Configuration of a Dynamic MOLS Algorithm for Bi-objective Flowshop Scheduling

Camille Pageau[1], Aymeric Blot[1], Holger H. Hoos[2], Marie-Eléonore Kessaci[1], and Laetitia Jourdan[1]

[1] Université de Lille, CNRS, UMR 9189 – CRIStAL, France
{camille.pageau,aymeric.blot,mkessaci,laetitia.jourdan}@univ-lille.fr
[2] LIACS, Leiden University, The Netherlands
hh@liacs.nl

Abstract. In this work, we propose a dynamic multi-objective local search (MOLS) algorithm whose parameters are modified while it is running and a protocol for automatically configuring this algorithm. Our approach applies automated configuration to a static pipeline that sequentially runs multiple configurations of the MOLS algorithm. In a series of experiments for well-known benchmark instances of the bi-objective permutation flowshop scheduling problem, we show that our dynamic approach produces substantially better results than static MOLS, and that longer pipeline (with a higher number of parameters) outperform shorter ones.

Keywords: Algorithm Configuration · Multi-Objective Combinatorial Optimisation · Local Search.

1 Introduction

Many metaheuristic algorithms for solving multi-objective optimisation problems have parameters that highly affect their performance, and that should be set to different values to achieve good performance for various types of problem instances. The problem of configuring such parameters for optimised performance can be approached in an off-line or on-line manner. Static algorithm configuration approaches can handle many parameters but provide configurations that can be highly specific to a given set or distribution of problem instances (see, e.g., [10,14]). Dynamic configuration approaches adapt parameters during the run of a given algorithm but generally consider only one or two parameters (see, e.g., [11]); they can, in principle, achieve robust performance over a broad range of problem instances. In this work, we leverage the advantages of both types of approaches by considering a framework in which we switch between different configurations of a multi-objective optimisation algorithm while it is running on a given problem instances. We determine these configurations, and the static schedule we use for switching between them, using a general-purpose, static algorithm configurator. Our approach thus represents a simple mechanism

for dynamically changing many algorithm parameters in a way that optimises overall performance on a given type of problem instances.

The bi-objective permutation flowshop scheduling problem (bPFSP), in which makespan and total flowtime are to be minimised, is a prominent and widely studied combinatorial multi-objective optimisation problem. The bPFSP can be solved effectively by multi-objective local search (MOLS) algorithms [6,12]; in the design of these algorithms, multiple design choices are encountered, and when using them, several parameters have to be set. Therefore, MOLS algorithms for the bPFSP provide an excellent test bed for our dynamic configuration approach.

The remainder of this article is organised as follows. First, in Section 2, we introduce our dynamic algorithm framework and a protocol to automatically configure it. Then, in Section 3, we describe the multi-objective local search algorithm. Sections 4 and 5 detail the setup of our experimental study and the results obtained from it, respectively. Finally, Section 6 provides some conclusions and perspectives on future work.

2 Automatic Design of a Dynamic Algorithm

2.1 Static *vs* Dynamic Design Approaches

Over the last decade, automatic algorithm configuration (AAC) techniques have been increasingly exploited in the off-line design of high-performance heuristic algorithms, such as metaheuristics. These algorithms present design choices, such as strategy components, and tunable parameters that heavily affect their performance. In the following, we will assume that all design choices have been exposed as parameters.

Given a parametrised *target algorithm* $\mathcal{A}$, a *configuration* θ is a specific setting of all the parameters of $\mathcal{A}$. The configuration space Θ of $\mathcal{A}$ is the set of all valid configurations. Automated algorithm configuration (AAC) can be seen as an optimisation problem, where the objective is to determine one or more configurations that lead to the best performance for a given set or distribution of problem instances. AAC can be seen as a supervised, off-line learning process, in which training instances are used to learn and determine the best configurations of the given target algorithm. This configuration is then fixed and used, in a completely static manner, whenever $\mathcal{A}$ is run on new problem instances. Prominent AAC procedures include irace [14] and ParamILS [10], which optimise a single configuration objective, and MO-ParamILS, a recent extension of ParamILS that handles multiple configuration objectives [1].

