Cluster Euclidean Instances	
S_c	{3-opt, random, firstimp, restart}

one of the best exhaustive configurations is used ({`2-opt`, `random`, `bestimp`, `restart`}, or ${}_{ex}\theta_3$) while, during the second phase T_2, the set of parameters {`shift`, `random`, `firstimp`, `kick`} is used. This set is quite surprising since the `shift` operator is not known to be an efficient neighborhood operator for TSP. Our hypothesis is that, given the fairly limited allocated time budget, less efficient but faster operators are selected by the configurator. This would also explain why the `3-opt` operator is not used more frequently.

For the S_c scenario, three time-dependent configurations have been returned by irace. All have three splits. Table 3 show that irace also tested half (24) of static configurations and configurations with 3 splits were performing better on training instances. Analyzing the time-dependent configurations, we notice that the only difference between ${}_{c}\theta_2$ and ${}_{c}\theta_3$ is the perturbation parameter (`restart` for ${}_{c}\theta_2$ and `kick` for ${}_{c}\theta_3$) of the first phase T_1. Here, none of the time-dependent configurations include the set of parameters corresponding to the best static configuration ${}_{ex}\theta$. We observe that the successive neighborhood operators look like the variable neighborhoods that might be used in Variable Neighborhod Search [7] where the neighborhood operators change in size and nature to allow for an escape from local optima.

Tables 10 and 11 give, for each scenario (S_u and S_c), the result of the statistical test between the static configuration(s) and the time-dependent configurations. For both scenarios S_u and S_c, the time-dependent configurations outperform the static ones. Interestingly for S_u, although the two time-dependent configurations start with the set of parameters of one of the static configurations (${}_{ex}\theta_3$), both of them manage to outperform the static configuration, meaning that the second set of configurations (during T_2, using the `shift` operator) enables the time-dependent ILS to reach better solutions.

7.3 Experiments Results of QAP

For the QAP, we consider only one scenario (S) where the instances are differentiated by their number of facilities as detailed in Section 6.

Table 12 gives the description of the best *static* ILS configuration (called ${}_{ex}\theta$) where the `swap` operator is preferred to the `shift` operator (a fact well-known by QAP specialists). As for the PFSP, the `firstimp` exploration strategy and the `random` neighborhood order give the best performance. However, here, the `restart` perturbation is selected instead of the `kick`.

Table 13 gives the three time-dependent configurations returned by irace (${}_{c}\theta_1$, ${}_{c}\theta_2$ and ${}_{c}\theta_3$). These configurations all have three splits and all of them start by the set of parameters used by ${}_{ex}\theta$. This set

Table 9: Best time-dependent configurations returned by irace for the TSP instances

Inst.	Conf.	Parameter Values			Time Split	
		T_1	T_2	T_3	t_1	t_2
Random Uniform Euclidean Instances						
S_u	${}_c\theta_1$	{2-opt, random, bestimp, restart}	{shift, random, firstimp, kick}	-	90%	-
	${}_c\theta_2$	{2-opt, random, bestimp, restart}	{shift, random, firstimp, kick}	-	50%	-
Random 10-cluster Euclidean Instances						
S_c	${}_c\theta_1$	{2-opt, random, bestimp, restart}	{shift, order, firstimp, restart}	{3-opt, order, firstimp, kick}	75%	25%
	${}_c\theta_2$	{2-opt, random, bestimp, restart}	{3-opt, random, firstimp, kick}	{shift, random, bestimp, kick}	25%	25%
	${}_c\theta_3$	{2-opt, random, bestimp, kick}	{3-opt, random, firstimp, kick}	{shift, random, bestimp, kick}	25%	25%

Occurrences of the best static configuration ${}_{ex}\theta_3$ are highlighted in blue. It is the only one that is reused within the time-dependent configurations.

