MO-ParamILS: A Multi-objective Automatic Algorithm Configuration Framework

Aymeric Blot[1,2,3(✉)], Holger H. Hoos[3(✉)], Laetitia Jourdan[1], Marie-Éléonore Kessaci-Marmion[1], and Heike Trautmann[4]

[1] Université de Lille, Inria, CNRS, UMR 9189 – CRIStAL, Lille, France
aymeric.blot@inria.fr,
{laetitia.jourdan,me.kessaci}@univ-lille1.fr
[2] École Normale Supérieure de Rennes, Rennes, France
[3] University of British Columbia, Vancouver, BC, Canada
hoos@cs.ubc.ca
[4] University of Münster, Münster, Germany
trautmann@wi.uni-munster.de

Abstract. Automated algorithm configuration procedures play an increasingly important role in the development and application of algorithms for a wide range of computationally challenging problems. Until very recently, these configuration procedures were limited to optimising a single performance objective, such as the running time or solution quality achieved by the algorithm being configured. However, in many applications there is more than one performance objective of interest. This gives rise to the multi-objective automatic algorithm configuration problem, which involves finding a Pareto set of configurations of a given target algorithm that characterises trade-offs between multiple performance objectives. In this work, we introduce MO-ParamILS, a multi-objective extension of the state-of-the-art single-objective algorithm configuration framework ParamILS, and demonstrate that it produces good results on several challenging bi-objective algorithm configuration scenarios compared to a base-line obtained from using a state-of-the-art single-objective algorithm configurator.

Keywords: Algorithm configuration · Parameter tuning · Multi-objective optimisation · Local search algorithms

1 Introduction

The performance of many algorithms strongly depends on the setting of their parameters. In particular, state-of-the-art solvers for prominent $\mathcal{NP}$-hard combinatorial decision and optimisation problems, such as propositional satisfiability, scheduling, vehicle routing and mixed integer programming critically rely on configurable heuristics whose parameters have a strong impact on the overall performance achieved on a given class of problem instances. In many cases, these parameters interact with each other in complex and non-intuitive ways,

P. Festa et al. (Eds.): LION 2016, LNCS 10079, pp. 32–47, 2016.
DOI: 10.1007/978-3-319-50349-3_3

and manually configuring them to optimise performance on a particular class of instances is a difficult and tedious task.

State-of-the-art automatic algorithm configuration procedures from the literature, such as ParamILS [6], SMAC [5] and irace [2], only consider a single performance objective when optimising the configuration of an algorithm; they are particularly frequently used for minimisation of running time or maximisation of solution quality. However, there are many situations in which multiple competing performance objectives matter when configuring a given algorithm, such as running time and memory consumption, and running time and solution quality, and there are well-known benefits in exploring the tradeoffs between such competing objectives.

To the best of our knowledge, automatic configuration for multiple performance objectives has been studied only recently, and only in the context of racing algorithms [13,14]. In this work, we introduce MO-ParamILS, an extension of the prominent ParamILS algorithm configuration framework [6,7] that allows us to deal with multiple performance objectives. Like the single-objective ParamILS framework that inspired it, MO-ParamILS implements two configuration procedures: MO-BasicILS and MO-FocusedILS. As we will demonstrate, both are effective in dealing with five bi-objective configuration scenarios, with MO-FocusedILS producing better results than MO-BasicILS.

The remainder of this paper is organised as follows. Section 2 describes the single- and multi-objective algorithm configuration problems and introduces definitions and notation required later. Section 3 presents the MO-ParamILS framework along with the MO-BasicILS and MO-FocusedILS multi-objective configuration procedures implemented within it. Section 4 presents an empirical evaluation of these configurators on five bi-objective configuration scenarios involving prominent solvers for MIP and SAT. Finally, Sect. 5 provides conclusions and an outlook on future work.

2 Automatic Algorithm Configuration

Automatic algorithm configuration deals with the optimisation of the performance of an algorithm through the automatic configuration of its parameters. In the following, we first describe the general context of algorithm configuration, before giving a formal definition of the algorithm configuration problem and its extension to a multi-objective setting.

2.1 General Context

Complex algorithms, especially ones for solving hard computational problems, often expose numerous parameters that can be optimised to achieve good performance in different application scenarios. General-purpose solvers for problems such as mixed integer programming (MIP) or propositional satisfiability (SAT) are usually designed to perform well across a broad range of instance types, but can be tuned manually for performance on particular sets of problem instances.