In parallel with AAC procedures, *dynamic* algorithm design techniques have been proposed [11] to permit the modification of strategy components or numerical parameters of a given target algorithm $\mathcal{A}$ while it is running. These so-called parameter control approaches use techniques such as multi-armed bandits [8] or adaptive pursuit [19] to dynamically determine good parameter settings in response to observations made while trying to solve a given problem instance. However, the number of configurations of $\mathcal{A}$ that can be handled by such approaches is very limited.

In this work, we are interested in algorithms that expose several design choices, in the form of categorical parameters. This scenario falls outside of most dynamic design scenarios, as they usually deal with a single numerical parameter or very few categorical choices. Nevertheless, we want to be able to dynamically modify parameters while running our target algorithm, and to this end, we introduce a framework that successively runs several configurations, in the form of a static pipeline, which we configure using a standard, general-purpose AAC procedure.

2.2 A Dynamic Algorithm Framework

Given a configurable algorithm $\mathcal{A}$ and its configuration space Θ, we use $\mathcal{A}_{\theta,T}$ to denote $\mathcal{A}$ under configuration $\theta \in \Theta$ with cut-off time T. Then, we define the dynamic algorithm $\mathcal{F}^{\mathcal{A}}_{(\theta_i,T_i)^k}$ as a pipeline with k stages, which sequentially runs $\mathcal{A}_{\theta_1,T_1}, \mathcal{A}_{\theta_2,T_2}, \ldots, \mathcal{A}_{\theta_k,T_k}$. Specifically, when applied to a multi-objective optimisation problem, we first run $\mathcal{A}$ under configuration θ_1, starting from a initial set of solutions, up to time T_1. At that point, we switch to configuration θ_2 and continue our computation from the current set of solutions, with a cut-off time of T_2. We note that $\mathcal{A}$ is *not* restarted when switching between configurations. Overall, the maximum running time of the dynamic algorithm is then $T = \sum_{i=1}^{k} T_i$.

Figure 1 depicts two examples of dynamic algorithms $\mathcal{F}^{\mathcal{A}}$ and $\mathcal{F}'^{\mathcal{A}}$. While $\mathcal{F}$ uses $k = 3$ configurations to divide the total time budget into three intervals of equal duration, $\mathcal{F}'$ uses $k = 4$ configurations, of which two are run quickly in the beginning, after which more time is allocated to last two configurations.

The configuration space of our framework comprises the Cartesian product Θ^k, the time budgets $T_1, \ldots, T_k$ and the integer $k \geq 1$. For $k = 1$, our framework degenerates to the original, static target algorithm $\mathcal{A}$.

2.3 Automatic Configuration of our Framework

The purpose of this work is to assess the performance gains that can be obtained by switching between different configurations of an algorithm $\mathcal{A}$ while it is running. Towards this end, we use a general-purpose, static algorithm configurator to configure the framework introduced in the previous section. Since the size of the configuration space exponentially increases with the maximum number of pipeline stages, K, we only consider a fixed number s_k of different cut-off times for each stage, where k is the number of actual pipeline stages used in a specific instantiation of our framework. This leads to a configuration space of size $\sum_{k=1}^{K} s_k \cdot |\Theta|^k$. Using this approach, we can also assess the influence of K and s_k (for $k = 1, \ldots, K$) on the performance achieved by automatically configuring our dynamic algorithm framework.

