An Analysis of Post-selection in Automatic Configuration

Zhi Yuan[*]
School of Information Systems, Singapore Management University, Singapore
zhiyuan@smu.edu.sg

Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
stuetzle@ulb.ac.be

Marco A. Montes de Oca
Dept. of Mathematical Sciences, University of Delaware, Newark, DE, USA
mmontes@math.udel.edu

Hoong Chuin Lau
School of Information Systems, Singapore Management University, Singapore
hclau@smu.edu.sg

Mauro Birattari
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
mbiro@ulb.ac.be

ABSTRACT

Automated algorithm configuration methods have proven to be instrumental in deriving high-performing algorithms and such methods are increasingly often used to configure evolutionary algorithms. One major challenge in devising automatic algorithm configuration techniques is to handle the inherent stochasticity in the configuration problems. This article analyses a post-selection mechanism that can also be used for this task. The central idea of the post-selection mechanism is to generate in a first phase a set of high-quality candidate algorithm configurations and then to select in a second phase from this candidate set the (statistically) best configuration. Our analysis of this mechanism indicates its high potential and suggests that it may be helpful to improve automatic algorithm configuration methods.

Categories and Subject Descriptors

[**Computing Methodologies**]: Artificial Intelligence-Search Methodologies

Keywords

Automatic Algorithm Configuration, Post-selection, Search

1. INTRODUCTION

The performance of many optimization algorithms depends on their parameter settings. Obtaining high performance requires that the parameters are appropriately set. In the last few years, it has been shown that the offline configuration of algorithm parameters can effectively been done by automatic algorithm configuration techniques such as ParamILS [15], F-Race [7] and iterated racing approaches [8], gender-based genetic algorithm [2], or model-based search approaches [12]. These aforementioned methods can be applied to general configuration tasks that include the setting of categorical, ordinal and numerical algorithm parameters. If only numerical parameters require to be determined, these methods remain applicable but they may be sidelined by methods from numerical optimization [21], estimation of distribution algorithms [17] or methods based on classical experimental designs such as CALIBRA [1]. Whatever the configuration task, these methods not only free humans from a tedious, manual trial-and-error process, but they often result in parameter configurations that substantially improve over default configurations proposed by the algorithm's designers.

In this paper, we focus on offline configuration [6], i.e. on finding a good parameter configuration before the algorithm is actually applied, a typical situation in algorithm design. The task in offline configuration is to identify an algorithm parameter configuration based on a set of training instances, such that the best-found configuration performs well on future, unseen problem instances. Offline configuration involves two sub-tasks, namely generation and evaluation. The first sub-task, the generation of candidate algorithm configurations, is typically done by search algorithms including, e.g. direct search methods [15, 8], model-based search methods [5, 14, 12], or modern continuous optimizers [21], etc. The second sub-task requires evaluating candidate algorithm configurations and at some point selecting the one with the best evaluation. This second sub-task is stochastic due to two main sources of randomness [6]: first, the algorithm to be configured may be a stochastic algorithm—this is always the case if randomized decisions are taken during the algorithm execution; second, the stochasticity due to estimating algorithm performance of candidate algorithm configurations by different training instances—these instances may be seen as drawn from some random distribution of problem instances. Most work on configuration problem fo-

[*]The main part of the research was carried out while Zhi Yuan was with IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium.

cuses on the first sub-task, while the second sub-task is less discussed and studied.

In this paper, we extend the analysis of post-selection [21], which is a promising mechanism designed to address the second sub-task in the offline configuration, the evaluation and selection of candidate configurations. The basic idea of the post-selection mechanism is to divide the configuration process into two phases: in the initial elite qualification phase, a number of elite algorithm configurations are identified; then, in the subsequent elite selection phase, the best of these elite algorithm configurations is carefully selected using, for example, a racing method. Initial results [21] indicated that with a careful elite selection in the final phase, the post-selection mechanism allows to use a more coarse evaluation of the candidate configurations in the elite qualification phase (that is, evaluating most of the candidate configurations on less training instances) As a result, more candidate configurations may be generated and, thus, potentially better configurations may be found. In the empirical study in this paper, we extend the analysis of the post-selection method in [21] to (i) study the impact of the maximum number of algorithm runs (called configuration budget) on the con gurator performance; (ii) examine the impact of using a very small number of training instances in the elite qualification phase; (iii) consider more search methods for generating the elite candidate configurations; (iv) empirically investigate some new settings of post-selection and derive a new high-performing configurator for setting numerical parameters.

The article is structured as follows. Section 2 reviews automatic algorithm configuration and Section 3 describes the experimental setup. Section 4 studies post-selection before we compare our post-selection configurators to iterated F-Race and ParamILS in Sections 5 and 6, respectively.

2. CONFIGURATION ALGORITHMS

A *configuration algorithm* (or *configurator*) typically combines a *search* method that generates candidate algorithm configurations and a mechanism for handling stochasticity through evaluating the configurations and selecting the most promising. In this article, we examine post-selection, a recent mechanism for combining search and evaluation methods. Here, we introduce briefly the search methods and the evaluation methods we consider for comparisons.

