Automatically Configuring Multi-objective Local Search Using Multi-objective Optimisation

Aymeric Blot, Alexis Pernet, Laetitia Jourdan, Marie-Éléonore Kessaci-Marmion, Holger H Hoos

Automatically Configuring Multi-objective Local Search using Multi-objective Optimisation

Aymeric Blot[1], Alexis Pernet[1], Laetitia Jourdan[1],
Marie-Éléonore Kessaci-Marmion[1], and Holger H. Hoos[2]

[1] Université de Lille, Inria, CNRS, UMR 9189 – CRIStAL, France
[2] University of British Columbia, Canada & Universiteit Leiden, The Netherlands
aymeric.blot@inria.fr
alexis.pernet@etudiant.univ-lille1.fr
{laetitia.jourdan,me.kessaci}@univ-lille1.fr
hoos@cs.ubc.ca

Abstract. Automatic algorithm configuration (AAC) is becoming an increasingly crucial component in the design of high-performance solvers for many challenging combinatorial optimisation problems. This raises the question how to most effectively leverage AAC in the context of building or optimising multi-objective optimisation algorithms, and specifically, multi-objective local search procedures. Because the performance of multi-objective optimisation algorithms cannot be fully characterised by a single performance indicator, we believe that AAC for multi-objective local search should make use of multi-objective configuration procedures. We test this belief by using MO-ParamILS to automatically configure a highly parametric iterated local search framework for the classical and widely studied bi-objective permutation flowshop problem. To the best of our knowledge, this is the first time a multi-objective optimisation algorithm is automatically configured in a multi-objective fashion, and our results demonstrate that this approach can produce very good results as well as interesting insights into the efficacy of various strategies and components of a flexible multi-objective local search framework.

Keywords: Algorithm Configuration, Multi-Objective Optimisation, Local Search, Permutation Flowshop Scheduling

1 Introduction

The performance of many single- and multi-objective optimisation methods strongly depends on the setting of their parameters. For most classes of problem instances, to achieve good performance, the values of these parameters must be specifically optimised. Thus, it is becoming increasingly common practice to use automatic algorithm configuration (AAC) procedures, such as irace [14], ParamILS [11] or SMAC [10]. While these configurators optimise a single performance metric only, such as solution quality or running time of a given target algorithm, recently, multi-objective AAC procedures, such as MO-ParamILS [2],

have become available and shown to be effective for optimising multiple performance metric simultaneously, using a multi-objective approach.

The performance of multi-objective optimisation (MOO) algorithms is generally assessed using multiple performance indicators, in order to characterise several distinct quality properties, such as convergence or diversity. However, so far, the automatic configuration of multi-objective algorithms has used standard, single-objective configurators to optimise either a single performance indicator or an aggregation of several indicators [1].

The hypothesis we investigate here is that automatic configuration of multi-objective optimisation algorithms is best achieved using a multi-objective configuration procedure, such as MO-ParamILS. To study this hypothesis, we consider multi-objective local search algorithms, which are known to be very efficient for a broad class of MOO problems, and specifically, multi-objective iterated local search (MO-ILS), a metaheuristic known to achieve excellent performance if its constituent components are chosen and configured carefully. We introduce a flexible, highly parametric MO-ILS framework for the bi-objective permutation flowshop scheduling problem (PFSP), a classic problem on which multi-objective local search algorithms are known to achieve excellent performance [8]. Our results show that using an MO configuration approach, we achieve better results than obtained from single-objective configuration approaches. Specifically, using the same configuration budget, we obtain a broader range of non-dominated trade-offs between two performance indicators, hypervolume and Δ spread, without significant loss in the quality of the configurations thus obtained. We also report new insights into the components of our local search framework that are effective for solving PFSP instances of different sizes.

2 Preliminaries

In multi-objective optimisation, multiple criteria (or objective functions) characterising the quality of solutions to a given problem are optimised simultaneously. The concept of *Pareto dominance* is used to capture trade-offs between those criteria: 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 exists 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*.

It is not straightforward to assess or compare the performance of multiple multi-objective algorithms. In the literature, many performance indicators have been proposed [12,17] and classified according to several properties: (i) cardinality, (ii) convergence and (iii) distribution. It has also been shown that it is generally not possible to aggregate such properties into a single indicator. Thus, it is recommended to consider multiple performance indicators, preferably ones that complement each other, in order to assess the efficiency multi-objective optimisation algorithms fairly.

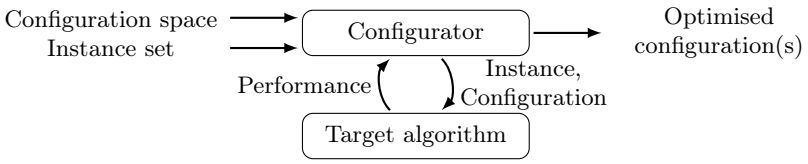

Fig. 1: Automatic configuration of a given, parameterised target algorithm for performance optimised for a given set problem instances.

