Automatic Design of Multi-Objective Local Search Algorithms

Case Study on a bi-objective Permutation Flowshop Scheduling Problem

Aymeric Blot
Université de Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, INRIA Lille, France
aymeric.blot@inria.fr

Laetitia Jourdan
Université de Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, INRIA Lille, France
laetitia.jourdan@univ-lille1.fr

Marie-Éléonore Kessaci
Université de Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, INRIA Lille, France
me.kessaci@univ-lille1.fr

ABSTRACT

Multi-objective local search (MOLS) algorithms are efficient metaheuristics, which improve a set of solutions by using their neighbourhood to iteratively find better and better solutions. MOLS algorithms are versatile algorithms with many available strategies, first to select the solutions to explore, then to explore them, and finally to update the archive using some of the visited neighbours. In this paper, we propose a new generalisation of MOLS algorithms incorporating new recent ideas and algorithms. To be able to instantiate the many MOLS algorithms of the literature, our generalisation exposes numerous numerical and categorical parameters, raising the possibility of being automatically designed by an automatic algorithm configuration (AAC) mechanism. We investigate the worth of such an automatic design of MOLS algorithms using MO-ParamILS, a multi-objective AAC configurator, on the permutation flowshop scheduling problem, and demonstrate its worth against a traditional manual design.

CCS CONCEPTS

•**Theory of computation** → **Design and analysis of algorithms;** *Randomized local search;* •**Applied computing** → Multi-criterion optimization and decision-making;

KEYWORDS

Metaheuristics, local search, parameter tuning, multi-objective optimisation

ACM Reference format:
Aymeric Blot, Laetitia Jourdan, and Marie-Éléonore Kessaci. 2017. Automatic Design of Multi-Objective Local Search Algorithms. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071323

1 INTRODUCTION

Multi-objective local search (MOLS) algorithms are metaheuristics specifically designed for multi-objective combinatorial optimisation problems. They have been used to tackle combinatorial optimisation problems such as the travelling salesman problem [14], the quadratic assignment problem [9], or the permutation flowshop scheduling problem (PFSP) [1, 9, 14]. Recently new ideas and new MOLS algorithms have arisen in the literature, which raises the question of the possible advantages of a unified MOLS algorithmic structure. This structure would comprise numerous strategies, not only from MOLS algorithms, but also from similar evolutionary algorithms, and should be able to easily include new unknown algorithms. The strength of such a structure is to enable to both instantiate existing algorithms and to design new hybrid algorithms.

The design of new algorithms often is a long and tedious task. Moreover, as the structure includes more and more possible components and parameters, it becomes increasingly impracticable to exhaustively analyse every possible combination of strategies, thus suggesting the use of an external tool to automatically search for the best configurations. Automatic algorithm configuration (AAC) has become in the last few years an increasingly efficient and popular preliminary step in the use of metaheuristics. Indeed, algorithms exposing numerous parameters can be automatically configured in order to improve their performance on given classes of problem instances. The benefit of this automatically design algorithms from a highly parametric structure having already been shown for single-objective SLS algorithms [17], we more specifically question here the automatic design of a multi-objective metaheuristic structure.

In this paper, we first propose a new highly parametric MOLS generalisation. Then, we investigate the performance of an AAC procedure in comparison to an exhaustive analysis of all feasible MOLS configurations. We use MO-ParamILS [2], a multi-objective AAC configurator, in order to account for the fundamental multi-objective nature of MOLS algorithms. The experiments are conducted on a well-known bi-objective permutation flowshop scheduling problem.

This paper is organised as follows. First, Section 2 presents multi-objective local search algorithms, and recent new ideas and algorithms, then details a generalised MOLS structure and enumerates its different design points together with their related strategies. Section 3 then lays out the design of the experiments, whose protocol is made explicit in Section 4, and whose results are discussed in Section 5. Finally, Section 6 concludes this paper and draws some perspectives.

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071323

2 MULTI-OBJECTIVE LOCAL SEARCH ALGORITHMS

In this section, we present a brief literature review on MOLS algorithms and report new recent strategies. Then, we propose a highly parameterisable generalisation of MOLS algorithms able to embed these new strategies.

2.1 Multi-Objective Combinatorial Optimisation

Multi-objective combinatorial problems appear when multiple criteria are considered. The Pareto dominance can be used to compare two solutions s_1 and s_2: s_1 is said to *dominate* s_2 if and only if s_1 is better or equal to s_2 according to all criteria, and there is at least one criterion according to which s_1 is strictly better than s_2. If neither s_1 dominates s_2 nor s_2 dominates s_1, both solutions are said *incomparable*. Various alternatives to the Pareto dominance can also be used, as for example, an aggregation between the criteria or a lexicographic order.

Solving a multi-objective combinatorial optimisation problem with the Pareto dominance aims to find the Pareto optimal set, defined as the set of feasible solutions not dominated by any other feasible solution. A Pareto set is then simply defined as a set of solutions in which no solution dominates another.

