# Automatic Configuration of Multi-Thread Local Search: Preliminary Results on Bi-objective TSP

Nicolas Szczepanski, Lucien Mousin, Nadarajen Veerapen, Laetitia Jourdan

# Automatic Configuration of Multi-Thread Local Search: Preliminary Results on Bi-objective TSP

Nicolas Szczepanski*, Lucien Mousin†*, Nadarajen Veerapen* and Laetitia Jourdan*
*Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, F-59000 Lille, France
{nicolas.szczepanski,nadarajen.veerapen,laetitia.jourdan}@univ-lille.fr
†Lille Catholic University, Faculté de Gestion, Economie et Sciences, Lille, France
lucien.mousin@univ-catholille.fr

***Abstract*—In solving combinatorial problems, the advent of multi-core machines has led to the development of new parallel methods offering higher performance as well as better solutions. However, finding the best parallel algorithm on such architectures for a given problem and programming these methods are still challenging tasks today. As a matter of fact, only a few parallel solvers have been designed so far to tackle multi-objective combinatorial optimisation problems. Therefore this paper first proposes a new highly parametric multi-thread and multi-objective local search algorithm dedicated to tackling optimisation problems. Specifically, this new parallel solver incorporates and combines most of methods available in the literature, such as single-walk and multi-walk parallel local search, which can be independent of each other or cooperative by sharing some solutions. In addition, this solver is also capable of making some innovative hybridisations by mixing both single-walk and multi-walk approaches.**

**The major problem is that it then becomes very difficult, if not impossible, for a human expert to determine the best configurations. This obstacle is due to the fact that the number of parameters in parallel methods is excessively too large. Indeed, all configuration possibilities include the combination of two kinds of parameters. The first ones are the parameters of the sequential algorithms within the parallel solver for multi-walk-based methods. The second ones are those controlling the main parallel components such as hybridisation, diversification or choice of communications. Fortunately, automatic algorithm configuration allows this problem to be taken into account. Thus, we use a configurator especially designed for multi-objective problems called MO-ParamILS in order to expose some best parallel configurations for the bi-objective travelling salesman problem.**

***Index Terms*—parallel computing, automatic configuration, multi-objective local search**

## I. Introduction

For NP-Hard multi-objective combinatorial optimisation problems, using an approximation (or incomplete) algorithm, such as metaheuristic, is one of the most popular optimization methods. Multi-objective local search (MOLS) algorithms are metaheuristics able to deal with multi-objective problems by handling the optimisation of several criteria simultaneously. More specifically, MOLS algorithms progressively improve the objective values of one or more solutions of a given optimisation problem through a local search algorithm in order to obtain high quality solutions. In the literature, MOLS algorithms have been used to tackle combinatorial optimisation problems such as the travelling salesman problem [19], the quadratic assignment problem [14], and the permutation flowshop scheduling problem (PFSP) [3], [14], [19]. Due to the complexity of these problems, improving MOLS algorithms is still a difficult and tedious task.

In order to improve the quality of the solutions within a reasonable amount of time, a basic approach from a hardware perspective consists in running existing solvers on more powerful processing units. This sounds interesting since Moore's Law states that the number of transistors in a processor will double every two years. Nevertheless, Moore's Law may not hold forever. There are physical limits to the ability to continually improve the overall processing power of computers [31]. Gordon Moore himself expects it will not hold beyond 2025. Thus, instead of increasing clock rates, processor manufacturers prefer to arrange multiple processors onto the same chip at the expense of some loss of performance (caused by synchronisations, atomic operation, load balancing, and data consistency) and more difficult programming. This is referred to as multi-core architecture.

Leveraging the power of multi-core architectures has had some success for solving hard combinatorial optimisation problems in both mono-objective and multi-objective contexts [8]. In the context of this paper, we are interested in parallel methods that focus on the local search algorithm. Parallel local search approaches distinguish between single-walk and multiple-walk methods. On the one hand, single-walk methods (fine-grained parallelism) are limited to the neighborhood of the current solution in the local search algorithm [21]. On the other hand, multi-walk methods (coarse-grained parallelism) consist in developing concurrent explorations of the search space by using distinct local search algorithms [32]. Let us also point out that multi-walk methods can be either independent or cooperative, with some communications between concurrent

This work has been achieved within the framework of CPER Data project. CPER Data is co-financed by European Union with the financial support of European Regional Development Fund (ERDF), French State and the French Region of Hauts-de-France.

processes [10]. In addition, other so-called hybrid methods are based on both single-walk and multi-walk at the same time in order to improve performance [1]. Nevertheless, there is little work comparing these methods, and it is still difficult to find the best parallel method for a given problem. Moreover, even if it seems natural to design parallel solvers to take advantage of technological advances, only a few parallel solvers have been designed so far for multi-objective problems. In fact, multi-threaded applications are generally hard to implement.

To overcome these disadvantages, we first propose a new highly parametric multi-thread MOLS algorithm which can instantiate the majority of known parallel methods with or without communications. However, as the number possible components and parameters increases, so does the difficult to analyze all possible combinations of strategies in order to find the best configurations.

To address this problem, the recent research field of automatic algorithm configuration (AAC) is an efficient and increasingly popular way to find the best configurations for metaheuristics. Indeed, today some algorithms containing an exponential number of configurations can use an AAC configurator in order to improve solver performances on a homogeneous set of instances representing a combinatorial problem. Thus, the advantages of using an AAC on a highly parametric algorithm have already been exploited in the past for single-objective SLS algorithms [25] but also on the MOLS algorithms [5]. In fact, this paper is a continuation of existing work [5] which aims to move from sequential to parallel computing by exploiting multi-thread architectures.

