Automatically Improving the Anytime Behaviour of Optimisation Algorithms

Manuel López-Ibáñez and Thomas Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2012-012

May 2012
Last revision: October 2013

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2012-012

Revision history:

TR/IRIDIA/2012-012.001	May 2012
TR/IRIDIA/2012-012.002	April 2013
TR/IRIDIA/2012-012.003	October 2013

Automatically Improving the Anytime Behaviour of Optimisation Algorithms (Extended Version)

Manuel López-Ibáñez[a,*], Thomas Stützle[a]

[a] *IRIDIA, Université Libre de Bruxelles (ULB), CP 194/6, Av. F. Roosevelt 50 – B-1050 Brussels, Belgium*

Abstract

Optimisation algorithms with good anytime behaviour try to return as high-quality solutions as possible independently of the computation time allowed. Designing algorithms with good anytime behaviour is a difficult task, because performance is often evaluated subjectively, by plotting the trade-off curve between computation time and solution quality. Yet, the trade-off curve may be modelled also as a set of mutually nondominated, bi-objective points. Using this model, we propose to combine an automatic configuration tool and the hypervolume measure, which assigns a single quality measure to a nondominated set. This allows us to improve the anytime behaviour of optimisation algorithms by means of automatically finding algorithmic configurations that produce the best nondominated sets. Moreover, the recently proposed weighted hypervolume measure is used here to incorporate the decision-maker's preferences into the automatic tuning procedure. We report on the improvements reached when applying the proposed method to two relevant scenarios: (*i*) the design of parameter variation strategies for MAX-MIN Ant System, and (*ii*) the tuning of the anytime behaviour of SCIP, an open-source mixed integer programming solver with more than 200 parameters.

Keywords: metaheuristics, anytime algorithms, automatic configuration, offline tuning

1. Introduction

Many optimisation algorithms are designed without a specific termination criterion, and generate a sequence of feasible solutions that are increasingly better approximations of the optimal solution. However, the performance of an algorithm is often crucially determined by the choice of the termination criterion and the parameters of the algorithm. If the parameter settings of an algorithm result in fast convergence to good solutions, this may prevent the algorithm from adequately exploring the search space to find better solutions if given ample time. On the other hand, parameter settings that give higher exploration capabilities may produce poor results if the termination criterion is too short. Hence, there is a trade-off between solution quality and the runtime of the algorithm that can be adjusted by appropriately setting the parameters of the algorithm.

[*] Corresponding author
Email addresses: `manuel.lopez-ibanez@ulb.ac.be` (Manuel López-Ibáñez), `stuetzle@ulb.ac.be` (Thomas Stützle)

In many practical scenarios, an optimisation algorithm may be terminated at an arbitrary time, and, upon termination, the algorithm returns the best solution found since the start of the run. In such scenarios, the termination criterion is not known in advance, and, hence, the algorithm should produce as high quality solutions as possible at any moment of its run time. Algorithms that show a better trade-off between solution quality and runtime are said to have a better *anytime behaviour* [66].

There are two classical views when analysing the anytime behaviour [38]. One view defines a number of termination criteria and analyses the quality achieved by the algorithm at each termination criterion. In this quality-over-time view, the anytime behaviour can be analysed as a series of plots of time-dependent solution quality distributions. A different view defines a number of target quality values and analyses the time required by the algorithm to reach each target. In this time-over-quality view, algorithms are often analysed in terms of a series of qualified runtime distributions.

In this paper, we consider a third view that does not favour time over quality or viceversa. Instead, this third view models the performance profile of an algorithm as a nondominated set in a multi-objective space. An algorithm has better anytime behaviour when it produces better nondominated sets, where "better" means better in terms of Pareto optimality. Surprisingly, this third view has received little attention [16, 20, 38], despite the important advances in theory and practice achieved in performance assessment of multi-objective optimisers in the last decade. Essentially, this model allows us to apply the same unary quality measures used in multi-objective optimisation to assign a single numerical value to the anytime behaviour of an algorithm's run. In this paper, we use the hypervolume measure as the unary quality measure for this purpose. The main reason is that the hypervolume is the quality measure with the highest discriminatory power among the known unary quality measures [68]. In addition, recent work has made possible to describe user preferences in terms of a weighted hypervolume measure [6], and, hence, our proposal allows incorporating user preferences when analysing the anytime behaviour of an algorithm. Moreover, as shown in this paper, evaluating the anytime behaviour of an algorithm in terms of the hypervolume allows applying automatic algorithm configuration methods to find parameter settings of an algorithm that optimise the trade-off between quality and time.

Recent advances in automatic configuration of algorithms (also called offline parameter tuning) have shown that such methods can save a significant amount of human effort and improve the performance of optimisation algorithms, when designing and evaluating new algorithms and when tuning existing algorithms to specific problems [9, 12, 24, 37, 39, 40]. Our proposal here is to combine automatic configuration with the use of the hypervolume as a surrogate measure of anytime behaviour in order to enable the automatic configuration of algorithms in terms of anytime behaviour.