The algorithm configuration problem is an optimisation problem that aims at finding the best possible parameter configuration of a given algorithm w.r.t. its performance on a given set or distribution of problem instances. Note that when talking about methods for solving this algorithm configuration problem, there are two levels of algorithms involved: the *target algorithm* – a lower-level algorithm for some problem, such as MIP or SAT, whose performance is to be optimised, and the *configurator* – a higher-level algorithm used for optimising the performance of the target algorithm.

There is a sizeable literature on automatic algorithm configurators, including procedures based on sequential model-based optimisation, such as SMAC [5]; racing algorithms, such as irace [10,12]; and model-free search algorithms, such as CALIBRA [1] and ParamILS [6,7]. These all address the *single-objective* automatic algorithm configuration problem, where the performance of the target algorithm is assessed by a single scalar value, such as the running time or solution quality, or a fixed aggregation of multiple scalar values. Recently the idea of a more general *multi-objective* automatic algorithm configuration problem has begun to emerge, *e.g.*, in the work of Zhang *et al.* on multi-objective configurators based on racing [13,14].

On the other hand, good examples of target algorithms are metaheuristics for $\mathcal{NP}$-hard problems, or commercial solvers such as CPLEX with a broad range of parameters. During the development of such algorithms, automated configurators can be used to assess and optimise the performance of different design choices, as well as to find good default parameter settings.

Target algorithm parameters can be numerous and varied in their type and function. They can control low-level aspects of target algorithm behaviour, such as probabilities for certain types of operations, up to high-level aspects, such as computation strategies or problem representations. We distinguish three main types of parameters: *categorical parameters*, which have a finite number of unordered discrete values, often used to select between alternative mechanisms, *integer parameters*, which have discrete and ordered domains, and finally, *continuous parameters* that take numerical values on a continuous scale. In addition, *conditional parameters* exist that depend on the setting of other parameters, as well as *forbidden parameters*, which describe forbidden parameter combinations, to avoid known incorrect or undesirable behaviour of the target algorithm.

2.2 Problem Statement and Notations

The single-objective algorithm configuration problem is defined as a tuple $< \mathcal{A}, \Theta, \mathcal{D}, o, m >$, where

- $\mathcal{A}$ is the parameterised target algorithm,
- Θ is the search space of possible configurations of $\mathcal{A}$,
- $\mathcal{D}$ is a distribution of problem instances,
- o is a cost function, and
- m is a statistical population parameter.

A configuration $\theta \in \Theta$ is one possible setting of the parameters of $\mathcal{A}$. The cost function o is the performance objective for a single execution of algorithm $\mathcal{A}$ on an instance $\pi \in \mathcal{D}$, such as the final accuracy or the total running time. The statistical parameter m is used to aggregate the values of the cost function o over a set of instances, *e.g.*, the arithmetic mean or the median. The aggregated cost of one configuration $\theta \in \Theta$ of an algorithm $\mathcal{A}$ over all instances π from $\mathcal{D}$ is then defined as $c(\theta) := m(O(\theta))$, where $O(\theta)$ is the distribution of costs induced by the function o on $\mathcal{D}$.

The single-objective automatic algorithm configuration problem then consists of finding a configuration $\theta^* \in \Theta$ such that $c(\theta^*)$ is optimised. While in general, performance measures may be minimised or maximised, in the following, we assume (without loss of generality) that $c(\theta)$ is to be minimised.

Unfortunately, the cost $c(\theta)$ of a configuration θ often cannot be computed directly, as $\mathcal{D}$ is usually not finite or much too large to explore exhaustively. Usually, the cost is estimated based on a finite set of instances from $\mathcal{D}$. In this context, we use R to denote a list of runs of a given configurator and represent each run by a triple $< \theta, \pi, o >$, where

- $\theta \in \Theta$ is the configuration considered,
- π from $\mathcal{D}$ is the instance on which the target algorithm is evaluated,
- o is the observed cost of the run.

The *estimated cost* $\hat{c}(\theta)$ of a configuration θ given a sequence of runs R is then determined as the aggregate m over the cost o of all runs $< \theta, \pi, o >$ for some π from $\mathcal{D}$. If the target algorithm is stochastic, a configuration is combined with a specific random seed in order to ensure fair comparisons.

As a simple example, let us consider the configuration of a general SAT solver in an application scenario where we want to minimise the average running time for a certain type of SAT instances. By using a SAT solver that is specifically configured to achieve this, we can save time on instances to be solved in the future compared to using the solver in its default configuration, and eventually these time savings will exceed the effort required for configuring the solver. Here, the target algorithm $\mathcal{A}$ is the SAT solver, with configuration space Θ, and $\mathcal{D}$ is the distribution of SAT instances of interest. The cost function o reflects the solving time for a given SAT instance, and m is defined as the arithmetic mean.