2.1 Black-box search methods

A *search* mechanism in a configurator iteratively generates candidate algorithm configurations. In this article, the algorithm to be configured use mainly numerical parameters, in which case we refer to the configuration problem also as tuning problem. For these numerical parameters, including continuous or quasi-continuous (e.g. integer, for which rounding may be applied) parameters, we consider the following (black-box) continuous optimizers as search methods:

Bound Optimization BY Quadratic Approximation (BOBYQA). BOBYQA [18] is a model-based trust-region continuous optimizer that iteratively builds and refines a quadratic model based on which the trial points are sampled.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES). CMA-ES [9] is a (μ, λ)-evolutionary strategy. It iteratively samples candidate solutions from a multivariate Gaussian distribution, with a sample mean as a linear combination of μ elite parents and a covariance matrix automatically adapted based on the search trajectory.

Mesh Adaptive Direct Search (MADS). MADS [3] is an extension of generalized pattern search algorithms. It is a mesh-based search method that systematically adapts the mesh coarseness, search radius, and search direction.

For discrete parameter configurations, an iterated local search method underlying ParamILS is considered in Sec. 6.

2.2 Evaluation method

Configurators typically have a limited *evaluation budget* B, which can be a maximum number of times the algorithm to be configured can be run on training instances. The evaluation method needs to determine how good candidate configurations are and select the one that performs best. How to allocate the available budget for evaluation is an important topic in the study of configuration algorithms, since on one side the evaluation budget is limited due to the high computational cost of each evaluation, but on the other side the evaluation error of a candidate reduces with the number of evaluations. A good compromise is to allocate more budget to promising candidates, so that they can be evaluated more carefully. The evaluation methods considered in this article include:

Repeated evaluation. It evaluates each candidate configuration by the same, fixed number of algorithm runs.

Racing. A *racing* method [16, 7, 6] evaluates candidate configurations instance by instance and eliminates inferior ones as soon as statistical evidence is gathered against them. Thus, better candidate also receive more evaluations. Racing methods differ in the statistical tests that are used to detect inferior candidates; e.g., F-Race adopts the Friedman and its post-hoc tests, and t-Race uses Student's t-test.

Intensification. *Intensification* mechanisms are used in methods such as FocusedILS [15], SPO+ [14], ROAR or SMAC [12]. It is used to compare a newly generated configuration to the incumbent, i.e. the best configuration found so far, and eliminate a new configuration as soon as it is worse than the incumbent in the sequence of instances the incumbent was already evaluated on; if a new candidate is not eliminated, its number of evaluations increases by, e.g. one, and compares with the incumbent again, until it reaches the same number of evaluations as the incumbent, then a new incumbent is determined.

2.3 Combination of search and evaluation

Given a search method and an evaluation method, a configurator essentially consists of an efficient, non-trivial combination of the two. We discuss two possibilities below, the second being the mechanism studied here.

Iterated selection. Iterated selection we call the approach where two distinctive phases are iterated: first new candidate configurations are generated, and then evaluated by an evaluation method, possibly updating the incumbent. Most of the established configurators are based on some form of iterated selection, including SPO [5] and SPO+ [14], iterated racing techniques such as iterated F-Race [4, 8], MADS/F-Race [22], and CMA-ES/F-Race [21], or FocusedILS [15]. These methods include the incumbent from iteration to iteration. Some of them consider using an intensification mechanism to preserve the incumbent (e.g. FocusedILS and SPO+). The possible drawbacks of iterated selection are that an incumbent may be lost if no specific mechanism

Algorithm 1 Post-selection

```
Phase 1: elite qualification. Run configurator and collect the best configurations as elite configurations. Each elite configuration $\theta_e$ is stored in $\Theta_e$. A budget of $R_n$ ($n = |\Theta_e|$) depending on the number of elite configurations is reserved for Phase 2. Go to Phase 2 when the budget for qualification phase finishes.
Phase 2: elite selection. Use an evaluation method, e.g. racing, to select the best $\theta^*$ from $\Theta_e$.
```

Table 1: The range of $\mathcal{MM}$AS parameters.

param.	α	β	ρ	m
range	$[0.0, 5.0]$	$[0.0, 10.0]$	$[0.0, 1.00]$	$[1, 1200]$
param.	γ	nn	q_0	
range	$[0.01, 5.00]$	$[5, 100]$	$[0.0, 1.0]$	

for incumbent preservation is used, while if an incumbent preservation mechanism is used, it may be too aggressive in eliminating potentially promising new candidates, leading to stagnation as observed occasionally in FocusedILS [15].

Post-selection. The basic idea of the post-selection mechanism is to divide the configuration process into two phases: a first elite *qualification* phase and a second elite *selection* phase. During the qualification phase, a number of elite configurations are identified by running a configurator. These elite configurations can be collected by, for example, enforcing quick convergence of the configurator and then taking the best configuration in each independent restart. Alternatively, different configurators may be run simultaneously and the best configurations returned by various configurators may be qualified as elites. In the elite selection phase, an evaluation method is applied to select the best from these elite configurations. See Algo. 1 for a summary of the post-selection mechanism. A number of configurators are devised following the post-selection approach and investigated in the following sections. We also compare post-selection configurators to iterated racing techniques and FocusedILS.