In the scenario described above, where the algorithm does not know its termination criterion in advance, techniques such as parameter adaptation are often applied to improve the anytime behaviour of the algorithm [3, 25, 62]. However, designing such parameter adaptation strategies is an arduous task, and they usually add new parameters to the algorithm that need to be tuned. The method proposed in this paper will help algorithm designers to compare and fine-tune such parameter adaptation strategies to find the settings that improve the anytime behaviour of the algorithm on the problem at hand.

In fact, the first case study reported here derives from our own efforts on designing parameter adaptation strategies for ant colony optimisation algorithms. This experience motivated us to develop the method proposed here, since the classical trial-and-error approach for designing such strategies proved extremely time-consuming.

The second case study reported here deals with a different scenario, in particular, a general purpose black-box solver (SCIP [1]) with a large number of parameters. The default parameter settings of such solvers are tuned for solving problem instances to optimality as fast as possible. However, in some practical scenarios, users may not want to wait until a problem instance is solved to optimality, and may decide to stop the solver at an arbitrary time. Using our method for fine-tuning the parameters of the solver with respect to anytime behaviour, users can improve the quality of the solutions found when the solver is stopped before reaching optimality, without knowing in advance the particular termination criterion.

The outline of the paper is as follows. We provide some background on automatic algorithm configuration, summarise the state of the art and describe the automatic configuration method (`irace`) used throughout this paper in Section 2. Section 3 introduces the two classical views of the analysis of anytime algorithms and the less-explored multi-objective view. In Section 4, we describe our proposal in detail. We explain the benefits of using the hypervolume to evaluate the anytime behaviour of an algorithm in the context of an automatic configuration method. We discuss the choice of reference point and how to combine `irace` with the hypervolume measure. An additional section summarises related work and highlights the differences with our proposed approach. Section 5 describes our first case study, where we apply this proposal to the design of parameter adaptation strategies for MMAS. Section 6 discusses how our proposal enables a decision maker to incorporate preferences regarding the anytime behaviour of an algorithm to the automatic configuration procedure. A second case study is considered in Section 7, where we tune the anytime behaviour of SCIP. Finally, Section 8 provides a summary of our results and discusses possible extensions of the present work.

2. Preliminaries: Automatic algorithm configuration

This section is an introduction to automatic algorithm configuration. We formally define the algorithm configuration problem, give an overview on the state of the art of automatic configuration methods, and briefly describe `irace`, the automatic configuration method used throughout this paper.

2.1. The algorithm configuration problem

Most algorithms for computationally hard optimisation problems have a number of parameters that need to be set. As an example, ACO algorithms [**?**] often require the user to specify not only numerical parameters like the evaporation factor and the number of ants, but also components like the type of heuristic information and update method. Another example is mixed-integer programming solvers, such as SCIP [1], which often have a large number of configurable parameters affecting the main algorithm used internally, e.g., selecting among different branching strategies. The process of designing complex algorithms from a framework of algorithm components can be seen as an algorithm configuration problem [42, 48, 55].

Given a parametrised algorithm with n parameters, X_i, $i = 1, \ldots, n$, where each of them may take different values (settings). A configuration of the algorithm $\theta = \{x_1, \ldots, x_n\}$ is a unique assignment of values to parameters, and Θ denotes the possibly infinite set of all configurations of the algorithm. When considering a problem to be solved by this parametrised algorithm, the set of possible instances of the problem may be seen as a random variable $\mathcal{I}$ from which instances to be solved are sampled. We are also given a cost measure $\mathcal{C}(\theta, \pi)$

that assigns a value to each configuration θ when applied to a single problem instance π (a realisation of $\mathcal{I}$). In the case of stochastic algorithms, this cost measure is a random variable. In optimisation problems, the cost value is often the best solution quality found within a given computation time. In the case of decision problems, it may correspond to the computation time required to reach a decision, possibly bounded by a maximum cut-off time. In any case, the cost measure assigns a cost value to one run of a particular configuration on a particular instance.

The goal of algorithm configuration is to find the configuration θ^* such that:

$$\theta^* = \arg\min_{\theta \in \Theta} f_{\mathcal{C},\mathcal{I}}(\theta) \ , \tag{1}$$

where $f_{\mathcal{C},\mathcal{I}}(\theta)$ is a function of $\mathcal{C}_{\mathcal{I}}(\theta)$, that is, the (random) cost of configuration θ over the distribution of problem instances given by $\mathcal{I}$. A typical definition of $f_{\mathcal{C},\mathcal{I}}(\theta)$ is $E[\mathcal{C}_{\mathcal{I}}(\theta)]$, the expected cost of θ over the distribution of $\mathcal{I}$. The precise form of $f_{\mathcal{C},\mathcal{I}}(\theta)$ is generally unknown, and it can only be estimated by sampling. This sampling is performed in practice by obtaining realisations of the random variable $\mathcal{C}(\theta, \pi)$, that is, by evaluating algorithm configurations on instances sampled from $\mathcal{I}$.

2.2. Automatic configuration methods

The traditional approach to algorithm configuration consists of ad-hoc experiments testing relatively few configurations. The use of experimental design techniques [2, 18, 35, 36] began a trend in which the task of finding the most promising configurations to be tested is performed automatically. The natural evolution of this trend has been to tackle algorithm configuration as an optimisation problem, and, thus, many optimisation techniques have been proposed in the literature, including evolutionary algorithms [4, 56], local search [39] or model-based search [9, 10, 11, 40]. It is becoming widely accepted that automatic configuration methods may save substantial human effort during the empirical analysis and design of optimisation algorithms, and, at the same time, lead to better algorithms [9, 12, 24, 37].