Here, we use two complementary indicators: unary hypervolume [20], a volume-based convergence performance indicator, and Δ spread [4], a distance-based distribution metric. In the following, we assume that all objective values have been normalised to $[0, 1]$ and are to be minimised, meaning that the nadir is at $(1, 1)$ and the ideal at $(0, 0)$. The unary hypervolume indicator [20] measures the hypervolume of the objective space between the solutions of a given Pareto set and the nadir point. Hypervolume is maximised when the Pareto set is reduced to the ideal point. The Δ spread indicator [4] has been proposed to measure the distance-based distribution of set of solutions in a bi-objective context. Given a Pareto set S, ordered regarding the first criterion, we define

$$\Delta := \frac{d_f + d_l + \sum_{i=1}^{|S|-1} |d_i - \bar{d}|}{d_f + d_l + (|S| - 1) \cdot \bar{d}},$$

where d_f and d_l are the Euclidean distances between the extreme positions $(1, 0)$ and $(0, 1)$, respectively, and the boundary solutions of S, and $\bar{d}$ 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 1 for Pareto sets with few or unevenly distributed solutions.

3 Multi-Objective Algorithm Configuration

The goal of *automatic algorithm configuration (AAC)* is to automatically determine a configuration (*i.e.*, parameter setting) optimising 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*. The general concept of AAC is illustrated in Figure 1.

Most applications of AAC consider a single-objective target algorithm, using a single performance metric – for optimisation algorithms, usually either the running time required to reach a specific solution quality or the solution quality achieved within a given running time. State-of-the-art AAC procedures from the literature include irace [14], SMAC [10] and ParamILS [11]. In principle, these and other single-objective AAC procedures can easily be applied

to multi-objective optimisation (MOO) target algorithms, using a single performance indicator or an aggregation of several indicators (*e.g.*, using the hypervolume of normalised indicators [1]). A conceptually attractive alternative is to directly optimise multiple performance indicators, requiring a multi-objective configurator, such as the recently proposed MO-ParamILS [2], an extension of the original, single-objective ParamILS configuration framework. To the best of our knowledge, multi-objective configurators have so far only been applied to single-objective target algorithms – for example, for simultaneously optimising solution quality and running time of single-objective optimisation algorithms. Here, we investigate the efficacy of the state-of-the-art MO configurator MO-ParamILS when applied to MOO algorithms, using multiple multi-objective performance indicators.

In the standard AAC framework shown in Figure 1, only the performance indicator values of the target algorithm are given to the configurator, rather than the entire target algorithm output. For multi-objective algorithms, this means that we cannot use binary performance indicators or performance indicators that use dynamic reference points. (Of course, this limitation could be overcome in future MO configurator designs.)

Algorithm configuration is a machine learning process that involves three separate phases: *training*, *validation* and *testing*. In the training phase, illustrated in Figure 1, the configurator is used to optimise the configuration of the target algorithm on a given set of training instances. However, configurators are based on stochastic search procedures that are sensitive to random decisions made during the search process, including the order in which training instances are considered. Thus, the training phase is performed multiple times, independently, each with a different pseudo-random number seed. In the validation phase, the performance of the final configurations thus obtained is measured on a common set of instances (usually a subset of the training set). Dominated configurations are discarded, resulting in a set of non-dominated configurations. In the final testing phase, the set of configurations thus obtained is evaluated on a set of instances that does not contain any of the instances used for training or validation.

Note that from each independent run of a single-objective configuration procedure, a single configuration is obtained, while a run of a multi-objective configurator results in a Pareto set of configurations. In the latter case, the validation phase over n independent configurator runs will result in n Pareto sets, which are then merged, eliminating any configuration that is dominated with respect to the given performance objectives. Likewise, in the testing phase, every configuration is evaluated w.r.t. all given performance objectives, and we do not report configurations that turn out to be dominated on the given test set.

