Adaptive Multi-Objective Local Search Algorithms for the Permutation Flowshop Scheduling Problem

Aymeric Blot[1], Marie-Éléonore Kessaci[1], Laetitia Jourdan[1], Patrick De Causmaecker[2]

[1] Université de Lille, CNRS, UMR 9189 - CRIStAL, France
[2] KU Leuven, CODeS and imec research groups, Kortrijk, Belgium
aymeric.blot@univ-lille.fr
{me.kessaci, laetitia.jourdan}@univ-lille1.fr
patrick.decausmaecker@kuleuven.be

Abstract. Automatic algorithm configuration (AAC) is an increasingly critical factor in the design of efficient metaheuristics. AAC was previously successfully applied to multi-objective local search (MOLS) algorithms using offline tools. However, offline approaches are usually very expensive, draw general recommendations regarding algorithm design for a given set of instances, and does generally not allow per-instance adaptation. Online techniques for automatic algorithm control are usually applied to single-objective evolutionary algorithms. In this work we investigate the impact of including control mechanisms to MOLS algorithms on a classical bi-objective permutation flowshop scheduling problem (PFSP), and demonstrate how even simple control mechanisms can complement traditional offline configuration techniques.

1 Introduction

Designing and tuning metaheuristics is a great challenge in the optimisation field. Offline configurators have been proposed to automatically configure optimisation algorithms according to a single indicator [**?**,**?**] and more recently, according to several indicators [**?**]. However, offline configuration requires a large amount of algorithm executions to test the different configurations and then, the output tuned algorithm is adapted to the training instances only. In this paper, we are interested in online configuration, also called parameter control, that adapts the algorithm during the execution [**?**,**?**,**?**].

Multi-objective local search (MOLS) algorithms are metaheuristics designed to solve multi-objective combinatorial optimisation problems. MOLS algorithms require the definition of several strategies such as the selection of the solution to explore, the exploration of the neighbourhood, the reference set to accept candidate neighbours, or the archive maintenance. They have successfully been used to tackle multi-objective permutation problems such as the travelling salesman problem, the quadratic assignment problem and the permutation flowshop

scheduling problem [?,?,?,?]. Parameter control has mostly been applied to bio-inspired algorithms for single objective optimisation. In this paper, we propose to control multi-objective local search algorithms to solve the bi-objective permutation flowshop scheduling problem.

This paper is organised as follows. First, a literature review on adaptive metaheuristics is presented. Section 3 details the static and the adaptive versions of the multi-objective local search algorithms. Section 4 presents the experimental setup and Section 5 gives and discusses the results. Finally, Section 6 concludes the paper and draws some perspectives.

2 Adaptive Metaheuristics

Parameter control mechanisms are generally classified between deterministic, adaptive and self-adaptive approaches [?]. In the following, we focus on adaptive approaches that are parameter and algorithm-independent.

2.1 Adaptive search

In their paper [?], Pisinger and Ropke describe a large neighbourhood search heuristic for a pick up and delivery problem which adapts during search through the selection probability of the neighbourhoods or "subheuristics" as the authors call them. The adaption strategy is steered by the history during the search and a reinforcement learning mechanism is employed to adjust the weight of each neighbourhood after a given number or a "segment" of iterations has been applied. The weight determines the probability of selecting a specific neighbourhood. Extra stochasticity is introduced through noise in some of the basic heuristics to create less deterministic behaviour and increase exploration.

An example of off-line learning can be found in [?] *e.g.*, a multi-mode scheduling problem is approached through an automaton model. Reinforcement learning agents learn the decision functions for these automatons off-line through observation of the history. The result is a fast constructive heuristic adapted to the real world situation from which problem instances originate. Off-line learning has taken on recently resulting in very strong results and frameworks. [?,?,?,?,?,?] and recently, tuning the tuner, in [?]. On-line as well as off-line learning are extensively used in hyperheuristics, see [?] for an overview.