2.4 Related Work

In addition to being conceptually related to adaptive algorithms or hyper-heuristics, since it enables modifications of the configuration of an algorithm while it is

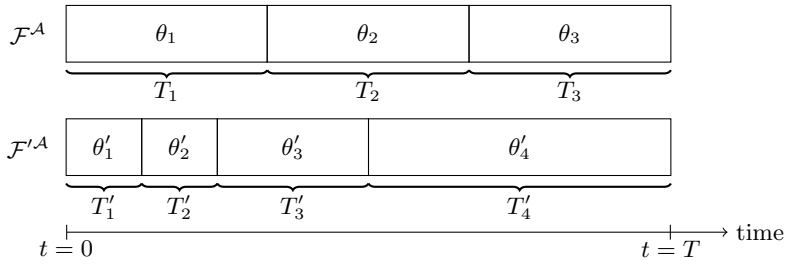

Fig. 1. Two examples of dynamic algorithms, $\mathcal{F}^{\mathcal{A}}$ and $\mathcal{F}'^{\mathcal{A}}$

running, our approach also bears resemblance to per-instance algorithm scheduling [13]. There are, however, several major differences. Firstly, per-instance algorithm scheduling uses instance features to determine which of a given set of distinct algorithms to run, one after the other, on a given problem instance; in contrast, our approach uses different configurations of a single algorithm and does not require instance features. Secondly, in per-instance algorithm scheduling, results are not passed from one stage of the schedule to the next, while in our pipeline approach, each stage continues from the result of the previous stage – as explained previously, it can thus be seen as a single algorithm whose parameter configuration changes while running on a given problem instance. Finally, the primary goal of per-instance algorithm scheduling is *robustness* resulting from performance complementarity between the algorithms in the schedule; the goal of our approach is to achieve improvements over the performance of the static version of the given target algorithm, which uses a single configuration for the entire run, based on the idea that different configurations are best suited for different phases of solving a given problem instance.

3 Multi-objective Local Search

In the following, we consider a Pareto optimisation approach to solve the bi-objective permutation floswhop scheduling problem. More precisely, we focus on multi-objective local search algorithms, since they are known to provide good solutions to classical multi-objective permutation problems [2,7,12].

3.1 The MOLS Framework

Stochastic local search (SLS) algorithms are widely used for solving a broad range of NP-hard problems, including many single-objective optimisation problems [9]. The key idea is to iteratively improve a candidate solutions, by choosing, in each step, a neighbouring solution to move to, making use of randomisation to balance intensification and diversification. SLS algorithms have also been developed for multi-objective optimisation problems, where they operate on a set of non-dominated candidate solutions dubbed an *archive*. Among the most widely

used SLS methods for multi-objective optimisation problems we find Pareto Local Search (PLS) [17] and its numerous variants, such as the stochastic PLS [5], the iterated PLS [16] and the anytime PLS [7], and the Dominance-based Multi-Objective local search [12].

Recently, Blot *et al.* have proposed a generic local search framework that encompasses most of the multi-objective local search (MOLS) algorithms of the literature as well as many new variants [3]. A MOLS algorithm iterates over several phases: *selection* of solutions within the current archive, *exploration* of these solutions, and *archiving* of the neighbouring solutions that have been visited. Similarly to single-objective local search algorithms, iterated local search (ILS) approaches have been developed, which add a perturbation phase designed to more effectively explore of the underlying search space [15]. Within the generic MOLS framework, different strategies can be selected for each of these phases in order to optimise performance for a given set or distribution of benchmark instances.

3.2 MOLS Component Strategies

In the following, we explain the different components of the MOLS algorithms and describe the strategies available for instantiating them in our experiments (see Section 5). Since our investigation is focussed on these strategies, all numerical parameters have been set to values determined in previous work [2].

Initialisation. First step of MOLS, in which one or more solutions are generated from which the search process is started. Here, 10 solutions are generated uniformly at random; these form the initial archive.

Selection. Solutions are chosen within the current archive according to strategy `select_strat`. One option is to select `all` solutions in the archive; alternatively, a subset of s solutions can be selected uniformly at `random`, or according to their age (*i.e.*, the time they have been in the archive), among the `newest` or the `oldest`. In our experiments (see Section 5), s has been set to 1.