However, in many cases, when optimising the performance of a given algorithm, there is more than one performance metric of interest, which gives rise to a multi-objective optimisation problem. To capture this, we consider an extension of the single-objective algorithm configuration problem that involves a vector of cost functions $o := (o_1, \ldots, o_n)$ where each o_i is a single-objective cost function, and a vector of statistical parameters $m := (m_1, \ldots, m_n)$. Theoretical cost vectors $c(\theta)$ and estimated cost vectors $\hat{c}(\theta)$ are defined based the component-wise scalar cost and estimated cost introduced earlier.

The multi-objective automatic algorithm configuration problem, given a dominance relation $\prec$ over configurations, then consists of finding a set of configurations $\Theta^* \subseteq \Theta$ such that no $\theta \in \Theta^*$ is dominated w.r.t. $\prec$ by any other $\theta' \in \Theta$.

In the following, the dominance relation $\prec$ we consider is Pareto dominance, *i.e.*, for $u := (u_1, \ldots, u_n)$ and $v := (v_1, \ldots, v_n)$, u is said to *dominate* v (denoted by $u \prec v$) if, and only if

$$\forall i \in \{1, \ldots, n\} : \; u_i \leq v_i \; \wedge \; \exists i \in \{1, \ldots, n\} : \; u_i < v_i$$

This relation is transferred to configurations by their costs: a configuration θ_1 dominates a configuration θ_2 iff $c(\theta_1)$ dominates $c(\theta_2)$. We will refer to a set of mutually non-dominated configurations as an *archive*. Adding the dominance relation to the multi-objective automatic algorithm configuration problem reflects the overall aim of generating configurations with performance characteristics (according to the given objectives) that are not dominated by any other configuration available. Thus, we are interested in (ideally all) existing tradeoff solutions. We note that, just as in the single-objective case, practical configurators may only find suboptimal solutions to a given configuration problem.

3 From ParamILS to MO-ParamILS

In this section, we first outline the existing single-objective ParamILS framework and then describe our new multi-objective framework, along with the two MO-ParamILS variants we study in the following, MO-BasicILS and MO-FocusedILS.

3.1 Single-Objective ParamILS

ParamILS [6] is an automatic algorithm configuration framework that optimises a single performance metric using iterated local search, a well-known stochastic local search method [11]. The configuration process starts by evaluating a given default configuration along with r configurations chosen uniformly at random from the given configuration space Θ. The best of these $r + 1$ configurations is used as the starting point for the iterated local search process, which can be seen as a sequence of three phases that is repeated until a given time budget is exhausted. Throughout the search process, we keep track of the *incumbent configuration* θ^*, *i.e.*, the best configuration seen so far. In the first phase, the current configuration θ is perturbed, by performing s random steps in the *one-exchange neighbourhood* (where two configurations are neighbours if, and only if, they differ by the value of a single parameter). In the second phase, randomised iterative first improvement local search is performed within the same neighbourhood, excluding all configurations that have been visited previously during the same local search phase. The local search process ends when all neighbours of a given configuration have been checked without achieving an improvement. If the configuration θ' thus obtained is better than the configuration θ from which the last perturbation phase was started, we set θ to θ' (and update the incumbent). To provide additional diversification to the search process and guarantee probabilistic approximate completeness, with a fixed probability $p_{restart}$, θ is reset to a configuration chosen uniformly at random from the entire space Θ.

We note that, in light of the usually high cost of evaluating configurations of the given target algorithm, ParamILS maintains a cache of the results from all target algorithm runs performed during the search process and only performs target algorithm runs after checking that the respective results are not available from that cache.

3.2 Multi-objective ParamILS

We now describe our multi-objective extension of the ParamILS framework. The main difference between ParamILS and MO-ParamILS (outlined in Algorithm 1) lies in the use of a multi-objective iterated local search process, in which an *archive* (*i.e.*, set of non-dominated configurations) is iteratively modified rather than a single configuration of the given target algorithm. Likewise, the incumbent is now an archive. Like ParamILS, MO-ParamILS exposes three parameters: the number r of initial random configurations, the number s of random search steps performed in each perturbation phase and the restart probability p_{restart}.

