Automatic Configuration of Multi-Objective Local Search Algorithms for Permutation Problems

Aymeric Blot aymeric.blot@univ-lille.fr
Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL – Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Marie-Éléonore Kessaci mkessaci@univ-lille.fr
Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL – Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Laetitia Jourdan laetitia.jourdan@univ-lille.fr
Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL – Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Holger H. Hoos hh@liacs.nl
LIACS, Leiden University, Leiden, 2333 CA, The Netherlands

Abstract

Automatic algorithm configuration (AAC) is becoming a key ingredient in the design of high-performance solvers for challenging optimisation problems. However, most existing work on AAC deals with configuration procedures that optimise a single performance metric of a given, single-objective algorithm. Of course, these configurators can also be used to optimise the performance of multi-objective algorithms, as measured by a single performance indicator. In this work, we demonstrate that better results can be obtained by using a native, multi-objective algorithm configuration procedure. Specifically, we compare three AAC approaches: one considering only the hypervolume indicator, a second optimising the weighted sum of hypervolume and spread, and a third that simultaneously optimises these complementary indicators, using a genuinely multi-objective approach. We assess these approaches by applying them to a highly-parametric local search framework for two widely studied multi-objective optimisation problems, the bi-objective permutation flowshop and travelling salesman problems. Our results show that multi-objective algorithms are indeed best configured using a multi-objective configurator.

Keywords

Algorithm configuration, local search, multi-objective optimisation, permutation flowshop scheduling problem, travelling salesman problem.

1 Introduction

Many optimisation tasks involve several competing objectives, and general-purpose methods for determining and characterising good tradeoffs between these objectives are of considerable academic and practical interest. Most algorithms for solving these multi-objective optimisation (MOO) problems have parameters, whose settings have substantial impact on performance, and finding good settings of those performance

parameters can be quite difficult. As for single-objective optimisation algorithms, performance parameters for MOO algorithms have traditionally been configured manually, based on human intuition and limited experimentation. More recently, general-purpose automated algorithm configuration procedures have become available and are increasingly widely used for this purpose (see, *e.g.*, López-Ibáñez et al., 2016).

Standard, general-purpose algorithm configurators, such as irace (López-Ibáñez et al., 2016), SMAC (Hutter et al., 2011) and ParamILS (Hutter et al., 2007, 2009), can and have been used for optimising the performance of high-performance MOO algorithms in terms of standard performance indicators, such as hypervolume, ε- or R-indicators (see, *e.g.*, Zitzler and Thiele, 1999; Knowles and Corne, 2002; Okabe et al., 2003). However, the performance of MOO algorithms is generally assessed using multiple performance indicators, in order to characterise complementary properties of the solutions produced by them, such as convergence and diversity (Zitzler et al., 2003). This suggests the use of multi-objective configuration procedures that can effectively explore the tradeoff between multiple performance indicators – the use of multi-objective configurators to optimise the performance of MOO algorithms (Blot et al., 2017b).

In this article, we present extensive evidence that multi-objective automatic algorithm configuration is preferable over single-objective automatic algorithm configuration for the performance optimisation of MOO algorithms when there are several performance indicators of interest. In particular, we consider two well-known MOO problems, the bi-objective permutation flowshop scheduling problem (PFSP) and the bi-objective travelling salesman problem (TSP), and study a highly parameterised framework of multi-objective iterated local search (MOLS) algorithms for those problems. For the PFSP, we consider two naturally correlated, commonly used objectives: makespan and flowtime, while in the case of the TSP, we use two uncorrelated optimisation objectives (López-Ibáñez and Stützle, 2012). In both cases, we configure our MOLS framework for two complementary performance metrics: hypervolume and Δ'-spread. We report results from experiments on a small configuration space of our algorithm framework, which can be searched exhaustively, demonstrating that our general-purpose multi-objective algorithm configuration procedure, MO-ParamILS (Blot et al., 2016), can find a broad range of optimal or close-to-optimal configurations. Furthermore, we show that within a much larger configuration space, using MO-ParamILS, better results are obtained than by using ParamILS (Hutter et al., 2007, 2009), the standard, single-objective configurator it was derived from. Finally, we provide some insights into the MOLS configurations obtained from MO-ParamILS.

Our work presented here builds on and extends two recent, more limited studies (Blot et al., 2017b,a). Here, we additionally consider the bi-objective TSP, to investigate to which extent our results for the PFSP generalise to another permutation-based MOP with uncorrelated objectives. To further strengthen our results, we also consider large instances of the PFSP and a slightly different measure of spread. Finally, we perform an exhaustive analysis of a small configuration space of our MOLS framework on our training and test instance sets for our bi-objective PFSP and TSP, in order to assess the results obtained with our automated configuration procedures against truly Pareto-optimal sets of configurations.

2 Preliminaries

In this section, we provide some background on multi-objective optimisation problems, performance evaluation of multi-objective algorithms and our multi-objective local search framework.

2.1 Multi-Objective Optimisation (MOO)

In multi-objective optimisation, multiple criteria (or objective functions) characterising the quality of solutions of a given problem are optimised simultaneously. A MOO problem involves minimising (w.l.o.g.) a vector of functions over a space of candidate solutions, *i.e.*, to determine

$$\text{argmin}_{x \in \mathcal{D}}(f_1(x), f_2(x), ..., f_n(x))$$

where n is the number of objectives ($n \geq 2$), $x = (x_1, x_2, \ldots, x_k)$ is a vector of decision variables (which may have discrete or continuous domains), $\mathcal{D}$ is the set of feasible solutions, and each function $f_i(x)$ has to be minimised.

The concept of *Pareto dominance* is used to capture trade-offs between the criteria f_i: solution s_1 is said to dominate solution s_2 if, and only if, (i) s_1 is better than or equal to s_2 according to all criteria, and (ii) there is at least one criterion according to which s_1 is strictly better than s_2. A set S of solutions in which there are no $s_1, s_2 \in S$ such that s_1 dominates s_2 is called a *Pareto set*, a *Pareto front*, or – in the context of multi-objective local search algorithms – an *archive*. The goal when solving an instance of a MOO problem is to determine the best such set, *i.e.*, the set $X \subset \mathcal{D}$ such that there is no $x' \in \mathcal{D}$ that dominates any of the $x \in X$.

In principle, MOO problems can be solved using single-objective optimisation algorithms, by aggregating the objectives into a single function (Murata and Ishibuchi, 1998). Using this approach, optimising a weighted sum of multiple objectives corresponds to searching for an optimal solution to a given MOO problem in a single direction in objective space. It is known that in some cases, there can be solutions in an optimal Pareto set S^* that cannot be obtained in this manner (namely, when S^* is not convex); therefore, dedicated multi-objective optimisation algorithms are usually preferable.

2.2 Performance Evaluation

Multi-objective algorithms provide a Pareto set of solutions. In order to assess the quality of such Pareto sets, different indicators have been proposed (Zitzler and Thiele, 1999; Knowles and Corne, 2002; Okabe et al., 2003). These usually characterise the final Pareto set produced by a multi-objective algorithm in terms of convergence, distribution or cardinality. Since no single quality indicator captures all of these properties, it is recommended to consider multiple indicators, preferably complementing each other, in order to assess the performance of multi-objective algorithms (Zitzler et al., 2003).

In this work, we use a combination of two indicators: the classical hypervolume (Zitzler and Thiele, 1999) and a complementary spread measure. These were chosen in light of their common usage, their complementarity, and the additional requirements for unary indicators that do not require reference sets arising in the context of the automatic algorithm configuration process at the core of our study.

Hypervolume (HV) is by far the most broadly used performance indicator in the literature on multi-objective optimisation (Riquelme et al., 2015). Assuming normalised objective values in $[0, 1]$, the unary hypervolume measures the volume between the Pareto set of solutions and the point $(1, 1)$ (see Figure 1 (left)). HV is primarily a convergence indicator, but also captures information about the diversity of the front of solutions.