Exploration. The neighbourhood of each solution that has been selected in the previous step is explored, and an archive of candidate solutions is created, containing some of the visited neighbours. The strategy for exploring the neighbourhood (`explor_strat`) can either involve exploring it *entirely* or *partially*, using different techniques for comparing new candidate solutions with those in the current archive. In the first case, the `all` and `all_imp` strategies evaluate all the neighbours of the selected solution and consider as candidates either all non-dominated or all dominating neighbours, respectively. On the other hand, the exploration may end before all the neighbours have been visited, when r non-dominated neighbours have been evaluated (`ndom`), or when r dominating neighbours have been found. In this last case, either only the dominating neighbours are kept (`imp`), or dominating neighbours as well as all visited non-dominated neighbours (`imp_ndom`) are considered as candidate solutions. In the following experiments (see Section 5), r has been set to 5.

Table 1. Configuration space of MOLS considered in our experiments.

Parameter	Values
select_strat	{rand, all, new, old}
explor_strat	{imp, ndom, imp_ndom, all, all_imp}
perturb_strat	{restart, kick, kick_all}

Archiving. All candidate solutions identified in the exploration phase are added to the current archive; then, dominated solutions are removed from the archive.

Perturbation. In order to facilitate exploration of the search space, the perturbation strategy (`perturb_strat`) can either *restart* the search process, or merely *kick* (*i.e.*, remove) solutions from the current archive. A `restart` is performed by forming a new archive, as in the initialisation phase. The *kick* strategy replaces one or more solutions by neighbours selected uniformly at random. It can be applied to either r solutions in the current archive (`kick`), or to all the solutions in the archive (`kick_all`). In the following experiments (see Section 5), r has been set to 1.

Table 1 shows all strategies we considered when configuring our MOLS framework; these jointly give rise to 60 ($4 \times 5 \times 3$) different configurations of MOLS.

4 Experimental Setup

Benchmark Sets for the bPFSP. As previously mentioned, we are considering a bi-objective version of the classical Permutation Flowshop Scheduling Problem (PFSP), which involves scheduling a set of n jobs $\{J_1, \ldots, J_n\}$ on a set of m machines $\{M_1, \ldots, M_m\}$. In the PFSP, each machine can only process one job at a time, and each job J_i is sequentially processed on each of the m machines, with fixed processing times $\{p_{i,1}, \ldots, p_{i,M}\}$. Furthermore, the jobs are processed in the same order on every machine. Therefore, each solution of a PFSP instance (called the schedule) can be represented by a permutation of jobs of size n. In the bi-objective PFSP (bPFSP), two objectives are considered: the makespan and the flowtime of the schedule, where makespan is the total completion time, and flowtime is the sum of the individual completion times of the n jobs. We use a widely studied set of benchmark instances proposed by Taillard [18]. It is known that the difficulty of these instances increases with the number of jobs. We evaluated our approach on 6 sets of 10 Taillard instances each, with 20 jobs and 20 machines, 50 jobs and 5 machines, 50 jobs and 10 machines, 50 jobs and 20 machines, 100 jobs and 10 machines and 100 jobs and 20 machines, respectively.

Dynamic MOLS for the bPFSP. We used the implementation of MOLS for the bPFSP provided by Blot *et al.* [2] and considered two instantiations of

our dynamic algorithm framework described in Section 2.2, with up to $K = 2$ and $K = 3$ pipeline stages, respectively, and three ways of dividing the overall running times between the pipeline stages: For $K = 2$, we used $(T_1, T_2) = (1/4, 3/4) \cdot T$, $(1/2, 1/2) \cdot T$ and $(3/4, 1/4) \cdot T$, where T is the overall cut-off time, while for $K = 3$, we considered $(T_1, T_2, T_3) = (1/3, 1/3, 1/3) \cdot T$, $(1/4, 1/4, 1/2) \cdot T$ and $(1/2, 1/4, 1/4) \cdot T$. Therefore, whilst the basic MOLS algorithm has 60 distinct configurations, the dynamic MOLS algorithm, dubbed D-MOLS, has $60 + 3 \cdot 60^2 \approx 1.1 \cdot 10^4$ configurations for $K = 2$, and $60 + 3 \cdot 60^2 + 3 \times 60^3 = 6.6 \cdot 10^5$ configurations for $K = 3$ stages. We note that this configuration space is very large compared to on-line algorithms from the literature, which typically involve only very few configurations. In our experiments, we chose an overall cut-off time of $T = n^2 \dot{m}/1000$ for D-MOLS.