Thus, our second contribution is to use an automatic algorithm configuration called MO-ParamILS [4] to automatically search for the best configurations of our parallel MOLS algorithm. To our knowledge, there is no work that uses such a tool for a multi-thread MOLS algorithm. On top of that, we propose to use the same AAC configurator to also find good configurations of the sequential MOLS algorithm in order to use them as parameters inside of a portfolio mode within our parallel MOLS algorithm.

The paper is organized as follows. In the next section, basic notions about multi-objective, as well as sequential and parallel, MOLS algorithms are provided. Section III describes the concept of automatic configuration (AC) and enumerates the different strategies implemented in the multi-thread MOLS algorithm. We explain the experimental protocol and present and discuss results in Section IV. We then conclude and provide perspectives in Section V.

## II. Multi-Objective Local Search Algorithms

### A. Multi-Objective Optimisation (MOO)

Multi-objective optimisation aims to simultaneously improve several criteria (objective functions) that directly affect the solution quality of a given problem. More precisely, a MOO problem consists in determining the set of solutions in which objective functions $f_i(x)$ reach their optimal values. We may consider minimization without loss of generally:

$$\underset{x \in D}{\operatorname{argmin}}(f_1(X_1), f_2(X_2), \ldots, f_n(X_n)) \tag{1}$$

where $n$ is the number of objectives ($n \geq 2$) and $D$ is the set of feasible solutions where a solution $x$ is represented by the vector of $k$ decision variables $x = (x_1, x_2, \ldots, x_k)$ of the given problem. Moreover, the sets $X_n \subseteq x$ represent several vectors of decision variables which can be different depending on the associated objective function. Note that mixed MOO problems, consisting of both some objective functions to maximize and minimize, can easily be transformed into minimization MOO problems by changing their sign, $f'_i(x) = -f_i(x)$.

The concept of Pareto dominance is used to distinguish solutions according to criteria of multi-objective combinatorial problems. A solution $s_1$ dominates another solution $s_2$ if, and only if, (i) $s_1$ is better than or equal to $s_2$ for all criteria, and (ii) $s_1$ is strictly better than $s_2$ for at least one criterion. A set of non-dominated solutions $\{s_1, s_2, \ldots, s_m\}$, i.e, in which there is no couple $(s_i, s_j)$ $(i \neq j)$ such that $s_i$ dominates $s_j$ is called a Pareto set, a Pareto front, or an archive in the context of multi-objective local search algorithms. Solving a MOO problem consists in finding the Pareto optimal set $S^* \subset D$, i.e, the best Pareto set in which there is no other feasible solution $x' \in D$ that dominates any $x \in S^*$.

To assess the Pareto set's quality, various indicators have been proposed. Hypervolume (HV) is one of the most broadly used performance indicators in the literature on multi-objective optimisation [26]. Assuming normalised objective values in $[0, 1]$, unary hypervolume measures the volume between a given Pareto set of solutions and the point $(1, 1)$. While HV is primarily a convergence indicator, it also captures information about the diversity of the set of solutions. Another indicator is a variant of $\Delta$ spread [11], used to capture the distributional properties of a Pareto set. Given a Pareto set $S$, ordered according to the first criterion, we define $\Delta' := \frac{\sum_{i=1}^{|S|-1} |d_i - \bar{d}|}{(|S|-1)\cdot\bar{d}}$, where $\bar{d}$ denotes the average over the Euclidean distances $d_i$ for $i \in [1, |S| - 1]$ between adjacent solutions on the ordered set $S$. This indicator has to be minimised; it takes small values for large Pareto sets with evenly distributed solutions, and takes values close to or greater than 1 for Pareto sets with few or unevenly distributed solutions.

Since many combinatorial optimisation problems are hard to solve due to their NP-hard complexity, one of efficient solving methods is the use of approximation algorithm in order to obtain high-quality solutions in a reasonable amount of time. These metaheuristics may be classified either as nature-inspired (evolutionary, genetic, ant colony, ...) algorithms or local search-based (tabu search, iterated local search, variable neighborhood search, ...) algorithms.

### B. Multi-Objective Local Search (MOLS)

Local search methods explore the search space by iteratively making small changes to a single solution. The procedure generally starts from a good initial solution constructed heuristically using, for example, a hill climbing algorithm. The set of all possible moves for a given solution is called a

neighborhood. Next, a better solution than the current one is selected from the neighborhood to become the new current solution. However, the neighborhood may not contain any improving solution. The current solution is then called a local optimum. To remedy this problem, a local search algorithm tries to escape from this local optimum by moving to another area of the search space. Iterative local search (ILS) deals with local optima by restarting from a new random solution or by performing a sufficiently large perturbation on the current solution.

Multi-objective local search (MOLS) algorithms are most often based on Pareto local search (PLS) [27], that gradually improves a Pareto set. In the literature, numerous extensions to PLS have emerged, such as the iterated PLS [12], the stochastic PLS [13], the anytime PLS [15] and the dominance-based multi-objective local search (DMLS) [19]. As opposed to single-objective local search, improving only one solution, DMLS maintains and improves multiple candidate and non-dominated solutions in an archive. The sequential iterated MOLS algorithm used as base in this paper is more recent and proposes a new generalisation of MOLS that exploits many strategies available in the literature [5]. This algorithm starts by creating an archive of two solutions and then improve them by iteratively executing four distinct components: *selection*, *exploration*, *archive* and *perturbation*.