2.2 Multi Armed Bandit

A good introduction on the subject of multi armed bandits (MAB), with the right balance between fundamental understanding and practical understanding, and without any unnecessary detail, can be found in [?]. MAB's, metaphorically referring to the infamous gambling machines, model a decision problem where the only (or main in some versions) source of information is history of previous selections. The only decision to be taken is which arm to pull next. The aim is to maximise the expected outcome of a finite series of decisions. This has

been the subject of a fascinating piece of research leading to a large number of convergence results for various policies and their variations. In an adaptive local search setting, multi armed bandits can be used to model the decisions to be taken on which subalgorithm or neighbourhood to select in the next step given the history of the current search. MAB's have been applied in evolutionary algorithms, see e.g. [**?**] as well as in evolution based hyperheuristic settings [**?**]. An early example of an application to combinatorial optimisation in networks is [**?**]. The design of algorithms using MAB's was studied in [**?**,**?**].

2.3 ϵ-greedy

MAB's, having only history to learn from, need to make decisions that both optimise the immediate result and optimise the lessons learned for better future results. Handling this exploitation/exploration dilemma is what makes a strategy for a MAB. A simple strategy is to pick the decision that has delivered the best result on average in the past. The MAB can then eventually start with an exploration phase where every handle is tried once or a predetermined number of times, after which the greedy strategy is used, adjusting the averages after every decision. In dynamical situations, the average may be weighted to introduce a bias towards recent history. These approaches are termed "greedy". In a slightly more explorative approach, a probability is introduced to allow for random selection of an arm, independently of its average success rate. This kind of approaches are called ϵ-greedy, see *e.g.* [**?**]. One possibility is to select at time $t+1$ the best performing arm with a probability of $(1-\varepsilon)$, leaving a probability of ε to uniformly select an arm at random (Eq. **??**).

$$p_i(t+1) = \begin{cases} (1-\varepsilon) + \varepsilon/N, & \text{if } i = arg\ max_j\ \overline{r}_j(t) \\ \varepsilon/N, & \text{otherwise} \end{cases} \tag{1}$$

2.4 Random

The most simple random control mechanism is based on uniform distribution of arms. However, other random approaches exist, such as *Softmax algorithms*, that use a specific distribution to select the next arm. Arms with better results on average in the past are assigned a higher probability to be selected. One possibility is to use a Boltzman distribution with the probability to select arm i at time $t+1$ is given in terms of the average rewards $\overline{r}_i(t)$ at time t as [**?**,**?**] (Eq. **??**).

$$p_i(t+1) = \frac{e^{\overline{r}_i(t)/\tau}}{\sum\limits_{arms\ j} e^{\overline{r}_j(t)/\tau}},\ \text{for all arms } i \tag{2}$$

where τ is a temperature parameter that can be taken constant or decreasing [**?**,**?**,**?**,**?**].

2.5 Adaptive pursuit

Pursuit algorithms relate to classical techniques from machine learning in that the probability to select any arm is adjusted after each selection of a specific arm. Given a learning rate β, the probabilities at $t+1$ are adapted after t as [**?**,**?**,**?**] (Eq. **??**).

$$p_i(t+1) = \begin{cases} p_i(t) + \beta(1 - p_i(t)), & \text{if } i = arg\ max_j\ \overline{r}_j(t) \\ p_i(t) + \beta(0 - p_i(t)), & \text{otherwise} \end{cases} \tag{3}$$

The probabilities at $t = 0$ are all equal. This works well for static systems, with convergence to one for the probability of selecting the best actions. It does not work well for dynamic systems exactly because of this convergence property. In order to alleviate this situation, *adaptive* pursuit was conceived [**?**]. In adaptive pursuit, a minimum and a maximum (P_{min} and P_{max}) for the probabilities p_i is introduced to leave room for exploration (Eq. **??**).

$$p_i(t+1) = \begin{cases} p_i(t) + \beta(P_{max} - p_i(t)), & \text{if } i = arg\ max_j\ \overline{r}_j(t) \\ p_i(t) + \beta(P_{min} - p_i(t)), & \text{otherwise} \end{cases} \tag{4}$$

Often, this is combined with a weighted average for the expected performances $\overline{r}_i$ coping better with dynamic situations.

2.6 Other approaches

Other approaches are reinforcement comparison [**?**,**?**] and upper confidence bound [**?**]. These approaches offer more possibilities or a better behaviour in case of uncertainty at the cost of somewhat higher complexity.

In the following we will concentrate on the simpler methods mentioned above. These result in acceptable behaviour at a relatively low implementation cost which is important for local search algorithms.