Automatic Configuration of D-MOLS. Blot et al. [2] showed that a multi-objective AAC is the best approach to automatically configure multi-objective algorithms such as MOLS. Therefore, in order to configure D-MOLS, we used the state-of-the-art multi-objective algorithm configurator MO-ParamILS[1], with two performance indicators: unary hypervolume[20], a volume-based convergence performance indicator, and Δ spread[4], a distance-based distribution metric. In order to simplify the use of MO-ParamILS and interpretation of results, we used a variant of hypervolume, denoted $1 - HV$, in which after normalisation to the interval $[0, 1]$, the hypervolume values are subtracted from 1, so that both indicators ($1 - HV$ and Δ) need to be minimised.

To obtain training sets to be used as the basis for automatic configuration, we generated uniformly at random a set 100 instances for each the six instance size we considered, following the same protocol as Taillard [18]. Since MO-ParamILS is a stochastic algorithm, we performed 20 independent runs for each configuration scenario, each with 1000 and 10 000 runs of D-MOLS for $K = 2$ and $K = 3$, respectively. Then, the best of the 20 resulting D-MOLS configurations (according to performance on the respective training set) was evaluated on the 10 Taillard instances in each of our testing sets, based on 15 independent runs. The performance indicators – hypervolume and spread – reported for a single D-MOLS configuration were obtained by averaging the respective values over the 15 independent runs and the 10 instances per set.

5 Experimental Results

First, we present results for D-MOLS, our dynamic version of MOLS, for the bPFSP for $K = 2$ and 3 pipeline stages, *i.e.*, one or two changes in configuration during each run. Next, we compare the results for D-MOLS with those for static MOLS.

5.1 Evaluation of Dynamic MOLS

Table 2 shows the number of D-MOLS configurations in the Pareto-optimal sets obtained from automatic configuration using MO-ParamILS; specifically,

Table 2. Number of non-dominated D-MOLS configurations determined through automatic configuration (see text for details).

Instances	$K=1$	$K=2$		$K=3$		
		$k=1$	$k=2$	$k=1$	$k=2$	$k=3$
20x20	20	5	4	1	7	9
50x5	9	-	7	-	5	9
50x10	9	-	7	-	10	8
50x20	11	-	12	-	3	8
100x10	8	1	9	-	5	5
100x20	8	-	13	N/A	N/A	N/A

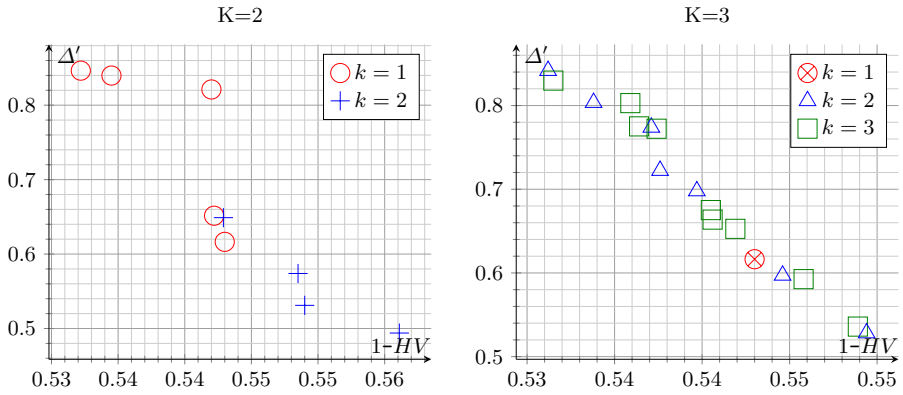

Fig. 2. Performance of Pareto-optimal D-MOLS configurations for the `20x20` benchmarks.

for $K = 2$ and $K = 3$, we report the number of non-dominated configurations with $k = 1$, 2 and 3 pipeline stages. For example, for the `20x20` scenario and $K = 3$, we obtained 1 configuration for static MOLS, 7 for dynamic MOLS with $K = 2$ and 9 for dynamic MOLS with $K = 3$ pipeline stages. For 8 of the 11 benchmark sets considered, all non-dominated D-MOLS configurations obtained from MO-ParamILS had at least 2 pipeline stages $k \geq 2$, which clearly indicates the performance advantage gained by switching between configurations during a single run of MOLS.