2.3. Iterated racing (`irace`)

Racing was first proposed in machine learning to deal with the problem of model selection [53]. Birattari et al. [13] proposed a racing procedure, called F-Race, for the selection of the best among a given set of algorithm configurations, by means of the non-parametric Friedman's two-way analysis of variance by ranks, and its associated post-hoc test [17]. This proposal was later improved by sampling configurations from the parameter space, and refining the sampling distribution by means of repeated applications of F-Race. The resulting automatic configuration method was called Iterated F-race (I/F-Race) [8, 14]. The `irace` software [50] implements a general *iterated racing* procedure, which includes I/F-Race as a special case. There are some notable differences between `irace` and the original description of I/F-Race, such as the use of truncated normal distribution for sampling numerical parameters, a restart mechanism for avoiding premature convergence, and other features described in the `irace` documentation [50].

Iterated racing is a method for automatic configuration that consists of three steps: (1) sampling new configurations according to a probability distribution, (2) selecting the best configurations from the newly sampled ones by means of racing, and (3) updating the probability distribution in order to bias the sampling towards the best configurations. These three

```
Require: Training instances: $\{\pi_1, \pi_2, \dots\} \sim \mathcal{I}$,
     parameter space: $X$,
     cost measure: $\mathcal{C}(\theta, i)$,
     tuning budget: $B$
 1: $\Theta_1 :=$ SampleUniform$(X)$
 2: $\Theta^{\text{elite}} :=$ Race$(\Theta_1,\ B_1)$
 3: $j := 2$
 4: while $B_{\text{used}} \leq B$ do
 5:     $\Theta^{\text{new}} :=$ UpdateAndSample$(X, \Theta^{\text{elite}})$
 6:     $\Theta_j := \Theta^{\text{new}} \cup \Theta^{\text{elite}}$
 7:     $\Theta^{\text{elite}} :=$ Race$(\Theta_j,\ B_j)$
 8:     $j := j + 1$
 9: end while
10: Output: $\Theta^{\text{elite}}$
```

Table 1: Algorithm outline of iterated racing.

steps are repeated until a termination criterion is met, usually a predefined budget of runs of the algorithm being tuned.

An outline of the iterated racing algorithm is given in Algorithm 1. Iterated racing requires as input: a set of training instances $\{\pi_1, \pi_2, \dots\}$ sampled from $\mathcal{I}$, a parameter space (X), a cost function ($\mathcal{C}$), and a tuning budget (B).

In the first iteration, the initial set of candidate configurations is generated by uniformly sampling the parameter space X. At each iteration, the best configurations are selected by means of racing. When a race starts, each configuration is evaluated on the first instance by means of the cost measure $\mathcal{C}$. Configurations are iteratively evaluated on subsequent instances until a number of instances have been seen. Then, a statistical test is performed on the results. If there is enough statistical evidence to identify some candidate configurations as performing worse than at least another configuration, the worst configurations are removed from the race, while the others, the *surviving* candidates, are run on the next instance. There are several alternatives for selecting which configurations should be discarded during the race, and among them, F-Race is the default in `irace`. Racing continues until reaching a minimum number of surviving configurations, or consuming the computational budget assigned to the current iteration.

At the end of each race, the surviving configurations are used to update the sampling distributions associated to each configurable parameter. The update biases the probabilities such that, in future iterations, the parameter values in the best configurations previously found are more likely to be sampled again.

In the next iteration, a number of new candidate configurations are sampled, and a new race is launched with the union of the new configurations and the best configurations found so far. The iterated racing algorithm stops when the overall budget is exhausted.

3. Preliminaries: Anytime Algorithms

This section is an introduction to the analysis of stochastic optimisation algorithms and, in particular, anytime algorithms. We introduce the concept of runtime distributions (RTD), performance profiles, the two classical views of the analysis of RTDs, and the lesser studied multi-objective view.

3.1. Classical views of the analysis of runtime distributions

The performance of a stochastic optimisation algorithm is formally described by its runtime distribution (RTD)

Definition 1 (***Runtime distribution (RTD) [38]***)**.** Given an optimisation algorithm A for an optimisation problem Π and a problem instance $\pi \in \Pi$, let us assume, without loss of generality, a bivariate random variable $(\mathcal{T}, \mathcal{Q})$, where $\mathcal{T}, \mathcal{Q} \in \mathbb{R}_0$, $\mathcal{T}$ is computational effort measured, for example, as CPU-time in seconds, and $\mathcal{Q}$ is solution quality measured, for example, as relative percentage deviation from the optimal solution quality. The *runtime distribution (RTD)* is the probability distribution of $(\mathcal{T}, \mathcal{Q})$ characterised by the runtime distribution function $rtd\colon \mathbb{R}_0 \times \mathbb{R}_0 \to [0, 1]$, and defined as

$$rtd(t, q) = \Pr\{\mathcal{T} \leq t \wedge \mathcal{Q} \leq q\} \ ,$$

that is, $rtd(t, q)$ denotes the probability of finding a solution of quality less than or equal to q in time less than or equal to t.