As a complementary indicator, we use a variant of spread to capture the distributional properties of the Pareto set (see Figure 1 (right), which shows two sets of solutions, one well-distributed (squares) and the other unbalanced (circles)). Given a Pareto

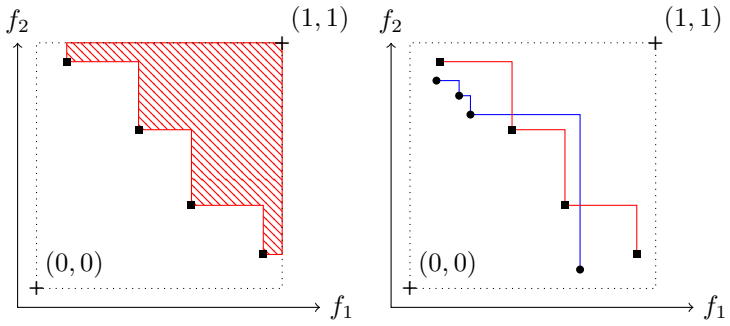

Figure 1: Properties (left: hypervolume; right: spread) of two sets of solutions (squares: well-distributed solutions; circles: unbalanced solutions)

set S, ordered regarding 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 is to be minimised; it takes small values for large Pareto sets with evenly distributed solutions, and values close to or greater than 1 for Pareto sets with few or unevenly distributed solutions. This slightly differs from the widely used spread indicator (Deb et al., 2002) in that it does not use extreme positions (after normalisation, the points $(1,0)$ and $(0,1)$) and only considers the distribution inside the Pareto set. Obviously, this indicator cannot be used alone to assess a Pareto set, but it complements the information captured by the hypervolume indicator. Using these two unary indicators, we can assess the performance of multi-objective algorithms in terms of the quality, diversity and distribution of the final Pareto sets obtained. In our experiments, in order to facilitate analysis, we will consider the minimising variant of hypervolume, calculated as $1 - HV$, so that performance of a multi-objective algorithm is optimised by minimising both indicators.

2.3 Multi-Objective Local Search Algorithms

The algorithm investigated in this paper is an iterated Multi-Objective Local Search (MOLS) algorithm, detailed in Algorithm 1 (Blot et al., 2017a).

The MOLS algorithm has four distinct components – *selection*, *exploration*, *archive* and *perturbation* – each of which exposes several parameters that potentially impact performance. These phases are presented hereafter, together with the space of parameter configurations (*i.e.*, combinations of parameter values) considered in our experiments, which are summarised in Table 1.

Selection. The first step of every iteration is the selection of a subset of solutions from the current archive to be further explored. How these solutions are selected is controlled by parameter `select-strat`. We can either select all solutions from the current archive, or only a specific number of them (specified by parameter `select-size`); in the latter case, solutions can be selected either uniformly at random, or according to the order in which they have been inserted in the archive (from oldest to newest). In all cases, a solution whose neighbourhood has been completely explored is never selected again.

Algorithm 1: Iterated Multi-Objective Local Search

Output: A set of solutions

```
archive, current ← init();
until termination criterion is met do
    until inner termination criterion is met do
        /* Selection */
        selected ← select(current);
        /* Exploration */
        candidates ← ∅;
        for solution ∈ selected do
            ref ← reference( solution, current );
            accepted ← explore( solution, ref );
            candidates ← candidates ⋃ accepted;
        /* Archive */
        current ← current ⋃ candidates;
        current ← pareto( current );
        current ← bound( current );
    archive ← archive ⋃ current;
    archive ← pareto( archive );
    /* Perturbation */
    current ← perturb( archive );
return archive;
```

Exploration. The neighbourhood of each selected solution is explored independently, in order to determine new neighbours as candidate solutions to be archived. The parameter `explor-strat` determines whether all or only some neighbours are explored (exhaustive *vs* partial exploration). If the exploration is exhaustive (values `all`, `all_imp`), all improving and non-dominated neighbours, or only all improving neighbours, are added as candidates, respectively. Otherwise, exploration is terminated when a given number (parameter `explor-size`) of either improving (values `imp`, `imp_ndom`) or non-dominated (value `ndom`) neighbours have been found and added as candidates. Non-dominated neighbours evaluated during an `imp_ndom` exploration are also added as candidates. The parameter `explor-ref` corresponds to the reference of the exploration and specifies if the "improving" and "non-dominating" criteria are computed using the current solution (being explored, value `sol`) or the current set of all selected solution (value `arch`).

Archive. When all solutions selected from the archive are explored, the resulting candidates are added to the archive, and Pareto-dominated solutions are filtered out. If the size of the archive exceeds the value of parameter `bound-size`, excess solutions are selected and removed uniformly at random, in order to limit exploration within the same region of the search space.

Perturbation. When the inner termination criterion is met, the neighbourhood exploration is stopped and a perturbation of the current archive is applied, in order to diversify the search. The parameter `perturb-strat` specifies the perturbation mechanism. Either new solutions, generated uniformly at random (value

Table 1: MOLS parameter space (10 920 configurations)

Phase	Parameter	Parameter values
Selection	`select-strat`	{`all`, `rand`, `newest`, `oldest`}
Selection	`select-size`	$\{1, 3, 10\}$
Exploration	`explor-strat`	{`all`, `all_imp`, `imp`, `imp_ndom`, `ndom`}
Exploration	`explor-ref`	{`sol`, `arch`}
Exploration	`explor-size`	$\{1, 3, 10\}$
Archive	`bound-size`	$\{20, 50, 100, 1000\}$
Perturbation	`perturb-strat`	{`kick`, `kick_all`, `restart`}
Perturbation	`perturb-size`	$\{1, 5, 10\}$
Perturbation	`perturb-strength`	$\{3, 5, 10\}$

Selection: $(1 + 3 \times 3)$ combinations ; Exploration: $(1 + 2 + 3 \times 2 \times 3)$; Perturbation: $(3 \times 3 + 3 + 1)$; Total: $10 \times 21 \times 13 \times 4 = 10920$ configurations

`restart`) are considered, or a kick move (values `kick`, `kick_all`) is applied to a given number of solutions (parameter `perturb-size`), or all solutions of the archive, respectively. A solution selected for a kick move is replaced by a solution reached by given number (parameter `perturb-strength`) of search steps, performed sequentially and uniformly at random within the given neighbourhood.

3 Automatic Algorithm Configuration for Multi-Objective Problems

The goal of *automatic algorithm configuration* (AAC) is to automatically determine a configuration (*i.e.*, parameter setting) that optimises the performance of a given algorithm for a given class of problem instances. In this context, we call the algorithm whose parameters are being optimised the *target algorithm* and the procedure that configures the target algorithm a *configurator*.

Algorithm configuration is a machine learning process, whose general concept is illustrated on Figure 2. It involves making a prediction of the optimal configuration of the target algorithm over a training dataset, usually relatively to a given running time or computational budget. The configurations resulting from this training are then re-evaluated on a disjoint test dataset to ensure the unbiasedness of the final prediction.

Given a parameterised target algorithm A, a space Θ of configurations of A, a distribution of instances $\mathcal{D}$, a performance indicator $o : \Theta \times \mathcal{D} \to \mathbb{R}$, and a statistical population parameter E; the algorithm configuration problem consists in optimising the aggregated performance of the target algorithm A across all instances $i \in \mathcal{D}$, as given in Eq. 1 (in which A_θ denotes the algorithm obtained by associating the configuration θ to the target algorithm A).

$$\begin{cases} \text{optimise} & E[o(A_\theta, i),\ i \in \mathcal{D}] \\ \text{subject to} & \theta \in \Theta \end{cases} \tag{1}$$

Algorithm configuration supposes that the limit implied by Eq. 1 exists and is finite. The most commonly used statistical parameter is the simple average of the performance of the target algorithm.

AAC has traditionally been defined as a single-objective optimisation problem, with either running time or solution quality as the optimisation objective. Recently

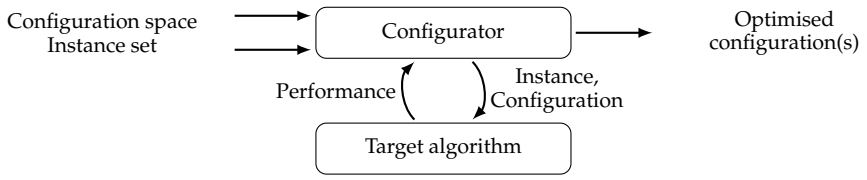

Figure 2: Automatic configuration of a given parameterised target algorithm for a given set problem instances.

(Blot et al., 2016), AAC has been extended to simultaneously deal with multiple performance metrics of a given target algorithm, giving rise to the notion of multi-objective automatic algorithm configuration (MO-AAC). While in single-objective AAC (SO-AAC), the performance of the target algorithm is represented as a scalar value and the output of the configurator is a single configuration, in MO-AAC, target algorithm performance is vector-valued, and the configurator produces a Pareto set of optimised configurations.

In our experiments, we will compare two SO-AAC approaches and one MO-AAC approach optimising the performance of a multi-objective local search algorithm. Specifically, as in previous work, we consider three distinct AAC approaches (Blot et al., 2017b):

***HV*,** a SO-AAC approach that optimises the hypervolume indicator only.

***HV*+Δ′,** a SO-AAC approach that optimises a weighted sum of hypervolume (with a 0.75 coefficient) and Δ' spread (with a 0.25 coefficient).