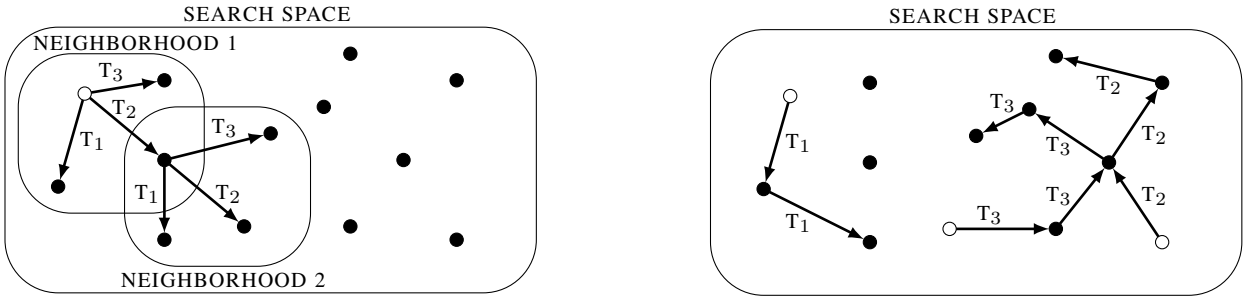
At each iteration, the first step is to select the solutions from the archive that will be explored (*selection* phase). In this phase, the `select-size` parameter allows the control of the number of solutions to be selected while `select-strat` represents the way to select them. The latter can be to select all solutions (`all`) or a certain number of solutions chosen either randomly (`rand`), or according to the time spent in the archive (`newest` and `oldest`). Thereafter the *exploration* phase seeks new candidate solutions to add to the archive by successively exploring the neighborhoods of the selected solutions. The `explor-strat` parameter determines which solutions should be added: either the improving ones (`all_imp` and `imp`), or the non-dominated ones (`ndom`) or the both improving and non-dominated ones (`all` and `imp_ndom`). Furthermore, the exploration is terminated when a given number `explor-size` of added solutions is reached except when the value of `explor-strat` is `all_imp` or `all`. In this case, there is no limit to the number of solutions added in the archive. In addition, the `explor-ref` parameter controls the comparison of neighbors for the improving and non-dominated criteria. This is carried out with respect to either the current solution (`sol`) or all the solutions in the archive (`arch`). Once the new solutions have been added to the archive, filtering is performed to keep only the non-dominated ones. Next, if the limit `archive-size` is exceeded then the archive is reduced by deleting uniformly at random some solutions (*archive* phase). This ensures the search space is kept within check for the next iteration. The last phase of the iteration, called *perturbation*, is then applied in order to diversify the search thanks to the parameter `perturb-strat`. Either a restart is performed by generating new random solutions (`restart`), or some kick moves are performed on all solutions of the archive (`kick_all`) or only on some of them (`kick`) thanks to the parameter `perturb-size`. Finally, the number of moves in a solution is carried out randomly according to the parameter `perturb-strength`. In the rest of this paper, we intend to exploit all of these parameters in a new Parallel Multi-Objective Local Search (PMOLS) algorithm.

### C. Parallel Multi-Objective Local Search (PMOLS)

Due to the large number of algorithmic components that do not contain any dependent tasks between them (some task A depends on B if it needs the result of B to be executed), local search methods naturally present various forms of parallelism. Parallel Multi-Objective Local Search (PMOLS) algorithms can thus take advantage of numerous opportunities such as fine-grained or coarse-grained parallelism, either with or without communication, .... Moreover, PMOLS can be used on various types of parallel architectures such as multi-cores, GPUs or cloud computing. Figure 1 presents the two main methods of parallel local search.

Single-walk methods, sometimes called neighborhood decomposition strategies, consist in using parallelism inside a single search process and exploit the independence between the tasks representing the neighborhood exploration of the local search. Indeed, this exploration can be easily divided into several parts where each part is computed by a thread (Figure 1a). In theory, this method can achieve, in some cases, linear speedup thanks to the independence between these parts. Nevertheless, as for all fine-grained parallel methods, an overload due to thread synchronisation can have a significantly detrimental impact on the speedup. Indeed, parallel processes need to be synchronized in order to choose the most promising neighbor to explore the next solution. Thus, this method can have very poor performance if the calculation cost of evaluations during the exploration is not high enough. However, single-walk methods have proved to be effective in solving different optimisation problems on both multi-thread [8] and GPU architectures [22]. Indeed, as the evaluation is the same algorithm but with different data (the solutions) and without dependencies, GPUs are typically dedicated to single-walk methods.

Contrary to single-walk, multi-walk methods are simply the execution of several and distinct local search algorithms, i.e, a portfolio of solvers launched in parallel (Figure 1b). Nevertheless, since the search space is not divided, several processes can explore the same solutions (as illustrated by the threads $T_2$ and $T_3$ on the figure). Conversely, too much diversification between the processes prevents local search from focusing on the best solutions. In the field of parallel computing of combinatorial problems, this problem is known as the diversification versus intensification dilemma between search processes. Multi-walk methods can be divided into two categories: independent and cooperative approaches. This last category adds a communication mechanism allowing to exchange information in order to improve either the diversification or the intensification between search processes [8].




(a) Single-walk parallelisms

(b) Multi-walk parallelisms

Fig. 1: Different parallel trajectories (arrows) of three threads ($T_1$, $T_2$ and $T_3$) for the single-walk and multiple-walk methods. The dots are solutions in the search space. Among them, unfilled circles represent the initial solutions of each thread.