3. EXPERIMENTAL SETUP

In this article, we focus on one configuration domain, where the algorithm to be configured is $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [20] applied to the traveling salesman problem (TSP). The numerical parameters in $\mathcal{MM}$AS that are considered in this study include: α and β, the relative importance of pheromone trail and heuristic information; ρ, the proportion of the pheromone evaporated after each iteration; m, the number of ants; γ, which controls the gap between the minimum and maximum pheromone trail limits in $\mathcal{MM}$AS; nn, the size of the nearest neighbor candidate list in the solution construction; and q_0, the probability with which an ant selects deterministically the best possible choice at each construction step. The range of the values considered for these parameters is listed in Table 1. In the configuration process, each search algorithm generates the parameter space with a precision of two significant digits.

From these seven numerical parameters we extracted a number of *case studies*, where a subset of parameters is to be set while the others assume their default values. More in detail, we extracted three case studies for $d \in \{2, 3, 4, 5, 6\}$ parameters to be set, resulting in $3 \times 5 = 15$ case studies. These case studies are listed in Table 2.

Table 2: The 15 case studies of configuring 2 to 6 parameters (each with 3 case studies) of $\mathcal{MM}$AS.

n.param.	case 1	case 2	case 3
2	$\alpha\ \beta$;	$\rho\ m$;	$\gamma\ nn$;
3	$\alpha\ \beta\ m$;	$\beta\ \rho\ nn$;	$\rho\ \gamma\ nn$;
4	$\alpha\ \beta\ \rho\ m$;	$\alpha\ \beta\ \gamma\ nn$;	$\rho\ m\ \gamma\ nn$;
5	$\alpha\ \beta\ \rho\ m\ nn$;	$\alpha\ \beta\ \rho\ m\ \gamma$;	$\alpha\ \beta\ m\ \gamma\ nn$;
6	$\alpha\ \beta\ \rho\ m\ \gamma\ nn$;	$\alpha\ \beta\ \rho\ m\ \gamma\ q_0$;	$\alpha\ \beta\ \rho\ m\ nn\ q_0$;

The instances are uniformly randomly distributed Euclidean TSP instances. Two sets of instances are considered in this article: the homogeneous (`hom`) set consists of uni-size instances of 750 nodes, 1 000 instances for the training phase, and 300 for the testing phase; the heterogeneous (`het`) set consists of instances ranging from 100 nodes to 1 200 nodes, 900 instances for training and 300 for testing. The computation time for $\mathcal{MM}$AS is 5 seconds. The $\mathcal{MM}$AS implementation is based on the ACOTSP software [19] with minor extensions to allow the usage of the parameter γ.

In each case study, seven *budget levels* are considered. The minimum level of the configuration budget is chosen to be $B_1 = 5\cdot(2d+2)$, which results in a budget $B_1 = 30$ when $d = 2$ and in a budget $B_1 = 70$ for $d = 6$. The other six levels of the configuration budget are $B_i = 2^{i-1}\cdot B_1, i = 2, 3, 4, 5, 6, 7$, which doubles the budget for each next level. Each budget level of each case study is considered as one *test domain*, resulting, thus, in $7 \times 15 = 105$ test domains. For each test domain, 10 trials were run. To reduce experimental variance, in each trial, the same random order of training instances is used for running each configurator, and each instance is evaluated with a common random seed. The instance order and random seed change from trial to trial. In each trial, an archive is used in order to prevent the same parameter configuration being evaluated twice on the same instance; in such case, the evaluation is read from the archive without consuming configuration budget.

When comparing testing results, we suppose every testing instance, large or small, is of the same importance. Therefore, in each test domain, we perform a standardized z-score normalization of the performance of configurations on each testing instance, such that for any given instance, the distribution of performance over tested configurations has mean zero and variance one. Whenever ranking results are presented, each rank is based on the mean value of the normalized performance in one test domain. Whenever results of statistical tests are reported in the following, we use Wilcoxon's signed-rank test with $\alpha = 0.05$, and with Holm's method in case of multiple comparison.

4. STOCHASTICITY HANDLING USING POST-SELECTION

In this section, we examine the impact post-selection has on the performance and the behavior of various search methods with which it is combined. Before the presentations of these details, we concisely show that, in general, there exist interactions between the budget level and settings of the

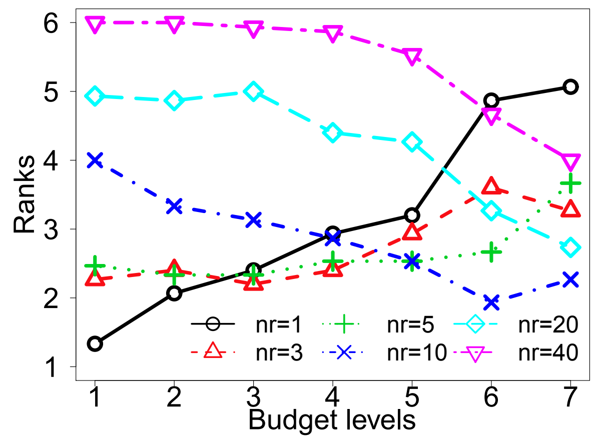


Figure 1: Average ranks of six nr settings ($nr \in \{1, 3, 5, 10, 20, 40\}$) for repeated evaluation over seven budget levels across 15 case studies of $\mathcal{MMAS}$.

evaluation method used. This sheds also some light on side-advantages of the post-selection mechanism, namely to make the configurator more robust to specific parameter settings for the evaluation method.