***HV*||Δ′,** a MO-AAC approach that simultaneously considers hypervolume and Δ' spread.

The latter two approaches are motivated by the previously mentioned belief that the performance assessment of multi-objective algorithms benefits from the use of multiple performance indicators (Zitzler et al., 2003). By comparing HV to the two other configuration approaches, we aim to assess this belief in the context of automatic configuration of MOLS algorithms. Furthermore, by comparing HV+Δ' and $HV||\Delta'$, we intend to assess the benefits of MO-ACC compared to SO-ACC with aggregated performance metrics. The aggregation coefficients (0.75 and 0.25) have been set since the Δ' indicator is a complementary measure to the hypervolume that enables to focus on convergence first and diversity second.

While numerous state-of-the-art SO-AAC procedures can be found in the literature, including irace (López-Ibáñez et al., 2016), based on statistical racing, SMAC (Hutter et al., 2011), based on regression models, and ParamILS (Hutter et al., 2009), based on iterated local search, only few MO-AAC procedures exist, including SPRINT-race (Zhang et al., 2015) and MO-ParamILS (Blot et al., 2016). In the following, we use ParamILS, one of the most widely used SO-AAC procedures, and MO-ParamILS, its recent multi-objective extension. Both configurators are based on iterated local search within the space of valid target algorithm configurations enabling a fair comparison between AAC approaches.

Indeed, SPRINT-race is a very simple and non-iterative multi-objective racing procedure. Its public implementation requires to fully evaluate every possible configura-

tion on every possible instance that it is not admissible in our context.

The core algorithm of MO-ParamILS is given by Algorithm 2. Like its predecessor ParamILS, it is based on an iterated local search (Lourenço et al., 2003; Hoos and Stützle, 2004), in which the incumbents (*i.e.*, the best solutions so far) are iteratively improved by mean of both local search and perturbation mechanisms. Three parameters are exposed: the number r of initial random configurations, a restart probability p_{restart}, and the number s of random search steps performed in each perturbation phase. The `update()` function performs target algorithms runs and ensures that the different configurations can be compared, while the `archive()` function simply discards dominated configurations.

Algorithm 2: Multi-objective ParamILS

Exposed parameters: r, p_{restart} and s
Input: Initial archive of configurations
Output: The archive of incumbents, *i.e.*, the overall best configurations found

```
/* Initialisation */
current_arch ← initial archive;
for i ← 1...r do
    tmp ← random configuration;
    update(tmp, current_arch);
    current_arch ← archive(current_arch, tmp);

/* Iterated local search */
until termination criterion is met do
    /* Perturbation (unless for the first iteration) */
    if first iteration then
        tmp ← current_arch;
    else
        with probability p_restart then // Restart
            /* incumbents are not forgotten */
            current_arch ← { random configuration };
            tmp ← current_arch;
        otherwise // Random walk
            config ← current_config;
            for i ← 1...s do
                config ← random neighbour of tmp;
            tmp ← { config };
    /* Local search */
    tmp ← local_search(tmp);
    foreach config ∈ tmp do
        update(config, current_arch);
        current_arch ← archive(current_arch, config);
return the archive of incumbent;
```

ParamILS and MO-ParamILS start by considering r random configurations, in or-

der to compare the initial (usually default) configuration to a few others to make sure of its relevance. Then, they apply a local search procedure, which is based on the *one-exchange neighbourhood*, *i.e.*, modifying a single parameter value at a time. A tabu mechanism is also used to ensure that the configurator is never stuck. Between iterations, there is a p_{restart} chance to restart the search from a new configuration, uniformly chosen at random from the search space. Otherwise, a perturbation of s random steps is performed. The main difference between ParamILS and MO-ParamILS is that the former focuses on optimising a single configuration with regard to a single performance indicator, while the later optimises an *archive* of configurations, *i.e.*, the set of the current best configurations with regard to the multiple performance indicators.

Specifically, we used the FocusILS variants of both configurators, since these usually give the best performance. For more details on those configurators, we refer the interested reader to Hutter et al. (2009) and Blot et al. (2016).

4 Experimental Setup

In this section, we present the two bi-objective permutation problems we used in our experiments: the permutation flow-shop scheduling problem (bPFSP) and the travelling salesman problem (bTSP). We choose these problems, because they are complementary in the sense that the classical bPFSP naturally involves two strongly correlated objectives, while the classical bTSP is constructed in way that gives rise to independent objectives. We also describe the protocol we use for configuration our multi-objective local search framework for these problems.

4.1 The Bi-Objective Permutation Flow-shop Scheduling Problem (bPFSP)

The permutation flow-shop scheduling problem (PFSP) involves scheduling a set of N jobs $\{J_1, \ldots, J_N\}$ on a set of M machines $\{M_1, \ldots, M_M\}$. Each job J_i is processed sequentially on each of the M machines, with fixed processing times $\{p_{i,1}, \ldots, p_{i,M}\}$, and machines can only process one job at a time. The sequencing of jobs is identical on every machine, so that a solution may be represented by a permutation of size N. The completion times $C_{i,j}$ for each job on each machine for a given permutation $\pi = (\pi_1, \ldots, \pi_n)$ are computed using Eq. 2 to Eq.5.

$$C_{\pi_1,1} := p_{\pi_1,1} \tag{2}$$

$$C_{\pi_1,j} := C_{\pi_1,j-1} + p_{\pi_1,j} \quad \forall j \in \{2, \ldots, m\} \tag{3}$$

$$C_{\pi_i,1} := C_{\pi_{i-1},1} + p_{\pi_i,1} \quad \forall i \in \{2, \ldots, n\} \tag{4}$$

$$C_{\pi_i,j} := \max(C_{\pi_{i-1},j},\ C_{\pi_i,j-1}) + p_{\pi_i,1} \quad \forall i \in \{2, \ldots, n\} \quad \forall j \in \{2, \ldots, m\} \tag{5}$$

The completion time C_k of a job J_k corresponds to its completion time on the last machine $C_{k,m}$. In the following, we consider the bi-objective PFSP (bPFSP), minimising two widely studied objectives, makespan $C_{\max}$ (Eq. 6) and flowtime FT (Eq. 7), where makespan is the total completion time of the schedule, and flowtime is the sum of the individual completion times C_i of the N jobs.

$$C_{\max} := \max_{i \in \{1,\ldots,N\}} C_i \tag{6}$$

$$FT := \sum_{i=1}^{N} C_i \tag{7}$$

Naturally, these two objectives are strongly coupled via the completion times for the individual jobs.

Arguably the most widely studied PFSP are those introduced by Taillard (1993), with numbers of jobs, $N \in \{20, 50, 100, 200, 500\}$ and numbers of machines, $M \in \{5, 10, 20\}$. There are 10 instance for every valid (N, M) combination in Taillard's benchmark set.

In the following, we consider three types of PFSP instance sets, characterised by their number of jobs, $N \in \{50, 100, 200\}$ and a fixed number of machines set to $M = 20$. The higher the number of jobs, the more challenging the instances tend to be. For the exhaustive analysis and the test phase of the three AAC approaches, we use the 10 Taillard instances for each of those N values. For the training phase of the AAC approaches, we used a different, completely disjoint, set of instances, composed by newly generated Taillard-like instances. We generated 30 of these instances for each $M \in \{50, 100, 200\}$, using the original generation procedure.

When running MOLS on bPFSP instances, we initialise the search using the 2-phase local search algorithm by Dubois-Lacoste et al. (2011), which is based an iterated greedy (IG) procedure (Ruiz and Stützle, 2007). This method is known to produce relatively good and well-distributed solutions sets. We use 25% of the overall time budget for this initialisation, and 75% for the remainder of each MOLS run. Classical PFSP neighbourhoods include the exchange neighbourhood, where the positions of two jobs are exchanged, and the insertion neighbourhood, where one job is reinserted at another position in the permutation. In this study, we consider a hybrid neighbourhood defined as the union of the exchange and insertion neighbourhoods, which is known to lead to better performance than considering each neighbourhood independently (Dubois-Lacoste et al., 2015).

4.2 The Bi-Objective Travelling Salesman Problem (bTSP)

The Travelling Salesman Problem (TSP) can be defined by a complete weighted graph G whose nodes represent cities, while edges corresponds to direct paths between cities. In the symmetric TSP, the graph is undirected, and edge weights correspond to distances between cities. Given a TSP instance G, the goal is to determine a round trip (or tour) passing through every city exactly once such that the overall distance travelled is minimised, *i.e.*, a minimum-weight Hamiltonian cycle in G. We note that any Hamiltonian cycle in G corresponds to a permutation of the cities. The TSP is one of the most widely studied combinatorial optimisation problems.

In the following, we consider the bi-objective symmetric TSP (bTSP), in which each instance is defined by a graph G as in the standard TSP, but each edge between two nodes i and j have two weights, $c^1_{i,j}$ and $c^2_{i,j}$. We now define the two objectives as the total distance covered by a given round trip according to each of the two distance matrices $(d^1)_{i,j}$ and $(d^2)_{i,j}$. Both objectives have to be minimised.