3 Adaptive Multi-Objective Local Search Algorithms

In this paper, we consider and solve multi-objective combinatorial optimisation (MOCO) problems using the Pareto dominance to compare two solutions s and s': s is said to dominate s' if and only if s is better or equal to s according to all criteria, and s is strictly better than s' at least for one criterion. If neither s dominates s' nor s' dominates s, both solutions are called incomparable. In order to handle the Pareto dominance, multi-objective metaheuristics, such as bio-inspired algorithms and multi-objective local search algorithms, can be applied to solve MOCO problems. The last ones have shown very good performance on multi-objective permutation problems such as the travelling salesman problem, the quadratic assignment problem and the permutation flowshop scheduling

problem [?,?,?]. Therefore, we decided to focus on multi-objective local search algorithms in the rest of the paper. We present first a static version of multi-objective local search algorithms that will be used in the experiments, and then an adaptive version in which control mechanisms have been integrated into.

3.1 Static Multi-Objective Local Search Algorithms

Multi-objective local search (MOLS) algorithms are originally adapted from stochastic local search algorithms [?] where the neighbourhood of a solution is explored to successively move to better and better solutions of the search space. Contrary to the single-objective case where a single current solution is considered only, an *archive* of solutions is maintained to store solutions found during the search. This archive is generally a set of Pareto solutions *i.e.*, all the solutions of the archive are incomparable. In the literature, many variants of multi-objective local search algorithms were proposed, such as PLS [?] and its numerous variants, iterated PLS [?], the stochastic PLS [?], and the anytime PLS [?]; or the DMLS [?].

The variants differ by several aspects: the selection strategy, the neighbourhood exploration strategy and the set of candidate neighbours to enter in the archive. Indeed, one, or several, or even all solutions of the archive can be explored according to the selection strategy. The neighbourhood of each solution selected in the previous phase can be fully or partially explored following the exploration strategy where a reference is used to compare the neighbours with the solution being explored. This reference is generally either the solution being explored or the entire archive. During the exploration, a policy can be used to accept neighbours as candidate solutions. This policy may be different from the termination of the neighbourhood exploration. For example, when the exploration is stopped as soon as a dominating neighbour has been found, both this neighbour and all the incomparable neighbours visited can be candidates.

Following the literature review presented in Section **??**, the on-line adaptation of an algorithm is mainly done by changing one or two strategies only during the execution contrary to off-line adaptation where thousand combinations can be tested. In this paper, we limit our study to three different termination conditions in the neighbourhood exploration and fix the other strategies. Only one solution is randomly selected in the archive to be explored. The neighbourhood exploration of the selected solution stops when one neighbour verifies the condition where the reference is fixed being the entire archive. Three strategies will be considered in this study:

`imp`, stops when the visited neighbour dominates one of the solutions of the archive; this neighbour only is candidate,

`imp_ndom`, stops when the visited neighbour dominates one of the solutions of the archive; this neighbour and all the visited neighbours not dominated by the solutions of the archive are candidates,

`ndom`, stops when the visited neighbour is not dominated by the solutions of the archive; this neighbour only is candidate.

Algorithm 1: Basic Multi-Objective Local Search (`mols`)

Input: A set of solutions, an exploration strategy
Output: A set of solutions

```
archive ← initial set of solutions;
until termination criterion is met do
    /* Select a random solution */
    selected ← select_1_rand(archive);
    /* Apply the given exploration strategy */
    reference ← archive;
    accepted ← exploration_strat(current, reference);
    /* Update the archive with the accepted neighbours */
    archive ← bounded_pareto(archive, accepted);
return archive;
```

Finally, through the `combine()` function, all the candidate solutions are added to the archive, and only the incomparable solutions of the two sets are kept in the archive for the next iteration. Algorithm **??** outlines the basic multi-objective described above. The termination criterion of the algorithm can be an allocated time or number of evaluations for example.

The main drawback of the stochastic local search is they focus their search in a small part of the search space, and then mechanisms have to be integrated to enable them to visit several parts the search space. The iterated local search (ILS) algorithm [**?**] 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 accept or not the new solution found. We give in Algorithm **??** a way to iterate a MOLS in this study, using a static exploration strategy and an initial archive. The final archive will contain the best Pareto solutions of the execution. The `kick_1()` function perturbs the execution of the `mols`: it randomly selects one solution from the final archive and it applies three random moves. Then, the algorithm has a chance to visit a new part of the search space. The `combine()` function merges the archive returned by the basic MOLS and the final archive of `s-mols` to keep the best Pareto solutions only. This MOLS is called *static* since all strategies are fixed before the execution.