In the literature, some cooperative multi-walk approaches are popular. For instance, there are some works on the metaheuristic parallelization which consist in creating multiple solver entities and sharing some information like elite solutions with the master node, which maintains a centralized shared memory for the best solvers [16]. Other parallel metaheuristic solvers prefer to exchange their configurations every $k$ iterations and each solver instance decides whether it adopts a received configuration or continues its current search process [9], [24]. More recently, [7] presents a parallel adaptive search in which only integer values are exchanged between entities in order to trigger or not restart procedures. Other works are agent-based approaches and are composed of a set of agents that perform specific tasks (search agent, intensifying agent and diversifying agent). Agents are often provided from others domains like machine learning techniques or genetic algorithms [29]. Finally, ParadisEO-PEO is a framework dedicated to multi-objective constrained combinatorial optimisation and supports different levels of parallel metaheuristics, from neighborhood decomposition (single-walk) to independent and cooperative multiple-walk [30].

In the paper, we have implemented a hybrid approach mixing both single-walk and multi-walk methods. Nevertheless, to the best of our knowledge, there is less work dealing with this hybridisation. Among them, Arbelaez and Codognet [1] take advantage of both GPU and CPU architectures at the same time by executing multiple instances of the adaptive search solver and by performing the evaluation of large neighborhoods in parallel.

## III. Automatic Configuration of Parallel Multi-Objective Local Search

### A. Multi-Objective Automatic Configuration

The Automatic Algorithm Configuration (AAC) research field provides tools called configurators allowing to determine the best configurations among all valid combinations of parameter values of a target algorithm. Indeed, since this configuration space is exponentially large, in most situations, finding the best configurations manually cannot be accomplished, even by experienced individuals. Thus AAC is becoming an essential ingredient in the design of powerful solvers for challenging optimisation problems. However, most of the existing work on AAC focuses on configuration procedures that optimize a sequential and single-objective algorithm. In contrast, in this paper, we deal with a more challenging AAC problem which aims to find the best configurations of a target algorithm combining both parallel computing and multi-objective optimisation.

On the one hand, to optimize the target algorithm, most AAC configurators in the literature (irace [23], ParamILS [17], or SMAC [18]) use a single indicator usually representing the resolution time or the solution quality. But more recently, as part of the work of Blot et al. [5], multiple performance indicators dedicated to multi-objective optimisation problems have shown their effectiveness through the MO-ParamILS configurator [4]. Moreover, the authors of [6] have achieved better performance with MO-ParamILS rather than with a single-objective configurator that performs an aggregation of two objectives. Let us also note that another tool called SPRINT-Race, based on multi-objective model racing, was used to identify the Pareto optimal parameter settings of Ant Colony Optimization algorithms [33].

On the other hand, AAC development and applications remain largely focused on sequential rather than parallel approaches. In parallel computing, some AAC configurators has been employed in order to determine the best parallel configurations [20]. More precisely, Automatic Construction of Parallel Portfolios (ACPP) involves optimizing some sequential parameters of each solver into the portfolio. For instance, Lindauer et al. [20] expose parameterized SAT solvers that produce significantly better SAT solvers than state-of-the-art parallel solvers built by human experts.

Finally, we have not encountered any work on the application of a configurator to a target algorithm that is both parallel and multi-objective, and this paper addresses this topic.

### B. Parameters of Parallel MOLS

The parameters are classified into two parts: parameters coming from the sequential MOLS algorithm and those dedicated to the parallel MOLS program. The first parameters are those controlling the search of a single solver, i.e. a single MOLS algorithm. Among them, we find the totality

of parameters used by Blot et al. [5] presented in section II-B such as `explor-strat` or `archive-size`. In addition, the other parameters deal with the main parallel components such as the single-walk, the multi-walk, the hybridisation, or the choice of communications.

Nevertheless, some parameters depend on other parameters coming from either sequential or parallel MOLS algorithms. For example, the parameter `explor-size` has to be used only when the value of `explor-strat` is `imp`, `imp_ndom` or `ndom`. In addition, the parallel single-walk method is inherently incompatible with certain MOLS methods. In fact, in order to have enough tasks with sufficient computational load in a single-walk method, the values of parameters `select-size` and `explor-size` have to be large enough. To alleviate this problem, we carry out two different experiments: one for the single-walk and hybrid methods, and one for the multi-walk methods. Thus, each of these experiments has different parameter values as exposed in the table I.

| Parameter | Single-Walk and Hybrid | Multi-walk |
|---|---|---|
| **Sequential** | | |
| select-strategy | $\{all, rand, newest, oldest\}$ | |
| explorer-strategy | $\{all, all\_imp, first, imp, imp\_ndom, ndom\}$ | |
| explor-ref | $\{sol, arch\}$ | |
| perturb-strat | $\{restart, kick, kick\_all\}$ | |
| perturb-strength | $\{3, 5, 10\}$ | |
| perturb-size | $\{1, 5, 10\}$ | |
| archive-size | $\{20, 50, 100, 1000\}$ | |
| select-size | $\{10, 15\}$ | $\{1, 3, 10\}$ |
| explor-size | $\{10\}$ | $\{1, 3, 10\}$ |
| **Parallel** | | |
| hybridisation | $\{\langle 1, 20\rangle, \langle 2, 10\rangle, \langle 4, 5\rangle, \langle 10, 2\rangle\}$ | $\{\langle 20, 1\rangle\}$ |
| configuration-1 | $\{1, 2, 3, 4, 5\}$ | |
| configuration-2 | $\{\ldots, \langle 1, 2\rangle, \ldots, \langle 3, 5\rangle, \ldots\}$ | |
| configuration-4 | $\{\ldots, \langle 1, 2, 3\rangle, \ldots\}$ | |
| configuration-10 | $\{\ldots, \langle 4, 4, 4, 4\rangle, \ldots\}$ | |
| configuration-20 | | {..., $\langle 1, 3, 4, 5, 5\rangle$, ...} |
| diversification | $\{solution, neighborhood, both\}$ | |
| communication | $\{none, all, half, one\}$ | |
| communication-send | $\{1, 2, 4, 8, 16\}$ | |

TABLE I: Sequential and parallel parameters. Respectively according to the single-walk and hybrid experiment and the multi-walk experiment, the total number of configurations is to 5460 and 10920 for the sequential part and is to 7500 and 7560 for the parallel part.