Figure 2 shows the Pareto fronts of D-MOLS configurations obtained in our experiments with $K = 2$ (left) and 3 (right), respectively, for the benchmark instances with 20 jobs and 20 machines. For $K = 2$, static MOLS ($k = 1$) achieves better hypervolume, while D-MOLS(2) obtains better spread; for $K = 3$, on the other hand, D-MOLS(2) and D-MOLS(3) yield better results w.r.t. both indicators. Figure 3 shows the our results for benchmark instances with 50 jobs and 20 machines. As also seen in Table 2, no configurations from static MOLS are found in the final Pareto sets; furthermore, the sets of configurations from both D-MOLS scenarios are well distributed over the Pareto front.

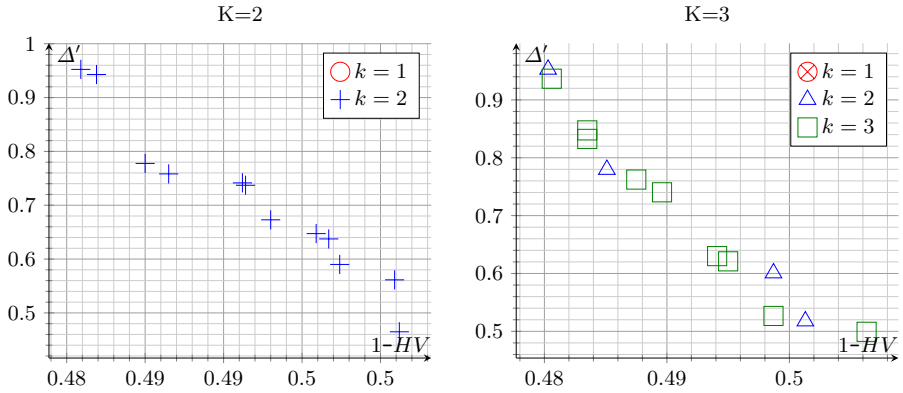

Fig. 3. Performance of Pareto-optimal D-MOLS configurations for the `50x20` benchmarks.

5.2 Performance of the Dynamic *vs* Static MOLS

In this section, we further assess the performance of our dynamic MOLS algorithm against static MOLS. Since there are only 60 configurations of static MOLS, we were able to evaluate all of them. Figure 4 shows the Pareto fronts of configurations for static MOLS ($K = 1$) *vs* dynamic MOLS for $K = 2$ and $K = 3$. Only few of the 60 configurations of MOLS ended up in the Pareto-optimal sets for each of our benchmarks. We further note that for each instance size, the Pareto fronts obtained for $K = 2$ and $K = 3$ are of roughly similar size. For `50x10`, `50x20`, `100x10` and `100x20`, the configurations obtained for D-MOLS are better distributed along the respective fronts. For `100x10` and `100x20`, the fronts obtained by static MOLS ($K = 1$) are very poorly distributed. Most of the configurations are tightly clustered; this is particularly pronounced for `100x20`, where there are two types of configurations that obtain either good hypervolume or good spread, but never both. The configurations for dynamic MOLS ($K \geq 2$), on the other hand, are well distributed and cover a broad range of tradeoffs between the objectives. Furthermore, the configurations for $K = 1$ are all dominated by those for $K \geq 2$. For the smallest instance size, `50x5`, we observed a large improvement in hypervolume, while spread remains comparable; this effect is less obvious for the `20x20` and `50x20` instances. For `50x20`, static MOLS dominates parts of the fronts for dynamic MOLS, likely as a result of the large configuration spaces for $K \geq 2$; nevertheless, for $K \geq 2$, more homogeneous Pareto fronts of configurations are obtained. For `20x20`, all three fronts are quite close to each other and reasonably well distributed, with the configurations of dynamic MOLS ($K \geq 2$) filling some of the gaps in the front obtained for static MOLS. We note that, even though the fronts for $K = 2$ and $K = 3$ are roughly similar in size, the one for $K = 3$ contains more configurations and is overall preferable.