3.2 Control in Multi-Objective Local Search Algorithms

In this paper, we are interested in an adaptive version of the previous static multi-objective local search where mechanisms are added to modify the exploration strategies during the execution. These mechanisms aim at adapting the exploration strategies of the MOLS during the execution. From the static MOLS algorithm, we designed an adaptive MOLS where a `control_arm` method and an `update_rewards` method to compute the rewards associated to each strategy are added. First, the rewards of each strategy have to be initialized: each exploration strategy is tested. Then, at each iteration of the MOLS, the `control_arm`

Algorithm 2: Static MOLS (`s-mols`)

Output: A set of solutions

```
archive ← initialisation();
until termination criterion is met do
    /* Perturb and apply the MOLS algorithm */
    current ← perturb(best_archive);
    current ← mols(current, exploration);
    /* Merge resulting archive */
    archive ← pareto(archive ⋃ current);
return archive;
```

Algorithm 3: Adaptive MOLS (`a-mols`)

Output: A set of solutions

```
archive ← initialisation();
until termination criterion is met do
    /* Select exploration strategy */
    exploration ← control_arm();
    /* Perturb and apply the MOLS algorithm */
    current ← perturb(best_archive);
    current ← mols(current, exploration);
    /* Merge resulting archive and update rewards */
    tmp ← pareto(archive ⋃ current);
    update_rewards(exploration, archive, tmp);
    archive ← tmp;
return archive;
```

method defines the exploration strategy for the next execution of the `mols`. The `kick_1()` and `combine()` functions are the same one as defined in `s-mols`. At the end of the iteration, the rewards of the exploration strategies are updated.

4 Experimental Setup

In this section, we present the bi-objective permutation optimisation problem tackled in this work, and detail the experimental protocol.

4.1 Permutation Flowshop Scheduling Problem

The Permutation Flowshop Scheduling Problem (PFSP) is a classical permutation problem which 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 $\{p_{i,1}, \ldots, p_{i,M}\}$; 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 (bPFSP), minimising two widely studied objectives, the makespan $C_{\max}$ (Eq. **??**), *i.e.*, the total completion time of the schedule, and the flowtime FT (Eq. **??**), *i.e.*, the sum of the individual completion times C_i of the N jobs.

$$C_{\max} := \max_{i \in \{1,\ldots,N\}} C_i \tag{5}$$

$$FT := \sum_{i=1}^{N} C_i \tag{6}$$

Classical permutation neighbourhoods include the exchange neighbourhood, where the positions of two jobs are exchanged, and the insertion neighbourhood, where one job is reinserted at another position in the permutation. In this study, we consider the hybrid neighbourhood defined as the union of the exchange and insertion neighbourhoods, which is known for the bPFSP to lead to better performance than considering each neighbourhood independently [**?**].

We use the classical PFSP Taillard instances [**?**], widely used in the literature. They span numbers of jobs $N \in \{20, 50, 100, 200, 500\}$ and numbers of machines $M \in \{5, 10, 20\}$. There are 12 valid (N, M) combinations, with 10 available instances each, for a total of 120 instances.

In this study, we limit the initialisation of the MOLS algorithms to the use of a simple single-objective greedy algorithm on the two objectives independently. Indeed, using smarter initialisation procedures, the starting solutions would be too close to the optimal Pareto front, which is undoubtedly detrimental to the current study since we aim to emphasise the impact of the control mechanisms over the algorithm itself. To obtain two solutions of reasonable quality, we have chosen the NEH procedure [**?**] for the two objectives independently. It is often used to seed state-of-the-art bPFSP initialisation procedures (*e.g.*, the 2-phase local search algorithm [**?**]).

4.2 MOLS Algorithms and Control Mechanisms

Section **??** presented the static and the adaptive versions of a MOLS algorithm where only exploration strategies can be set. Indeed, all other strategies are fixed since earlier work [**?**,**?**] has shown that the exploration strategy is the most impactful MOLS component for the bPFSP.