The three columns of Table I represent, respectively, the name of the parameter and the parameter values, first for the single-walk and the hybrid experiment, and then for the multi-walk experiment. The lines of the table summarize firstly in a first group the sequential parameters used, whereas the second group exposes the new parameters implemented in our parallel MOLS algorithm.

`hybridisation`. This parameter is composed of two numbers: the first is defined as the number of threads representing the portfolio of the multi-walk method, noted $n$, where each thread is a MOLS algorithm, while the second is the number of threads dedicated to the single-walk methods for each MOLS algorithm of the portfolio, noted $m$. Thus the total number of threads used in our PMOLS algorithm is the product of these two numbers $n \times m$. For example, the values $\langle 4, 5\rangle$ represent a PMOLS algorithm of $n = 4$ MOLS algorithms in parallel, and each of them divides the computational load of its neighborhood exploration across $m = 5$ threads thanks to the single-walk method, making a total of 20 threads. Two extreme cases of this parameter should also be noted: the pure single-walk method and the pure multi-walk method without hybridisation and respectively represented by the values $\langle 1, 20\rangle$ and $\langle 20, 1\rangle$.

`configuration-`$n$. This set of parameters is used to choose the values of sequential parameters (i.e. configurations) for each of $n$ MOLS algorithms in the portfolio among $x$ available sequential configurations. In the example of Table I and in our experiments, $x = 5$ in order to limit the exponential explosion of an excessively large number of sequential configurations. By definition, these parameters depend on the value of parameter `hybridisation`, i.e., `configuration-`$n$ is taken into account (i.e. used as parameter) only if the value of `hybridisation` is equal to $\langle n, m\rangle$ (i.e, contains $n$ MOLS algorithms in the portfolio). For example, when `hybridisation` is $\langle 10, 2\rangle$, only `configuration-10` is used as parameter, and other `configuration-`$n$ such as $n \neq 10$ are not possible parameters. A possible value of the parameter `configuration-`$n$ is a set of distinct numbers between 1 and 5 where each number represents a sequential configuration among the $x$ available sequential configurations. Moreover, the values of `configuration-`$n$ are all combinations of size 1, 2, 3, 4 or 5 of the five available sequential configurations, respectively for $n$ equal to 1, 2, 4, 10 or 20. For instance, the values of `configuration-4` are of size 3 and some of them can be, for example, $\langle 1, 2, 3\rangle$ or $\langle 2, 4, 5\rangle$. In addition, we remove also those that are equivalent ($\langle 1, 2, 3\rangle$ and $\langle 2, 3, 1\rangle$). This allows to browse through a wide range of sequential configuration combinations for the portfolio without adding unnecessary or equivalent configurations. When the number of MOLS algorithms $n$ is greater than the number of available sequential configurations $x$, then our parallel algorithm repeats the distribution in the order of configurations given by parameter values of `configuration-`$n$. For example, for hybridisation $\langle 10, 2\rangle$, the chosen parameter is `configuration-10`, and if this parameter is $\langle 1, 2, 2, 5\rangle$, then the sequential configurations of 10 MOLS algorithms are the configurations 1, 2, 2, 5, 1, 2, 2, 5, 1 and 2.

`diversification`. This parameter specifies the mechanism of the diversification of each MOLS in the portfolio. Either only their initial solutions are distinct (`solution`) or only their neighborhood explorations are performed in different orders (`neighborhood`) or `both`.

`communication` and `communication-send`. These parameters control the amount of exchanged solutions at the end of each neighborhood exploration. The communication can be turned off thanks to the value `none` of `communication`. In the other cases, solutions are received either by `one` or `all` MOLS algorithms or `half` of them. In addition, the parameter `communication-send` is the number of solutions to send to others. Note also that the

solutions sent are among the best from the current archive according to the Pareto-dominance.

## IV. Experiments

### A. Case Study

The Symmetric Travelling Salesman Problem (TSP) can be modelled as an undirected weighted and complete graph $G$ where the vertices denote the cities whereas the edges (pairs of non-ordered vertices) correspond to distances between cities. Given a TSP instance $G$, the goal is to determine the shortest possible route that visits each city (vertex) and returns to the origin city, i.e, finding a minimum-weight Hamiltonian cycle in $G$. The decision version is an NP-complete problem and a possible solution (not necessarily the best route) is represented by a permutation of cities, i.e., a Hamiltonian G-cycle.

The experiments in this paper focus on the bi-objective symmetric TSP (bTSP) problem, which is defined as the TSP but with the peculiarity of having two weights to minimize per edge instead of one. In other words, the aim is to minimize the total distance covered by a round trip according to each of the two objectives. Note also that these two objectives are not correlated since they were computed by combining two independently generated distance matrices computed using Euclidean distance between cities randomly placed on a two-dimension grid.

In order to fairly test the final parallel configurations found by the configurator on other different instances, we have two disjoint sets of distinct bTSP instances: training instances and test instances. Furthermore in each of these sets, we classify the instances according to the number of cities which are: $100$, $300$ and $500$ cities.