4.1 Repeated evaluation

The simplest evaluation method is probably repeated evaluation, where each candidate configuration is evaluated nr times. We consider here values of $nr \in \{1, 3, 5, 10, 20, 40\}$ and evaluate their performance on the 105 test domains. To illustrate the trade-offs incurred between the setting of nr and different configuration budgets, we use a uniform random search (we observed similar behavior with other search methods). Fig. 1 shows the average ranks of the six settings of nr. The relative performance of different nr settings depends strongly on the configuration budget: while for the lowest budget levels B_1 and B_2 the setting of $nr = 1$ appears to be best, the performance of low nr settings downgrades as the configuration budget increases. The clearest example is the setting $nr = 1$, which is the best for B_1 and B_2 but becomes the worst for the two highest budgets B_6 and B_7. On the contrary, large nr settings are the worst for low budgets but they improve as the configuration budget increases. Similar trade-offs were also observed in [13].

4.2 Effectiveness of Post-selection

For the experiments with post-selection, we adopted whenever possible the settings used in [21]: The budget reserved for elite selection phase is set to

$$R_n = \begin{cases} 2 \cdot n^2 & \text{if } n < 10 \\ 20 \cdot n & \text{if } n \geqslant 10 \end{cases} \quad (1)$$

and the first Friedman test starts at $f_n = \min\{n + 2, 10\}$-th instance, where n is the number of candidates for the elite selection. The first line in Equation 1 extends [21] to ensure reasonable settings for the low budget levels. In post-selection, the default is that only restart-best configurations qualify as candidates for the elite selection, where a restart-best solution is the best solution in one independent restart of the algorithm–restarts are triggered by convergence of the

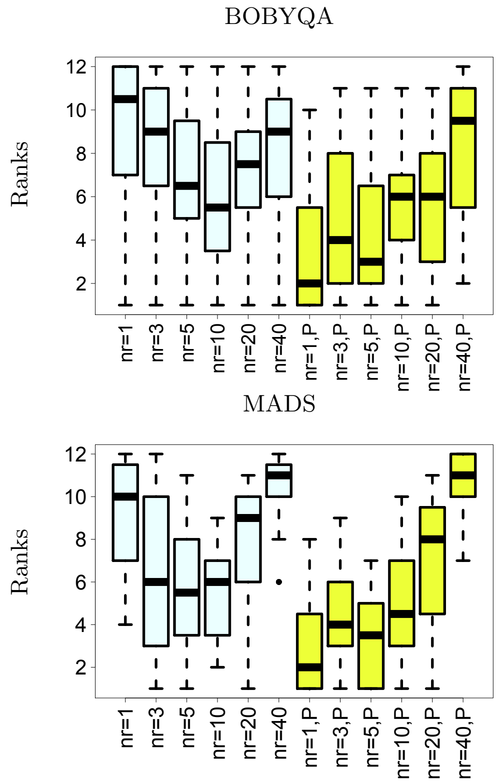


Figure 2: The rank distribution of 12 settings: repeated evaluation without post-selection (the left six light-blue box-plots in each plot) or with post-selection (the right six yellow box-plots dubbed with "P"), using $nr \in \{1, 3, 5, 10, 20, 40\}$. BOBYQA (top) and MADS (bottom) are tested on eight case studies, each with budget levels from B_4 to B_7.

algorithm. Finally, each configuration generated in the elite qualification phase is evaluated by nr same instances.

We present results with the search methods BOBYQA and MADS with either repeated evaluation or post-selection. Each search method is restarted when it stagnates. Stagnation can be detected, for example, if the search radius drops to less than the degree of significant digits (two in this work). In earlier work [21], post-selection was studied with nr ranging from 5 to 40, and it was shown that post-selection is effective when nr is small. Here, we explore smaller settings of nr equal to one and three on a subset of the test domains on the four high budget levels from B_4 to B_7 and taken from eight of the case studies and the homogeneous instance set. The box-plots for the ranking of each of the explored settings are given in Figure 2.

Considering the versions without post-selection (left six boxes in each plot), the best setting of nr appears to be 5 or 10, in accordance to what was observed in the previous section for uniform random sampling. However, for the versions with post-selection (right six yellow boxes dubbed with "P" for post-selection), the best setting is $nr = 1$, resulting in the best ranking improving also over the *a posterior* best settings of nr for BOBYQA and MADS without post-selection. Hence, the overall best performance with post-selection is obtained when during the run of the search method each candidate configuration is evaluated on one same instance.