4 Multi-Objective Local Search Algorithms

Stochastic local search (SLS) algorithms have been widely used for single-objective optimisation [9], and extensions to multi-objective optimisation are known to achieve excellent performance. The most popular multi-objective SLS algorithms

Algorithm 1: Multi-Objective Iterative Improvement (`moii_algo`); as described in detail in Section 4.3, 'exploration' makes use of a reference.

```
Input: Initial archive
Output: Archive of non-dominated solutions
archive ← initial archive;
until termination criterion is met do
    /* Selection */
    selection ← select(archive);
    /* Exploration */
    candidates ← empty list;
    foreach solution ∈ selection do
        candidates ← candidates ∪ exploration(solution);
    /* Archive */
    archive ← combine(archive, candidates);
return archive;
```

include Pareto local search (PLS, 2004) [18] and its numerous variants, such as the iterated PLS (2010) [5], stochastic PLS (2012) [6], anytime PLS (2015) [7], and dominance-based multi-objective local search (DMLS, 2012) [13]. In the following, we use a parametrised general local search framework that incorporates most of the strategies used by these algorithms. Our framework can be configured to replicate the behaviour of efficient algorithms of the literature, while making possible numerous intermediate behaviours by combining known building blocks in novel ways. More complex and problem-specific hybrid strategies based on local search strategies have also been studied in the literature. We do not consider these here, as our goal is to automatically configure effective yet widely applicable local search procedures rather than to design a state-of-the-art local search algorithm for a given problem using problem-specific expert knowledge.

Single-objective local search algorithms often get trapped in or around local optima of the search space they are exploring. This can be overcome in many ways, among which iterated local search (ILS) is well-known for its efficacy and versatility; as a general stochastic local search method, ILS provides the basis for state-of-the-art algorithms for many challenging combinatorial optimisation problems [15]. Our highly parametric ILS framework, described in the following, facilitates the design of powerful multi-objective local search algorithms based on an iterative improvement procedure.

4.1 Multi-Objective Iterative Improvement

Algorithm 1 outlines a general procedure for iteratively improving an archive (*i.e.*, a Pareto set) until a stopping condition is satisfied. It works in three phases: (i) the *selection* phase, in which solutions are selected from the archive; (ii) the *exploration* phase, in which the neighbourhood of each selected solution is explored, based on a reference set of solutions, and some neighbours are accepted

Algorithm 2: Multi-Objective Iterated Local Search

```
Input: Initial archive
Result: Archive of the best solutions
current_archive ← initial archive;
current_archive ← moii_algo(current_archive);
until termination criterion is met do
    tmp_archive ← perturb(current_archive);
    tmp_archive ← moii_algo(tmp_archive);
    current_archive ← combine(current_archive, tmp_archive);
return current_archive;
```

as candidates; and (iii) the *archive* phase, in which the current archive is updated using the candidate neighbours.

Many strategies exist for these three phases and will be described in Section 4.3. We note that structurally, Algorithm 1 resembles DMLS [13]; however, as explained in Section 4.3, we have augmented the exploration procedure to make use of a reference point or set in order to replicate the behaviour of PLS algorithms.

4.2 Multi-Objective Iterated Local Search

Our multi-objective iterated local search framework is obtained by embedding the iterative improvement procedure from Algorithm 1 into a typical ILS algorithm, as shown in Algorithm 2. After an initial run of the iterative improvement procedure, three search phases are iterated: (i) a perturbation phase, in which a copy of the current archive is perturbed, (ii) an iterative improvement phase, during which the perturbed archive is optimised, (iii) an acceptance phase, during which the current archive is updated based on the previous and newly optimised archives. In our case, the acceptance phase consists of merging the two archives followed by pruning based on the Pareto criterion.

4.3 Parameters and Configuration Space

We now describe the different strategies included in our multi-objective ILS framework, its parameters and their instantiation for simulating the behaviour of prominent local search algorithms from the literature.

Initialisation. We always initialise our iterative improvement procedure with 10 solutions generated uniformly at random. It is known that additional performance improvements can be realised by using an auxiliary optimisation algorithm to generate initial solutions [8], but we decided to not include this type of initialisation mechanism in order to focus our investigation on the core ILS procedure and its configuration.

Selection. In the selection phase, we choose a subset of solutions from the archive on which exploration will be performed. We then distinguish two main selection strategies (`selectStrat`): The first of these selects `all` solutions from the archive for exploration. The second strategy chooses a subset of $k \in \{1, 2, 3\}$ solutions from the archive (parameter `selectSize`), either uniformly at `random`, or the k `newest` or `oldest` solutions. Additionally, if some solutions have already been fully explored, we filter these out before starting the selection process.

Exploration. In this phase, the neighbourhood of each selected solution is explored: a set of neighbours is evaluated, and some of these are then added to the candidate set, which is later merged with the archive. We consider two types of exploration strategies (`explorStrat`), one of which evaluates all neighbours, while the other limits exploration to a subset of the neighbours. If all neighbours are evaluated, we can then either add all non-dominated neighbours to the candidate set (strategy `all`), or only add all dominating neighbours (strategy `all_imp`). Otherwise, we consider two different termination criteria for the exploration. The first of these ends exploration when k dominating neighbours have been evaluated. In that case, we can either only add these dominating neighbours to the candidate set (strategy `imp`), or also include the non-dominated neighbours that have been evaluated so far (strategy `imp_ndom`). The second criterion terminates exploration when k non-dominated neighbours have been evaluated, which are then added to the candidate set (strategy `ndom`). For both termination criteria, the value of k is specified by the parameter `explorSize`.