The training instances are new instances generated by the original generator from the DIMACS challenge and contain $30$ instances for each number of cities representing a total of $435$ possible combinations for the bTSP problem per city size. In contrast, test instances taken from the literature are used to measure the performance of bTSP algorithms [19], [27]. This set consists of $6$ instances for each number of cities totaling $15$ different pairwise independent combinations per city size.

### B. Protocol

In order to configure our parallel MOLS without aggregating hypervolume and spread into a single performance indicator, we have made the choice to use the recent multi-objective AAC configurator MO-ParamILS, itself based on a MOLS algorithm. Thanks to this feature, MO-ParamILS is, in effect, fully capable of exploiting the multi-objective aspect. The usual MO-ParamILS configuration protocol uses three phases: training, validation and test.

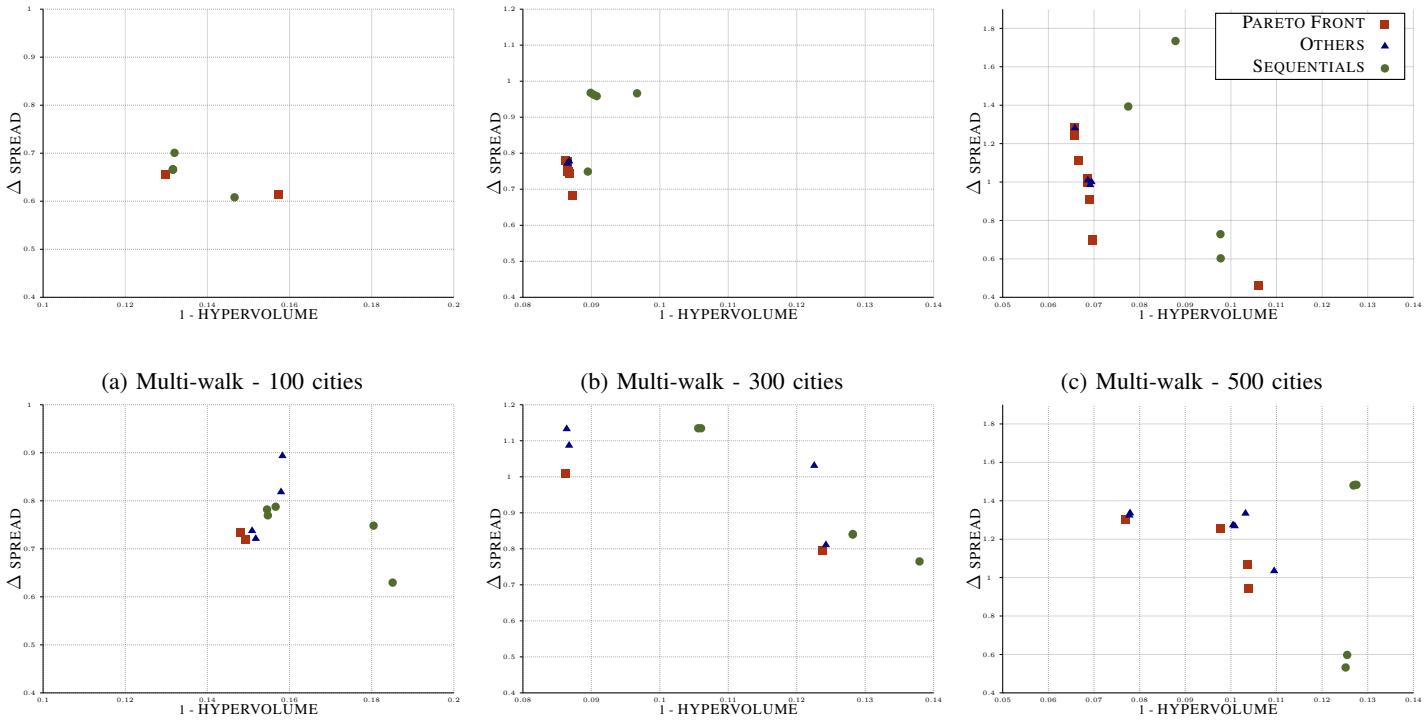
In the training phase, we run the MO-ParamILS configurator on the training instances many times with different random seeds in order to find diverse and various Pareto fronts representing some distinct good configurations. In order to complete this phase within a reasonable period of time, note that MO-ParamILS does not test the configurations on all instances but only on a subset of them. Next, to reduce the number of configurations by taking only the best of them, a second phase allowing a fairer comparison of configurations is necessary. This is achieved by the validation phase which consists in executing the configurations found by the training phase on the complete set of training instances. The result thus obtained is in the form of a single Pareto front of some non-dominated configurations. To finish, the Pareto set of configurations of the validation phase is evaluated on the test instances in order to check the quality of these configurations on other different instances.

However, this protocol usually used for sequential algorithms has to be adapted to the parallel features of our PMOLS program. We have therefore chosen to divide the protocol into two steps: one for the sequential MOLS and one for the parallel MOLS.

The first step is composed of a training phase followed by a validation phase and aims to find five good configurations of the sequential MOLS algorithm (among the parameters of the first part of the table I) intended for the parameter `configuration-`$n$ of the parallel MOLS algorithm. These five configurations are first chosen in the Pareto front of the validation phase and then among other dominated configurations if this Pareto front is out of stock. The second step consists of finding the best configurations of the parallel MOLS algorithm by using the five good configurations of the previous step. However, due to limited resources, the training and validation phases have been revised by dividing by four the number of training instances representing a total of $30$ instances.