Algorithm 1. Multi-objective ParamILS

```
Data: Initial archive, algorithm parameters r, p_restart and s
Result: Archive of incumbents, i.e., overall best configurations found
current_arch ← initial archive;
for i ← 1...r do
    conf ← random configuration;
    update(conf, current_arch);
    archive(conf, current_arch);
until termination criterion is met do
    if first iteration then
        arch ← current_arch;
    else
        if with probability p_restart then // Restart
            conf ← random configuration;
            current_arch ← {conf};
            arch ← current_arch;
        else // Random sampling and random walk
            /* Incumbents are not forgotten between restarts */
            conf ← random configuration of current_arch;
            for i ← 1...s do
                conf ← random neighbour of conf;
            arch ← {conf};
    arch ← local_search(arch);
    foreach conf in arch do
        update(conf, current_arch);
        archive(conf, current_arch);
return the archive of incumbents;
```

Function 2. archive(new_conf, arch)

```
Data: Single configuration new_conf, archive arch
Result: Updated archive arch
foreach conf in arch do
    if dominates(new_conf, conf) then
        arch ← arch \ {conf};
    else if dominates(conf, new_conf) then
        return arch;
arch ← arch ∪ {new_conf};
return arch;
```

The initialisation of the search process does not change conceptually, except that an initial set of default configurations can be provided and is combined, with the r randomly chosen configurations, into an archive. We ensure that whenever we add a new configuration to an archive a, all Pareto-dominated configurations in a are discarded (see Function 2), so that an archive always contains only non-dominated configurations.

MO-ParamILS prominently uses the two following functions: `dominates()` and `update()`. The function `dominates()` compares two configurations using strict Pareto dominance on the respective cost (estimate) vectors. The function `update()` is, unless explicitly specified otherwise, the only function that runs the target algorithm and updates the cost vector of a configuration; it ensures that a given configuration can subsequently be compared to another configuration using `dominates()`. It also maintains a cache of all target algorithm runs performed throughout the multi-objective search process and ensures that the overall best configurations, the archive of incumbents, is kept up-to-date. We will discuss the instantiations of `update()` and `dominates()` for MO-BasicILS and MO-FocusedILS, the two MO-ParamILS variants we used in our experiments, later in this section.

We use a simple variant of the perturbation mechanism from ParamILS, in which a single configuration is selected uniformly at random from the current archive and modified by a sequence of s random search steps in the 1-exchange neighbourhood; the resulting configuration is then stored as a new archive, which forms the starting point of the subsequent local search phase [4]. The restart mechanism remains unchanged, except that it now replaces the current archive with one containing a single configuration chosen uniformly at random from the entire configuration space Θ. As in ParamILS, we use default values of $r := 10$, $p_{restart} := 0.01$, and $s := 3$ in our experiments.

The subsidiary local search process used in MO-ParamILS is outlined in Function 3. From a wide range of existing multi-objective local search procedures [9], we chose this one, because it is conceptually simple and resembles the subsidiary local search procedure used in ParamILS; it has also been shown to be very efficient [3]. At each step of the local search process, all configurations in the

Function 3. localSearch(init_arch)

```
Data: Initial archive of configurations init_arch
Result: Best archive of configurations found
Side effect: Change or update the incumbent if necessary

current_arch ← init_arch;
tabu_set ← current_arch;
repeat
    /* Selection */
    current_set ← current_arch;
    candidate_set ← ∅;
    foreach current in current_set do
        foreach neighbour in randomised neighbourhood of current do
            /* Exploration */
            if neighbour ∈ tabu_set then
                next ;
            tabu_set ← tabu_set ∪ {neighbour};
            update(neighbour, current);
            if dominates(neighbour, current) then
                candidate_set ← candidate_set ∪ {neighbour};
                break ;
            if not dominates(current, neighbour) then
                candidate_set ← candidate_set ∪ {neighbour};
    /* Archive */
    foreach conf in candidate_set do
        archive(conf, current_arch);
until candidate_set = ∅;
return current_arch;
```

current archive are explored individually. When exploring a configuration θ, its neighbours are evaluated in random order (excluding any configurations already visited earlier in the same local search phase), until one is found that strictly dominates θ or all neighbours have been visited. All non-dominated neighbours encountered during this process are added to the current archive, making sure that dominated solutions are removed. (Notice how this can be seen as a generalised version of the acceptance criterion used in the single-objective ParamILS framework.) The local search then stops when there is no more unvisited neighbour that can be added to the archive.

3.3 MO-BasicILS

The key idea behind BasicILS(n) is to evaluate configurations on a fixed set of n training instances, selected uniformly at random (without replacement) from the given training set $\mathcal{D}$ [7]. This can be easily carried over to the MO-ParamILS framework of Algorithm 1, by defining `update()` and `dominates()` the way that

the latter always compares configurations based on their quality vectors on the same instances set, and the former ensures that all target algorithm runs required in this context are performed.

The disadvantage of the resulting MO-BasicILS procedure, as in the case of BasicILS, lies in the difficulty of choosing n: if n is too small, solution quality estimates can be inaccurate, leading to poor generalisation of the performance of the configurations obtained from MO-BasicILS to unseen test instances; if n is too large, much effort is wasted on evaluating poorly performing configuration, compromising the efficiency of the search process. In our experiments, we used a default setting of $n := 100$.

