`MADS/F-Race`: Mesh Adaptive Direct Search Meets F-Race

Zhi Yuan, Thomas Stützle, and Mauro Birattari

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{zyuan,stuetzle,mbiro}@ulb.ac.be

Abstract. Finding appropriate parameter settings of parameterized algorithms or AI systems is an ubiquitous task in many practical applications. This task is usually tedious and time-consuming. To reduce human intervention, the study of methods for automated algorithm configuration has received increasing attention in recent years.

In this article, we study the mesh adaptive direct search (MADS) method for the configuration of parameterized algorithms. MADS is a direct search method for continuous, global optimization. For handling the stochasticity involved in evaluating the algorithm to be configured, we hybridized MADS with F-Race, a racing method that adaptively allocates an appropriate number of evaluations to each member of a population of candidate algorithm configurations. We experimentally study this hybrid of MADS and F-Race (MADS/F-Race) and compare it to other ways of defining the number of evaluations of each candidate configuration and to another method called I/F-Race. This comparison confirms the good performance and robustness of MADS/F-Race.

1 Introduction

The performance of parameterized algorithms for computationally hard problems depends on the appropriate choice of parameter values. Parameter settings are still often searched in a trial-and-error fashion and typically this is a time-consuming, labor intensive process. In this article, we focus on the *offline* algorithm configuration problem (ACP) [1]. (We call an *algorithm configuration* the full specification of all algorithm parameters, that is, a fully instantiated algorithm.) In the offline ACP, one distinguishes two phases. In a first *training phase*, an algorithm configuration is determined that minimizes the expected cost of the solutions generated by the algorithm. This algorithm configuration is then used in a *production phase*, where the algorithm is run on unseen instances. It is important to stress that the crucial aspect here is generalization, that is, the ability of the tuning method to find parameter settings that work well on previously unseen instances.

In the settings we are interested in, the evaluation of a candidate algorithm configuration is stochastic. Stochasticity may arise due to two main reasons. Firstly, the algorithm may take randomized decisions during the search. This is a main characteristic of stochastic local search (SLS) algorithms [2], but also of any randomized search algorithms. Secondly, the performance of an algorithm, even if it is deterministic, usually depends on the particular instance being tackled. In fact, the particular instance can naturally be seen as being drawn according to some, typically unknown, probability distribution; it is, in this sense, a second stochastic factor. Tackling this stochastic aspect of the problem was the objective of the `F-Race` method [3]. The crucial idea is to evaluate a set of candidate algorithm configurations in parallel on a stream of instances and to use a particular racing method for determining the best one. In `F-Race`, at each evaluation step, that is, after each additional instance, the non-parametric family-wise Friedman test is applied to check whether there is evidence that at least one of the configurations is significantly different from the others. If the null hypothesis of no differences is rejected, Friedman post-tests are applied to eliminate those candidate configurations that are significantly worse than the best one.

`F-Race` is a general method that can be used to select the best candidate under stochastic evaluation. It can be used independent of the way the candidate algorithm configurations are actually generated. In this article, we adopt an iterative sampling method called mesh adaptive direct search (`MADS`) [4]. `MADS` is a direct search method for continuous function optimization. In fact, in this article we consider only configuration tasks with numerical parameters with large domains: either these parameters are continuous, or they have a (large) integer domain such that rounding real numbers to the closest integers is reasonable. The latter is usually the case for many algorithm parameters such as population sizes or tabu list lengths. One contribution of the article is the hybridization of `MADS` with `F-Race` for the evaluation of the candidate algorithm configurations, resulting in the `MADS/F-Race` algorithm. Another contribution is the application of the leave-one-out cross-validation for experimental comparison of `MADS/F-Race` to alternative uses of `MADS`. Finally `MADS/F-Race` is also compared to the state-of-the-art tuning method `I/F-Race` [5].