These strategies are further elaborated using another parameter, `explorRef`, which specifies the reference point for both *dominating* and *non-dominated* neighbours. This reference point can be either set to the current solution being explored (`sol`), or to a set of solutions, in which case we implicitly mean *dominating every solution in the set* and *non-dominated regarding each solution in the set*; we support taking the current archive as reference during exploration (`arch`) as well as taking the subset of the solutions that have been selected (`select`).

Archive. After all selected solutions have been explored, the set of candidates is merged with the current archive, and every dominated solution is removed. Thus, the archive always contains the best non-dominated solutions found during the search process. We note that the size of the archive is unbounded.

Termination Criteria. The termination criterion of the iterative multi-objective improvement procedure (Algorithm 1) is satisfied either when all the solutions in the current archive have been entirely explored, or when a given number of iterations have been performed without any modification of the archive. The termination criterion of the multi-objective iterated local search (Algorithm 2) is simply time-based.

Perturbation. In Algorithm 2, the starting point of every subsidiary local search is obtained by applying perturbation to the current archive. The pertur-

Table 1: Parameters of our MO-ILS framework and their possible values

Phase	Parameter	Parameter values
Initialisation	`initStrat`	`rand`
Initialisation	`initSize`	`10`
Selection	`selectStrat`	{`all`, `rand`, `newest`, `oldest`}
Selection	`selectSize`	$\{1, 2, 3\}$
Exploration	`explorStrat`	{`all`, `all_imp`, `imp`, `imp_ndom`, `ndom`}
Exploration	`explorRef`	{`sol`, `select`, `arch`}
Exploration	`explorSize`	$\{1, 2, 3\}$
Perturbation	`perturbStrat`	{`restart`, `kick`, `kick_all`}
Perturbation	`perturbSize`	$\{1, 2, 3\}$
Perturbation	`perturbStrength`	$\{3, 5\}$

bation strategy (`perturbStrat`) can either be `restart`- or `kick`-based. In the first case, we consider 10 new solutions, generated uniformly at random, following the initialisation strategy. Otherwise, we either select $k \in \{1, 2, 3\}$ solutions from the archive (strategy `kick` with parameter `perturbSize`) or take all the solutions of the archive (strategy `kick_all`), before performing a *kick* move on each of them; when a solution is *kicked*, it is replaced by one of its neighbour selected uniformly at random. The parameter `perturbStrength` specifies how many times the selected solutions are kicked.

Overall Configuration Space. Table 1 shows all parameters exposed by our multi-objective iterated local search framework (Algorithms 1 and 2) and their possible values; these jointly give rise to $1 \cdot (1+3 \cdot 3) \cdot (1+3+3 \cdot (3 \cdot 3)) \cdot (1+3 \cdot 2+2) = 2790$ valid configurations.

5 Experiments

In our experiments, we focus on the bi-objective permutation flowshop scheduling problem, for which multi-objective local search algorithms are known to be very efficient [8]. In the following, we first give a brief description of this classical MOO problem, followed by the experimental protocol we used to compare our MO configuration approach with two single-objective approaches.

5.1 The Bi-objective Permutation Flowshop Scheduling Problem

The Permutation Flowshop 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 is represented by a permutation of size N. Here, we consider the bi-objective PFSP, minimising

both the makespan and the flowtime of the schedule, two objectives widely investigated in the literature [16], where makespan is the total completion time of the schedule, and flowtime is the sum of the individual completion times of all N jobs.

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 perturbation. In the following, we consider the union of these two classical neighbourhoods, as this hybrid neighbourhood has been shown to lead to better solutions than either of the two constituting neighbourhoods by itself [7].

5.2 Experimental Design

Benchmark Instances and Configuration Scenarios. We considered two sizes of bi-objective PFSP instances, leading to two configuration scenarios: one with instances of 20 jobs and a configuration budget (training time) of 12 CPU hours, and the other one with instances with 50 jobs and a configuration budget of 24 CPU hours. The classical PFSP instances in the literature are the widely-used Taillard instances [19]. Because the set of training and testing instances need to be completely disjoint and independent, we used the original Taillard instances only in the testing phase and generated new Taillard-like instances for training and validation phases.