Table 10: Statistical comparison of configuration performance for the random uniform Euclidean TSP scenario

Scenario	${}_{ex}\theta_1$	${}_{ex}\theta_2$	${}_{ex}\theta_3$	${}_c\theta_1$	${}_c\theta_2$
S_u	−	−	−	+	+

Table 11: Statistical comparison of configuration performance for the random 10-cluster TSP scenario

Scenario	${}_{ex}\theta$	${}_c\theta_1$	${}_c\theta_2$	${}_c\theta_3$
S_c	−	+	+	+

Table 12: Best configuration for QAP for the static algorithm

Instances	Best Configuration
S	{swap, random, firstimp, restart}

is run for either 75% or 95% (90% of the time budget + 50% of the remaining budget) of the whole runtime. Surprisingly, the `shift` neighborhood operator is considered by two out of three returned configurations. Moreover, the values reported in Table 3 show that irace tested half (12) of static configurations (column $K = 1$) and eliminated them in favor of an ILS with three time splits.

Table 14 reports the result of the statistical test between the static configuration (${}_{ex}\theta$) and the three configurations returned by irace (${}_c\theta_1$, ${}_c\theta_2$ and ${}_c\theta_3$) on the validation instances. As for FSP and TSP, the time-dependent configurations are statistically better than the static configuration. The results show also that even if the static configuration is used at the beginning of run, other sets of parameters have to be run in order to get better performance.

8 CONCLUSION AND FUTURE WORK

Parameter setting maybe be split into parameter tuning – offline – and parameter control – online. In traditional parameter tuning, the parameter configuration is set before the start of an algorithm's execution and does not change thereafter. We investigated the introduction of time-dependent parameter configuration changes during the execution, something that is usually done via online parameter control or hyperheuristics, using pretuned time splits. In particular, our contribution consisted in considering various possible time period lengths, which are themselves tunable parameters, where previous work only considered a fixed number of time periods with fixed lengths. Our results, across the Permutation Flowshop Scheduling Problem, Traveling Salesman Problem and Quadratic Assignment Problem, show that it is generally preferable to use time-dependent control over a single configuration for the whole execution.

In the experiments on the PFSP, we also investigated the impact of using configurations obtained via tuning on different training instances. We observed that, as expected, training on instances with specific characteristics and then using the configurations on unseen instances with similar characteristics worked well. In addition, training across all types of instances also worked reasonably well.

Another interesting observation, across all three problems, was that different configurations worked better across the different time splits. We also observed that the single configuration for the entire run found to be the best via exhaustive exploration usually, but not always, appeared during one of the time splits, but never throughout all of them. Some less common operators, such as the worst-improvement neighbor selection, also performed well in very specific phases. All of this highlights that, as always, we are faced with the No-Free-Lunch theorem, and that parameter tuning and control can be one of the tools that help us in trying to overcome it.

This work is an initial step. Future work will delve into combining additional parameter control features, such as the detection of stationary or non-improving phases of the search process, into our automatic algorithm configuration framework. We are also interested in how we can transpose the ideas expressed here into a multi-objective context and how meta-learning approaches could help to overcome some of the hurdles currently faced by both parameter tuning and parameter control.

ACKNOWLEDGMENTS

The first author is supported by a PhD scholarship from the Royal Thai Government.

Table 13: Best time-dependent configurations returned by irace for the QAP instances

Inst.	Conf.	Parameter Values			Time Split	
		T_1	T_2	T_3	t_1	t_2
S	${}_c\theta_1$	{swap, random, firstimp, restart}	{swap, random, firstimp, kick}	{shift, order, bestimp, kick}	75%	90%
	${}_c\theta_2$	{swap, random, firstimp, restart}	{swap, order, firstimp, kick}	{swap, random, bestimp, kick}	75%	50%
	${}_c\theta_3$	{swap, random, firstimp, restart}	{swap, random, firstimp, restart}	{shift, order, bestimp, kick}	90%	50%

Occurrences of the best static configuration ${}_{ex}\theta$ are highlighted in blue.