2 Combining `MADS` and `F-Race`

The `MADS` class of algorithms tackles global optimization problems that can be stated as

$$\min_{p \in \Omega} f(p) \tag{1}$$

for $f : \Omega \in \mathbb{R}^d \rightarrow \mathbb{R} \cup +\infty$, where $d = |\Omega|$ is the dimensionality of the variable space. The algorithm does not require derivative information or continuity of f. In fact, the evaluation function f can be considered as a black-box. `MADS` algorithms are reported to be robust for non-smooth optimization problems [4], and its convergence properties given only bound constraints is shown in [6]. It is an extension to the generalized pattern search [7] by allowing a more general exploration of the variable space.

Algorithm 1. The mesh adaptive direct search

```
Step 0: Initialization Given $p_0$, a feasible initial point, $\Delta_0 > 0$, set $k = 0$ and go to Step 1.
Step 1: search step Let $L_k \subset M_k$ be the finite set of mesh points, and let $q^* = \arg\min_{q \in L_k} f(q)$ be the iteration best point from search. If $f(q^*) < f(p_k)$, declare $k$ successful and set $p_{k+1} = q^*$ and go to Step 3; otherwise go to Step 2.
Step 2: poll step Generate the frame $F_k$ as in Eq. 3, and let $q^* = \arg\min_{q \in F_k} f(q)$ be the iteration best point from poll. If $f(q^*) < f(p_k)$, declare $k$ successful and set $p_{k+1} = q^*$; otherwise declare $k$ unsuccessful. Go to Step 3.
Step 3: Parameter update If iteration $k$ is unsuccessful, set $p_{k+1} = p_k$, otherwise set $p_{k+1}$ as the improved mesh point. Update mesh size parameter $\Delta_k$ according to Eq. 4, $k \leftarrow k + 1$ and go to Step 1.
```

2.1 A General Overview of MADS

MADS is an iterative algorithm that generates at each iteration a population of trial points. The trial points must lie on the *current mesh*, whose coarseness is controlled by a *mesh size parameter* $\Delta_k \in \mathbb{R}_+$. At iteration k, the set of mesh points is given by

$$M_k = \bigcup_{p \in S_k} \{p + \Delta_k z : z \in \mathbb{Z}^d\} \tag{2}$$

where S_k is the point set that was evaluated in the previous iterations. At each iteration of MADS, two steps are applied, the `search` step and the `poll` step. In the `search` step, trial points are randomly sampled on the current mesh. In the `poll` step, trial points are generated around the incumbent one. The trial points of `poll` form the set called *frame*:

$$F_k = \{p_k \pm \Delta_k b : b \in B_k\} \tag{3}$$

where $B_k = \{b^1, b^2, \ldots, b^d\}$ is a basis in $\mathbb{Z}^d$. At iteration k, the mesh size parameter is updated as follows:

$$\Delta_{k+1} = \begin{cases} \Delta_k/4 & \text{if no improved mesh point is found;} \\ 4\Delta_k & \text{if an improved mesh point is found, and } \Delta_k \le \frac{1}{4}; \\ \Delta_k & \text{otherwise.} \end{cases} \tag{4}$$

An outline of the MADS algorithm is given in Algorithm 1.

2.2 The Hybrid of F-Race and MADS

The stochasticity associated to the evaluation of candidate algorithm configurations usually requires to reevaluate a configuration several times to reduce the variance of the performance estimate. The most straightforward way of doing so is to apply each candidate algorithm configuration a fixed, same number of times; this variant we call MADS(fixed). The number of times an algorithm candidate configuration is run in MADS(fixed) we denote by *nr*, where *nr* is a parameter.

Table 1. Parameters used in `MADS`. d denotes the dimension of the parameter space.

parameter	value
Initial search type	Latin hypercube search
#Points in initial search	d^2
Iterative search type	random search
#Points in iterative search	$2d$
#Points in `poll` step	$2d$
Speculative search	yes
Initial mesh size parameter	1.0

Two disadvantages of `MADS(fixed)` are that (i) a priori an appropriate value of nr is not known, and (ii) the same amount of computation time is allocated to good and bad performing candidates, wasting useful computation.