Metaheuristics are generic methods, often preferred to exact methods on combinatorial problems, since they allow to quickly find very good solutions. They are divided into two categories: the bio-inspired methods, such as evolutionary algorithms, ant colonies and particle swarms optimisation algorithms, and the stochastic local search methods.

2.2 Recent Ideas in MOLS

Stochastic local search (SLS) algorithms have been widely used for single-objective optimisation [10], and their extensions to multi-objective optimisation are known to achieve excellent performance by iteratively improving a Pareto set. Many popular multi-objective local search algorithms (MOLS) are based on the Pareto local search (PLS) [21] and its numerous variants, such as the iterated PLS [6], the stochastic PLS [7], and the anytime PLS [9]. The dominance-based multi-objective local search (DMLS) [14] was proposed to generalise many of the PLS algorithms and some other very similar evolutionary algorithms, such as the Pareto archived evolution strategy (PAES) [13].

DMLS algorithms iterate three phases: (i) the *selection* phase, in which solutions are selected from the archive; (ii) the *exploration* phase, in which the neighbourhood of the selected solutions are explored and candidate neighbours are extracted; and (iii) the *archive* phase, in which the current archive is updated with the candidate neighbours. However, this generalisation is not able to include all recent MOLS of the literature (*e.g.*, in [1, 9]) since new ideas fall outside its scope.

These ideas are reported in the following. In phase (ii) of a DMLS, only the quality of the solution explored is considered while it has been shown that comparing the neighbours to the whole archive often leads to better performance [9]. A reference point should then be added to the exploration strategy. When several solutions are selected, the neighbourhoods are explored one by one. New exploration strategies may appear if we consider the union of the neighbourhood of all solutions as a unique neighbourhood. Likewise, all selected solutions are explored in phase (ii) no matter what happens. Therefore, it could be interesting to add a stopping criterion for this phase, *e.g.*, taking into account the neighbours already found.

In most literature MOLS algorithms, including the DMLS generalisation, candidate neighbours necessarily need to be archived before being able to be explored later – the three phases being executed in the same order iteratively. However, archiving may discard some useful neighbours, especially if the exploration strategy enables multiple candidate neighbours to be archived in a single iteration. It has been shown that exploring neighbours that might be immediately discarded can also lead to increasing performance [1], *i.e.*, it may be efficient to restart from phase (i) again directly after phase (ii).

Additionally, in the DMLS structure the acceptance criteria of archiving candidate neighbours is to merge both sets of solutions and accept any non-dominated solutions for inclusion, then use an optional criteria to avoid large archives. Therefore, strategies such as only accepting dominating neighbours [9] as candidate solutions for archiving also fall outside of this scope.

Finally, in all MOLS algorithms, the archive (*i.e.*, the current set of solutions) is a Pareto local set, meaning that no solution of the archive dominates another. Therefore, the archive always contain the best solutions found (locally). However, this additional Pareto constraint might sometimes hinder the diversity of solutions in the archive, *e.g.*, when it becomes necessary to bound the size of the archive. In some cases, especially near local optima, finding out new dominating solutions is very difficult and it could be beneficial to temporary *relax* the Pareto constraint to allow exploring the best dominated solutions that would have been otherwise simply discarded. For this reason, in our generalisation the archive is a simple set of solutions, and not a Pareto local set.

2.3 MOLS Structure

Algorithm 1 outlines our generalised MOLS structure where the three phases of the DMLS are kept and completed. Indeed, following the previous section, the exploration phase (ii) and the archive phase (iii) are generalised. In phase (ii), the `pick()` function selects one or many solutions from the ones selected in phase (i). The `refer()` function enables to choose the reference point, *e.g.*, the current archive or the current picked solution(s). At last, the use of the `update()` function and additional stopping conditions in both Algorithm 1 and the `explore()` function allows to shorten the entire exploration when useful or necessary.

Stochastic local search are exploitation methods, that needs to be diversified to perform better. The iterated local search (ILS) algorithm [16] gives a way to do that for single-objective optimisation by adding a phase to perturb the current solution and so, diversify the search, and another to accept or not the new solution found. We propose in Algorithm 2 a general way to iterate a MOLS, through an extended MOLS algorithm making use of the diversity. The `perturb()` function plays the same role as in single-objective optimisation while the `combine()` and `combineBest()` functions are needed to manage the archives following the archiving policy.