In the testing phase, we used 30 classical Taillard instances for both 20 and 50 jobs scenarios, with 5, 10 or 20 machines (10 different instances for each combination). For the training phase, we generated 80 new instances with 5 to 20 machines (18 different instances for 5, 10 and 20 machines, and 2 instances for each intermediate instances sizes) for both 20 and 50 jobs scenarios. We also limited the maximum number of training runs of a given configuration to 240 (*i.e.*, 3 runs on each instance) in order not to spend too much time on too few instances. For the validation phase, a single target algorithm run was performed on each of the 80 training instances. The final test assessment was performed spending 5 runs on each of the 30 Taillard instances for a total of 150 runs.

In all our configuration experiments, we considered the configuration space defined by the parameters and parameter values specified in Table 1 (Section 4.3), containing a total of 2790 valid configurations. The maximum running time for all target algorithm runs was dynamically fixed to $n \cdot m/50$ CPU sec, where n and m are the number of jobs and the number of machines of the instance being solved. We note that these running times were chosen to be smaller than those commonly found in the literature ($n \cdot m/10$ in [16] and [7], or from $6 \cdot n \cdot m$ to $0.9 \cdot n \cdot m$ in [13] for instances of similar size), in order to permit the configurator to perform more runs and thus consider a larger number of configurations. We set the maximum number of consecutive iterations of Algorithm 1 without improvement to n. Finally, our algorithm framework and all of its components have been implemented in ParadisEO [3], a white-box, object-oriented framework dedicated to the flexible design of metaheuristics, in order to facilitate fair comparisons between arbitrary instantiations of our MO-ILS framework.

Performance assessment. We used the hypervolume and Δ spread indicators to assess the performance of each MO-ILS configuration. Let us recall that the hypervolume measures the convergence, and the spread indicator is only used as a complementary indicator that measures the distribution along the Pareto set; we note that Δ spread is meaningless when used alone. To achieve a formulation where both indicators are to be minimised, we used $HV = 1 -$ hypervolume, *i.e.*, the complement between hypervolume and the hypervolume of the ideal point.

AAC Experimental Protocol. We evaluated three configuration approaches: $HV||\Delta$, a multi-objective approach, in which both hypervolume and Δ spread are minimised, using MO-ParamILS [2]; HV, a single-objective approach, in which only hypervolume is minimised; and HV+Δ, a single-objective approach, in which we minimise a weighted sum of hypervolume and Δ-spread. For both single-objective approaches, we used single-objective ParamILS [11], as implemented in our MO-ParamILS framework. In HV+Δ, we normalised both performance indicators and use weights of 0.75 for the hypervolume and 0.25 for the spread, as we see the latter as a secondary performance indicator. For each of these configuration approaches, we performed 30 independent configurator runs each with a total budget of 12 CPU hours for the 20-job benchmark and 24 CPU hours for the 50-job benchmark. The resulting configurations were evaluated on the entire given training set, and Pareto-dominated configurations were removed.

6 Results and Discussion

Figure 2 shows the performance of the non-dominated configurations obtained in our three configuration scenarios for the 20- and 50-job Taillard benchmark sets. The single-objective configuration approach optimising hypervolume only (HV) resulted in 3 and 2 final configurations on the 20- and 50-job instance sets, respectively. While the hypervolume values achieved by these configurations were amongst the best obtained by any of our approaches, we observed somewhat average spread values on the 20-job instances, and poor spread on the 50-job set. Two of the three configurations for the 20-job set obtained exactly the same hypervolume and spread values.

The single-objective HV+Δ configuration approach optimising a weighted combination of hypervolume and spread also produced small numbers of non-dominated configurations (2 on the 20-job and 3 on the 50-job benchmarks). As expected, these configurations achieved different tradeoffs between the two performance metrics. While it is clear that a broader range of tradeoffs could be obtained by using different weight vectors, this would require additional, costly configurator runs. Additionally, the HV+Δ approach requires non-trivial normalisation of both performance metrics, which we achieved based on the configurations obtained by HV and $HV||\Delta$; this is unproblematic only because the purpose of studying HV+Δ was to provide a second baseline for our multi-objective configuration approach.

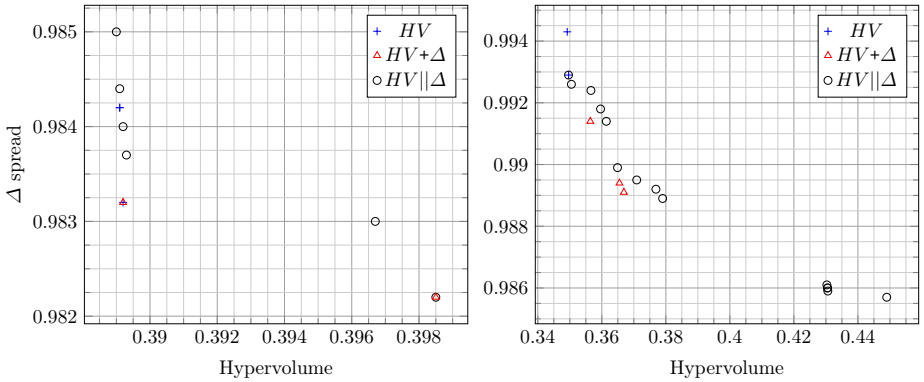

Fig. 2: Performance of automatically determined, non-dominated configurations on test sets of 20-job (left) and 50-job (right) Taillard instances. Each point shows the mean performance of a single configuration over the test set.