Dean and Boddy [19] describe an anytime algorithm as one that, first, may be interrupted at any moment and return a solution and, second, it keeps steadily improving its solution until interrupted, eventually finding the optimal. Most metaheuristics and other optimisation algorithms satisfy this condition, and, hence, they are anytime optimisation algorithms. A concept of anytime behaviour more useful in the context of metaheuristics was introduced by Zilberstein [66], who highlights that algorithms with *good anytime behaviour* return as high-quality solutions as possible at any moment of their execution. A single run of an anytime algorithm generates a sequence of solutions that are increasingly better approximations of the optimal solution. Hence, in the context of anytime algorithms, an algorithm run is often described as a performance profile:

Definition 2 (***Performance profile [66]***)**.** Let us consider a single run r of an anytime optimisation algorithm A on a problem instance π, and record the time (t_i) and the solution quality (q_i) whenever a new best-so-far solution is found during the run of the algorithm. The set $P_r = \{(t_1, q_1), (t_2, q_2), \dots\}$ is called the *performance profile* of run r, where (t_i, q_i) are sampled with probability $rtd(t, q)$, and $t_i < t_j \wedge q_i > q_j$, $\forall i < j$.

Figure 1 gives an example of performance profiles for three runs A, B, and C. The above definition of performance profile does not favour quality over time, or viceversa. It would be equivalent to plot quality or time on either axis. Nonetheless, performance profile plots traditionally place time on the x-axis and quality on the y-axis, and we follow this custom here. A problem arises, however, when one wants to aggregate the information from several performance profiles in order to analyse the behaviour of an algorithm.

Analysing a bivariate distribution is remarkably more difficult than univariate distributions. Hence, in the literature, the RTD is usually transformed into a univariate distribution in one of two ways: either considering cross-sections of the RTD for fixed runtimes or for fixed quality targets. These two orthogonal transformations are often characterised as two views of the behaviour of an optimisation algorithm [38]. The solution-quality-over-time (SQT) view examines the solution quality distribution (SQD), which is the marginal probability of the

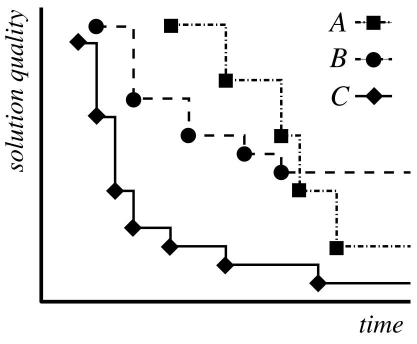


Figure 1: Performance profiles of three algorithms A, B, and C (or three independent runs of the same stochastic algorithm).

RTD over fixed run-times, e.g., plotting the development of the mean, the median or specific quantiles of the SQD over time. This is equivalent to aggregating performance profiles over fixed run-times. This SQT view is the most popular in combinatorial optimisation, where it is common to compare the anytime behaviour of optimisation algorithm by visually inspecting mean SQT curves over time [38, 52, 62, 63].

A second classical view is to aggregate performance profiles over fixed quality targets. Although plots of mean time over quality are possible [38], the most common approach is to examine the *qualified run time distribution* (QRTD), which is the marginal probability of the RTD over fixed quality targets. The analysis based on QRTDs is popular for benchmarking continuous black-box optimisers [34].

3.2. A multi-objective view of runtime distributions

A third alternative is to not favour either of the classical views, but instead use techniques from multi-objective optimisation for evaluating the bivariate RTD [16, 20]. Thus, we do not aggregate the performance profiles over fixed run-times or fixed quality-targets. Instead, we describe the performance profiles generated by an optimisation algorithm in terms of a random nondominated set called *random performance profile.* First, let us introduce some basic definitions borrowed from multi-objective optimisation, but adapted to the analysis of RTDs.

Definition 3 (***Weak dominance***)**.** Given two vectors (t,q), $(t',q') \in \mathbb{R}_0 \times \mathbb{R}_0$, we say that (t,q) *weakly dominates* (t',q'), and denote it by $(t,q) \leq (t',q')$ iff $t \leq t' \wedge q \leq q'$.

Definition 4 (***Nondominated set***)**.** A set $X = \{(t_1,q_1),(t_2,q_2),\dots\}$ is called *nondominated* iff $\nexists (t_i,q_i),(t_j,q_j) \in X$, $i \neq j$ such that $(t_i,q_i) \leq (t_j,q_j)$.

According to the definition of performance profile above (Def. 2), the elements of a performance profile, besides being realisations of the bivariate random variable $(\mathcal{T}, \mathcal{Q})$, must also be mutually nondominated. Therefore, we can analyse performance profiles as nondominated sets using the same techniques used in multi-objective optimisation.