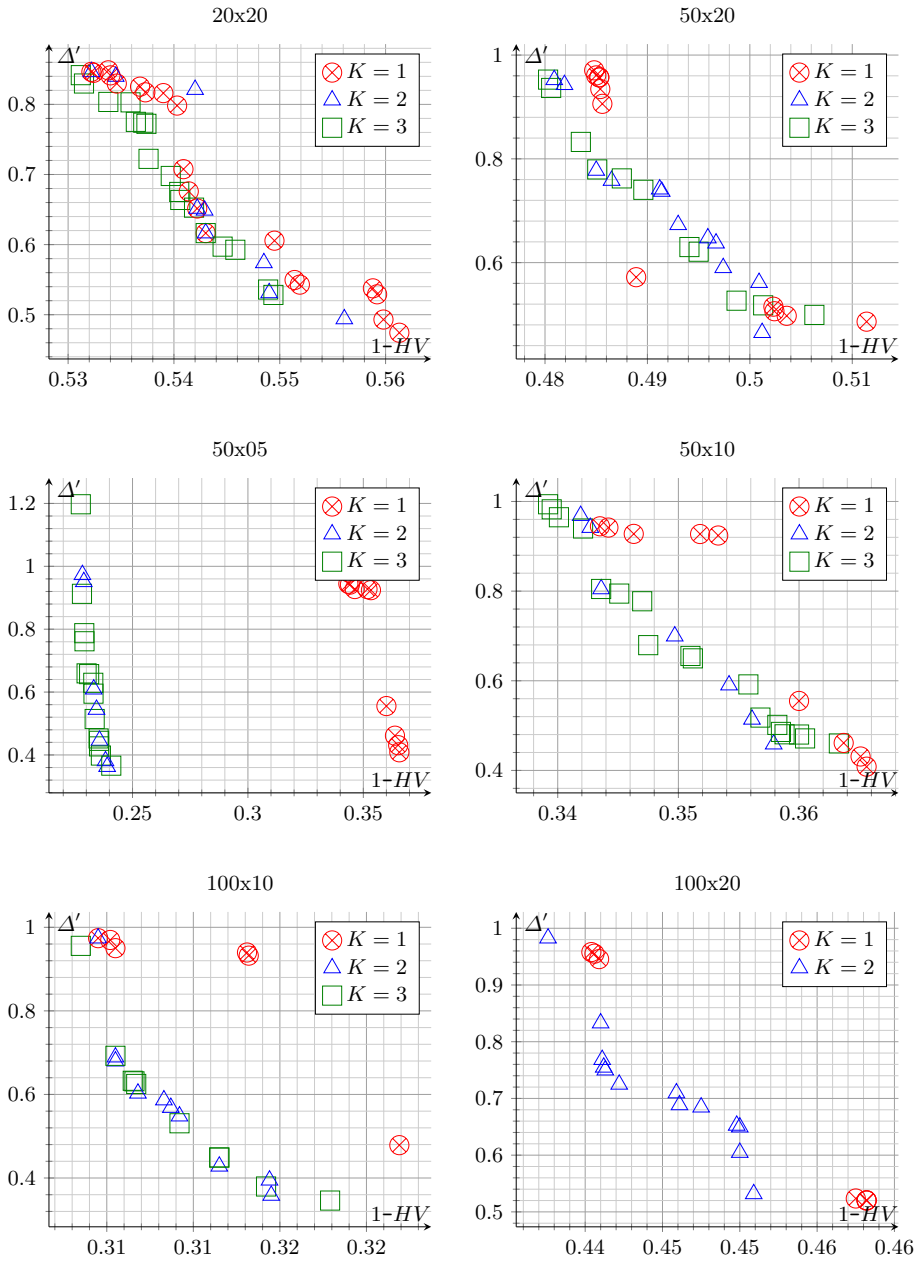

Fig. 4. Pareto fronts of static ($K = 1$) and dynamic ($K \geq 2$) MOLS configurations; for details see text.