These disadvantages are reduced by hybridizing `F-Race` and `MADS`, resulting in the `MADS/F-Race` algorithm. This hybrid can be obtained in a rather straightforward way. At each `MADS` iteration, independent of whether it is the `search` or `poll` step, the population of candidate configurations sampled by `MADS` is evaluated by `F-Race`, together with the incumbent point p_k. In this hybrid, `F-Race` identifies the best point among the sampled ones and $\{p_k\}$. If `F-Race` determined p_k as the best, the iteration is declared unsuccessful; otherwise the iteration is successful, and the `MADS` parameters are updated accordingly.

2.3 Implementation Details of `MADS/F-Race`

For the setting of the `MADS` algorithm, we follow [6]. The specific parameter values are listed in Table 1. For `MADS/F-Race`, we essentially only need to set the maximum number of algorithm evaluations we allow in each application of `F-Race`. Here, we explore a fixed setting of 10 times the number N_l of trial points generated from `MADS` at the l-th iteration.

3 Case Study: `MADS/F-Race` vs. `MADS(fixed)`

We have compared the performance of `MADS/F-Race` and `MADS(fixed)` on six benchmark *configuration problems*. Each *configuration problem* consists of a parameterized algorithm to be configured, and an optimization problem to which this algorithm is applied. The six configuration problems are listed below:

– MMASTSP: the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) algorithm [8] applied to the traveling salesman problem (TSP). There are six parameters of $\mathcal{MM}$AS to be configured (listed in Table 2). No local search is applied.
– MMASTSP_ls: the $\mathcal{MM}$AS applied to TSP with 2-opt local search on solutions constructed by each ant. The six parameters to be configured are the same as MMASTSP.
– ACSTSP: the ant colony system (ACS) algorithm [9] for the TSP. The six parameters to be configured are listed in Table 2. No local search is applied.

Table 2. Range of each parameter considered for configuring MMASTSP (including MMASTSP_ls) and ACSTSP (including ACSTSP_ls)

`MMASTSP`		`ACSTSP`	
parameter	range	parameter	range
α	$[0.0, 5.0]$	α	$[0.0, 5.0]$
β	$[0.0, 10.0]$	β	$[0.0, 10.0]$
ρ	$[0.0, 1.00]$	ρ	$[0.0, 1.00]$
γ	$[0.01, 5.00]$	q_0	$[0.0, 1.0]$
m	$[1, 1200]$	m	$[1, 1200]$
nn	$[5, 100]$	nn	$[5, 100]$

Table 3. Range of each parameter considered for configuring ILSQAP. The individual parameters refer to parameters concerning the tabu search algorithm, the simulated-annealing type of acceptance criterion of the ILS, and the perturbation strength.

parameter	range	parameter	range	parameter	range	parameter	range
tc	$[0, 5]$	ia	$[3, 99]$	iu	$[0, 2]$	iy	$[0, 1]$
ti	$[0, 20]$	it	$[0, 10]$	ix	$[1, 100]$	iz	$[0, 1]$

– ACSTSP_ls: ACS applied to TSP with 2-opt local search. The six parameters to be configured are the same as in ACSTSP.
– ILSQAP: an iterated local search (ILS) algorithm [10] applied to the quadratic assignment problem (QAP). This ILS algorithm applies a robust tabu search as the local search and a simulated annealing type acceptance criterion. The eight parameters to be configured are listed in Table 3;
– SAQAP: the simulated annealing (SA) for QAP with three parameters to be configured (Table 4).

In our experiments, each of the configuration problems contains 12 *domains*. These are defined by considering for each configuration problem four computation time limits (1, 2, 5, and 10 CPU seconds) for stopping the algorithm to be configured, and three values of *configuration budget* (1000, 2000, and 4000). The term *configuration budget* is determined as the maximum number of times that the algorithm to be configured can be applied during the configuration process. Take `MADS(fixed)` for example, given a configuration budget of 1000, and $nr = 20$, then each parameter configuration will be evaluated 20 times, and in total 50 different parameter configurations will be evaluated.