Algorithm 1: Multi-Objective Local Search Algorithm

```
Input: A set of solutions
Output: A set of solutions
archive ← initial set;
until termination criterion is met do
    /* phase (i) */
    selection ← select(archive);
    /* phase (ii) */
    candidates ← ∅;
    until condition or selection = ∅ do
        current ← pick(selection);
        reference ←
            refer(archive, current, candidates);
        neighbours ← exploration(current, reference);
        candidates, selection ← update( candidates,
            selection, current, reference, neighbours);
    /* phase (iii) */
    archive ← combine(archive, candidates);
return archive;
```

Algorithm 2: Iterated Multi-Objective Local Search Algorithm

```
Input: A set of solutions
Output: A set of solutions
current_archive ← initial archive;
current_archive ← local_search(current_archive);
best_archive ← current_archive;
until termination criterion is met do
    tmp_archive ←
        perturb(current_archive, best_archive);
    tmp_archive ← local_search(tmp_archive);
    current_archive ←
        combine(current_archive, tmp_archive);
    best_archive ←
        combineBest(best_archive, current_archive);
return best_archive;
```

2.4 MOLS Strategies

The role of the strategies corresponding to every unspecified sub-functions of Algorithms 1 and 2 are detailed and some example of instantiation are given. The strategies used in the following sections are also specified (see Section 4 Table 1).

Selection. The *selection* phase of Algorithm 1 selects the solutions of the current archive whose neighbourhood will be explored in the following phase of the algorithm. Either all the solutions of the archive (strategy `all`), or only a subset of solutions may be selected. The choice of the subset can be done either uniformly at random (strategy `rand`), or according to indicators, such as their lifetime in the archive (strategies `oldest` and `newest`), or their contribution to hyper-volume [9], crowding or sharing in the archive [4].

It is also possible, following an *activation/desactivation* scheme, to flag solutions as to ensure they will not be selected, and thus not explored, for example to avoid exploring again a solution already fully explored, or to speed up the algorithm by avoiding unrequired explorations [19]. In that case, the selection phase only consider activated solutions, and the state of a solution is generally handled in the exploration phase (*e.g.*, desactivating explored solutions, or newly found neighbours).

Pick. This function replaces the implicit mechanism of DMLS that iteratively explores the neighbourhood of every selected solutions one by one. One or more solutions are picked among ones selected by the selection function. The union of the neighbourhood of each picked solution will be considered in the exploration function. Let us remark that the DMLS process can be imitated: only one solution is picked without replacement during an iteration.

Reference. The neighbourhood explorations is conditioned by a reference to which is compared the neighbours. This reference point can be either set to the current picked solutions or to the current archive. Candidate neighbours found during the iteration (phase (ii)) can also been used as reference point.

Exploration. This function enables the exploration of the joined neighbourhood of the solutions picked. Explorations strategies are determined according to how much of the neighbourhood they explore, and to which neighbours they select. They can either explore the neighbourhood entirely (strategy `all`), or only partially until their termination criterion is met. For partial exploration strategies, the most popular strategies include stopping after the firsts dominated neighbours (strategy `imp`) or after the first non-dominated neighbours (strategy `ndom`), returning for both all non-dominated neighbours visited.

Update. At the end of every exploration, the sets of selected solutions and candidate neighbours are updated. In the DMLS framework, the picked solutions were removed from the set of selected solutions, and selected neighbours were simply merged into the candidate set. Here, the picked solutions are allowed to remain in the selection set – to be picked again –, and neighbours can be included in the selection set directly before phase (iii).

Combination / Archive. In phase (iii) of Algorithm 1 the current archive is updated with the set of candidate neighbours. This combination can be as simple as merging the two sets of solutions and discarding any dominated solutions, or can include any filtering mechanisms such as in the selection phase. As mentioned earlier, the archive is here a simple set of solutions, with no hard restriction of being a Pareto set at all time.

Termination Criteria. Algorithm 1 can include a natural termination criterion when no solution from the archive can be selected again. Otherwise, common popular termination criteria including time spent by the algorithm, number of iterations, number of iterations without improvement, or number of global evaluation can also be set.

Initialisation. The initial archive from which Algorithm 2 starts is necessarily non empty. It is usually composed either by solutions taken uniformly at random, or by high-quality solutions, often obtained with a scalarisation-based subsidiary algorithms [9].

Perturbation. In Algorithm 2, the starting point of every subsidiary local search is determined by applying a perturbation to either the current archive or the best archive, or also by restarting from a new archive. Following single-objective strategies to escape local optima, perturbations are generally based on *kick*s, where the current solution is replaced by one of its neighbours selected uniformly at randomly, multiple times. Kick-based multi-objective perturbations select a subset of an archive, then apply a kick on each of the selected solutions.

Other examples of possible perturbations strategies include restarting from new solutions obtained by an initialisation strategy, or using a *deconstruction/reconstruction* mechanism to alter the current solutions.

3 EXPERIMENTAL DESIGN

In this section, we briefly present automatic algorithm configuration, then describe the case study problem used in the experiments, its objectives, and the multi-objective performance indicators used to compare the different configurations.

3.1 Automatic Algorithm Configuration

The goal of automatic algorithm configuration (AAC) is to determine a configuration (*i.e.*, a parameter setting), optimising a given algorithm over a given class of problem instances. The algorithm performing the AAC procedure is called the *configurator*, whereas the algorithm being configured is called the *target algorithm*. A AAC procedure is typically performed when the analysis of the configuration space of the target algorithm – all the valid combinations of its parameter values – is too expensive to be carried out manually.