A benchmark set of Euclidean instances (available online[1]) has been widely used in the literature to assess the performance of bTSP algorithms (Paquete et al., 2004; Lust and Teghem, 2010; Dubois-Lacoste et al., 2015). These instances were constructed by combining two independently generated distance matrices, computed using Euclidean distance between cities randomly placed on a two-dimension grid; therefore, unlike in the case of the bPFSP discussed earlier, the two optimisation objectives are uncorrelated. The bTSP benchmark set contains six instances each for 100, 300 and 500 cities, meaning 15 different pairwise independent combinations of two instances per benchmark size. For our exhaustive analysis and the test phase of the three AAC approaches, we used these 15 instances. For the training phase of the AAC approaches, we used a

[1] `https://eden.dei.uc.pt/~paquete/tsp`

different, completely disjoint, set of newly generated instances containing 30 instances for each number of cities, obtained using the original generator from the DIMACS challenge.

Each run of MOLS is initialised using the well-known nearest neighbour heuristic applied to only one of the two objectives, resulting in two greedily constructed tours. Subsequently, we perform local search steps in the classical *two-opt* neighbourhood, according to which the neighbours of a given tour are obtained by removing two non-adjacent edges and reconnecting the resulting fragments by two new edges such that a different tour is obtained (Croes, 1958).

4.3 AAC Protocol

Our MO-ParamILS multi-objective algorithm configuration protocol proceeds in three phases: *training*, *validation*, and *test*.

Training: In the training phase, MO-ParamILS is independently run multiple times on a given training set of instances; each of these runs produces a Pareto set of configurations. Multiple runs of MO-ParamILS are used, because individual runs can suffer from stagnation and to make effective use of parallel computing resources. However, as different MO-ParamILS runs usually use different subsets of the given training set, the configurations obtained from them cannot be fairly compared to each other.

Validation: To fairly compare configurations obtained from different MO-ParamILS runs and to reduce the number of configurations ultimately evaluated in the test phase, every configuration from the training phase is evaluated on the same, fixed subset of training instances. Based on the performance measurements thus obtained, Pareto-dominated configurations are removed.

Test: The Pareto set of configurations obtained from the validation phase is evaluated again, on a set of test instances that does not contain any of the instances used for training or validation. Again, Pareto-dominated configurations are removed.

Using this protocol, we compare the three AAC approaches described in Section 3. In the case of the HV approach, ParamILS is used during the training phase to only optimise the hypervolume of the target algorithm, since the configuration scenario is single-objective. However, during the subsequent validation and test phases, the configurations resulting from the single-objective training are assessed using both hypervolume and Δ' spread separately, in a Pareto way. Ultimately, it is expected that this approach only finds good hypervolume values and disregards the Δ' spread value. Similarly, the HV+Δ' approach uses ParamILS during the training phase to optimise an aggregation of hypervolume and Δ' spread, while the assessment of the validation and test phases are then performed in a Pareto way.

These two SO-AAC approaches constitute the baseline against which our MO-AAC approach is compared to. While they focus on specific directions of the multi-objective space and use an optimisation criterion less complete than the final evaluation, they represent the performance that our multi-objective approach will have to at least match. We note that this makes effective use of an off-the-shelf SO-AAC procedure, while taking into account the multi-objective nature of the algorithm configuration problem we are ultimately trying to solve.

Table 2: Small version of the MOLS parameter space (300 configurations)

Phase	Parameter	Parameter values
Selection	`select-strat`	{`all`, `rand`, `oldest`}
Selection	`select-size`	$\{1, 10\}$
Exploration	`explor-strat`	{`imp`, `imp_ndom`, `ndom`}
Exploration	`explor-ref`	{`sol`, `arch`}
Exploration	`explor-size`	$\{1, 10\}$
Archive	`bound-size`	$\{1000\}$
Perturbation	`perturb-strat`	{`kick`, `kick_all`, `restart`}
Perturbation	`perturb-size`	$\{10\}$
Perturbation	`perturb-strength`	$\{3, 10\}$

Selection: $(1 + 2 \times 2)$ combinations ; Exploration: $(3 \times 2 \times 2)$; Perturbation: $(2 \times 2 + 1)$; Total: $5 \times 12 \times 5 = 300$ configurations

In the following, this protocol is used in conjunction with two different configurations spaces (in the training phase). First, we consider a *small* configuration space of 300 configurations, described in Table 2. This smaller configuration space has been defined based on preliminary experiments, in which we informally identified parameters and parameter values most likely to result in good performance of our target algorithm framework. The small size of this search space permits us to perform validation and testing in an exhaustive way, where the performance of each of the 300 configurations is assessed on the entire training and test instance sets, respectively. This exhaustive analysis of the configuration space enables the comparison of the configurations resulting from the training phase to ones that may otherwise be never considered. As stated previously, our goal with this analysis is to demonstrate that our AAC approaches can effectively configure a multi-objective local search for solving bi-objective permutation problems.

The second configuration space, detailed in Table 1 and already presented in Section 2, enables the comparison between our three AAC approaches in a much richer space with $10,920$ configurations. We note that, because of the relatively high running times for each configuration and the stochastic nature of our target algorithm, searching this space exhaustively would require a computational budget orders of magnitude beyond that used for the experiments described in the following.

Table 3 summarises the details of our AAC protocol for both configuration spaces. The main differences concern the training phase. For the small (large) configuration space, ParamILS starts by evaluating a single (10) random configuration, and can execute 100 (1000) MOLS runs before stopping, where each selected configuration cannot be run more than 10 (100) times. Due to the reduced size of the small configuration space, only 10 independent runs of ParamILS are performed, compared to 20 runs for the large space. In the validation phase, the configurations resulting from the training phase are evaluated on all training instances, running every configuration once on each instance. In the test phase, each of the configurations in the Pareto set obtained from the validation phase is run 10 times on every test instance. For both validation and test phases, the performance of each configuration is assessed based on the average hypervolume and spread values over the runs. Obviously, for the small configuration space, our exhaustive analysis ensures that the performance of all configurations are known

Table 3: AAC Experimental Protocol

Phase	Small configuration space	Large configuration space
Training	No default configuration	No default configuration
	1 random configuration	10 random configurations
	10 ParamILS runs	20 ParamILS runs
	100 MOLS runs budget	1000 MOLS runs budget
	max 10 MOLS run per config.	max 100 MOLS run per config.
Validation	1 run per instance	1 run per instance
Test	10 runs per instance	10 runs per instance

Table 4: CPU running time

Scenario	1 MOLS run	100 MOLS runs	1000 MOLS runs
bPFSP 50	50 seconds	1.39 hours	13.89 hours
bPFSP 100	3.33 minutes	5.56 hours	2.31 days
bPFSP 200	13.33 minutes	22.22 hours	9.26 days
bTSP 100	1.5 minutes	2.5 hours	1.04 days
bTSP 300	4.5 minutes	7.5 hours	3.12 days
bTSP 500	7.5 minutes	12.5 hours	5.21 days

for all training and test instances, and we will directly use these results in the validation and test phases to avoid recomputing the performance of configurations selected in the training phase.

Table 4 details the CPU running time for a single run, 100 runs, and 1000 runs of a MOLS algorithm. The last two columns correspond to the running time of a single run of (MO-)ParamILS on the small and large configuration spaces, without taking into account the slight system time overhead induced by launching the MOLS executable. Note that to obtain the total training time, theses durations should be multiplied by the number of (MO-)ParamILS runs (10 and 20, respectively). Moreover, the time required for the validation and test steps is not pointed out since it depends on the number of configurations returned by the training step.

Finally, Table 5 reports the bounds used for each scenario to compute the aggregation in the case of the $HV{+}\Delta$ approach. These bounds have been determined using preliminary data from the exhaustive analysis on the training sets of instances.

5 Experimental Results

We now present and discuss the results from our computational experiments, first for the small configuration space, and then for the large configuration space. All experiments have been conducted in parallel on the *grace* cluster of the ADA research group at the Leiden Institute of Advance Computer Science (LIACS). Each of the 32 nodes of *grace* is equipped with two 16-core 2.10GHz Intel Xeon E5-2683 v4 CPUs with 40 MB L3 cache and 94 GB RAM, running CentOS 7.4.1708.

Table 5: Indicator bounds used in the HV+Δ approach.

Scenario	(1-HV) lower	(1-HV) upper	Δ lower	Δ upper
bPFSP 50	0.48	0.5	0.2	1
bPFSP 100	0.44	0.46	0.1	1.1
bPFSP 200	0.355	0.375	0.3	1.3
bTSP 100	0.13	0.24	0.6	1.7
bTSP 300	0.09	0.2	0	2
bTSP 500	0.08	0.18	0	2

5.1 Small Configuration Space

Here, we compare our three AAC approaches (HV, HV+Δ' and $HV||\Delta'$) to each other and in relation to the results of our exhaustive analysis made possible by the size of the small configuration space. We recall that we transform the hypervolume (HV) into a minimisation measure ($1 - HV$) to simplify the analysis of our results, and thus, when speaking of good hypervolume values, we refer to high HV (*i.e.*, low values of $1-HV$).