In contrast to HV and $HV{+}\Delta$, our multi-objective configuration approach, $HV||\Delta$, resulted in substantially larger numbers of configurations covering a wide range of tradeoffs between hypervolume and spread. (The same phenomenon was observed when evaluating the configurations produced by ParamILS and MO-ParamILS on the respective training sets of instances.) We note that, while on the 20-job benchmark, $HV||\Delta$ and $HV{+}\Delta$ achieved very similar extreme values for both performance indicators, on the more challenging 50-job benchmark, the configurations found by $HV||\Delta$ spanned a wider range of hypervolume and spread values. Overall, for both benchmarks, we obtained two rather well-separated clusters of non-dominated configurations, but for the set of larger instances, the second cluster – characterised by low Δ spread and high hypervolume – was only reached by the multi-objective $HV||\Delta$ configuration approach, while the single-objective $HV{+}\Delta$ achieved excellent results only in the direction of the weight vector used for aggregation.

Whereas Figure 2 shows average performance across each benchmark set, we also examined average performance on subsets of instances with the same number of machines and on individual instances within these sets. This analysis revealed significant variation in both performance metrics with number of machines as well as between instances. For example, while the largest difference in Δ spread for 50-job instances, averaged over the entire set, that we measured for any pair of configurations found by our three approaches is below 0.01 (as seen in Figure 2), we observed differences as large as 0.03 between instances within the set for individual configurations produced by any of our approaches. Furthermore, similar spread values were observed across and within instance subsets with 5, 10 and 20 machines. This indicates that the small differences in mean spread observed between the configurations found by $HV{+}\Delta$ and the closest configurations produced by $HV||\Delta$ are somewhat insignificant. This finding is further supported by our observation that differences in the spread values measured for pairs of configurations are inconsistent across the instances within each set.

Table 2: Optimised configurations for Taillard instances with 20 jobs

HV	Δ	Approach	Selection		Exploration			Perturbation		
0.3891	0.9842	HV	`rand`	2	`imp_ndom`	`select`	1	`kick`	1	5
0.3891	0.9842	HV	`oldest`	2	`imp_ndom`	`sol`	1	`restart`	.	.
0.3892	0.9832	HV	`all`	.	`imp_ndom`	`select`	1	`kick_all`	.	5
0.3892	0.9832	HV+Δ	`all`	.	`imp_ndom`	`select`	1	`kick_all`	.	5
0.3985	0.9822	HV+Δ	`all`	.	`ndom`	`arch`	1	`restart`	.	.
0.389	0.985	$HV\|\|\Delta$	`oldest`	1	`imp_ndom`	`sol`	1	`kick`	1	5
0.3891	0.9844	$HV\|\|\Delta$	`all`	.	`imp_ndom`	`select`	1	`kick`	1	5
0.3892	0.984	$HV\|\|\Delta$	`oldest`	1	`imp_ndom`	`sol`	1	`kick`	3	3
0.3893	0.9837	$HV\|\|\Delta$	`oldest`	1	`imp_ndom`	`sol`	1	`restart`	.	.
0.3967	0.983	$HV\|\|\Delta$	`all`	.	`ndom`	`arch`	1	`kick`	1	3
0.3985	0.9822	$HV\|\|\Delta$	`all`	.	`ndom`	`arch`	1	`restart`	.	.

In contrast, the differences in mean hypervolume observed between the two clusters of configurations in Figure 2 are not only quite large, but also correspond well to the differences in hypervolume for instance subsets with the same, fixed number of machines and on individual instances within these sets. Furthermore, the hypervolume values for instance subsets with 5, 10 and 20 machines fall into intervals with very little overlap, and those intervals differ markedly between configurations from the two clusters seen in Figure 2 (right). Overall, this indicates that these clusters of configurations are indeed well separated, which highlights the significance of our finding that, at least for the larger, 50-job instances, only the multi-objective $HV||\Delta$ approach finds the second cluster of configurations, characterised by high hypervolume and low Δ spread values.