Notorious literature single-objective configurators include tools such as irace [15], ParamILS [12], or SMAC [11]. They optimise the estimated quality of the target algorithm using a single quality indicator, such as the average running time or quality. Recently, multi-objective configurators such as SPRINT-race [23] or MO-ParamILS [2] have shown the possibility and the benefits of using multiple performance indicators. It has also been shown that, on multi-objective AAC scenario optimising multiple multi-objective performance indicators of multi-objective target algorithms, using a multi-objective configurator should be preferred to using a single-objective configurator with an aggregation of the performance indicators [3].

3.2 Case Study: 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 may be represented by a permutation of size N. In the following, we consider the bi-objective PFSP, minimising both the makespan and the flowtime of the schedule, two objectives widely investigated in the literature [18], where makespan is the total completion time of the schedule, and flowtime is the sum of the individual completion times of the 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. We consider here the union of these two classical neighbourhoods, as this hybrid neighbourhood has been shown to lead to better performance than considering each neighbourhood independently [9].

3.3 Multi-Objective Automatic Algorithm Configuration

We use two complementary indicators to compare the performance of MOLS algorithms on the tackled bi-objective PFSP: the unary hypervolume [24], a volume-based convergence performance indicator, and the Δ spread [5], a distance-based distribution metric. Both indicators were required to be unary performance indicators as a current constraint of all literature AAC procedures. 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 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. However, in the following, in order to achieve a configuration scenario where both indicators are to be minimised we will always consider the complement HV between hypervolume and the hypervolume of the ideal point ($HV = 1 -$ hypervolume).

The Δ spread indicator measures 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.

4 EXPERIMENTAL PROTOCOL

In this section, we specify the sets of instances used in our experiments, together with a description of the considered training and validation PFSP instances, the considered MOLS parameter space, and the exhaustive and AAC approaches.

4.1 PFSP Instances

The classical PFSP instances widely used in the literature are the Taillard instances [22]. Their number of jobs range in $N = \{20, 50, 100, 200, 500\}$, while their number of machines range in $M = \{5, 10, 20\}$. There are 10 available Taillard instance for every valid (N, M) combination.

Table 1: Investigated parameters leading to 189 configurations of MOLS

Phase	Parameter	Parameter values
Initialisation	`initStrat`	{`rand`, `neh`, `ig`}
Selection	`selectStrat`	{`all`, `rand`, `newest`, `oldest`}
Selection	`selectSize`	{1, 3}
Exploration	`explorStrat`	{`all`, `imp`, `ndom`}
Exploration	`explorRef`	{`pick`, `arch`}
Exploration	`explorSize`	{1, 3}

In the following, we consider three types of PFSP instance sets, characterised by their number of jobs: $N = 20$, 50, and 100 jobs.

For the exhaustive approach and the validation of the AAC approach, we use the Taillard instances, and include 10 instances for each number of machines ($M = 5$, 10, 20), for a total of 30 instances by instance set. For the training of the AAC approach, we used a different, completely disjoint, set of instances, composed by newly generated Taillard-like instances. We generated 80 instances for each number of jobs (18 different instances for $M = 5$, 10 and 20 machines, and 2 different instances for each intermediate instance size).

4.2 MOLS Configuration Space

The MOLS strategies and parameters considered in our experiments are reported in Table 1. These strategies account for a total of $3 \times (1 + 3 \times 2) \times (1 + 2 \times 2 \times 2) = 189$ valid MOLS configurations.

As for the initialisation of our MOLS iterative structure (Algorithm 2), we considered three strategies. First, a strategy where 10 initial solutions are generated uniformly at random (strategy `rand`). Then, a strategy where 5 solutions are obtained using NEH, a well known flowshop constructive heuristic [20], according to 5 uniformly distributed aggregations of the two objectives: one optimising the sole makespan, one optimising the sole flowtime, and three optimising a mixture of both. The third strategy combine each of the 5 NEH solutions with the application of a simple iterated greedy (IG) algorithm. We used the IG algorithms and parameter values of Dubois-Lacoste *et al.* [8] (*i.e.*, $d = 4, 5, 5$; $T_c = 0.4, 0.5, 6$; and $N_{LS} = \infty, 3, 1$ for the pure makespan IG, the pure flowtime IG, and the aggregation-based IG, respectively).

The global termination criterion of the MOLS algorithm is set to $T_{\text{run}} = 0.1 \times N \times M$ seconds. Regarding the termination criterion of the inner MOLS (Algorithm 1), it stops when either a total of $N \times N$ iterations are reached, or N successive iterations are performed without improvement. We choose to consider then a kick-based perturbation: a single solution is selected uniformly at random from the current archive, kicked 3 times, and the search starts again from the resulting solution.

The quality of a given configuration is obtained by computing the arithmetic means of the hypervolume and the Δ spread over all the runs the configuration has been run on. These indicators having already been computed on normalised objectives, no additional normalisation has been performed.