The multi-objective view considers the bivariate RTD without aggregation, which is a well-known approach to the analysis of optimisation algorithms [38], but much less studied due to the inherent difficulty of analysing a bivariate distribution. Chiarandini [16] considered such a view for analysing the anytime behaviour of several metaheuristics for graph colouring, concretely using the attainment function [32]. First, let us introduce a special case of the definition of random nondominated point set (RNP-set) [32]:

Definition 5 (***Random performance profile***)**.** A random performance profile is a random set defined as

$$\mathcal{P} = \{(t_1, q_1), (t_2, q_2), \ldots, (t_N, q_N) \in \mathbb{R}_0 \times \mathbb{R}_0 \colon \Pr\{(t_i, q_i) \leq (t_j, q_j)\} = 0, i \neq j\}$$

where the number of elements $N \in \mathbb{N}_0$ is random, and the elements (t_i, q_i) are realisations of the bivariate random variable $(\mathcal{T}, \mathcal{Q})$, and no element in the set weakly dominates any other element.

How to best analyse the distribution of RNP-sets is an ongoing research effort in multi-objective optimisation [26, 28, 32, 33, 49]. The best-known proposal is to characterise the distribution of an RNP-set in terms of the attainment function:

Definition 6 (***Attainment function [33]***)**.** Given a RNP set $\mathcal{P}$, the attainment function $\alpha_{\mathcal{P}} \colon \mathbb{R}_0 \times \mathbb{R}_0 \to [0, 1]$ is defined as

$$\alpha_{\mathcal{P}}(t, q) = \Pr\{(t', q') \in \mathcal{P} : (t', q') \leq (t, q)\} \ ,$$

that is, the probability that an element of a realisation of $\mathcal{P}$ weakly dominates an arbitrary point in $\mathbb{R}_0 \times \mathbb{R}_0$.

In the case of a single performance profile P, the attainment function is a binary function $\alpha_P(t, q) \colon \mathbb{R}_0 \times \mathbb{R}_0 \to \{0, 1\}$ defined as

$$\alpha_P(t, q) = \mathbf{I}\{\exists (t', q') \in P : (t', q') \leq (t, q)\} \ ,$$

where $\mathbf{I}\{\cdot\}$ is the indicator function.

There has been substantial theoretical work on the attainment function and its properties [32]. For example, the attainment function is a generalisation of the multivariate cumulative distribution function [31]. Practical applications include statistical analysis and graphical explorations methods [43, 44, 49]. Although these methods can be used to analyse anytime algorithms [16], they require substantial human interaction, and, thus, they are difficult to use as the basis of an automatic configuration procedure.

4. Our proposal: Automatic configuration of anytime algorithms by means of the hypervolume measure

As mentioned above, directly using the attainment function as the basis of an automatic configuration tool for anytime algorithms seems difficult. Instead, we identify the hypervolume measure as the best available choice for this task. The main reasons are its high discriminatory power, being Pareto-compliant, and the possibility of incorporating user preferences into the automatic configuration process.

First, we explain the hypervolume measure and its properties. Second, we discuss how an automatic configuration method (in particular, `irace`) should use the hypervolume measure to improve the anytime behaviour of an algorithm. Third, we explain how user preferences can be incorporated into the automatic configuration process by means of the weighted hypervolume.

4.1. Hypervolume measure of performance profiles

As a first step, let us define the classical Pareto-dominance relation on performance profiles:

Definition 7 (***Better in terms of Pareto-optimality,*** $\lhd$)**.** Given two performance profiles P_i and P_j, which are the result of running two algorithms A_i and A_j (or two times the same stochastic algorithm) on the same problem instance π, we say that P_i is *better*, in terms of Pareto-optimality, than P_j ($P_i \lhd P_j$) iff $P_i \neq P_j$, and $\forall (t_j, q_j) \in P_j$, $\exists (t_i, q_i) \in P_i$, such that $(t_i, q_i) \leq (t_j, q_j)$.

It is often the case, however, that neither performance profile is better than the other, i.e., they are *incomparable*. These relations are independent of the classical views of fixed-quality (first view) or fixed-runtimes (second view) as described above, and only make sense in the third view that considers both quality and runtime in terms of Pareto-optimality.

In the following, we assume that, without any a priori information about the actual termination criterion of the algorithm or the preferred trade-off between solution-quality and computation time, a performance profile that is better than another in terms of Pareto-optimality is also better in terms of anytime behaviour, in the sense that the former is always preferred to the latter.

In order to apply an automatic configuration tool from the literature for improving the performance profiles produced by an algorithm, we would desire a unary quality measure that unequivocally indicates whether a performance profile is better than another. Unfortunately, a well-known result from multi-objective optimisation states that no unary quality measure (or finite combination thereof) can indicate whether a performance profile is better, as defined above, than another [68]. The most powerful of the unary quality indicators can at most indicate that a performance profile is not worse than (better than or incomparable to) another. The hypervolume measure [67] is the only unary quality measure known to have such discriminatory power [68].

In the context of performance profiles, the hypervolume measures the area enclosed between the performance profile and bounded above by a reference point. As shown by Zitzler et al. [69], the hypervolume quality measure can also be defined as the volume enclosed by the attainment function and the axes [67, 69]:

Definition 8 (***Hypervolume measure***)**.** The hypervolume measure of a performance profile P with reference point (t_r, q_r) is computed as

$$H(P) = \int_0^{t_r} \int_0^{q_r} \alpha_P(t, q)\, dq\, dt$$

In this paper, we only consider single-objective optimisation algorithms. Nonetheless, it is trivial to extend the above discussion to multi-objective algorithms, where there is more than one measure of solution quality. In fact, we have applied the method proposed here to automatically improve the anytime behaviour of multi-objective evolutionary algorithms [57].