The results from this analysis for training and test instance sets are shown in Figures 3 and 4. Generally, the shapes of the Pareto sets in objective space are similar between validation and test results, indicating that our AAC approaches do not suffer from over-fitting. Therefore, we will focus our discussion on the test results seen in Figure 4. Table 6 details how many unique (in parentheses: non-unique) configurations were found by each AAC approach, and how many of these survived the validation and test phases. (Recall that Pareto-dominated configurations are pruned in those phases). Figure 5 shows the parameter distribution of the 300 configurations on test instances according to our three selection mechanisms (crosses: $+ \times \star$, polygons: $\square\Delta\diamond$, and circles: $\circ\oplus\otimes$) and our three exploration strategies (red: $+\square\circ$, green: $\times\Delta\oplus$, and blue: $\star \diamond \otimes$). Finally, Tables 7 to 12 list the Pareto-optimal configurations within the small configuration space. A "$*$" symbol indicates that the value of the respective parameter does not impact the performance of the configured MOLS when the other parameter values are held fixed at the values shown. Conversely, when a specific parameter is shown, any deviation from it will reduce performance.

First, we will discuss the results for the bPFSP. None among the 300 possible configurations simultaneously achieves good hypervolume and spread values (see Figure 4); the Pareto front is distinctly non-convex. While for the smallest scenario with 50 jobs, most of all configurations achieve good hypervolume values (*i.e.*, low $1 - HV$), such configurations get rarer as the number of jobs increases. This result was expected, since it is known that larger bPFSP instance are harder for multi-objective local search. Examining these results in more detail, we observe that the `imp` exploration strategy always obtains rather bad hypervolume values (see Figure 5). For 50 jobs, this strategy leads to better spread values; however, this tends to be no longer true for larger instances. For the three instance sizes, the `imp-ndom` and `ndom` strategies appear to give better performance in terms of hypervolume.

All three approaches find very good, even near-optimal configurations – in particular, $HV||\Delta'$, which achieves spreads over the entire Pareto-front. The 10 configurator runs of HV and HV+Δ' produce close to 10 unique configurations each (see

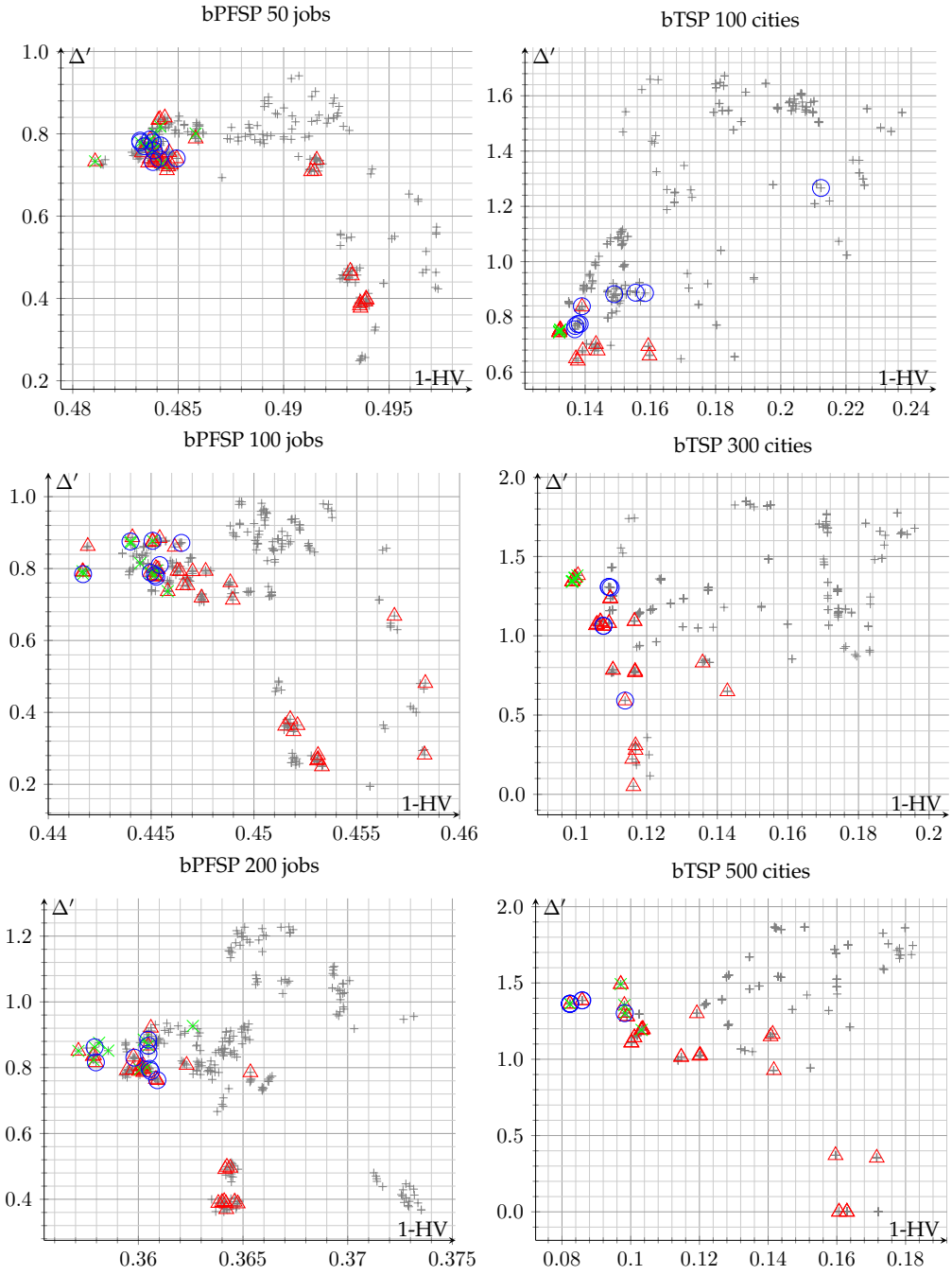

Figure 3: Exhaustive analysis on training instances (left: bPFSP; right: bTSP)
x: HV approach, o: HV+Δ' approach, Δ: $HV||\Delta'$ approach, $+$: exhaustive analysis