4.3 Exhaustive Approach

The baseline of our experiments is the exhaustive analysis of all MOLS algorithm configurations on all Taillard instances. We performed 30 runs of every valid MOLS configuration for all instances. All configurations having a termination criterion of T_{run}, 30 runs being performed on every 10 instances of each size, for every of the 189 valid MOLS configurations, the exhaustive analysis of the three sets of validation instances has taken a total computation time of $189 \times T_{\text{test}}$, with

$$T_{\text{test}} = \sum_{M \in \{5, 10, 20\}} T_{\text{run}} \times 10 \times 30$$

seconds; *i.e.*, approximately 46 days, 115 days, and 230 days for each number $N \in \{20, 50, 100\}$ of jobs, respectively.

Note that the time required by the exhaustive approach increases exponentially with the number of parameters and their individual number of possible parameter values. Here it would have been excessively expensive to consider additional parameters (*e.g.*, more detailed initialisation, exploration or perturbation strategies) or additional parameter values(*e.g.*, more numerical values, or new strategies).

4.4 AAC Approach

We choose to use MO-ParamILS [2], a recent multi-objective AAC configurator itself based on a MOLS algorithm, as the configurator for the AAC approach, in order to take advantage of its intrinsic multi-objective nature in order to configure our multi-objective structure, and to avoid aggregating the hypervolume and the spread into a single performance indicator.

MO-ParamILS protocol uses three phases. First, it learns the best configurations of the given algorithm over a given training instance set. Because AAC procedures are generally very sensitive to the order in which the instances are considered, and given the stochasticity of ParamILS, this first step is carried out multiple times, with different random seeds and permutations of the training instance set. Then, because the quality of the final configurations of each of these ParamILS runs have been computed using ultimately different subsets of the training set and different number of algorithm runs, a second phase in necessary in order to compare all final configurations fairly. This second phase re-use the same training set of instances, but simply reassess the quality of every final configuration on a joint subset of instances, to discard any dominated configuration. At last, the quality of every non-dominated configuration is then assessed in a third phase on a second set of instances, disjoint from the training set.

For the training, 30 runs have been performed on each set of training instances, with a total training time per run of $T_{\text{training}} = 36$ minutes, 1 hour and 30 minutes, and 3 hours for each instance set, respectively. Each MO-ParamILS run was set to start from a single configuration, drawn uniformly at random. Then, each final configuration of the 30 runs was reassessed using a single run on each of the 80 instances of the training set of instances. As there are 18 instances of size $M = 5$, 10 and 20, and 2 instance for other intermediary number of machines, the total reassess computational

time by unique final training configuration is

$$T_{\text{reassess}} = \sum_{M \in \{5,10,20\}} T_{\text{run}} \times 18 + \sum_{\substack{6 \leq M \leq 9 \\ 11 \leq M \leq 19}} T_{\text{run}} \times 2$$

seconds. As for the validation on the Taillard instances, we reused the data of the exhaustive analysis, which would have taken T_{test} seconds by unique final configuration.

In comparison to the exhaustive approach, the time required by an AAC approach can be more or less arbitrary. Indeed, a great part of the time spent here by MO-ParamILS lies in the validation of the training solutions, whose number is generally small and does not increase, whereas considering a bigger search space only lengthen the training phase of the AAC mechanism, with a rate that can be arbitrarily determined. AAC configurators can handle search spaces of configurations many times bigger than the one considered in this paper (*e.g.*, ParamILS and MO-ParamILS optimising the performance of the CPLEX solver within up to 10^{48} valid configurations in Blot *et al.* [2]).

5 EXPERIMENTAL RESULTS

All the experiments have been conducted in parallel on the 24 cores of two (12 × 1.9GHz, 32GB RAM) Intel E5-2609v3 machines.

On the set of 20-job training instances, the 30 MO-ParamILS runs have resulted in 38 unique configurations, that have thus been reassessed on the full training test. Of these 38 configurations, a single one was non-dominated on the classical Taillard instances. In total, the training, reassessment and test phases then account for $30 \times T_{\text{training}} + 38 \times T_{\text{reassess}} + 1 \times T_{\text{test}} \approx 1$ day and 22 hours . On the set of 50-job training instances, the 30 MO-ParamILS runs have resulted in 41 unique configurations, 5 of them being non-dominated on the validation set. The total time is then approximately 7 day and 5 hours. Finally, on the set of 100-job training instances, the MO-ParamILS runs have resulted in 59 unique configuration, 4 of them being non-dominated on the validation set. The total time is then approximately 15 days and 4 hours. In comparison, the exhaustive approach was much longer and took approximately 46 days, 115 days, and 230 days on the three validation sets, respectively.