4.1.1. The choice or reference point

In the context of anytime algorithms, the choice of reference point is application specific. The bivariate runtime distribution is defined without any limits on how bad the solution quality might be or how long it may take to generate any solution. In practice, however, large deviations from the optimal quality may be of little interest (no matter how fast they can be generated) and algorithms need to be stopped at some point (cut-off time). These limitations are not specific to any of the three views discussed above. A default approach is to consider a cut-off quality that corresponds to the worst solution quality found by any run of the algorithms under analysis, and a cut-off time that is slightly larger than the maximum time that would be reasonable for a single run of the algorithm. The reference point would then be defined as a factor larger than the cut-off quality and cut-off time.

How much larger this factor should be is an open question in multi-objective optimisation. There are some theoretical results on how the choice of reference point affects the distribution of elements within a nondominated set that maximises the hypervolume [7]. However, it is not clear how these results extend to the relations between nondominated sets. In any case, the only effect of the reference point is to bias the preference between incomparable performance profiles. In that sense, our suggestion (and the usual practice in multi-objective optimisation) is to define the reference point in a consistent manner and bias this preference by other means, such as the weighted hypervolume (Sec. 6).

4.2. Automatic configuration of anytime algorithms

The use of the hypervolume to compare performance profiles in terms of Pareto-optimality has the additional benefit of providing a unary scalar measure to evaluate anytime behaviour. Integrating such a measure in most automatic configuration methods should be straightforward. Here, we discuss the practical aspects of the integration of the hypervolume in `irace` for improving the anytime behaviour of single-objective optimisation algorithms.

Our procedure requires to specify a maximum cut-off time for the algorithm being tuned by `irace`. As discussed above, this cut-off time is necessary because anytime algorithms may in principle run forever. The cut-off time could be dynamic or different for each run, however, for the sake of simplicity, we do not explore these possibilities here. Each run of the algorithm must produce a performance profile as defined in Def. 2.

Within a single race in `irace`, a set of algorithm configurations are evaluated on a sequence of training instances. After evaluating an instance, some configurations may be discarded. Let Θ_π denote the configurations that have *not* been discarded *before* evaluating instance π. Let $P_{\theta,\pi}$ denote the performance profile generated by running configuration θ on instance π. First, we normalise the performance profiles $P_{\theta,\pi}$, for each $\theta \in \Theta_\pi$ to the range $[0.0, 0.9]$. Thus, after normalisation we obtain for each θ a new performance profile:

$$P'_{\theta,\pi} = \{(t'_i, q'_i) \mid \forall (t_i, q_i) \in P_{\theta,\pi}\}$$

where

$$\begin{cases} t'_i = 0.9 \cdot (t_i - t_{\min})/(t_{\max} - t_{\min}) \\ q'_i = 0.9 \cdot (q_i - q_{\min})/(q_{\max} - q_{\min}) \end{cases}$$

where $t_{\min}$ is usually zero, $t_{\max}$ is the cut-off time, $q_{\min} = \min\{q_i \mid \forall (t_i, q_i) \in P_{\theta,\pi}, \forall \theta \in \Theta_\pi\}$ and $q_{\max}$ is defined similarly for the maximum solution quality found after running all configurations in Θ_π on instance π.

Finally, we compute the hypervolume $hv(\theta, \pi)$ of each $P'_{\theta,\pi}$ using $(1.0, 1.0)$ as the reference point. In our proposal, this $hv(\theta, \pi)$ value becomes the cost measure $\mathcal{C}(\theta, \pi)$ used by `irace` (Sec. 2.3 on page 4).

Neither quality values, nor hypervolume values from different instances are directly compared because normalisation is done within each instance π, as defined above, and the application of the F-test within `irace` transforms the hypervolume values into ranks per instance. Hence, quality values or hypervolume values may have different ranges on each instance without introducing a bias. This approach also allows for instance-dependent cut-off times, although we do not explore this possibility in this paper.

4.3. Related Work on Automatic Configuration of Anytime Algorithms

There is substantial work on automatic configuration for decision problems and single-objective optimisation problems. We refer to recent overviews [24, 37] and books [9, 12] for a complete bibliography. By comparison, there are relatively few works on tuning multi-objective optimisation algorithms. Wessing et al. [64] automatically tuned the variation operator of a multi-objective evolutionary algorithm applied to a single problem instance. Simultaneously, López-Ibáñez and Stützle [46, 48] automatically instantiated new designs of multi-objective ant colony optimisation (MOACO) algorithms for the bi-objective travelling salesman problem from a framework of MOACO algorithmic components. More recently, Dubois-Lacoste et al. [23] applied this latter approach to outperform the state of the art in several bi-objective permutation flow-shop problems. These works share with our proposal the use of unary quality measures, such as the hypervolume, as the cost function $\mathcal{C}(\theta, i)$ used by the automatic configuration method.

On the other hand, our proposal should not be confused with parameter tuning as a multi-objective problem [22], where the aim is to produce a set of parameter configurations that are mutually nondominated with respect to multiple criteria. In this paper, our aim is to produce a single parameter configuration that generates an anytime behaviour that is as good as possible.