The experiments have been conducted on a cluster of four computers each containing $20$ Intel XEON cores running at $2.2$ GHz and with $500$ GiB RAM. The cluster is equipped with an Ethernet controller at $1$ GiB/s. The sequential MOLS algorithms have been rewritten from [5] into a new library called MH-builder. This library now takes advantage of the object-oriented programming in order to be fully modular and thus help in the building of many MOLS algorithms. The parallel methods and the communications have been implemented thanks to the pFactory library available at https://github.com/crillab/pfactory [2].

For each MOLS or PMOLS algorithm, the base times are $20$, $60$ and $100$ seconds, respectively, for $100$, $300$ and $500$ cities. Note that the training phases are consist of $10$ MO-ParamILS runs achieving each $100$ MOLS algorithm. The total time of the validation phases depends of the number of configurations found in the training phases. Thus, for $500$ cities and the single walk and hybrid experiment, the sequential step has lasted about $9$ days: $1$ day for the training phase ($1000 \times 100$ seconds) and $8$ days for the validation phase ($15$ configurations $\times 455$ instances $\times 100$ seconds). For the sake of comparison, an exhaustive approach on the training instances will last $15.06$ years ($10920$ configurations $\times 435$ instances $\times 100$ seconds). Next, the parallel step has taken $2$ days: about $1$ day for the training phase ($1000 \times 100$ seconds), $1$ day for the validation phase ($25$ configurations $\times 30$ instances $\times 100$ seconds) and about $1$ hour ($2$ configurations $\times 15$ instances $\times 100$ seconds)




(a) Multi-walk - 100 cities (b) Multi-walk - 300 cities (c) Multi-walk - 500 cities

(d) Hybridisation - 100 cities (e) Hybridisation - 300 cities (f) Hybridisation - 500 cities

Fig. 2: Results of the single-walk and hybrid experiment and the multi-walk experiment on 100, 300 and 500 cities.

for the test phase. Once again, note that an exhaustive approach on the training instances would have lasted 0.71 years (7500 configurations $\times 30$ instances $\times 100$ seconds). In the same way, each other experiment takes between 4 and 15 days of computation.

### C. Results and Discussion

Let us recall that we have performed two experiments, the hybrid one, and the multi-walk one. Each of these experiments follows the protocol previously indicated, consisting of a sequential step and a parallel step. Figure 2 shows the results of the test phases of the parallel step (the triangles and squares) and, for the sake of comparison, the results of the five sequential configurations (the circles) used in the associated parallel methods on the same test instances with identical run times (20, 60 and 100 seconds, respectively, for 100, 300 and 500 cities). Each point represents the averages of $1-$hypervolume and spread values of each configuration for all test instances. Thus, the closer the points are to the origin (with small values), the better their hypervolume and spread. Note that the comparisons between the sequential and parallel methods are performed only w.r.t. the hypervolume and spread values, and not on the speedup in terms of time.

On the one hand, the multi-walk experiment is represented by the first three sub-figures 2a, 2b and 2c. We can see that the parallel methods are better than the sequential ones for both the hypervolume and spread values. On the other hand, the hybrid experiment (sub-figures 2d, 2e and 2f) exposes greater discrepancies between sequential and parallel methods, notably for the hypervolume values. Furthermore, for all experiments, the larger the city size, the greater the gap. For the single-walk methods, this is due to the fact that they are more efficient when the computation load of the function evaluation is higher, i.e, for 500 cities. More generally, the parallel performances are also less strong for 100 cities because sequential methods are already very close to optimality. By comparing both experiments, we can observe that the multi-walk methods are the best, even though some hybrid configurations are similar in terms of hypervolume for 300 cities. We conclude that both hybrid and multi-walk methods are both useful and powerful.

Concerning sequential parameters, the `imp-ndom`, `ndom` and `all-imp` strategies with different kick sizes and select strategies are the most commonly used. The archive size is often set to 1000. We also found `explor-ref`, which controls what to compare neighbors against when evaluating improvement and non-domination, to be equal to either the archive or the solution. For parallel parameters, communications are useful because they are used in most of the best parallel configurations by often sending one solution to some other solvers of the portfolio. The parameter `diversification` is never set to `solution` and often set to `both` (i.e. a combination of distinct initial solutions and performing neighborhood exploration in different orders). Concerning hybridisation, the best parallel configurations use

a portfolio of 4 solvers where each solver performs a single-walk of 5 threads for 100 cities (value 4_5). For the other city sizes, the parameter hybridisation is often also equal to 4_5 but sometimes to 10_2 or 2_10. We can see also two pure single-walk methods (1_20) in the configurations of 500 cities. Therefore, this shows that all parallel components may be useful, within a given context, and they can improve the parallel MOLS algorithm performances when configured correctly.

## V. Conclusion and Perspective

This paper deals with, and obtains encouraging preliminary results, on a problem that had never been tackled until now: how to find the best configurations of a local search algorithm that is both parallel and multi-objective. As a first step, we have implemented a new highly parametric multi-thread MOLS dedicated to tackling optimisation problems. In addition, this solver is also capable of making some innovative hybridisations by mixing both single-walk and cooperative multi-walk approaches. Next we have exposed some best parallel configurations by using MO-ParamILS thanks to two distinct experiments. In the near future, we plan to build upon this work and explore various perspectives. Firstly, we will test other sequential configurations for the portfolio of the parallel MOLS algorithm because, as we can see from our results, it is not necessarily the best sequential configurations that form the best parallel solver. Secondly, we are also going to add the possibility for solvers to communicate their search space in order to improve their diversification potential. Finally, the performance of our parallel MOLS algorithm should also be investigated on other problems, such as the rule mining problem [28]. Indeed, because the evaluation steps of this problem are more expensive, we believe that hybrid methods would be more suitable.