Figures 1, 2 and 3 show the final quality of the 189 feasible configurations described in Table 1 on the three validation sets of Taillard instances , respectively. This final quality is obtained by computing the arithmetic means of both hypervolume and Δ spread of the 900 runs performed by every configuration (30 runs on all 30 Taillard instances per validation set). Pluses indicate the final quality of every configuration as computed by the exhaustive approach. Red triangles show Pareto optimal configurations, whereas blue circles show the MO-ParamILS final configurations. Details of the Pareto optimal configurations and MO-ParamILS final configurations are given in the top and bottom half of Tables 2, 3, and 4. We show these configurations sorted according to their quality in the corresponding validation Taillard instances. Additionally, a check mark (✓) in the P column means that a configuration found by MO-ParamILS is Pareto optimal. Note that we choose to associate the `all` exploration strategy with the `arch` reference point, even if it does not use one in practice, as it always explores and selects all (non-dominated) neighbours; however, this is in philosophy equivalent to using directly the current archive as reference point.

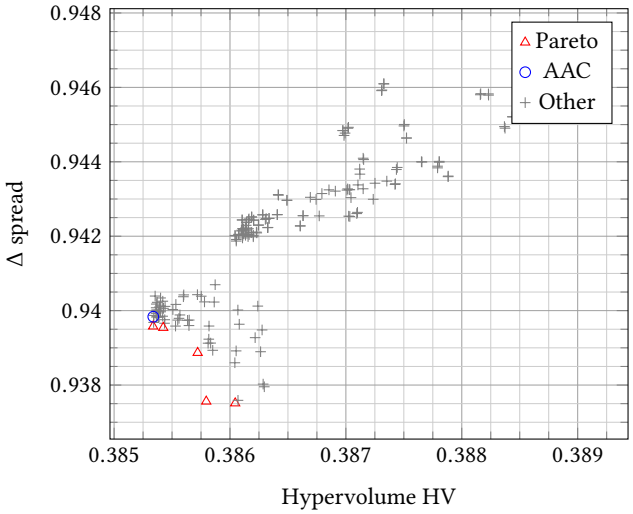

Figure 1: Exhaustive performance – 20-job Taillard instances

Table 2: Optimised configurations – 20-job Taillard instances

HV	Δ	Init	Selection		Exploration		P
Pareto optimal configurations							
0.385337	0.939578	`ig`	`oldest`	1	`imp`	1	`pick`
0.385424	0.939541	`ig`	`newest`	3	`imp`	1	`arch`
0.385721	0.938868	`ig`	`newest`	3	`ndom`	1	`arch`
0.385795	0.937560	`ig`	`newest`	3	`ndom`	3	`arch`
0.386045	0.937512	`ig`	`newest`	1	`ndom`	1	`arch`
AAC configurations							
0.385338	0.939837	`ig`	`rand`	3	`imp`	3	`pick`

On the easier instance set of Taillard instances with $N = 20$ jobs (Figure 1, Table 2), the feasible MOLS configurations are well separated into two separate clusters, one almost completely dominating the other. The exhaustive analysis found 5 optimal configurations, whereas MO-ParamILS resulted in a single, non-optimal, configuration. However, the hypervolume of this configuration is very good, and Figure 1 shows that it is, although non-optimal, very close to the optimal front.

On the instance set of Taillard instances with $N = 50$ jobs (Figure 2, Table 3), the feasible MOLS configurations are separated into two separate clusters and few configurations with a good hypervolume and an excellent spread. The exhaustive analysis found 10 optimal configurations, whereas MO-ParamILS resulted in 5 configurations. 4 of these 5 configurations corresponds to the Pareto front of the better cluster, while the 5th configuration correspond to one of the few configurations outside the two clusters, and achieved the optimal spread. A part of the optimal Pareto front was not found by MO-ParamILS (see Figure 3 the configurations with hypervolume ≈ 0.344 and Δ spread ≈ 0.947); however, the configurations within being well separated from the other could explain why they were much harder to reach for a neighbourhood-based configurator such as MO-ParamILS.