Finally, we examined the non-dominated configurations found by HV, HV+Δ and $HV||\Delta$ in detail. Tables 2 and 3 show the performance metrics and parameter settings for each of the configurations shown in Figure 2; inactive conditional parameters are indicated by dots (*e.g.*, when all the solutions are selected, the selection size parameter is inactive). For the 20-job scenario (see Table 2), the configurations found by our three approaches are quite varied. The selection strategy is mostly `oldest` or `all`, the exploration strategy either `imp_ndom` or `ndom`, and other parameters take various values. Overall, this suggests that for these small instances, a broad range of design choices within our framework achieves good tradeoffs between hypervolume and spread. For the larger, 50-job scenario (see Table 3), the situation appears to be markedly different. There are two distinct types of configurations, corresponding to the two clusters seen in Figure 2. The first of these consistently uses the `newest` selection strategy and the `ndom` exploration strategy, whereas the second employs the `all` selection strategy and (with one exception) the `all` exploration strategy; almost all configurations make use of the `kick` or `kick_all` perturbation strategy. The `restart` perturbation strategy, which seems to work well on the 20-job instances,

Table 3: Optimised configurations for Taillard instances with 50 jobs

HV	Δ	Approach	Selection		Exploration			Perturbation		
0.3492	0.9943	HV	`newest`	1	`ndom`	`select`	2	`kick`	2	3
0.3496	0.9929	HV	`newest`	1	`ndom`	`sol`	1	`kick_all`	.	3
0.3564	0.9914	HV	`all`	.	`ndom`	`arch`	1	`kick_all`	.	3
0.3655	0.9894	HV+Δ	`newest`	1	`ndom`	`arch`	3	`kick`	2	3
0.3669	0.9891	HV+Δ	`newest`	1	`ndom`	`arch`	2	`kick_all`	.	3
0.3496	0.9929	$HV\|\|\Delta$	`newest`	1	`ndom`	`sol`	2	`kick_all`	.	3
0.3505	0.9926	$HV\|\|\Delta$	`newest`	1	`ndom`	`sol`	1	`kick`	1	5
0.3566	0.9924	$HV\|\|\Delta$	`newest`	3	`ndom`	`arch`	2	`kick_all`	.	3
0.3596	0.9918	$HV\|\|\Delta$	`newest`	2	`ndom`	`arch`	2	`kick_all`	.	3
0.3614	0.9914	$HV\|\|\Delta$	`newest`	3	`ndom`	`arch`	1	`kick_all`	.	3
0.3649	0.9899	$HV\|\|\Delta$	`newest`	1	`ndom`	`arch`	3	`kick`	3	3
0.3669	0.9891	$HV\|\|\Delta$	`newest`	1	`ndom`	`arch`	2	`kick_all`	.	3
0.379	0.9889	$HV\|\|\Delta$	`newest`	1	`ndom`	`arch`	1	`kick`	1	3
0.4303	0.9861	$HV\|\|\Delta$	`all`	.	`all`	.	.	`kick`	3	3
0.4305	0.986	$HV\|\|\Delta$	`all`	.	`all`	.	.	`kick`	1	5
0.4305	0.986	$HV\|\|\Delta$	`all`	.	`all`	.	.	`kick`	3	5
0.4306	0.9859	$HV\|\|\Delta$	`all`	.	`all`	.	.	`kick`	1	3
0.449	0.9857	$HV\|\|\Delta$	`all`	.	`imp_ndom`	`arch`	2	`restart`	.	.

appears to be less effective on the more challenging 50-job instances. These findings suggest that, as instance size and difficulty increases, effective combinations of design choices become more constrained, and finding these combinations more challenging. Our multi-objective configuration approach, $HV||\Delta$, appears to be considerably more effective at exploring this design space than the two single-objective configuration approaches, HV and HV+Δ, as witnessed by the size and diversity of the sets of non-dominated configurations found by each approach. We note that `rand` is a selection strategy commonly used in the literature; interestingly, it was chosen only once by our configuration approaches, which, in contrast, favoured `newest` and `oldest`. The prevalence of the `ndom` and `imp_ndom` exploration strategies in the configurations obtained from our approaches highlights the importance of considering the non-dominated neighbours in multi-objective iterated local search for the bi-objective PFSP.

The DMLS recommendations [13] regarding design choices within multi-objective local search algorithms depending on instance size are mostly consistent with the results obtained from our automatic configuration approaches, with some interesting differences. For small instances, the recommendation is to use exploration until the first dominating neighbour is found (*i.e.*, exploration strategy `imp_ndom` with `selectSize` = 1), with either the selection of a single random solution from the archive or the selection of the entire archive (*i.e.*, selection strategy `all` or `rand` with `selectSize` = 1). For larger instances, the recommendation is to explore until the first non-dominated neighbour is found

(exploration strategy `ndom` with `selectSize` = 1). In both cases, the only reference considered is the current solution being explored (`explorRef` = `sol`), and the recommendation is to use full restarts. In contrast, our results indicate that the random selection strategy is not a good choice, and that either the entire archive should be selected, or a selection of the oldest or newest solutions from the archive should be considered instead. Furthermore, we found no evidence that the full restart strategy is effective for larger instances; on the other hand, using the current archive as a reference for exploration – something not considered in DMLS – appears to be very effective, especially for larger instances.