Saving evaluations allows to evaluate more configurations and to obtain more restart-best configurations, which then are evaluated more carefully in the elite selection phase.

4.3 Advanced settings of post-selection

Next, we extended the study in Sec. 4.2 by considering also the low budget levels B_1, B_2, and B_3. Here, we empirically examine several advanced settings of post-selection, including nr setting (Sec. 4.3.1), alternating instances (Sec. 4.3.2, dubbed A in Figure 3), early qualification (Sec. 4.3.3, dubbed E in Figure 3), and iterated selection hybrid (Sec. 4.3.4, dubbed IS in Figure 3).

4.3.1 nr setting

As high nr settings perform poorly with post-selection (see Fig. 2), here we consider only values of $nr \in \{1, 3, 5\}$. In Fig. 3(a-d), it can generally be observed that the best post-selection configurator in each plot uses $nr = 1$. Considering the curves identified by "$nr = 1, P$", "$nr = 3, P$", and "$nr = 5, P$" in Fig. 3(a-d), $nr = 1$ ranks either best or very well, and $nr = 3, 5$ rank slightly better only in the smallest budget levels of BOBYQA-based configurators (Fig. 3(a,b)) and in high budget levels of CMA-ES-based configurators (Fig. 3(d)). The reason for the latter is further discussed and addressed in Sec. 4.3.3.

4.3.2 Alternating instances

Note that in our basic setting each configurator restart uses the same instances (or the same instance if $nr = 1$) and only in the final elite selection phase different instances would be used. This may lead to poor results especially on instance sets where instances are heterogeneous (as in our `het` set—for this instance set good $\mathcal{MM}$AS settings are known to depend on instance size).

Here we consider a different variant, where we use *alternating instances* instead of fixed instances; hence, each elite configuration is qualified through nr different instances. We compared this approach empirically with the basic post-selection using fixed instances, taking BOBYQA as case study across 105 test domains of configuring $\mathcal{MM}$AS in both homogeneous instance set (`hom`, see Fig. 3(a)) and the heterogeneous instance set (`het`, see Fig. 3(b)). As $nr = 1$ is the best nr setting for post-selection, we compared directly $nr = 1$ fixed instance ("$nr = 1, P$" in Fig. 3(a, b)) with alternating instances ("$nr = 1, P, A$'). In `hom`, using alternating instance performs as well as using a same, fixed instance and no statistically significant difference was detected (p-value 0.3). However, in `het`, using alternating instances leads to significant improvement. We included also $nr = 3$ alternating instances ("$nr = 3, P, A$") for `het`; it again performs significantly better than using $nr = 3$ same instances ("$nr = 3, P$"), but significantly worse than $nr = 1$ with alternating instance. To sum up, using alternating instances results in better performance, especially when the target instance set is heterogeneous.

4.3.3 Early qualification

CMA-ES is the only of the three search methods, where the setting $nr = 1$ does not perform as well as $nr > 1$ especially for higher budget levels (see Fig. 3(d)). One reason is probably that CMA-ES is slower than MADS and BOBYQA to converge and restart and in our basic setting only restart-best configurations qualify for the elite selection. However, one may obtain more configurations for post-selection by qualifying configurations earlier, as done, e.g., by picking all iteration-best configurations instead of only the restart-best. Besides, as suggested in Sec. 4.3.2 for BOBYQA, each iteration may use *alternating* instances for evaluation. This new setting "$nr = 1, P, A, E$" (E for *early qualification*) of CMA-ES configurator is shown in Fig. 3(d) to be the significantly best-performing configurator on all budget levels.