3.4 MO-FocusedILS

The key idea behind FocusedILS is to avoid the potential problems arising from the use of a fixed number of instances for evaluating configurations by starting comparisons between configurations on a small initial set of instances and then increasing the number of instances as better and better configurations are found [7]. Based on the same idea, MO-FocusedILS allows poor configurations to be dominated very soon, while promising configurations are evaluated increasingly more accurately as the search process progresses.

Towards this end, MO-FocusedILS uses a slightly weaker dominance relation in the function `dominates()`, which adds the condition that a configuration θ dominates a configuration θ' if, and only if, θ has been run on every instance θ' has been run and θ dominates θ' on those instances. Note that when θ and θ' have been run on the same instances, this corresponds to standard Pareto domination, and as the number of instances grows, it approximates Pareto domination on the true (theoretical) cost vectors arbitrarily accurately.

In practice, new runs are performed for the configuration that has been evaluated on fewer instances so far, until either of the two configurations being compared dominates another; the instances for these new runs are chosen according to a random permutation of the training instance set that has been determined when initialising MO-FocusedILS and then remains fixed. This ensures that for two configurations θ and θ' and their respective sequences of runs R_θ and $R_{\theta'}$, either $R_\theta \subseteq R_{\theta'}$ or $R_\theta \supseteq R_{\theta'}$, and that $R_\theta \cap R_{\theta'}$ is either equal to R_θ or to $R_{\theta'}$.

The `update()` function of MO-FocusedILS handles the comparison of a single configuration θ and an archive a by adding new runs of θ until there is at least one configuration $\theta' \in a$ for which $R_\theta \supseteq R_{\theta'}$ or θ' dominates θ.

Like Focused-ILS, MO-FocusedILS additionally requires an intensification mechanism that ensures that over the course of the search process, good configurations are evaluated on an increasing number of instances. This mechanism is outlined in Procedure 4: it simply performs new runs for a given configuration until its new cost vector Pareto dominates its cost vector before intensification. Procedure 4 is called at the beginning of every local search phase, to help start the local search process with a better cost estimate, after every local search phase, to further increase the accuracy of cost estimates, and each time the

Procedure 4. intensify(conf)

Data: Single configuration `conf`
Side effect: Updates the level of detail of `conf`

repeat
  `old_cost` ← `cost(conf)`;
  perform a new run of `conf`;
  `new_cost` ← `cost(conf)`;
until *pareto_dominates(old_cost, new_cost)*;

`update()` function compares two configurations with the same number of runs. (Alternative, but less efficient intensification mechanisms might perform a fixed number of new runs, or a number of runs given by a function of the time spent since intensification was last performed.)

4 Experiments

In this section, we present results for MO-ParamILS for two different multi-objective automatic algorithm configuration problems. First, we study the trade-off between running time and solution quality for an anytime optimisation algorithm. Our second example involves the simultaneous optimisation running time and memory usage. In both cases, we consider two optimisation objectives; however, MO-ParamILS is not restricted to such bi-objective algorithm configuration problems.

4.1 Experimental Protocol

To assess the performance of MO-ParamILS we consider five configuration scenarios, described by Table 1. These scenarios use three datasets and two target algorithms, which belongs to ACLib[1], a comprehensive algorithm configuration library, and are already known and have been studied in single-objective algorithm configuration.

Details of the two algorithms are precised by Table 2. Note that the neighbourhood relation of ParamILS considers all parameters as categorical; henceforth for integer or continuous parameters the set of values have been discretised before all experiments.

Our experimental protocol involves three consecutive steps, namely *training*, *validation* and *test*. In the training step, the configurator is run 25 times on 25 different permutations of the training set, resulting in 25 archives. Because the configurations produced at the end of the training phase have not necessarily been evaluated on precisely the same training instances, in the validation step, each final configuration of the 25 training runs is reassessed on the same subset of the training instances. At the end of this validation phase, all configurations

[1] http://aclib.net

Table 1. Configuration scenarios.

Dataset	Algorithm	Walltime	Performance objectives	Abbrv
Regions200	CPLEX	1 day	Quality, Cutoff	RCut
Regions200	CPLEX	1 day	Quality, Running Time	RRun
CORLAT	CPLEX	1 day	Quality, Cutoff	CCut
CORLAT	CPLEX	1 day	Quality, Running Time	CRun
QUEENS	CLASP	1 day	Memory usage, Running Time	QUEENS

Table 2. Target algorithm parameters (with number of possible values).