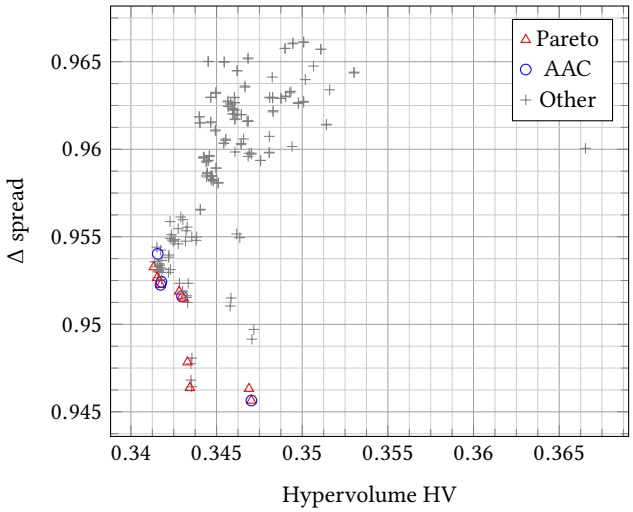

Figure 2: Exhaustive performance – 50-job Taillard instances

Table 3: Optimised configurations – 50-job Taillard instances

HV	Δ	Init	Selection		Exploration		P	
Pareto optimal configurations								
0.341332	0.953271	ig	oldest	3	imp	3	pick	
0.341530	0.952665	ig	rand	3	imp	3	pick	
0.341728	0.952256	ig	all	-	imp	1	arch	
0.342817	0.951881	ig	newest	3	ndom	3	pick	
0.342983	0.951602	ig	all	-	all	-	arch	
0.343086	0.951471	ig	rand	1	ndom	1	arch	
0.343305	0.947833	ig	newest	3	ndom	3	arch	
0.343443	0.946379	ig	newest	3	ndom	1	arch	
0.346891	0.946321	ig	oldest	3	ndom	1	arch	
0.347032	0.945645	ig	oldest	3	ndom	3	arch	
AAC configurations								
0.341565	0.954026	ig	oldest	1	imp	3	pick	
0.341728	0.952256	ig	all	-	imp	1	arch	✓
0.341778	0.952433	ig	all	-	imp	3	arch	
0.342983	0.951602	ig	all	-	all	-	arch	✓
0.347032	0.945645	ig	oldest	3	ndom	3	arch	✓

On the much more challenging instances of Taillard instances with $N = 100$ jobs (Figure 3, Table 4), the feasible MOLS configurations are separated into a single cluster and few other separate configurations. The exhaustive analysis found 7 optimal configurations, whereas MO-ParamILS resulted in 4 configurations. Of these 4 configurations, the first is not optimal, but achieved an excellent hypervolume, the next two are optimal, one in the main cluster of solutions and the other being one a the separate configurations. At last, the last configurations achieved a very poor hypervolume, but is excellent regarding the Δ spread.

In overall, the AAC approach always lead to excellent configurations, either optimal or dominated by very few other configurations. These configurations spread well over the real optimal Pareto front

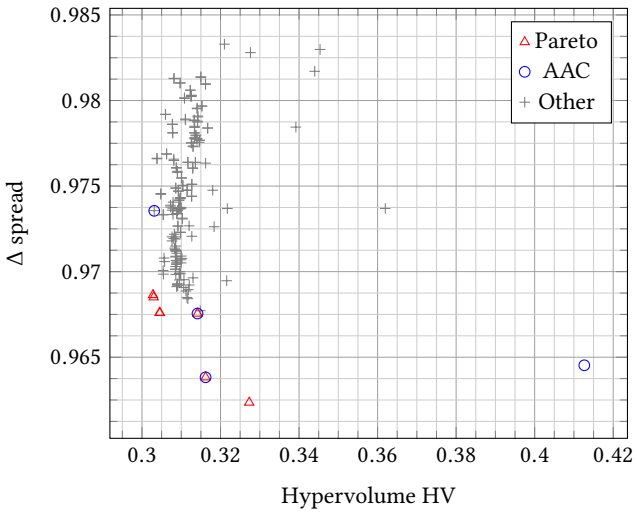

Figure 3: Exhaustive performance – 100-job Taillard instances

Table 4: Optimised configurations – 100-job Taillard instances

HV	Δ	Init	Selection		Exploration		P	
Pareto optimal configurations								
0.302815	0.968639	ig	newest	1	ndom	1	pick	
0.302981	0.968505	ig	newest	1	ndom	3	pick	
0.304468	0.967600	ig	newest	1	ndom	3	arch	
0.304574	0.967595	ig	newest	1	ndom	1	arch	
0.314142	0.967553	ig	oldest	3	all	-	arch	
0.316194	0.963828	ig	all	-	all	-	arch	
0.327318	0.962354	neh	all	-	all	-	arch	
AAC configurations								
0.303123	0.973554	neh	newest	1	ndom	3	pick	
0.314142	0.967553	ig	oldest	3	all	-	arch	✓
0.316194	0.963828	ig	all	-	all	-	arch	✓
0.412647	0.964528	rand	all	-	all	-	arch	

for the two sets of bigger instances, with a single configuration in the case of the set of smallest instances that can be explained by the performance similarity of the high-performing configurations induced by the easier instances.

Regarding the optimal configurations and the final MO-ParamILS configurations, they greatly differ for each set of instances, with some notable similarities. First, the IG initialisation was used on virtually all optimal configurations, with a single configuration using the NEH initialisation, despite it exposing many parameters (*e.g.*, number of aggregation, running time, parameters of the different inner IG algorithms) being not considered here. There is no doubt even better results can be achieved by optimising these additional parameters for each set of instances, and by considering other complex and similarly parametric initialisation mechanisms (*e.g.*, TPLS [8]). Most of the optimal configurations use the `newest` and `oldest` selection strategies, rather than the classical `rand` and `all` strategies found in the literature. As for the explorations strategies,

on both sets of 20-job and 50-job instances, better hypervolume is achieved using the `imp` strategy while a better spread is achieved with the `ndom` strategy. On the set of bigger 100-job instances, a similar conclusion is reached regarding the `ndom` and the `all` strategies. At last, a vast majority of the final configurations use the current archive as reference point during the exploration, unlike most non-exhaustive exploration strategies of the literature, which use the current solution as reference point. However, we note that on every set of instances, the optimal configuration regarding the hypervolume always use the picked solutions (*i.e.*, here, the current solution) as reference point.