4.3.4 Iterated selection with post-selection

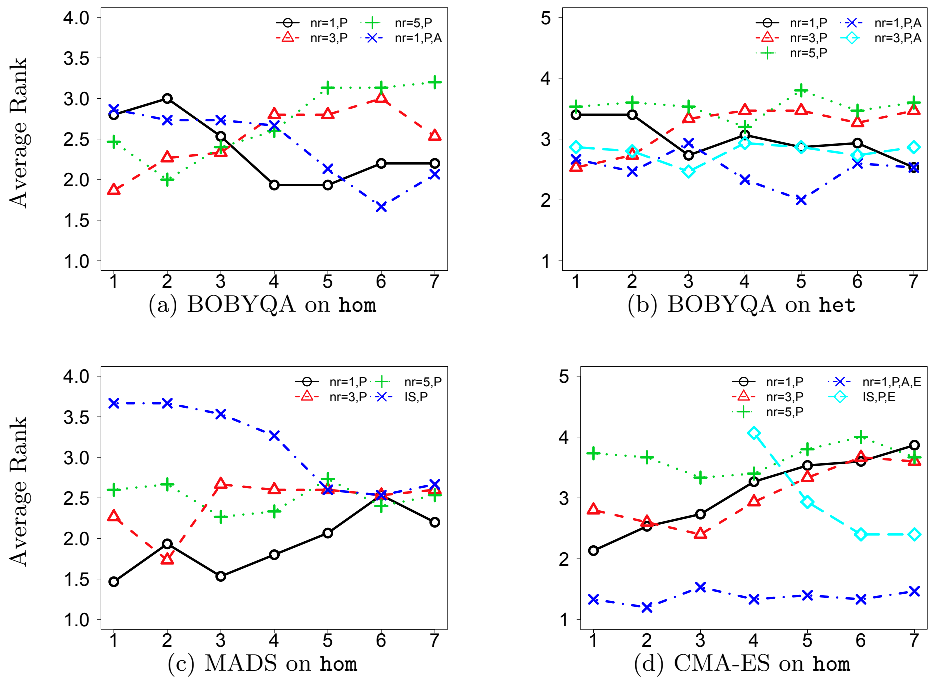
Instead of using a fixed number of nr instances, one may apply iterated selection during the elite qualification phase. Such examples include MADS/F-Race [21, 22] and CMA-ES/F-Race [21], where F-Race is not only used in the elite selection phase to select the best of the elite configurations, but also used within each iteration of the search method in the elite qualification phase to select the best among the incumbent and newly-generated configurations.[1] Besides, our CMA-ES/F-Race applies also the idea of *early qualification* (Sec. 4.3.3), i.e. both iteration-best and restart-best configurations are qualified as elites. However, this interesting hybrid, either MADS/F-Race ("IS, P" in Fig. 3(c)) or CMA-ES/F-Race ("IS, P, E" in Fig. 3(d)), does not perform well compared with the other post-selection variants derived in this work. MADS/F-Race is significantly outperformed by post-selection with $nr \leq 5$, despite the better performance of MADS/F-Race over MADS with fixed nr evaluations without post-selection [22]. CMA-ES/F-Race is also significantly outperformed by post-selection CMA-ES with one alternating instance and early qualification.

5. POST-SELECTION VS. I/F-RACE

We compare the best post-selection configurators with I/F-Race, a state-of-the-art iterated selection configurator [8]. Additionally, we compare also to U/F-Race, which generates configurations uniformly at random and then selects the best by F-Race. As the best post-selection configurators we select the best setting for each of the three search methods found in Sec. 4.3, including BOBYQA with one alternating instances ("$nr = 1, P, A$" in Fig. 3(a,b)), CMA-ES with one alternating instance and early qualification ("$nr = 1, P, A, E$" in Fig. 3(d)) and MADS with $nr = 1$ ("$nr = 1, P$" in Fig. 3(c), only shown in `hom`).

Fig. 4 shows the comparison of these configurators in dependence of the budget level (top row) and the number of parameters to be configured (bottom row) on the homogeneous (left column) and the heterogeneous instance set (right column). The clear winner is the CMA-ES configurator: it significantly outperforms all other configurators in almost every budget level and every number of parameters being configured. BOBYQA generally performs well in case studies with 2, 3, or 4 parameters being configured, but its performance declines in case studies with 5 or 6 parameters, as shown in Fig. 4(c) and Fig. 4(d). I/F-Race is only applicable in the four high budget levels due to its default parameter settings, and it is outperformed by CMA-ES. MADS is not considered in the experiments of `het` due to its unsatisfactory performance in `hom`. All the above-mentioned configurators outperform U/F-Race.

[1]Note that in BOBYQA, each configuration has to be evaluated on the same number of instances due to the way its quadratic model is built; therefore, F-Race cannot be combined with BOBYQA in the iterated selection manner.



(a) BOBYQA on `hom` (b) BOBYQA on `het` (c) MADS on `hom` (d) CMA-ES on `hom`

Figure 3: Average ranks of different settings for post-selection. These settings include nr values (Sec. 4.3.1), "P" for post-selection (Algo. 1), "A" for alternating instances (Sec. 4.3.2), "E" for early qualification (Sec. 4.3.3), or "IS" for iterated selection hybrid (Sec. 4.3.4). This study is done using three search methods as testbed, BOBYQA, MADS, and CMA-ES, and each setting is tested on seven budget levels (shown in the X-axis) for 15 case studies of $\mathcal{MM}$AS with either homogeneous (`hom`) or heterogeneous (`het`) instance sets.

6. POST-SELECTION IN PARAMILS

For a final set of experiments we introduce post-selection into ParamILS with the goal of comparing it to the *intensification* mechanism used in FocusedILS. We adopted the version 2.3.5 of ParamILS [11], kept the search mechanism (ILS), and adapted its *intensification* mechanism into a post-selection mechanism in a straightforward manner. Since the best setting of post-selection found in Sec. 4.2 is using $nr = 1$, alternating instance, BasicILS(1) is adopted for the elite qualification phase of post-selection. The main question then is how to define configurations that qualify as elite. In this study, only the best configuration found in each restart is qualified. We adopted three restart schemes.