Algorithm	Categorical	Integer	Continuous	Total configurations
CPLEX	5 (2)	65 (2–7)	2 (5–6)	$2.26 \cdot 10^{46}$
CLASP	15 (2–5)	43 (2–16)	8 (6–14)	$9.96 \cdot 10^{48}$

have been assessed on the same set of problem instances and can therefore be meaningfully compared in order to identify the ones that are Pareto-optimal w.r.t. performance objectives on solved problem instances and percentage of unsolved problem instances. Then, in the test step, the configurations of the archive obtained from the validation step are reassessed on a disjoint set of testing instances. We use this protocol to compare the configurations obtained from MO-BasicILS, from MO-FocusedILS, from an approach only using SO-FocusedILS, as well as the default configuration. In each of these cases, we use the same 25 permutations of the 1000 training instances, the same subset of 100 training instances for the validation, and the same 1000 testing instances.

Regarding MO-BasicILS, its parameter n is set to 100, meaning that estimations of configuration performance use 100 training instances. Regarding the SO approach, we used SO-FocusedILS with every available improvement (*e.g.*, aggressive capping). For the four CPLEX scenarios, we ran SO-FocusedILS separately on the 5 different cutoff values chosen to obtain a total wall-clock time of one day (that is, for 1, 2, 3, 5 and 10 CPU seconds cutoffs, the walltime for the 1 CPU second cutoff is $^1/_{(1+2+3+5+10)} \times 24\,\text{h}$). For the CLASP scenario, we ran SO-FocusedILS separately on each of the two objectives for 12 hours.

In the CLASP scenario, failure by CLASP to find a solution within 300 seconds in a particular instance is penalised by counting any such run as 10 times the cutoff time (*i.e.*, using the well-known PAR10 performance metric [6]). In the CPLEX scenarios, we penalised failure by CPLEX to return a MIP gap value by setting the MIP gap value to 10^{10} for such runs, thus making sure that such configurations tend to be avoided by our configuration approaches.

Performance assessment has been carried out using the PISA framework [8]. For the CPLEX scenarios, we used the data without timeout. For validation and test steps, the final fronts are compared using the hypervolume and ε indicators. First, all fronts for a given step and scenario are normalised so the values of every

objective vector lie in the interval $[1, 2]$. Then, a reference front is computed by merging every front and applying Pareto dominance. The indicator values are then computed between each front and the reference front.

SPRINT-Race [14] is a recent multi-objective racing algorithm, and we originally considered including it in our performance comparison. However, both CPLEX and CLASP algorithms have very large configuration spaces (10^{46} and 10^{48} configurations, respectively), which implies that the only way to apply SPRINT-Race would be in combination with a sampling technique. Furthermore, the implementation of SPRINT available from its authors requires as input the exhaustive evaluation of all configurations on all instances, making it impractical to use for our configuration scenarios.

4.2 Results

Empirical results from the test phases are shown in Fig. 1, considering only instances solved before the given timeout. The corresponding number of unsuccessful runs are given in Table 3. Table 4 shows the performance assessment for test results for both indicators. For each scenario, the best value is highlighted.

As can be seen from Table 4, MO-FocusedILS finds considerably better Pareto fronts for the test sets of all our multi-objective configuration scenarios than our baseline single-objective approach in terms of hypervolume and ε indicator. In all but one case, MO-FocusedILS also produces better results than MO-BasicILS,

Table 3. Average percentages of timeouts for final CPLEX configurations.

	Validation				Test			
Approach	RCut	RRun	CCut	CRun	RCut	RRun	CCut	CRun
MO-FocusedILS	1.3	0.7	4.2	3.6	0	0	1.06	2.89
MO-BasicILS	0.1	0.6	3.6	2.9	0.04	0	0.47	3.78
SO approach	0.3	0.4	4.8	5.1	0.12	0	1.87	1.87
Default	0	0	2.2	2.2	0	0	0.14	0.14

Table 4. Hypervolume (top) and ε indicator values (bottom) for final test fronts.

Approach	RCut	RRun	CCut	CRun	Queens
MO-FocusedILS	9.02e−03	**2.07e−03**	**2.37e−02**	**7.63e−04**	**1.57e−02**
MO-BasicILS	**2.46e−03**	5.41e−02	5.53e−02	1.02e−01	5.49e−02
SO approach	3.82e−02	5.82e−02	3.35e−01	1.72e−01	3.04e−02
Default	2.43e−01	3.57e−01	2.70e−01	5.30e−01	1.08e+00
MO-FocusedILS	**1.44e−02**	**9.05e−03**	**9.00e−02**	**8.06e−04**	**2.64e−02**
MO-BasicILS	1.80e−02	1.71e−01	1.11e−01	1.48e−01	8.35e−02
SO approach	5.77e−02	1.38e−02	3.33e−01	1.42e−01	6.52e−02
Default	2.22e−01	2.69e−01	2.33e−01	3.90e−01	1.00e+00