6 Conclusions and Future Work

In this work, we have investigated the use of automatic algorithm configuration techniques for generating dynamic algorithms that modify their parameters while solving a given problem instance. Specifically, we proposed a dynamic algorithm framework that can be automatically configured with a standard, general-purpose algorithm configurator. Given a parameterised static algorithm, using our approach, it is easy to automatically construct a dynamic version of the algorithm whose parameter configuration is adjusted, according to an optimised, static schedule, while it is running.

We evaluated this approach by applying it to a multi-objective local search (MOLS) algorithm for the bi-objective permutation flowshop scheduling problem. Our experiments show that the dynamic MOLS algorithm obtained using our approach shows better performance than the underlying static MOLS procedure on the widely studied Taillard instances.

In future work, we plan to analyse the behaviour of our dynamic MOLS algorithm to further understand how the optimised configurations used by it contribute to its overall performance. We also intend to apply our approach to single- and multi-objective metaheuristic algorithms for other challenging combinatorial problems.

References

1. Blot, A., Hoos, H.H., Jourdan, L., Kessaci-Marmion, M.É., Trautmann, H.: Mo-paramils: A multi-objective automatic algorithm configuration framework. In: Festa, P., Sellmann, M., Vanschoren, J. (eds.) Learning and Intelligent Optimization. pp. 32–47. Springer International Publishing, Cham (2016)
2. Blot, A., Jourdan, L., Kessaci, M.E.: Automatic design of multi-objective local search algorithms: Case study on a bi-objective permutation flowshop scheduling problem. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 227–234. GECCO '17, ACM, New York, NY, USA (2017)
3. Blot, A., Kessaci, M.É., Jourdan, L., De Causmaecker, P.: Adaptive multi-objective local search algorithms for the permutation flowshop scheduling problem. In: Learning and Intelligent Optimization Conference (LION 12) (2018)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
5. Drugan, M.M., Thierens, D.: Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies. Journal of Heuristics **18**(5), 727–766 (2012)
6. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Computers & Operations Research **38**(8), 1219–1236 (2011)
7. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Anytime pareto local search. European journal of operational research **243**(2), 369–385 (2015)
8. Fialho, Á., Da Costa, L., Schoenauer, M., Sebag, M.: Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary

algorithms. In: International Conference on Learning and Intelligent Optimization. pp. 176–190. Springer (2009)
9. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations and applications. Elsevier (2004)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36**(1), 267–306 (2009)
11. Karafotias, G., Hoogendoorn, M., Eiben, Á.E.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Transactions on Evolutionary Computation **19**(2), 167–187 (2015)
12. Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., Talbi, E.G.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics **18**(2), 317–352 (2012)
13. Lindauer, M., Bergdoll, R.D., Hutter, F.: An empirical study of per-instance algorithm scheduling. In: International Conference on Learning and Intelligent Optimization. pp. 253–259. Springer (2016)
14. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
15. Lourenço, H.R., Martin, O.C., Stützle, T.: Handbook of Metaheuristics, chap. Iterated Local Search, pp. 320–353. Springer US (2003)
16. Olson, R.S., Moore, J.H.: Tpot: A tree-based pipeline optimization tool for automating machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Proceedings of the Workshop on Automatic Machine Learning. Proceedings of Machine Learning Research, vol. 64, pp. 66–74. PMLR, New York, New York, USA (2016)
17. Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) Metaheuristics for Multiobjective Optimisation. pp. 177–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
18. Taillard, É.D.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278 – 285 (1993), project Management anf Scheduling
19. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation. pp. 1539–1546. ACM (2005)
20. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation **3**(4), 257–271 (1999)