6 CONCLUSION

In this paper, we presented a method to automatically design MOLS algorithms for combinatorial optimisation problems.

First, we proposed a new MOLS generalisation, unifying existing MOLS algorithms with new ideas and designed to easily include new components and strategies. Our structure exposes numerous components, strategies and parameters, that must be tuned for specific classes of problem instances in order to achieve increased performance. As such, the question of using AAC mechanisms to automatically tune this structure arises naturally. We investigated the performance of automatically designed MOLS algorithms, in comparison to the performance obtained by exhaustively analysing all feasible MOLS configurations. We used three sets of instances of a well-known bi-objective permutation flowshop scheduling problem, and MO-ParamILS, a multi-objective AAC procedure.

Our experiments showed that for the three sets of instances the AAC procedure was able to systematically design efficient MOLS algorithms, matching the performance of optimal configurations. Moreover, the exhaustive analysis required to constrain the feasible configurations considered to a small subset of the possible MOLS configurations. AAC procedures being typically not constrained by the size of the search space, we strongly believe that similar results would be achieved when considering many more feasible configurations.

Based on our results, we conclude in the strong promises of using an AAC procedure to automatically design MOLS algorithms for given classes of problem instances. Furthermore, we believe that this promise also applies to other types of highly parametric multi-objective metaheuristics. Regarding the MOLS structure itself, we believe that including strategies from other algorithms, such as for example evolutionary algorithms mechanisms, could enable the design of strong hybrid metaheuristics. Finally, the performance of automatically designed (hybrid-)MOLS algorithms should also be investigated on problems where manually designed metaheuristics already perform well (*e.g.*, scheduling problems, routing problems) and compared to the performance of state-of-the art algorithms.

REFERENCES

[1] Aymeric Blot, Hernán E. Aguirre, Clarisse Dhaenens, Laetitia Jourdan, Marie-Eléonore Marmion, and Kiyoshi Tanaka. 2015. Neutral but a Winner! How Neutrality Helps Multiobjective Local Search Algorithms. In *EMO 2015*. 34–47.

[2] Aymeric Blot, Holger H. Hoos, Laetitia Jourdan, Marie-Éléonore Marmion, and Heike Trautmann. 2016. MO-ParamILS: A Multi-Objective Automatic Algorithm Configuration Framework. In *LION 10*.

[3] Aymeric Blot, Alexis Pernet, Laetitia Jourdan, Marie-Eléonore Kessaci-Marmion, and Holger H. Hoos. 2017. Automatically Configuring Multi-Objective Local Search Using Multi-Objective Optimisation. In *EMO 2017*.

[4] Kalyanmoy Deb. 2001. *Multi-objective optimization using evolutionary algorithms*. Vol. 16. John Wiley & Sons.

[5] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[6] Madalina M Drugan and Dirk Thierens. 2010. Path-guided mutation for stochastic Pareto local search algorithms. In *PPSN 11*. Springer, 485–495.

[7] Mădălina M. Drugan and Dirk Thierens. 2012. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics* 18, 5 (2012), 727–766.

[8] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. 2011. A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research* 38, 8 (2011), 1219–1236.

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

[10] Holger H. Hoos and Thomas Stützle. 2004. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

[11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION 5*. 507–523.

[12] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36 (2009), 267–306.

[13] Joshua D. Knowles and David Corne. 2000. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation* 8, 2 (2000), 149–172.

[14] Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, and El-Ghazali Talbi. 2012. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics* 18, 2 (2012), 317–352.

[15] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58.

[16] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2003. Iterated local search. In *Handbook of metaheuristics*. Springer, 320–353.

[17] Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle. 2013. Automatic Design of Hybrid Stochastic Local Search Algorithms. In *HM 2013*. 144–158.

[18] Gerardo Minella, Rubén Ruiz, and Michele Ciavotta. 2008. A Review and Evaluation of Multiobjective Algorithms for the Flowshop Scheduling Problem. *INFORMS Journal on Computing* 20, 3 (2008), 451–471.

[19] Laurent Moalic, Alexandre Caminada, and Sid Lamrous. 2013. A Fast Local Search Approach for Multiobjective Problems. In *LION 7*. 294–298.

[20] Muhammad Nawaz, E Emory Enscore, and Inyong Ham. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* 11, 1 (1983), 91–95.

[21] Luis Paquete, Marco Chiarandini, and Thomas Stützle. 2004. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*. Springer, 177–199.

[22] Eric Taillard. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64, 2 (1993), 278–285.

[23] Tiantian Zhang, Michael Georgiopoulos, and Georgios C. Anagnostopoulos. 2015. SPRINT Multi-Objective Model Racing. In *GECCO 2015*. 1383–1390.

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