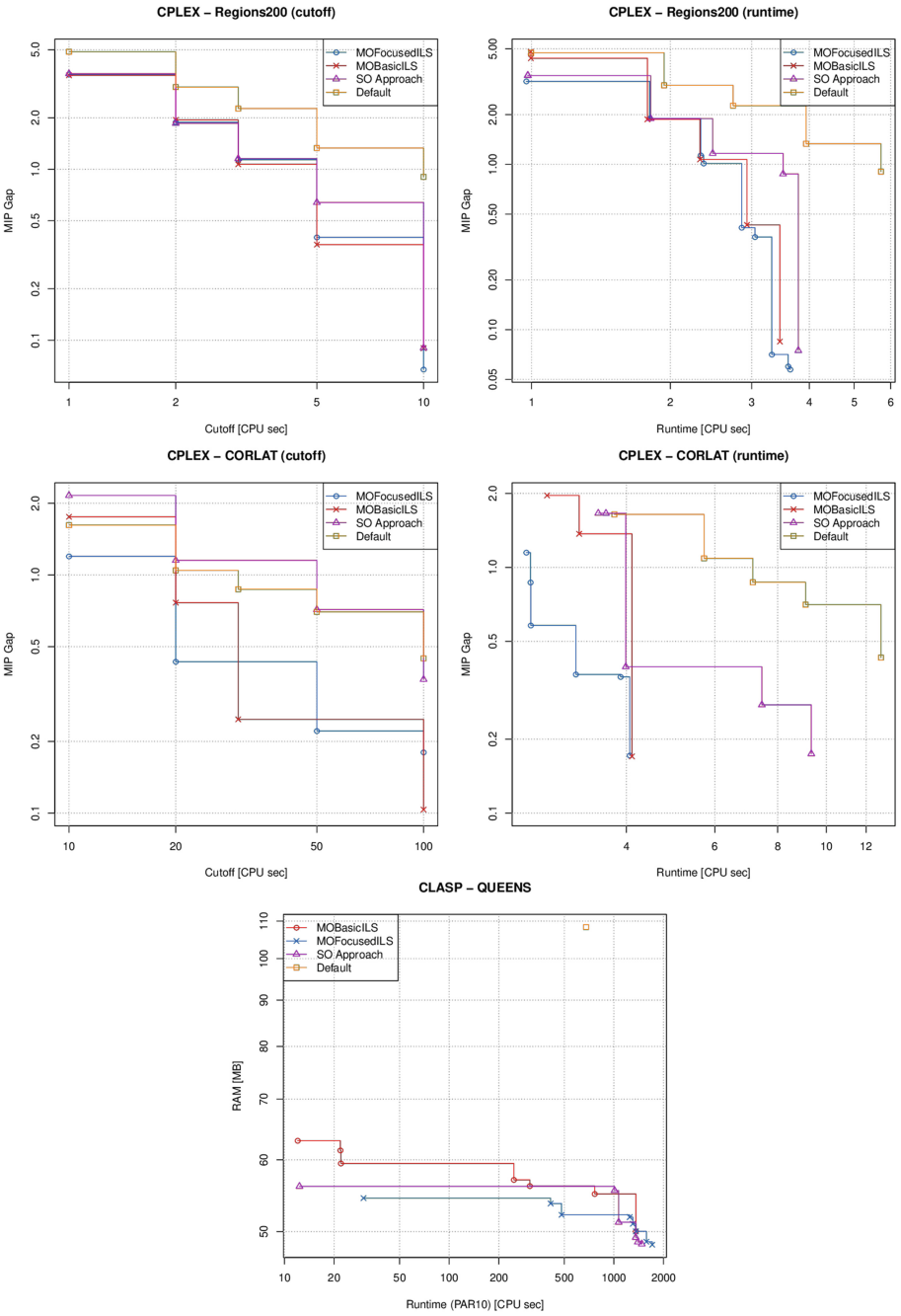

Fig. 1. Final test fronts for all five scenarios

which, in most cases, still produces better results than the single-objective approach, but with less of a margin. Figure 1 provides additional details by showing the Pareto fronts for all three multi-objective configuration approaches as well as the performance (trade-off) achieved by the default configuration; we note that the latter still produces a trade-off curve, because of the anytime nature of CPLEX.

When analysing these results, we noticed that MO-FocusedILS evaluates many more unique configurations than MO-BasicILS (4752 *vs* 166 on average, over all five scenarios). This clearly indicates the efficacy of the way in which MO-FocusedILS controls the number of runs per configuration performed and mirrors analogous findings for BasicILS *vs* FocusedILS in the single-objective case [6].