## References


[1] Alejandro Arbelaez and Philippe Codognet. A gpu implementation of parallel constraint-based local search. In PDP '14, page 648–655, USA, 2014. IEEE Computer Society.

[2] Gilles Audemard, Gael Glorian, Jean-Marie Lagniez, Valentin Montmirail, and Nicolas Szczepanski. Pfactory: A generic library for designing parallel solvers. 2019.

[3] Aymeric Blot, Hernan Aguirre, Clarisse Dhaenens, Laetitia Jourdan, Marie-Eléonore Marmion, and Kiyoshi Tanaka. Neutral but a Winner! How Neutrality Helps Multiobjective Local Search Algorithms. In *EMO 2015*, volume 9018, pages 34–47, Guimarães, Portugal, 2015.

[4] Aymeric Blot, Holger Hoos, Laetitia Jourdan, Marie-Éléonore Marmion, and Heike Trautmann. MO-ParamILS: A Multi-objective Automatic Algorithm Configuration Framework. In *LION*, volume 10079 of *Lecture Notes in Computer Science*, pages 32–47, Ischia, Italy, May 2016.

[5] Aymeric Blot, Marie-Eléonore Kessaci, Laetitia Jourdan, and Holger Hoos. Automatic Configuration of Multi-Objective Local Search Algorithms for Permutation Problems. *Evolutionary Computation*, 2019.

[6] Aymeric Blot, Alexis Pernet, Laetitia Jourdan, Marie-Éléonore Kessaci-Marmion, and Holger H Hoos. Automatically Configuring Multi-objective Local Search Using Multi-objective Optimisation. In *EMO 2017*, volume 10173, pages 61–73, Münster, Germany, March 2017.

[7] Yves Caniou, Codognet Philippe, Florian Richoux, Daniel Diaz, and Salvador Abreu. Large-scale parallelism for constraint-based local search: the costas array case study. *Constraints*, 20:30–56, 01 2014.

[8] Philippe Codognet, Danny Munera, Daniel Diaz, and Salvador Abreu. Parallel local search. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 381–417. 2018.

[9] Jean-François Cordeau and Mirko Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers OR*, 39:2033–2050, 09 2012.

[10] Teodor Gabriel Crainic, Michel Gendreau, Pierre Hansen, and Nenad Mladenovic. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10, 05 2004.

[11] Kalyan Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6:182 – 197, 05 2002.

[12] Madalina Drugan and Dirk Thierens. Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18, 10 2012.

[13] Madalina M. Drugan and Dirk Thierens. Path-guided mutation for stochastic pareto local search algorithms. In *PPSN XI*, pages 485–495, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[14] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. Anytime pareto local search. *European Journal of Operational Research*, 243(2):369 – 385, 2015.

[15] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. Anytime pareto local search. *European Journal of Operational Research*, 243, 06 2015.

[16] Michel Gendreau, François Guertin, Jean-Yves Potvin, and Éric Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33:381–390, 11 1999.

[17] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 01 2009.

[18] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, 2011.

[19] Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, and El-Ghazali Talbi. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 18(2), 2012.

[20] Marius Lindauer, Holger Hoos, Kevin Leyton-Brown, and Torsten Schaub. Automatic construction of parallel portfolios via algorithm configuration. *Artificial Intelligence*, 244, 05 2016.

[21] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. Local search algorithms on graphics processing units. a case study: The permutation perceptron problem. 04 2010.

[22] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. Gpu computing for parallel local search metaheuristics. *IEEE Trans. Computers*, 2013.

[23] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3, 01 2011.

[24] Rui Machado, Salvador Abreu, and Daniel Diaz. Parallel Performance of Declarative Programming using a PGAS Model. In *Practical Aspects of Declarative Languages (PADL)*, Rome, Italy, 2013.

[25] Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle. Automatic design of hybrid stochastic local search algorithms. pages 144–158, 05 2013.

[26] C. von Lücken N. Riquelme and B. Barán. "performance metrics in multi-objective optimization,".

[27] Luis Paquete, Marco Chiarandini, and Thomas Stützle. *Pareto Local Optimum Sets in the Biobjective Traveling Salesman Problem: An Experimental Study*, volume vol. 535, pages 177–199. 01 2004.

[28] Tari Sara, Szczepanski Nicolas, Mousin Lucien, Jacques Julie, Kessaci Marie-Eleonore, and Jourdan Laetitia. Multi-objective automatic algorithm configuration for the classification problem of imbalanced data. In *IEEE Congress on Evolutionary Computation CEC*, 2020.

[29] El-Ghazali Talbi and Vincent Bachelet. Cosearch: A parallel cooperative metaheuristic. *J. Math. Model. Algorithms*, 5:5–22, 04 2006.

[30] El-Ghazali Talbi, S. Cahon, and Nouredine Melab. Designing cellular networks using a parallel hybrid metaheuristic on the computational grid. *Computer Communications*, 30:698–713, 02 2007.

[31] M. Mitchell Waldrop. The chips are down for Moore's law. *Nature*, 530(7589):144–147, 2016.

[32] Mehdi Yazdani, Maghsoud Amiri, and Mostafa Zandieh. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37:678–687, 01 2010.

[33] Tiantian Zhang, Michael Georgiopoulos, and Georgios C. Anagnostopoulos. Sprint multi-objective model racing. GECCO '15, page 1383–1390, New York, NY, USA, 2015.