In the experiments, we compare the three deterministic instantiations of Algorithm **??**, each using a single exploration strategy (denoted simply by `imp`, `imp_ndom`, and `ndom`, respectively), to two adaptive algorithms (see Algorithm **??**), using a basic random control mechanism or a ε-greedy control mechanism respectively. While many other more sophisticated mechanisms could have been compared (*e.g.*, see [**?**,**?**]), they would likely be similar because only three arms were considered, which considerably limits the number of dissimilar decision strategies.

In the random control mechanism decisions are uniformly taken at random, without any feedback from the search, in contrary to the ε-greedy control mechanism that uses feedback to take decisions. This feedback is computed every iteration using the hypervolume difference between the hypervolume of the new archive and the one of the previous iteration. It is then used to update the reward associated to the current strategy using a learning rate $\alpha = 0.8$ (Eq. **??**, with r_i the reward of the arm i, and $f(t)$ the feedback obtained using the current arm).

$$r_i(t+1) = \begin{cases} \overline{r}_i(t) + \alpha(f(t) - \overline{r}_i(t)), & \text{if } i \text{ the current arm} \\ \overline{r}_i(t), & \text{otherwise} \end{cases} \quad (7)$$

In this study, we set $\varepsilon = 0.1$ so that the best performing strategy (*i.e.*, the arm $arg\ max_i\ \overline{r}_i(t)$) is chosen with 90% probability, either strategy being selected uniformly at random otherwise.

For both control strategies, we consider four different variants, that differ by the subset of exploration strategies that are available. First, the three exploration strategies are available for both adaptive algorithms (`rand_3`, `greedy_3`). Note that it is already known that the `imp` strategy leads to poorer results on the bPFSP. But, we still decide to make available this bad strategy in order to evaluate the control mechanism without any a priori knowledge. Secondly, we use this expertise and only make available the two strategies `imp_ndom` and `ndom` for both adaptive algorithms (`rand_2`, `greedy_2`). Finally, the last two variants introduce a *long-term learning* scheme, beginning with the three strategies but switching to only use the two best strategies during the search. Two possibilities are evaluated: either after half the total running time of both adaptive algorithms (`rand_ltl_50`, `greedy_ltl_50`), or after twenty percent of the total running time (`rand_ltl_20`, `greedy_ltl_20`).

The termination criterion of the three static algorithms and the eight variants of the adaptive algorithm is a total running time fixed to $n^2m/500$ seconds. The termination criterion of the inner MOLS (see Algorithm **??**) is a combination of either n^2 solution evaluations or n iterations without improvement. This criterion is well adapted to the bPFSP since it enables a sufficient number of iterations of both the inner algorithm and the control mechanism. In the following experiments, about 1600 executions of the inner MOLS for instances with 20 jobs have been done, then about 750, 400, 250 and 100 iterations for instances with 50, 100, 200 and 500 jobs, respectively. This decrease of the number of executions when the number of jobs increases is explained by the exploration step that becomes more and more long and challenging as the size of the neighbourhood quickly grows.

4.3 Experimental Protocol

In this work, we propose to compare the performance of using traditional deterministic mechanisms (through three static algorithms) with different control mechanisms (through eight adaptive algorithms). Experiments are conducted

Table 1. Experimentations summary

Type	Approach	3 arms	2 arms	LTL
Deterministic	`imp`	✓		
Deterministic	`imp_ndom`	✓	✓	
Deterministic	`ndom`	✓	✓	
Random	`rand_3`	✓		✓
Random	`rand_2`		✓	✓
Random	`rand_ltl_50`			✓
Random	`rand_ltl_20`			✓
ε-greedy	`greedy_3`	✓		✓
ε-greedy	`greedy_2`		✓	✓
ε-greedy	`greedy_ltl_50`			✓
ε-greedy	`greedy_ltl_20`			✓

across all classical bPFSP Taillard instances, separated in twelve benchmarks of 10 instances sharing the same number of jobs and machines.

The experimental protocol is reduced to the exhaustive comparison of all approaches on all benchmark instances. Because of the stochasticity of both the algorithm and the control mechanisms, all approaches are run 20 times on each instance, using a given set of 20 random seeds.