3.1 Comparison of `MADS/F-Race` to Random `MADS(fixed)`

As a first benchmark test for `MADS/F-Race`, we compare its performance to `MADS(fixed)`, for which six values for nr are tested: $nr \in \{1, 5, 10, 20, 30, 40\}$. Given no a priori information about the best choice for nr, we select one of the six values randomly for each problem domain, and compare it with `MADS/F-Race`

Table 4. Range of each parameter considered for configuring SAQAP; the parameters define the annealing schedule

parameter	range	parameter	range	parameter	range
sb	$[0.0, 10.0]$	sc	$[0.0, 1.0]$	sd	$[1, 100000]$

across all domains, with blocking on each instance. The comparison is done using the Wilcoxon signed rank test with continuity correction; the α-level chosen is 0.05. The experimental results show that `MADS/F-Race` statistically significantly performs better than `MADS(fixed)` with randomly chosen nr on each individual domain. The average percentage improvement of the cost obtained by `MADS/F-Race` over the cost obtained by the random `MADS(fixed)` is around 0.25%.

3.2 The Leave-One-Out Cross-Validation

The dominance of `MADS/F-Race` over a random selection of the value nr for `MADS(fixed)` is convincing if no a priori knowledge about an appropriate value for nr is available. As our knowledge of `MADS(fixed)` grows, we may learn a good setting for `MADS(fixed)`, with which it can be applied to an unknown problem domain. In the sequel, we study how well the knowledge of `MADS(fixed)` over the learned domains can be generalized to an unknown domain. To this end, a statistical technique named leave-one-out cross-validation is applied.

Cross-validation is a technique that is commonly used in machine learning and statistics, to assess how the results of statistical analyses are generalized to an independent data set. To do so, on each cross-validation iteration, the available data set is partitioned into two sets, the training set based on which the quality of the candidate settings are observed and analyzed, and the test set on which the previously assessed results are tested. This process is iterated, using different partitions in each iteration, to reduce the variability of the validation.

There are various ways how the cross-validation can be performed. In our case, the common leave-one-out cross-validation is applied. In each iteration, one domain is picked for testing, and the rest of the domains serve as the training set. The best candidate chosen in the training set is applied on the test domain, and its results are collected into a validation set. The process repeats until each domain has collected its validation data. A more formal description of how we apply the leave-one-out cross-validation is given in Algorithm 2.

The leave-one-out cross-validation in our experiments takes as candidates the `MADS(fixed)` with six values of nr. The goal is to collect the validation set as described above, and compare it with the results obtained by `MADS/F-Race`. Note that the data collected in the validation set are first trained, while `MADS/F-Race` is not. The unfairness of the comparison will show the advantage of `MADS/F-Race` in robustness over the naive `MADS(fixed)`.

Algorithm 2. The leave-one-out cross-validation

```
Given a set of domains $D$, a set of candidate algorithms $A^c$, and the function $c(a, E)$
referring to the expected cost of algorithm $a \in A^c$ on a set of domains $E \in D$.
Goal is to collect the validation set $V^c$
Set $V^c \leftarrow \emptyset$
for $d \in D$ do
    Select $\bar{a} = \arg\min_{a \in A^c} c(a, D \setminus \{d\})$
    Set $V^c \leftarrow V^c \cup \{c(\bar{a}, d)\}$
end for
Return $V^c$ for assessment
```

3.3 Comparison of MADS/F-Race to Tuned MADS(fixed)

Here we compare MADS/F-Race with the tuned version of MADS(fixed). Firstly we applied the leave-one-out cross-validation for MADS(fixed) over all the six configuration problems. This is done as follows. Across $12 \cdot 6 = 72$ domains, the validation set of each domain is collected by applying MADS(fixed) with the value of nr, which has the best performance for the other 71 domains. We observed that the best value of nr determined by each combination of the 71 domains is always five. As previously, we apply the Wilcoxon test. It shows that MADS/F-Race significantly outperforms the tuned MADS(fixed). The observed difference in the mean performance between the tuned MADS(fixed) and MADS/F-Race is 0.17%.