There have been some recent attempts at tackling the problem of tuning anytime algorithms. As mentioned above, Chiarandini [16] used the attainment function to analyse the anytime behaviour of several metaheuristics for graph colouring. However, it is far from obvious how to effectively use the attainment function in an automatic configuration method. The proposal closest to ours is by den Besten [20], who combined racing and a performance measure based on the *binary* ϵ-indicator. The use of a binary measure involves computing a matrix of ϵ-measure values, comparing each alternative with the rest, and transforming it into ranks. More recently, Branke and Elomari [15] combined a meta-level evolutionary algorithm and an ad-hoc ranking procedure for tuning the mutation rate of a lower-level algorithm for multiple termination criteria in a single tuner run. Their ranking method is not based on any multi-objective quality measure. Instead, it ranks each configuration with respect to the number of discrete time steps in which the configuration was better than other configurations. In that sense, it is an example of the classical fixed-runtimes view (what we call first view above).

5. Case Study: Design of parameter variation strategies for MAX-MIN Ant System on the TSP

Many anytime algorithms use parameter adaptation strategies [3, 25], that is, the variation of parameter settings while solving a problem instance, to adapt the parameters to different phases of the search, and to balance exploration of the search space and exploitation of the best solutions found.

Designing and comparing parameter adaptation strategies is, however, an arduous and complex task. Traditionally, the analysis is performed in terms of one (or both) classical views, that is, either measuring solution quality over fixed-runtimes [3], or runtime (CPU-time or function evaluations) over fixed quality-targets [5].

In previous work [54, 62], we studied parameter adaptation strategies for ant colony optimisation (ACO) algorithms using the classical solution quality over fixed-runtimes view. In particular, we studied the anytime behaviour of MAX-MIN Ant System (MMAS) on the travelling salesman problem (TSP) by experimenting with various static parameter settings and parameter variation strategies. Our analysis relied on visually comparing the mean SQT (solution-quality-over-time) curves of various strategies that were deemed interesting. Needless to say, this was a human intensive task that required many iterations of experimentation and analysis. We roughly estimate that the overall effort for obtaining the best configurations was close to one person-year. The result of this effort is that the best configurations are significantly better in terms of anytime behaviour than the default settings of MMAS [54, 62].

The effort of designing and comparing these parameter adaptation strategies could have been significantly reduced by using an automatic algorithm configuration tool for improving the anytime behaviour. This case study was, in fact, our main motivation for developing the method proposed in this paper. In this section, we describe the case study in detail, we examine the setup required for applying automatic algorithm configuration and, finally, we compare the parameter adaptation strategies identified as the best by the automatic configuration procedure versus the ones identified in our previous work.

5.1. MAX-MIN Ant System

MMAS is an ACO algorithm that incorporates an aggressive pheromone update procedure and mechanisms to avoid search stagnation. When applying MMAS to the TSP, each ant starts at a randomly chosen initial city, and constructs a tour by randomly choosing at each step the city to visit next according to a probability defined by pheromone trails and heuristic information. In particular, the probability that ant k chooses a successor city j when being at city i is given by

$$p_{ij} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{h \in N^k} [\tau_{ih}]^{\alpha} \cdot [\eta_{ih}]^{\beta}} & \text{if } j \in N^k \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where τ_{ij} is the pheromone trail strength associated to edge (i, j), η_{ij} is the corresponding heuristic information; α and β are two parameters that influence the weight given to pheromone and heuristic information, respectively; N^k is the feasible neighbourhood, that is, a candidate list of cities not yet visited in the partial tour of ant k.

Following previous work [58, 60], we also incorporate the *pseudo-random action choice rule* of ACS [21], which allows for a greedier solution construction. With a probability q_0, an ant chooses next a city $j \in N^k$ such that

$$j = \arg\max_{h \in N^k} \left\{ [\tau_{ih}]^{\alpha} \cdot [\eta_{ih}]^{\beta} \right\}; \tag{3}$$

otherwise, the ant performs the probabilistic selection based on Eq. 2. A value of $q_0 = 0$ reverts back to the original MMAS.

Table 2: Default settings of the parameters under study for MMAS with 2-opt local search.

Algorithm	Time_{CPU}	α	β	ρ	m	q_0
MMAS	500 s	1.0	2.0	0.2	25	0.0

The pheromone update of MMAS updates all pheromone trails as

$$\tau_{ij} \leftarrow \max\bigl\{\tau_{\min}, \min\{\tau_{\max}, (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{\text{best}}\}\bigr\}, \tag{4}$$

where ρ, $0 < \rho \leq 1$, is a parameter called evaporation rate and

$$\Delta\tau_{ij}^{\text{best}} = \begin{cases} 1/f(s^{\text{best}}) & \text{if edge } (i,j) \in s^{\text{best}}, \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where $f(s)$ is the tour length of solution s, and s^{best} is either the iteration-best solution, the best-so-far solution or the best solution since a re-initialisation of the pheromone trails (restart-best). In MMAS, these solutions are chosen alternately [61].

Finally, solutions constructed by the ants may be further improved by the application of a local search algorithm. In this paper, we will use MMAS with 2-opt local search, as was done in previous work [54, 62].