In total, eleven approaches are compared, in 4 successive steps, as detailed in Table **??**. First, we compare the three deterministic approaches with the two adaptive approaches that use all three explorations strategies. Then, we focus on the two best strategies, and compare the respective two deterministic approaches with the two adaptive approaches that use them only. Finally, we investigate the potential of a *long-term learning* scheme for the two control mechanisms independently, first by switching from three arms to two arms after half of the runtime has passed (`rand_ltl_50`, `greedy_ltl_50`), then after only twenty percent of the runtime (`rand_ltl_20`, `greedy_ltl_20`).

5 Experimental Results

Table **??** presents the rankings resulting of the four steps presented in the experimental protocol (see Section **??**) for the 12 instance sizes together with the resulting average ranks. For each instance size, the ranking is computed using pairwise Wilcoxon signed rank tests and the Friedman test post hoc analysis checks the statistical equivalence between algorithms ranked 1, and their difference with the others.

First, we focus on the 3-arm adaptive approaches, `rand_3` and `greedy_3`, comparing them to the three respective deterministic approaches `imp`, `imp_ndom` and `ndom`. As shown on Table **??**, the `imp` and `ndom` approaches always perform very poorly and very well, respectively. Meanwhile, the `imp_ndom` approach performs rather poorly in small instances, but achieves very good results on the

Table 2. Experimental ranking

Instance		3 arms					2 arms				LTL rand				LTL ε-greedy			
N	M	imp	imp_ndom	ndom	rand_3	greedy_3	imp_ndom	ndom	rand_2	greedy_2	rand_3	rand_2	rand_ltl_50	rand_ltl_20	greedy_3	greedy_2	greedy_ltl_50	greedy_ltl_20
20	5	5	4	1	1	1	4	1	1	1	4	1	3	1	1	1	1	1
20	10	5	4	1	1	1	4	1	1	1	4	1	1	1	1	1	1	1
20	20	5	3	3	1	1	3	3	1	1	2	1	2	2	1	1	1	1
50	5	5	4	1	1	1	4	1	1	1	4	1	1	1	1	1	1	1
50	10	5	4	1	1	1	4	1	1	1	4	1	1	1	4	1	1	3
50	20	5	4	1	1	1	4	1	1	1	4	1	1	1	4	1	1	1
100	5	5	4	1	1	1	4	1	1	1	4	1	3	1	4	1	3	1
100	10	5	1	1	1	1	4	1	1	1	4	1	3	1	4	1	3	1
100	20	5	2	1	2	2	4	1	2	2	4	1	3	1	4	1	3	1
200	10	5	1	1	3	3	4	1	1	1	4	1	2	2	4	1	3	1
200	20	5	2	1	3	3	4	1	1	1	4	1	3	2	4	1	2	2
500	20	5	1	1	3	3	1	1	1	1	3	1	3	2	3	1	2	3
average		5	2.8	**1.2**	1.6	1.6	3.7	**1.2**	**1.1**	**1.1**	3.8	**1**	2.2	1.3	2.9	**1**	1.8	1.4

largest ones. Surprisingly, the two adaptive approaches (`rand_3` and `greedy_3`) equivalently perform. More precisely, they perform very well on the first eight instances (rank 1), but their performance are more debatable on the four largest ones. Indeed, for instances with 100 jobs and 20 machines, they are outperformed by the deterministic approach `ndom` and equivalently perform with the `imp_ndom` approach. But, for instances with 200 and 500 jobs, they are also outperformed by this latter approach. In these cases, they are still better than the `imp` approach. This results show that the `imp` approach affects more the adaptive algorithms when the problem gets harder.

Then, in the second step, the `imp` approach is discarded and we focus on the 2-arm adaptive approaches (`rand_2` and `greedy_2`) that have only the choice between the `imp_ndom` and `ndom` exploration strategies. Once again, the two adaptive approaches equivalently perform. However, they are rank 1 for all the instances except for the 100-jobs 20-instances (rank 2). Considering only the well performing exploration strategies largely improves the adaptive approaches for the largest instances.