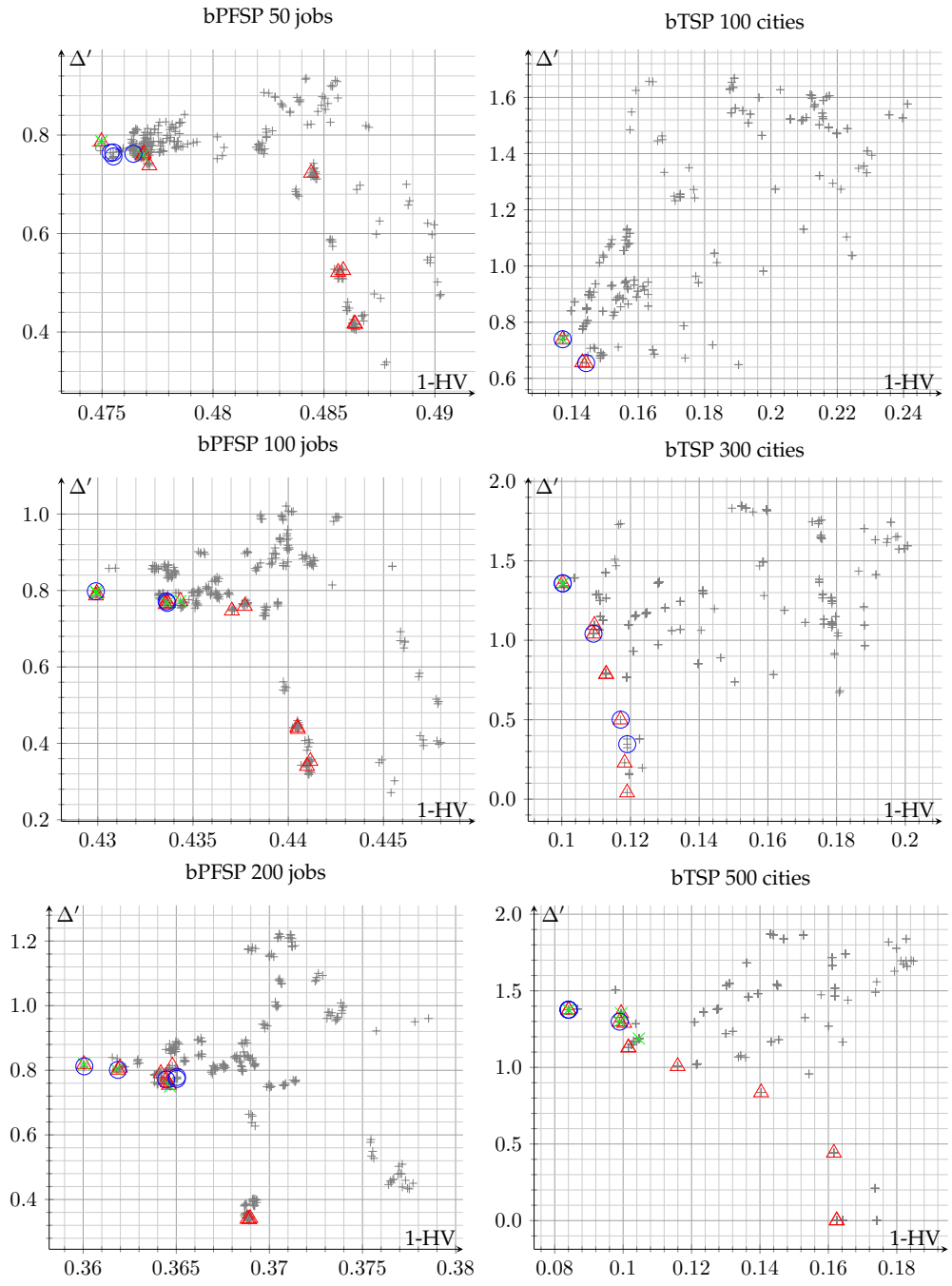

Figure 4: Exhaustive analysis on test instances (left: bPFSP; right: bTSP)
x: HV approach, o: HV+Δ' approach, Δ: $HV||\Delta'$ approach, $+$: exhaustive analysis

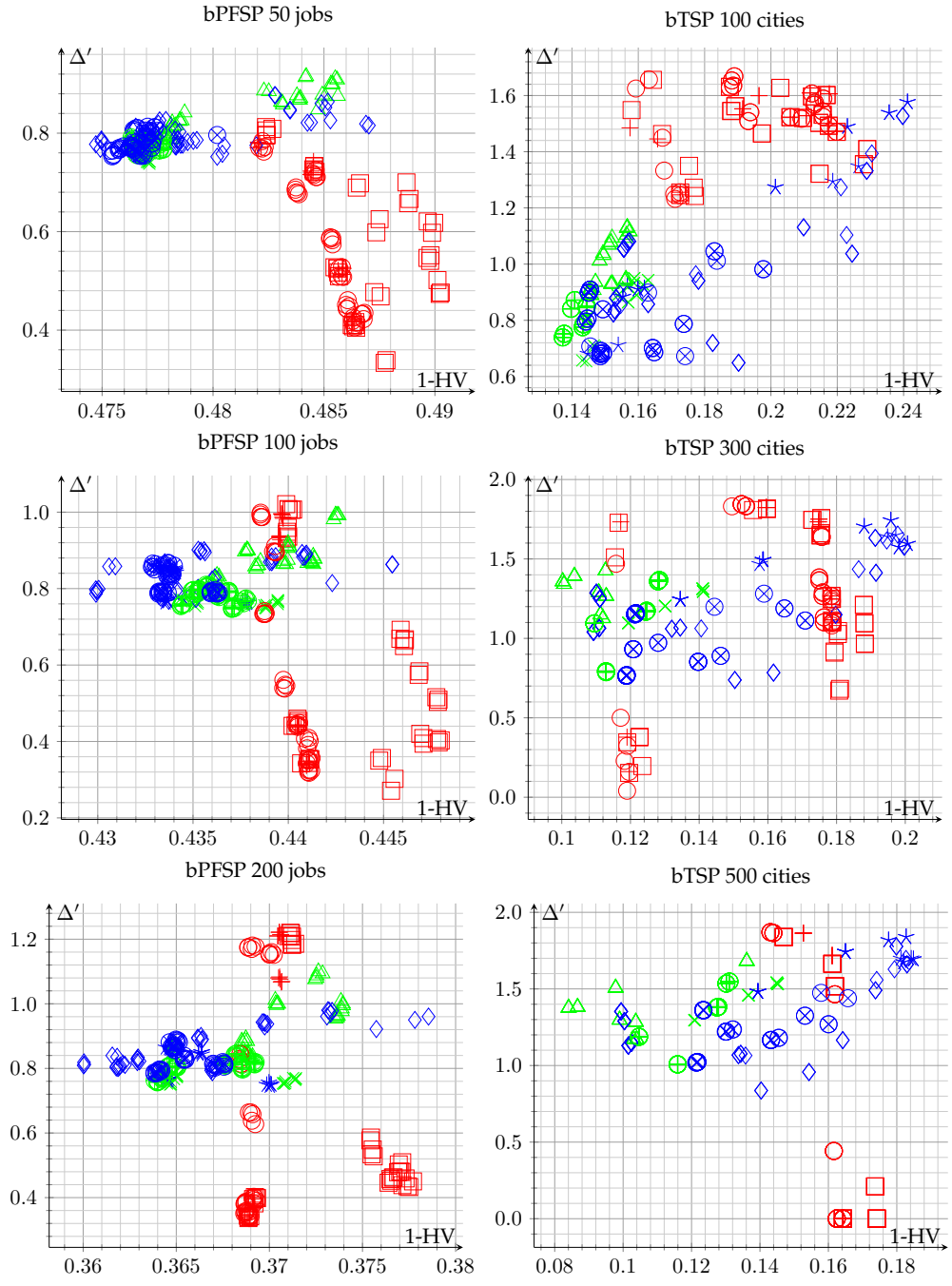

Figure 5: Exhaustive analysis parameter distribution on test instances (left: bPFSP; right: bTSP); Selection strategy: +×⋆: `all` (crosses), □△◇: `oldest` (polygons), ○⊕⊗: `rand` (circles); Exploration strategy: +□○: `imp` (red), ×△⊕: `imp_ndom` (green), ⋆◇⊗: `ndom` (blue)

Table 6: Number of configurations after training, validation and testing

Scenario	Approach	Small space			Large space		
		Configs	Pareto	Final	Configs	Pareto	Final
bPFSP 50	HV	10	2	2	20	2	2
	HV+Δ'	10	4	2	10	2	2
	$HV\|\|\Delta'$	32 (38)	9	7	145	14	11
bPFSP 100	HV	10	4	3	19 (20)	1	1
	HV+Δ'	8 (10)	3	3	20	4	2
	$HV\|\|\Delta'$	36 (42)	12	6	171 (172)	27	19
bPFSP 200	HV	10	3	3	20	4	3
	HV+Δ'	9 (10)	5	3	16 (20)	2	2
	$HV\|\|\Delta'$	29 (39)	11	8	111 (117)	14	9
bTSP 100	HV	6 (10)	1	1	15 (20)	2	1
	HV+Δ'	6 (10)	2	2	15 (20)	6	4
	$HV\|\|\Delta'$	16 (26)	3	3	62 (73)	11	5
bTSP 300	HV	9 (10)	2	2	13 (20)	4	2
	HV+Δ'	9 (10)	5	5	12 (20)	5	2
	$HV\|\|\Delta'$	33 (41)	8	6	107 (130)	18	12
bTSP 500	HV	6 (10)	5	4	16 (20)	4	4
	HV+Δ'	8 (10)	5	4	14 (20)	3	2
	$HV\|\|\Delta'$	36 (40)	12	11	135 (145)	25	22

Table 6), and all of these show good hypervolume values. However, after validation and testing, for both AAC approaches, few configurations remain, and those tend to have good hypervolume but average spread. On the other hand, our MO-AAC approach, $HV||\Delta'$, produces many more configurations after the training, validation and test phases. Compared to the other approaches, $HV||\Delta'$ clearly achieves better coverage of the optimal Pareto set of configurations (Figure 4). Note that all three approaches use the same time budget for configuration, the number of final solutions being strongly dependant of the kind (single-objective or multi-objective) of AAC used for training.

Regarding the nature of the configurations, we observe a trend across the three instance sizes (Tables 7 to 9): The best hypervolume is always reached with the `oldest` selection strategy, the `ndom` exploration strategy and the `arch` exploration reference set choice. Slightly worse hypervolume, but better spread is achieved using the `imp_ndom` exploration strategy. Finally, the best spread values are obtained from configurations using the `imp` exploration strategy, although this comes at the cost of rather bad hypervolume. In almost every case, the perturbation strategy did not significantly impact the performance of the non-dominated configurations.