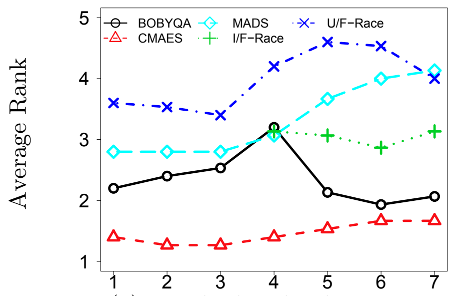
Natural restart. We restart ParamILS either when it is naturally restarted as triggered by the parameter $p_{restart}$ (set to 0.01 by default) or when the search falls into a local optimum and perturbation starts. Post-selection ParamILS with natural restart is denoted as `PPn`.

Fixed early restart. We enforce ParamILS to restart earlier so as to qualify more elites. The simplest way to enforce early restart is to restrict the maximum number of evaluation B_r for each run to a small value. Considering that each ParamILS run evaluates 10 uniformly random initial configurations before starting ILS, $B_r = 30$ appears to be a setting that allows reasonable exploitation while keeping reasonably frequent restarts. This version is denoted as `PP30`.

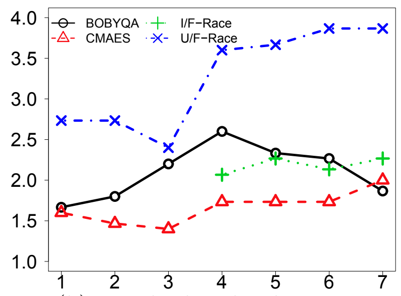
Incremental early restart. Besides fixing B_r, we also consider incrementing B_r by 10 from restart to restart, i.e. let $B_r = 10$ in the first restart, increment B_r to 20 in the second restart, $B_r = 30$ in the third, etc. Post-selection ParamILS with incremental early restart is denoted as `PPi`.

We compared FocusedILS with the three versions of post-selection ParamILS, `PPn`, `PP30`, and `PPi` on the six case studies of $\mathcal{MM}$AS with five or six parameters to be configured. Both homogeneous and heterogeneous instance sets are considered. Since ParamILS handles only discrete parameters, each parameter of our case studies is discretized into 10 equidistant values. ParamILS does not support standardized z-score normalization, and so we adopted the *mean* algorithm performance as the objective measure. Accordingly, the post-selection applies a t-Race without adjustment for multiple comparisons [6] instead of F-Race.

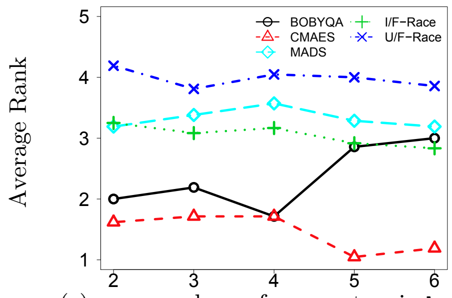
The results are presented in Fig. 5. They show that post-selection with early restart, especially `PP30`, is clearly the best configurator in budget levels B_1 to B_6. FocusedILS performs better than `PP30` only in the highest budget level B_7. `PPn` doesn't perform very well as expected, since it usually takes around 100 to 400 evaluations to reach a natural restart; this leads to very few elite configurations, which greatly worsens the impact of post-selection. Enforcing early restart in `PP30` and `PPi` proves to be a more successful setting of post-selection than natural restart. However, frequent restart may weaken the exploitation ability in finding promising configurations during the elite qualification phase. A better approach than enforcing early restart is to use *early qualification* as done for CMA-ES in Sec. 4.3.3, qualifying elite configurations without interrupting the search procedure. However, we leave this possibility for future research.



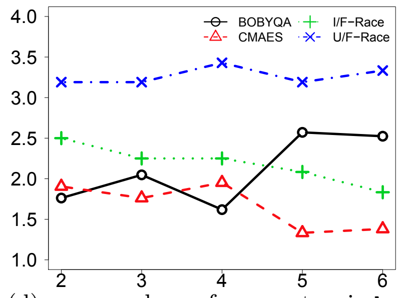
(a) over budget levels in `hom`



(b) over budget levels in `het`



(c) over numbers of parameters in `hom`



(d) over numbers of parameters in `het`

Figure 4: Comparison of best post-selection configurators with I/F-Race U/F-Race for configuring 15 case studies of $\mathcal{MM}$AS with either homogeneous (`hom`) or heterogeneous (`het`) instance sets.

7. DISCUSSION AND CONCLUSIONS

Post-selection adopts two distinct phases in the automatic algorithm configuration process. In the elite qualification phase, a number of elite configurations are identified, for example, by independent runs of some algorithm configurator. The subsequent elite selection phase tries to identify then the best of these elite configurations, for example, by a racing method. In this paper, we have examined in more detail such a post-selection mechanism, proposed earlier in [21], using the example application of algorithm configurators for setting numerical parameters of $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System applied to the traveling salesperson problem (TSP). Our analysis of post-selection showed that it is enough to evaluate candidate configurations on rather few instances during the elite qualification stage. In our case studies only one instance was even enough, but we expect that on other configuration tasks with more heterogeneous instances than in the TSP a larger number of instances in the elite qualification stage may be better. If the configurator in the elite qualification phase cannot gather many elite configurations, enforcing early restarts or an *early qualification* mechanism, as proposed in this paper, may be useful. Overall, our results showed that post-selection is a promising approach that should receive further attention. In addition, we identified a post-selection CMA-ES configurator with alternating instances and early qualification, as a high-performing configurator for setting numerical parameters.