Moreover, we observed that the best setting of nr in MADS(fixed) may differ for each individual configuration problem. To further fine-tune nr for MADS(fixed), we did a blocking on the set of domains by configuration problems, and applied the leave-one-out cross-validation on 12 domains of each configuration problem separately. Since nr is better tuned, the results of MADS(fixed) in this case are better than obtained by MADS(fixed) tuned across all configuration problems. The comparison results of MADS/F-Race and MADS(fixed) tuned in individual configuration problems are listed in Table 5, together with the best value of nr for each configuration problem. It shows that MADS/F-Race obtains significantly better results than tuned MADS(fixed) on individual configuration problems MMASTSP, ACSTSP and SAQAP, while in the rest of the configuration problems MMASTSP_ls ACSTSP_ls and ILSQAP, MADS/F-Race has significantly worse results than tuned MADS(fixed). Also note that when tuning nr for individual configuration problem, the tuned nr values vary a lot from 5 to 40.

The fact that the advantage of MADS/F-Race is more clear in some domains than in others motivates us to examine factors of the configuration problems that may affect the performance of the configuration algorithms MADS(fixed) and MADS/F-Race. Given that F-Race is particularly designed to deal with the stochastic aspect of the problem, one conjecture is that for problem domains where the stochasticity is relatively high, better performance is obtained by using F-Race. As a scale-free indicator of the variability of the configurations in a problem domain, we have investigated the *variation coefficient*. The variation coefficient of each configuration problem is obtained as follows. Firstly, 200

Table 5. Comparison results of `MADS/F-Race` and `MADS(fixed)` with nr tuned in six individual configuration problems separately by leave-one-out cross-validation. The column entries with label "best.nr" show the tuned nr value for each configuration problem. It is worth noticing that, in each cross-validation process, this nr value selected by each training set can vary. But in our case, it happened to remain constant. The column entries with the label `per.dev` show the mean percentage deviation of cost obtained by `MADS/F-Race` comparing with tuned `MADS(fixed)`. $+x$ $(-x)$ means that the solution cost of `MADS/F-Race` is x% more (less) than tuned `MADS(fixed)`, i.e. `MADS/F-Race` performs x% worse (better) than tuned `MADS(fixed)`.

problem	`per.dev`	best.nr	problem	`per.dev`	best.nr
MMASTSP	−0.12	5	MMASTSP_ls	+0.055	5
ACSTSP	−0.64	40	ACSTSP_ls	+0.027	40
SAQAP	−0.030	30	ILSQAP	+0.13	40

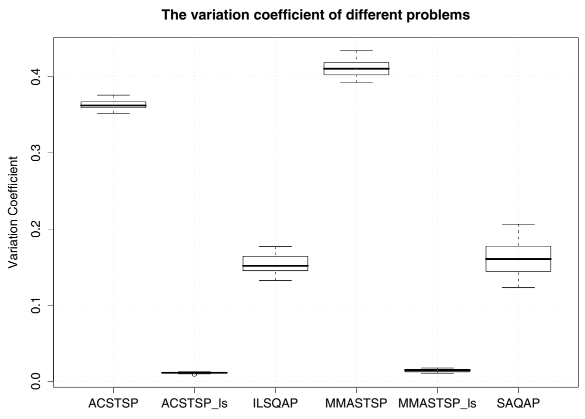

Fig. 1. The variation coefficient of each problem used

randomly generated parameter settings are applied on 25 generated instances. Then, for each of the instances we compute the ratio of the standard deviation over the mean cost of the 200 parameter settings. A high value of the variation coefficient indicates steep slopes in the search landscape, while a low value of the variation coefficient indicates a flat landscape, which is often the case for algorithms when a strong local search is adopted. The box-plot of the measured variation coefficients on the six configuration problems is shown in Figure 1.

Interestingly, the two configuration problems MMASTSP and ACSTSP, on which `MADS/F-Race` performs particularly well, are the two with the highest variation coefficient. Although the variation coefficient of SAQAP is not significantly higher than that of ILSQAP, it is still much larger than that of ACSTSP_ls and MMASTSP_ls. This gives further evidence for our conjecture that the impact of `F-Race` is the larger the higher the variability of algorithm performance.