Our results on the bTSP differ markedly from those on the bPFSP. Firstly, we observe that the shape of the Pareto-optimal front of configurations varies with instance size: While it is convex for 100 cities with some degree of correlation between hypervolume and spread, for larger instances, the correlation between the two performance indicators decreases, and the front becomes non-convex. In contrast to the bPFSP, where the two objectives were correlated, for our bTSP benchmark sets, the objectives are

Table 7: 50-job bPFSP optimal configurations (small space)

1-HV	Δ'	Selection		Exploration			Archive	Perturbation		
0.4747	0.7775	oldest	10	ndom	arch	1	1000	$*$	10	$*$
0.4754	0.7640	all		ndom	arch	1	1000	$*$	10	$*$
0.4770	0.7420	all		imp_ndom	sol	10	1000	$*$	10	$*$
0.4837	0.6798	rand	1	imp	arch	10	1000	$*$	10	$*$
0.4853	0.5856	rand	1	imp	sol	10	1000	$*$	10	$*$
0.4855	0.5277	$*$	10	imp	arch	1	1000	$*$	10	$*$
0.4860	0.4433	rand	1	imp	arch	1	1000	$*$	10	$*$
0.4862	0.4093	$*$		imp	sol	1	1000	$*$	10	$*$
0.4877	0.3336	oldest	1	imp	sol	1	1000	kick	$*$	10

Table 8: 100-job bPFSP Pareto-optimal configurations (small space)

1-HV	Δ'	Selection		Exploration			Archive	Perturbation		
0.4299	0.7865	oldest	10	ndom	arch	1	1000	kick	10	3
0.4299	0.7979	oldest	10	ndom	arch	1	1000	kick_all		$*$
0.4332	0.7802	oldest	1	ndom	arch	1	1000	kick	10	$*$
0.4336	0.7640	all		ndom	arch	1	1000	$*$	10	$*$
0.4344	0.7541	rand	10	imp_ndom	arch	1	1000	$*$	10	$*$
0.4351	0.7540	all		imp_ndom	sol	1	1000	$*$	10	$*$
0.4370	0.7470	rand	10	imp_ndom	arch	10	1000	$*$	10	$*$
0.4387	0.7338	rand	1	imp	arch	10	1000	$*$	10	$*$
0.4397	0.5396	rand	1	imp	sol	10	1000	$*$	10	$*$
0.4402	0.4409	$*$	10	imp	arch	1	1000	$*$	10	$*$
0.4407	0.3428	oldest	10	imp	sol	1	1000	$*$	10	$*$
0.4410	0.3201	rand	1	imp	sol	1	1000	$*$	10	$*$
0.4410	0.3371	all		imp	sol	1	1000	$*$	10	$*$
0.4454	0.2711	oldest	1	imp	sol	1	1000	kick	10	$*$

Table 9: 200-job bPFSP Pareto-optimal configurations (small space)

1-HV	Δ'	Selection		Exploration			Archive	Perturbation		
0.3600	0.8093	oldest	1	ndom	arch	1	1000	restart/kick	10	$*$
0.3618	0.8027	oldest	10	ndom	arch	1	1000	$*$	10	$*$
0.3638	0.7628	rand	1	imp_ndom	arch	1	1000	$*$	10	$*$
0.3645	0.7534	all		imp_ndom	arch	1	1000	$*$	10	$*$
0.3686	0.3511	rand	1	imp	sol	1	1000	$*$	10	$*$
0.3687	0.3456	$*$	10	imp	sol	1	1000	$*$	10	$*$

Table 10: 100-city bTSP Pareto-optimal configurations (small space)

1-HV	Δ'	Selection		Exploration			Archive	Perturbation		
0.1372	0.7389	rand	10	imp_ndom	sol	10	1000	$*$	10	$*$
0.1431	0.6572	all		imp_ndom	sol	10	1000	restart		
0.1443	0.6544	all		imp_ndom	arch	10	1000	restart		
0.1902	0.6488	oldest	1	ndom	sol	1	1000	kick	10	3

Table 11: 300-city bTSP Pareto-optimal configurations (small space)

1-HV	Δ'	Selection		Exploration			Archive	Perturbation		
0.1003	1.3582	oldest	10	imp_ndom	sol	1	1000	$*$	10	$*$
0.1006	1.3417	oldest	10	imp_ndom	sol	10	1000	$*$	10	$*$
0.1092	1.0409	oldest	10	ndom	sol	10	1000	$*$	10	$*$
0.1128	0.7933	rand	10	imp_ndom	arch	1	1000	$*$	10	$*$
0.1129	0.7880	rand	1	imp_ndom	arch	1	1000	$*$	10	$*$
0.1171	0.5003	rand	1	imp	sol	10	1000	restart		
0.1183	0.2288	rand	1	imp	sol	1	1000	restart		
0.1190	0.0409	rand	1	imp	arch	1	1000	restart		

completely independent; therefore, the final archives are much bigger, as there exist a richer space of trade-off solutions. The impact on spread is evident from Figure 4; values above 1 correspond to two tightly clustered sets of solutions separated by a large gap that the respective configuration of MOLS failed to cover, and spread values of 0 correspond to final sets containing only two solutions, which are produced when the `imp` exploration strategy fails to sufficiently diversify.

Our HV configuration approach produced few configurations, achieving near-optimal hypervolume. HV+Δ' produced weak training results on the 100-city instances, but worked well on the 300-city instances, because of the shape of the Pareto-optimal front. As for the bPFSP, $HV||\Delta'$ found many more configurations and achieved far better coverage of the Pareto front. In our test instances from the literature, all three AAC approaches produced optimal configurations for 100-city instances, HV+Δ' and $HV||\Delta'$ still did on 300-city instances, and only $HV||\Delta'$ managed to find most of the optimal configurations on the 500-city instance (see Figure 4).

Analysing the MOLS configurations in more detail, those that achieve the best hypervolume values always use the `imp_ndom` exploration strategy with the `sol` reference set. While for 300- and 500-city instances, the `oldest` selection strategy is preferred, for 100 cities, the more common `rand` selection strategy performs better. Similarly to the bPFSP, the choice of perturbation mechanism does not significantly impact the performance of optimal configurations.

For both problems, within the small configuration space, all three AAC approaches are able to find configurations very close to the true Pareto-front. The two SO-AAC approaches strongly favours the hypervolume indicator, while the MO-AAC approach is able to accurately cover the full range of Pareto-optimal configurations.

5.2 Large Configuration Space

In this section, our three AAC approaches are now tested against a much larger search space, in order to study their scalability.