7 Conclusions

Based on our empirical investigation, we conclude that there is substantial promise in the use of multi-objective (MO) automated algorithm configuration procedures, such as MO-ParamILS, for optimising the performance of highly heuristic algorithms for multi-objective optimisation (MOO) problems. While standard, single-objective algorithm configurators are effective tools for optimising a single performance metric of an MOO algorithm, such as hypervolume, in many circumstances, it can be important to pay attention to multiple performance indicators – in particular, to ones that assess the overall quality of a set of solutions and its diversity. Conceptually, this turns the configuration problem into a multi-objective optimisation problem, and our results suggest that this problem is solved most effectively using an MO configuration procedure.

For our experiments, we devised a highly parameterised iterated local search framework for the widely studied bi-objective permutation flow-shop problem. Our framework comprises a broad range of building blocks from the literature as well as a few simple novel choices. It is important to emphasise that in this work, our goal was not to improve the state of the art in solving the bi-objective PFSP, but rather to determine whether the automated configuration of flexible and powerful algorithm frameworks for this type of MOO problem should make use of multi-objective algorithm configuration procedures, such as MO-ParamILS. We firmly believe that automated algorithm configuration will greatly facilitate improvements in the state of the art for solving challenging MOO problems, such as the bi-objective PFSP – and this belief is amply supported by prior work on automatically configuring single- and multi-objective optimisation algorithms. Moreover, based on our findings reported in this work, we believe that MOO algorithms should be configured using an MO algorithm configuration procedure for the Pareto-optimisation of multiple performance indicators – an approach which differs from prior work on automated configuration of MOO algorithms.

As a secondary contribution of our work, based on a detailed analysis of the configurations of our multi-objective iterated local search framework, we proposed several revisions to the recommendations from prior work on dominance-based multi-objective local search [13]. In particular, we found evidence that simple alternatives to random selection provide improved overall performance. This type of finding confirms that the use of effective automated algorithm pro-

cedures and protocols can yield valuable insights into the efficacy of various algorithmic strategies and components for solving challenging MOO problems.

References

1. Bezerra, L.C.T.: A component-wise approach to multi-objective evolutionary algorithms, PhD thesis, IRIDIA, Université Libre de Bruxelles, Belgium, July 2016
2. Blot, A., Hoos, H.H., Jourdan, L., Marmion, M.É., Trautmann, H.: MO-ParamILS: A multi-objective automatic algorithm configuration framework. In: LION 10. LNCS, vol. 10079, pp. 32–47 (2016)
3. Cahon, S., Melab, N., Talbi, E.: Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. JoH 10(3), 357–380 (2004)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE TEVC 6(2), 182–197 (2002)
5. Drugan, M.M., Thierens, D.: Path-guided mutation for stochastic Pareto local search algorithms. In: PPSN 11. LNCS, vol. 6238, pp. 485–495 (2010)
6. Drugan, M.M., Thierens, D.: Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. JoH 18(5), 727–766 (2012)
7. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Anytime Pareto local search. EJOR 243(2), 369–385 (2015)
8. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. C&OR 38(8), 1219–1236 (2011)
9. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Elsevier / Morgan Kaufmann (2004)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: LION 5. LNCS, vol. 6683, pp. 507–523 (2011)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. JAIR 36, 267–306 (2009)
12. Knowles, J., Corne, D.: On metrics for comparing nondominated sets. In: IEEE CEC. vol. 1, pp. 711–716 (2002)
13. Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., Talbi, E.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. JoH 18(2), 317–352 (2012)
14. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
15. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of metaheuristics, pp. 320–353. Springer (2003)
16. Minella, G., Ruiz, R., Ciavotta, M.: A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. IJOC 20(3), 451–471 (2008)
17. Okabe, T., Jin, Y., Sendhoff, B.: A critical survey of performance indices for multi-objective optimisation. In: IEEE CEC. vol. 2, pp. 878–885 (2003)
18. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Metaheuristics for Multiobjective Optimisation, pp. 177–199. Springer (2004)
19. Taillard, E.: Benchmarks for basic scheduling problems. EJOC 64(2), 278–285 (1993)
20. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE TEVC 3(4), 257–271 (1999)