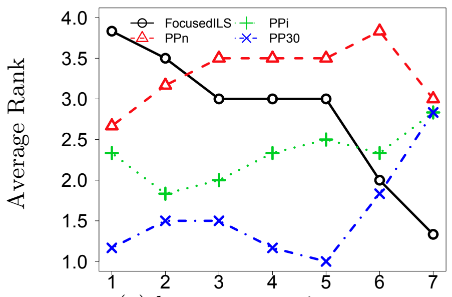
In future work, we plan to test the effectiveness of post-selection on other configuration tasks with more parameters and to explore complementing other configuration methods with post-selection. The fact that with post-selection, each run of automatic algorithm configuration method in the elite qualification stage may use few same instances, makes post-selection also applicable to model-based search methods such as SPO, which cannot easily be enhanced by iterated selection methods that evaluate candidate configurations with different numbers of instances (see BOBYQA in Sec. 4.3.4 as an example). Furthermore, post-selection can be seen as a form of stochasticity handling and it may also be useful for optimization problems with noise, e.g., it may be integrated into deterministic algorithms for optimizing noisy functions such as those of the black-box optimization benchmarking workshop series [10]. The positive results obtained with post-selection in this paper indicate that the directions outlined above are promising ideas to pursue.

Acknowledgments

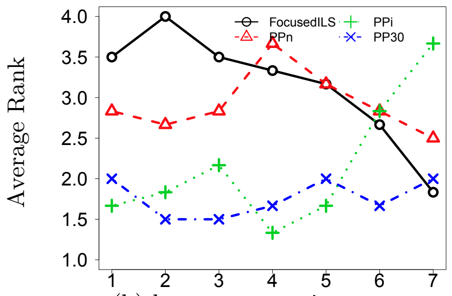
This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium and support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*". This research and its results have also received funding from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Zhi Yuan, Mauro Birattari, and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS. Zhi Yuan also acknowledges support by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

8. REFERENCES

[1] B. Adenso-Díaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.



(a) homogeneous instances



(b) heterogeneous instances

Figure 5: Average ranks of four ParamILS-based configurators over seven budget levels for configuring six case studies of $\mathcal{MMAS}$ with either (a) homogeneous or (b) heterogeneous instance set. The four ParamILS versions include FocusedILS, and post-selection variants PPn, PP30, and PPi.

[2] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of solvers. In I. P. Gent, editor, *CP 2009*, volume 5732 of *LNCS*, pages 142–157. Springer, Heidelberg, Germany, 2009.

[3] C. Audet and J. J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[4] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein et al., editors, *HM'2007*, volume 4771 of *LNCS*, pages 108–122. Springer, Heidelberg, Germany, 2007.

[5] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In *CEC 2005*, pages 773–780. IEEE, 2005.

[6] M. Birattari. *Tuning Metaheuristics: A machine learning perspective*. Springer, Berlin, Germany, 2009.

[7] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *GECCO 2002*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[8] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and iterated F-Race: An overview. In T. Bartz-Beielstein et al., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Germany, 2010.

[9] N. Hansen. The CMA evolution strategy: a comparing review. In J. Lozano et al., editors, *Towards a new evolutionary computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 75–102. Springer, Berlin, Germany, 2006.

[10] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions. Technical Report RR-6869, INRIA, 2009.

[11] F. Hutter. Software ParamILS. `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS`, 2010.

[12] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. LION 2011. Volume 6683 of LNCS*, pages 507–523. Springer, 2011.

[13] F. Hutter, H. Hoos, and K. Leyton-Brown. Tradeoffs in the empirical evaluation of competing algorithm designs. *Annals of Mathematics and Artificial Intelligence*, 60(1):65–89, 2010.

[14] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In F. Rothlauf, editor, *GECCO 2009*, pages 271–278. ACM press, New York, 2009.

[15] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[16] O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J. D. Cowan et al., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 59–66. Morgan Kaufmann Publishers, San Francisco, USA, 1994.

[17] V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proc. of IJCAI 2007*, pages 975–980. AAAI Press/IJCAI, Menlo Park, CA, 2007.

[18] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report NA2009/06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK, 2009.

[19] T. Stützle. Software ACOTSP. `http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html`, 2002.

[20] T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[21] Z. Yuan, M. Montes de Oca, M. Birattari, and T. Stützle. Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, 6(1):49–75, 2012.

[22] Z. Yuan, T. Stützle, and M. Birattari. MADS/F-Race: mesh adaptive direct search meets F-race. In M. Ali et al., editors, *Proceedings of IEA-AIE 2010*, volume 6096 of *LNAI*, pages 41–50. Springer Verlag, Heidelberg, Germany, 2010.