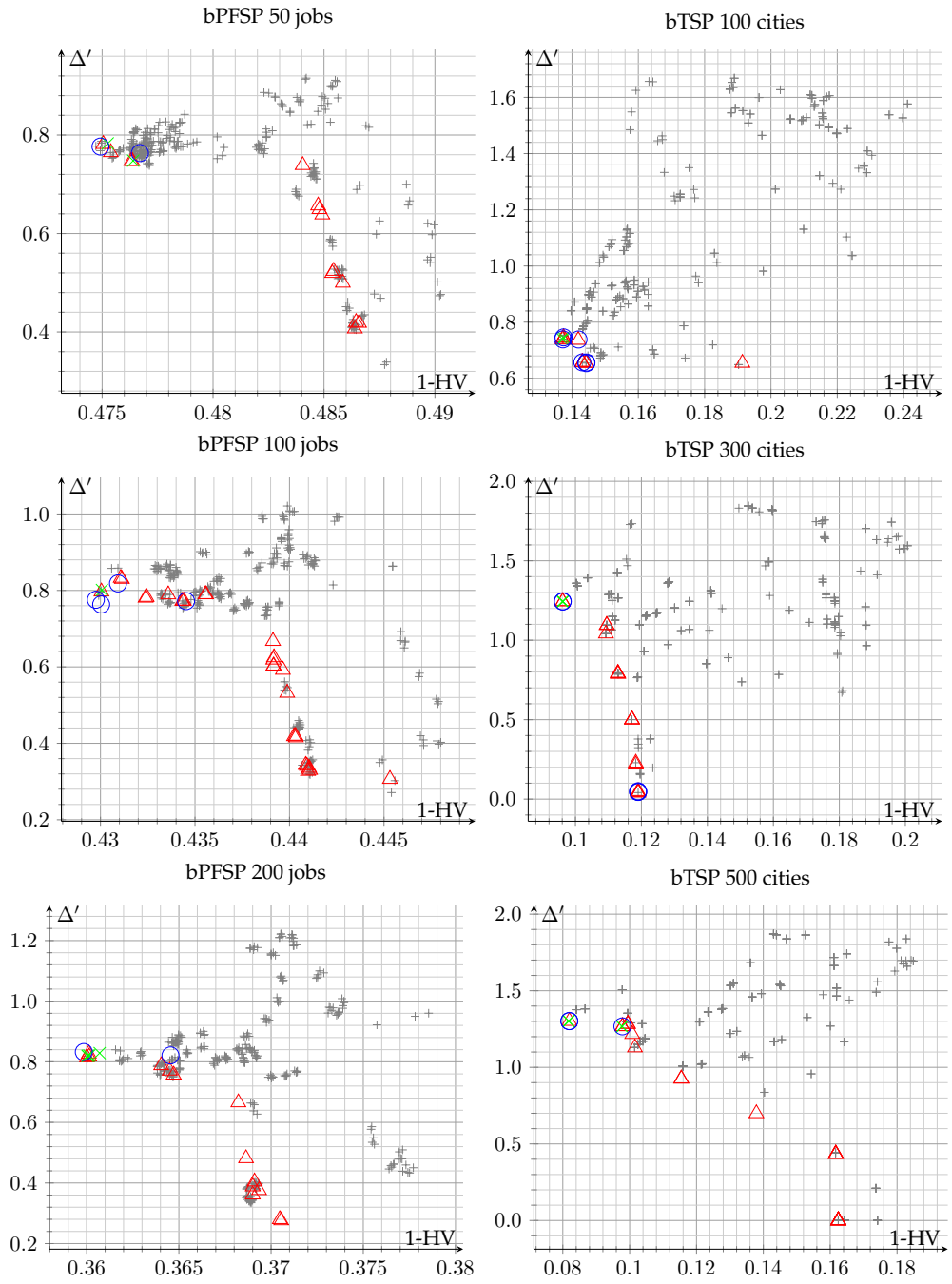

Figure 6: Large-scale analysis on test instances (left: bPFSP; right: bTSP)
x: HV approach, o: HV+Δ' approach, Δ: $HV||\Delta'$ approach, +: exhaustive analysis

Table 12: 500-city bTSP Pareto-optimal configurations (small space)

1-HV	Δ'	Selection		Exploration			Archive	Perturbation		
0.0841	1.3767	oldest	10	imp_ndom	sol	1	1000	$*$	10	$*$
0.0989	1.2983	oldest	1	imp_ndom	arch	10	1000	$*$	10	$*$
0.1003	1.2897	oldest	10	ndom	arch	10	1000	$*$	10	$*$
0.1015	1.1290	oldest	10	ndom	sol	10	1000	$*$	10	$*$
0.1159	1.0080	rand	$*$	imp_ndom	arch	1	1000	$*$	10	$*$
0.1403	0.8468	oldest	10	ndom	arch	1	1000	kick	10	$*$
0.1616	0.4420	rand	1	imp	sol	10	1000	$*$	10	$*$
0.1624	0.0000	rand	1	imp	$*$	1	1000	$*$	10	$*$

Figure 6 shows the final configurations produced by all three AAC approaches for our six benchmarks (two problems, three instance sizes). We also show the configurations of the smaller set of configurations that we exhaustively evaluated in Section 5.1, in order to show that these final configurations map very closely those found within the small space, which suggests that the small space indeed captures the high-performance configurations from the much larger space and, more importantly, demonstrates that our AAC approaches effectively finds such configurations. In the following, we will focus on the performance of our three AAC approaches.

Both SO-AAC approaches, HV and HV+Δ', produced few non-dominated configurations in their final testing phase – typically between 2 and 4 on each instance set (see Table 6). As one might expect, HV always finds a final configuration with near-optimal hypervolume. The results for HV+Δ' are similar to those for HV for the bPFSP, but markedly different on our bTSP benchmarks. For 100-city bTSP instances, HV+Δ' covers the Pareto front, while for 300 cities, it finds the two extreme configurations, due to accidentally well-adapted weights used for aggregating hypervolume and spread. However, due to the non-convex shape of the front, no trade-off configurations are found between these extremes. For 500-city instances, HV+Δ' only finds configurations with near-optimal hypervolume, similar to what we observed for the bPFSP.

On the other hand, our MO-AAC approach, $HV||\Delta'$, consistently provides many more non-dominated configurations, except for the small 100-city bTSP instance set, where the Pareto front is completely covered by all three approaches. In all cases, the sets of configurations found by $HV||\Delta'$ are very well distributed over the entire front of optimal configurations. Although HV+Δ' sometimes finds better configurations (*e.g.*, on the 100- and 200-jobs bPFSP scenarios), $HV||\Delta'$ always produces configurations with similar performance.

Overall, our MO-AAC approach, $HV||\Delta'$, produces substantially better results than the two SO-AAC approaches, HV and HV+Δ'. HV finds excellent sets of configurations with respect to hypervolume, but only provides very few of those and consequently fails to achieve good spread. HV+Δ' sometimes provides better results and, under favourable circumstances, can cover the entire set of Pareto-optimal configurations; however, especially for more challenging scenarios, its performance is similar to that of HV. The main drawback of this approach is the requirement of a costly preliminary step for calibrating the weights used for aggregating the two optimisation

objectives. Finally, $HV||\Delta'$, our *native* MO-AAC approach, always efficiently covers the entire Pareto-front of configurations, while still finding sets of configurations with excellent hypervolume, as produced by the two SO-AAC approaches.

6 Conclusion

In this article, we have studied the question how to best approach the automated configuration of multi-objective optimisation algorithms. From the literature, it is known that no single metric completely characterises the performance of MOO algorithms, and that therefore, it is best to evaluate such algorithms using multiple complementary performance metrics. Consistent with this insight, we have studied the use of multi-objective configuration techniques for MOO algorithms, contrasting two approaches built around a standard, single-objective configuration procedure with one using a natively multi-objective algorithm configurator. Substantially expanding on our own, preliminary work in this area, we have conducted extensive experiments with a highly-parametric local search framework, MOLS, applied to two well-known MOO problems, the bi-objective permutation flowshop and travelling salesman problems. We have found strong evidence that automated configuration of highly-parametric MOO frameworks such as ours is effectively possible using existing techniques and, more importantly, that indeed, by using a natively multi-objective algorithm configurator – here, MO-ParamILS (Blot et al., 2016) – the best configuration results can be achieved. Specifically, using MO-ParamILS, we could consistently obtain large and diverse sets of configurations of our MOLS framework.

In future work, we intend to study how the degree of correlation between the objectives of a given MOO problem impacts the performance of our automated algorithm configuration approaches and that of the configurations thus obtained. Our results presented here already provide some evidence for qualitative differences arising from very high and very low degrees of correlation, as found in the bi-objective PFSP and TSP, respectively, but it is yet to be determined to which degree these findings are independent of other differences between these benchmark problems.

We also believe that it would be very interesting to expand the investigation presented here to other types of MOO problems – notably, to problems from machine learning and data mining, whose design and calibration usually gives rise to trade-offs between several performance objectives, such as sensitivity and specificity, thus providing a new direction for the area of automated machine learning (AutoML).

References

Blot, A., Hoos, H. H., Jourdan, L., Marmion, M.-É., and Trautmann, H. (2016). MO-ParamILS: A multi-objective automatic algorithm configuration framework. In *Proceedings of the Learning and Intelligent Optimization 10 Conference*, pages 32–47.

Blot, A., Jourdan, L., and Kessaci-Marmion, M.-É. (2017a). Automatic design of multi-objective local search algorithms: case study on a bi-objective permutation flowshop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation 2017 Conference*, pages 227–234.

Blot, A., Pernet, A., Jourdan, L., Kessaci-Marmion, M.-É., and Hoos, H. H. (2017b). Automatically configuring multi-objective local search using multi-objective optimisation. In *Proceedings of the Evolutionary Multi-Criterion Optimization 2017 Conference*, pages 61–76.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2011). A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236.

Dubois-Lacoste, J., López-Ibáñez, M., and Stützle, T. (2015). Anytime pareto local search. *European Journal of Operational Research*, 243(2):369–385.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proceedings of the Learning and Intelligent Optimization 5 Conference*, pages 507–523.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

Hutter, F., Hoos, H. H., and Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1152–1157.

Knowles, J. and Corne, D. (2002). On metrics for comparing nondominated sets. In *IEEE Congress on Evolutionary Computation*, volume 1, pages 711–716.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

López-Ibáñez, M. and Stützle, T. (2012). The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875.

Lourenço, H., Martin, O., and Stützle, T. (2003). Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer.

Lust, T. and Teghem, J. (2010). Two-phase pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510.

Murata, T. and Ishibuchi, H. (1998). A multi-objectives genetic local search algorithm and its application flow-shop scheduling. *IEEE Transaction System*, 28(3):392–403.

Okabe, T., Jin, Y., and Sendhoff, B. (2003). A critical survey of performance indices for multi-objective optimisation. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 878–885.

Paquete, L., Chiarandini, M., and Stützle, T. (2004). Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, pages 177–199. Springer.

Riquelme, N., von Lücken, C., and Barán, B. (2015). Performance metrics in multi-objective optimization. In *Proceedings of the 2015 Latin American Computing Conference*, pages 1–11.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Zhang, T., Georgiopoulos, M., and Anagnostopoulos, G. C. (2015). SPRINT multi-objective model racing. In *Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, pages 1383–1390.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.