Table 14: Statistical comparison of configuration performance for the QAP

Scenario	${}_{ex}\theta$	${}_c\theta_1$	${}_c\theta_2$	${}_c\theta_3$
S	−	+	+	+

REFERENCES

[1] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss. 2010. *The Sequential Parameter Optimization Toolbox.* Springer Berlin Heidelberg, Berlin, Heidelberg, 337–362. https://doi.org/10.1007/978-3-642-02538-9_14

[2] Benjamin Doerr and Carola Doerr. 2020. *Theory of Parameter Control for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices.* Springer International Publishing, Cham, 271–321. https://doi.org/10.1007/978-3-030-29414-4_6

[3] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. 2020. Recent advances in selection hyper-heuristics. *European Journal of Operational Research* 285, 2 (2020), 405 – 428. https://doi.org/10.1016/j.ejor.2019.07.073

[4] Mădălina M. Drugan. 2013. Instance generator for the quadratic assignment problem with additively decomposable cost function. In *2013 IEEE Congress on Evolutionary Computation.* 2086–2093. https://doi.org/10.1109/CEC.2013.6557815 ISSN: 1941-0026.

[5] A.E. Eiben, R. Hinterding, and Z. Michalewicz. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3, 2 (July 1999), 124–141. https://doi.org/10.1109/4235.771166

[6] Brian W. Goldman and Daniel R. Tauritz. 2011. Meta-Evolved Empirical Evidence of the Effectiveness of Dynamic Parameters. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '11).* Association for Computing Machinery, New York, NY, USA, 155–156. https://doi.org/10.1145/2001858.2001945

[7] Pierre Hansen and Nenad Mladenović. 2018. *Variable Neighborhood Search.* Springer International Publishing, Cham, 759–787. https://doi.org/10.1007/978-3-319-07124-4_19

[8] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2012. Parallel Algorithm Configuration. In *Learning and Intelligent Optimization*, Youssef Hamadi and Marc Schoenauer (Eds.). Vol. 7219. Springer Berlin Heidelberg, Berlin, Heidelberg, 55–70. https://doi.org/10.1007/978-3-642-34413-8_5

[9] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stuetzle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36 (Oct. 2009), 267–306. https://doi.org/10.1613/jair.2861

[10] Frank Hutter, Holger H. Hoos, and Thomas Stützle. 2007. Automatic Algorithm Configuration Based on Local Search. (2007). https://www.aaai.org/Library/AAAI/2007/aaai07-183.php

[11] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. 2010. Iterated Local Search: Framework and Applications. In *Handbook of Metaheuristics*, Michel Gendreau and Jean-Yves Potvin (Eds.). Springer US, Boston, MA, 363–397. https://doi.org/10.1007/978-1-4419-1665-5_12

[12] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (Jan. 2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[13] Gabriela Ochoa, Sébastien Verel, and Marco Tomassini. 2010. First-Improvement vs. Best-Improvement Local Optima Networks of NK Landscapes. In *Parallel Problem Solving from Nature, PPSN XI*, Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 104–113. https://doi.org/10.1007/978-3-642-15844-5_11

[14] Camille Pageau, Aymeric Blot, Holger H. Hoos, Marie-Eléonore Kessaci, and Laetitia Jourdan. 2019. Configuration of a Dynamic MOLS Algorithm for Bi-objective Flowshop Scheduling. In *Evolutionary Multi-Criterion Optimization (Lecture Notes in Computer Science)*, Kalyanmoy Deb, Erik Goodman, Carlos A. Coello Coello, Kathrin Klamroth, Kaisa Miettinen, Sanaz Mostaghim, and Patrick Reed (Eds.). Springer International Publishing, Cham, 565–577. https://doi.org/10.1007/978-3-030-12598-1_45

[15] E. Taillard. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64, 2 (Jan. 1993), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M

[16] Sara Tari, Matthieu Basseur, and Adrien Goëffon. 2018. Worst Improvement Based Iterated Local Search. In *Evolutionary Computation in Combinatorial Optimization (Lecture Notes in Computer Science)*, Arnaud Liefooghe and Manuel López-Ibáñez (Eds.). Springer International Publishing, Cham, 50–66. https://doi.org/10.1007/978-3-319-77449-7_4