Table 6. Comparison of MADS/F-Race and I/F-Race for configuring six benchmark configuration problems. The column entries with the label per.dev shows the mean percentage deviation of cost obtained by MADS/F-Race comparing with I/F-Race. $+x$ $(-x)$ means that the solution cost of MADS/F-Race is x% more (less) than I/F-Race, i.e. MADS/F-Race performs x% worse (better) than I/F-Race.

problem	per.dev	problem	per.dev
MMASTSP	−0.34	MMASTSP_ls	+0.053
ACSTSP	+1.12	ACSTSP_ls	−0.064
SAQAP	−0.80	ILSQAP	+0.48

3.4 Comparison of MADS/F-Race and I/F-Race

Finally, we compared the proposed MADS/F-Race to I/F-Race, another state-of-the-art configuration algorithm [5]. I/F-Race was applied to the same benchmark configuration problems as MADS/F-Race. In each domain one configuration procedure is run, and the tuned parameters are evaluated on 300 test instances. All the results are compared using paired Wilcoxon's signed rank test. When measured across all test domains, we could not detect statistically significant differences between MADS/F-Race and I/F-Race (p-value was 0.82). However, analyzing the results separately for each configuration problem (see Table 6), MADS/F-Race is significantly better than I/F-Race on domains MMASTSP, ACSTSP_ls and SAQAP, and significantly worse on domains ACSTSP, MMASTSP_ls and ILSQAP.

4 Conclusions

In this article, we have explored the hybridization of MADS with F-Race. Computational results have shown that this hybridization is successful. Further analysis of the performance variability among the benchmark configuration problems has shown that it is particularly useful for tackling the stochastic aspect of the problem. This insight shed also more light on situations under which similar combinations of optimization methods with statistical methods such as racing [11,12] are particularly useful.

In future works, we will examine possibilities for improving MADS/F-Race and also I/F-Race by considering alternative ways of defining the races by, for example, re-using previous evaluations. Another direction is the study of other continuous optimization techniques for algorithm configuration to widen the set of available methods for automated algorithm configuration. We strongly believe that the integration of appropriate statistical methodologies with effective sampling approaches will allow a further boost of the performance of automated algorithm configuration tools.

Acknowledgments. We would like to thank Mohamed Saifullah bin Hussin to make available his iterated local search and simulated annealing implementations for the quadratic assignment problem. This work has been supported by META-X, an ARC project funded by the French Community of Belgium. The authors acknowledge support from the fund for scientific research F.R.S.-FNRS of the French Community of Belgium, of which they are research associates (MB and TS), or an aspirant (ZY), respectively.

References

1. Birattari, M.: Tuning Metaheuristics: A machine learning perspective. Springer, Berlin (2009)
2. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann Publishers, San Francisco (2005)
3. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Langdon, W.B., et al. (eds.) Proceedings of GECCO 2002, pp. 11–18. Morgan Kaufmann Publishers, San Francisco (2002)
4. Audet, C., Dennis, J.: Mesh adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization 17(1), 188–217 (2007)
5. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: An overview. In: Bartz-Beielstein, T., et al. (eds.) Experimental Methods for the Analysis of Optimization Algorithms. Springer, Berlin (to appear)
6. Audet, C., Orban, D.: Finding optimal algorithmic parameters using derivative-free optimization. SIAM Journal on Optimization 17, 642 (2006)
7. Audet, C., Dennis, J.: Analysis of generalized pattern searches. SIAM Journal on Optimization 13, 889–903 (2000)
8. Stützle, T., Hoos, H.H.: $\mathcal{MAX}--\mathcal{MIN}$ Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
9. Dorigo, M., Gambardella, L.M.: Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
10. Stützle, T.: Iterated local search for the quadratic assignment problem. European Journal of Operational Research 174(3), 1519–1539 (2006)
11. Yuan, B., Gallagher, M.: Combining Meta-EAs and racing for difficult EA parameter tuning tasks. In: Lobo, F., Lima, C., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms, pp. 121–142. Springer, Berln (2007)
12. Smit, S., Eiben, A.: Comparing parameter tuning methods for evolutionary algorithms. In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, pp. 399–406. IEEE Press, Piscataway (2009)