Having validated that the `imp` arm should not be used on larger machines instances, we finally investigate a long-term learning scheme where arms can be discarded if they are worse than the others. Two approaches have been tested: the discard of the worst strategy is done after either fifty percent (`rand_ltl_50`, `greedy_ltl_50`) or twenty percent (`rand_ltl_20`, `greedy_ltl_20`) of the total running time. In order to effectively analyse this long-term learning scheme, the

Table 3. Complete ranking

Instance												
N	M	imp	imp_ndom	ndom	rand_3	rand_2	rand_ltl_50	rand_ltl_20	greedy_3	greedy_2	greedy_ltl_50	greedy_ltl_20
20	5	11	10	1	9	1	8	1	1	1	1	1
20	10	11	10	1	9	1	1	1	1	1	1	1
20	20	11	9	9	6	1	6	6	1	1	1	1
50	5	11	10	1	9	1	1	1	1	1	1	1
50	10	11	10	1	9	1	5	5	8	1	1	5
50	20	11	10	1	7	1	1	1	9	1	7	1
100	5	11	9	1	10	1	7	5	8	1	5	1
100	10	11	7	1	10	1	7	1	9	1	6	1
100	20	11	6	1	9	2	6	2	9	2	8	2
200	10	11	4	1	9	1	6	6	9	1	6	4
200	20	11	7	1	10	1	7	1	9	1	6	1
500	20	11	6	1	9	2	6	2	9	2	8	2
average		11	8.2	**1.7**	8.8	**1.2**	5.1	2.7	6.2	**1.2**	4.3	**1.75**

two adaptive approaches are investigated and ranked separately. Unsurprisingly, the 2-arm versions of both adaptive approaches always statistically outperform their respective 3-arm versions and so for the versions using the long-term learning. Introducing long-term learning to only keep well-performing arms is efficient. The ranking between the two control mechanisms `rand` and `greedy` are not the same size by size, but the average ranks show that it is more efficient to discard an arm sooner since `rand_ltl_20` and `greedy_ltl_20` are better ranked than `rand_ltl_50` and `greedy_ltl_50` respectively. These results demonstrate how control mechanisms can effectively identify and evaluate the performance of strategies during the search.

Table **??** summarises all experiments and shows the overall ranking of the eleven approaches on each instance size and shows the final average ranks. Regarding the three deterministic approaches, the `imp` approach is always ranked last; the `ndom` approach is almost always ranked first, only beaten on the 20-jobs 20-machines instance where it is outperformed by all adaptive approaches. Regarding the adaptive approaches, both the 2-arm approaches `rand_2` and `greedy_2` are the best performing, then are ranked the `ltl_20` ones and the `ltl_50` ones. The approaches using the random control mechanism generally perform worse than the ones using the ε-greedy mechanism, especially for the 3-arm variants and the long-term learning variants. Interestingly, even the random adaptive approach performs really well when considering the two `imp_ndom`

and `ndom` arms, potentially meaning that the adaptive algorithms will achieve very good results as long as there is no critically bad performing arm available. However, the long-term learning variants show that it is possible to identify, remove and recover in such event.

6 Conclusion

In this paper we presented an adaptive version of MOLS algorithms by introducing control mechanisms. If many control mechanisms can be found in the literature, they have mostly been applied to single objective bio-inspired algorithms; we investigated here their impact on a multi-objective iterative local search algorithm on the classical bi-objective bPFSP. We restrained the control on a single parameter, the exploration strategy, using two different types of control mechanisms: one uniform random and one feedback-based.

Our results show that on the studied problem, taken individually, the three considered explorations strategies performs very differently, and that adaptive approaches can achieve results statistically equivalent to the best strategy. We verified that identifying the best performing strategy before the search, using for example an offline automatic algorithm configurator, can lead to substantial improvement of the algorithm performance. Moreover, we show that very basic control mechanisms can achieve similar outcomes as long as the worst performing strategies are quickly discarded. Identification of the arm qualities continues to critically impact the performance of the overall algorithm, but the long-term learning approaches show that it is possible to introduce this knowledge during the search instead. It means that if a preliminary offline configuration of the algorithm results on multiple high performing mechanisms, it should be possible to simply postpone the decision and to adaptively decide during the search for each tackled instance.

In future work, it would be necessary to first investigate the addition of more possible arms to the MOLS structure, for example by differentiating the exploration reference, in order to compare more complex and potentially more efficient control mechanisms. Moreover, it would be helpful not to fix strategies a priori. Therefore, we would investigate the potential of control mechanisms to simultaneously manage different strategies. Finally, a single bi-objective problem was considered in this paper, in which the performance of each exploration strategy did not vary much. As MOLS algorithms have also been applied to other problems such as the travelling salesman problem or the quadratic assignment problem, for which the optimal MOLS configurations differs, it would be interesting to investigate adaptive MOLS on these problems.