On all five scenarios, the default configuration of CPLEX or CLASP produce few unsuccessful runs on training or test instances. Our three approaches lead to configurations generating about as many timeouts as the default configuration. However, by also taking in account the configurations returned that have both more timeouts and better performances on successful instances, we were able to achieve even better results at the cost of a small loss of generality, as shown in Table 3. While our CLASP scenario uses PAR10 scores to take into account instances that could not be solved within the given cutoff time, as previously mentioned, the final Pareto fronts we produce for the CPLEX scenarios do not reflect a small number of instances for which no MIP gap was obtained within the allotted running time. The fraction of the validation and test sets on which this happened is shown in Table 3; as seen there, timeouts generally occur for a small fraction of instances, and while that fraction tends to increase as we configure CPLEX, it remains low enough in all cases to not raise serious concerns.

5 Conclusion

We have introduced MO-ParamILS, an extension of the prominent ParamILS automatic algorithm configuration framework for solving the multi-objective algorithm configuration problem. To the best of our knowledge, while MO-ParamILS is not the first multi-objective algorithm configurator, it is the first to be able to effectively deal with the highly-parameterised target algorithms usually considered in standard, single-objective algorithm configuration scenarios, as demonstrated in our experiments on five bi-objective configuration scenarios involving CPLEX and clasp, two prominent solvers for mixed integer programming (MIP) and propositional satisfiability (SAT) problems, respectively.

As is the case for their single-objective analogues, MO-FocusedILS typically performs better tha MO-BasicILS, but both approaches are able to produce sets of non-dominated configurations that cover an interesting range of trade-offs in all five scenarios we studied, and were considerably more effective in doing so than a base-line approach using a state-of-the-art single-objective configurator.

We believe that automatic multi-objective configurators, such as MO-FocusedILS, will be very useful in many application situations where there is

no clear and obvious way to trade off multiple performance criteria for a given target algorithm. In future work, it might be interesting to apply multi-objective configuration to multi-objective optimisation procedures as target algorithms; these are notoriously difficult to configure, and we believe that doing so automatically, based on multiple performance objectives, could be quite attractive. It would also be interesting to exploit the potential for parallelisation inherent in the MO-ParamILS framework; while using standard configuration protocols, the current version of MO-ParamILS can exploit parallel computing resources (just like single-objective ParamILS), there is considerably more room for easy parallelisation in the multi-objective extension presented here. Furthermore, we believe that it might be interesting to explore advanced methods for ensuring effective coverage of the true tradeoff curves (or surfaces) of a given multi-objective configuration scenario within the MO-ParamILS framework.

Finally, we are interested in exploring multi-objective extensions of sequential model-based algorithm configuration methods, in particular SMAC [5]. We also see potential value in effective multi-objective extensions of configuration procedures such as irace [2].

References

1. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Oper. Res. **54**(1), 99–114 (2006)
2. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Berlin (2010)
3. Blot, A., Aguirre, H., Dhaenens, C., Jourdan, L., Marmion, M.-E., Tanaka, K.: Neutral but a winner! How neutrality helps multiobjective local search algorithms. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C.C. (eds.) EMO 2015. LNCS, vol. 9018, pp. 34–47. Springer, Heidelberg (2015). doi:10.1007/978-3-319-15934-8_3
4. Geiger, M.J.: Foundations of the Pareto iterated local search metaheuristic. CoRR abs/0809.0406 (2008). http://arxiv.org/abs/0809.0406
5. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25566-3_40
6. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. JAIR **36**, 267–306 (2009)
7. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: AAAI 2007, pp. 1152–1157 (2007)
8. Knowles, J., Thiele, L., Zitzler, E.: A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, revised version (2006)
9. 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. J. Heuristics **18**(2), 317–352 (2012)

10. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical report, TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
11. Lourenço, H., Martin, O., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics, vol. 2, pp. 363–397. Springer, New York (2010)
12. Marmion, M.-E., Mascia, F., López-Ibáñez, M., Stützle, T.: Automatic design of hybrid stochastic local search algorithms. In: Blesa, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (eds.) HM 2013. LNCS, vol. 7919, pp. 144–158. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38516-2_12
13. Zhang, T., Georgiopoulos, M., Anagnostopoulos, G.C.: S-Race: a multi-objective racing algorithm. In: GECCO 2013, pp. 1565–1572 (2013)
14. Zhang, T., Georgiopoulos, M., Anagnostopoulos, G.C.: SPRINT multi-objective model racing. In: GECCO 2015, pp. 1383–1390 (2015)