References

1. Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

2. Jany Belluz, Marco Gaudesi, Giovanni Squillero, and Alberto Tonda. Operator selection using improved dynamic multi-armed bandit. In *GECCO 2015*, pages 1311–1317, 2015.
3. Aymeric Blot, Holger H. Hoos, Laetitia Jourdan, Marie-Éléonore Marmion, and Heike Trautmann. MO-ParamILS: A multi-objective automatic algorithm configuration framework. In *LION 10*, pages 32–47, 2016.
4. Aymeric Blot, Laetitia Jourdan, and Marie-Éléonore Kessaci-Marmion. Automatic design of multi-objective local search algorithms: case study on a bi-objective permutation flowshop scheduling problem. In *GECCO 2017*, pages 227–234, 2017.
5. Aymeric Blot, Alexis Pernet, Laetitia Jourdan, Marie-Éléonore Kessaci-Marmion, and Holger H. Hoos. Automatically configuring multi-objective local search using multi-objective optimisation. In *EMO 2017*, pages 61–76, 2017.
6. Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.
7. Nicolò Cesa-Bianchi and Paul Fischer. Finite-time regret bounds for the multi-armed bandit problem. In *ICML 1998*, pages 100–108, 1998.
8. Nguyen Thi Thanh Dang, Leslie Pérez Cáceres, Thomas Stützle, and Patrick De Causmaecker. Configuring irace using surrogate configuration benchmarks. In *GECCO 2017*, 2017.
9. Madalina M Drugan and Ann Nowé. Designing multi-objective multi-armed bandits algorithms: A study. In *IJCNN 2013*, pages 1–8, 2013.
10. Madalina M Drugan and Dirk Thierens. Path-guided mutation for stochastic Pareto local search algorithms. In *PPSN XI*, pages 485–495. Springer, 2010.
11. Madalina M. Drugan and Dirk Thierens. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics*, 18(5):727–766, 2012.
12. Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. *Computers & Operations Research*, 38(8):1219–1236, 2011.
13. Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. Anytime Pareto local search. *European Journal of Operational Research*, 243(2):369–385, 2015.
14. Ágoston E. Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
15. Aguston Eiben, Zbigniew Michalewicz, Marc Schoenauer, and Jim Smith. Parameter control in evolutionary algorithms. *Parameter setting in evolutionary algorithms*, pages 19–46, 2007.
16. Yi Gai, Bhaskar Krishnamachari, and Rahul Jain. Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking*, 20(5):1466–1478, 2012.
17. Angeliki Gretsista and Edmund K. Burke. An iterated local search framework with adaptive operator selection for nurse rostering. In *LION 11*, pages 93–108, 2017.
18. Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.
19. Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION 5*, pages 507–523, 2011.

20. Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.
21. Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2015.
22. Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *CoRR*, abs/1402.6028, 2014.
23. Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, and El-Ghazali Talbi. 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.
24. Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.
25. Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.
26. Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle. Automatic design of hybrid stochastic local search algorithms. In *HM 2013*, pages 144–158, 2013.
27. Muhammad Nawaz, E Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.
28. Luis Paquete, Marco Chiarandini, and Thomas Stützle. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*, pages 177–199. Springer, 2004.
29. Kanagasabai Rajaraman and P. S. Sastry. Finite time analysis of the pursuit algorithm for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(4):590–598, 1996.
30. Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.
31. Nasser R. Sabar, Masri Ayob, Graham Kendall, and Rong Qu. A dynamic multi-armed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, 45(2):217–228, 2015.
32. Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1):6:1–6:25, 2009.
33. Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.
34. Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
35. Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *GECCO 2005*, pages 1539–1546, 2005.
36. Joannès Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *ECML 2005*, pages 437–448, 2005.
37. Tony Wauters, Katja Verbeeck, G. Vanden Berghe, and Patrick De Causmaecker. Learning agents for the multi-mode project scheduling problem. *Journal of the Operational Research Society*, 62(2):281–290, 2011.
38. Saba Q. Yahyaa, Madalina M. Drugan, and Bernard Manderick. Annealing-pareto multi-objective multi-armed bandit algorithm. In *ADPRL 2014*, pages 1–8, 2014.