5.2. Parameter variation strategies in MMAS

Following our previous work [62], we focus on two basic schemes for parameter variation in MMAS, which we call henceforth delta and switch strategies. During a single run of MMAS, the variation strategy called delta applied, for example, to parameter β increases the value of β at each iteration of the algorithm by a certain amount $\Delta\beta$, starting from the value β_{start} and stopping at the value β_{end}. If $\beta_{\text{start}} > \beta_{\text{end}}$, then the value of β is *decreased* at each iteration by $\Delta\beta$ instead of increased. Conversely, the variation strategy called switch changes, at iteration β_{switch}, the value of parameter β from the value β_{start} to the value β_{end}. An additional parameter β_{var} controls the variation strategy, which is either delta, switch or none, where none means that the parameter value of β stays constant throughout the run of the algorithm. As a result, we add to MMAS five additional parameters for varying β: β_{start}, β_{end}, $\Delta\beta$, β_{switch} and β_{var}.

We apply the same parameter variation strategies to parameters β, ρ, the number of ants (m), and q_0. Hence, we add five additional static parameters for each parameter that is dynamically varied. Table 3 describes the domains of all parameters, and Table 2 describes the default values [61].

5.3. Automatic configuration of parameter adaptation strategies

We consider random uniformly generated instances of the symmetric TSP with 3 000 cities [41]. Our instances are available in the supplementary material page [47]. We generate 50 training (tuning) instances and 50 test instances.

We apply our proposed method for the automatic configuration of the anytime behaviour (Sec. 4.2). The automatic configuration tool is the implementation of I/F-Race provided by the `irace` software package [50]. As explained above, we incorporate the hypervolume

Table 3: Parameter space for variation strategies of MMAS.

Parameter	**Domain**	**Constraint**
m_{var}, β_{var}, ρ_{var}, $q_{0\text{var}}$	{ delta, switch, none }	
m	[1, 100]	if var = *none*
β	[0, 20]	
ρ	[0.01, 1.0]	
q_0	[0.0, 1.0]	
Δm	$\{0.01, 0.05, 0.1, 0.25, 0.5, 1, 2, 5\}$	if var = *delta*
$\Delta\beta$	$\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0\}$	
$\Delta\rho$	$\{0.001, 0.002, 0.005, 0.01\}$	
Δq_0	$\{0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.005\}$	
m_{switch}, β_{switch}, ρ_{switch}, $q_{0\text{switch}}$	[1, 500]	if var = *switch*
m_{start}	1	if var $\in \{delta, switch\}$
m_{end}	[1, 500]	
β_{start}	[0, 20]	
β_{end}	[0, 5]	
ρ_{start}	[0.001, 1.0]	
ρ_{end}	[0.001, 1.0]	
$q_{0\text{start}}$	[0.0, 1.0]	
$q_{0\text{end}}$	[0.0, 1.0]	

measure to `irace` in order to evaluate the anytime behaviour of a single run of MMAS. We use a publicly available implementation of the hypervolume measure [27], and the MMAS implementation is based on the ACOTSP software [59]. All experiments are run on Intel Xeon E5410 CPUs (2.33 GHz, 2×6MB L2 cache) running Cluster Rocks Linux version 6/CentOS 6.3, 64bits. Each individual run of the algorithms being tuned uses just one core.

In a first step, we tune separately the variation strategy of one dynamic parameter at a time, while other parameters are fixed to their default values as given in Table 2. That is, we perform one run of `irace` for each parameter $\{m, \beta, q_0, \rho\}$. For example, in the run of `irace` that tunes the variation strategy of β, the parameters tuned are β_{var}, β, $\Delta\beta$, β_{switch}, β_{start}, β_{end}, whereas the other parameters (m, q_0, ρ) are fixed to their default values (Table 2) and their variation strategies ($param_{\text{var}}$) are set to `none`, and, hence, their corresponding variation parameters ($\Delta param$, $param_{\text{switch}}$, $param_{\text{start}}$, $param_{\text{end}}$) are not considered (see Table 3). We give each run of `irace` a budget of 1 000 runs of MMAS. Each run of MMAS is stopped after Time_{CPU} seconds (Table 2).

In a second step, we automatically configure all parameter variation strategies at the same time, that is, we configure 24 parameters instead of six. Since the parameter space is much larger now, we assign a larger tuning budget to this run of `irace`, specifically 10 000 runs of MMAS.

After each run of `irace` finishes, we apply the resulting parameter configurations (Table 5) to the test instances. In addition, we also run, on the test instances, the *default* parameter configuration of MMAS (Table 2) without any variation strategy and several variation strategies previously found by manual ad-hoc experimentation (Table 4) [62]. We present these results in the following sections.

5.4. Analysis of the results

5.4.1. Automatic configuration vs. manual configuration

Our goal is to improve the anytime behaviour of MMAS over the whole set of test instances. Hence, we analyse the overall results by plotting the average solution quality over time (SQT) for all test instances at once. However, as explained above, our proposed approach *does not rely* on these SQT curves for improving the anytime behaviour, and it does not favour solution-quality over time or viceversa. We could also visualise our results in terms of qualified runtime distributions [34, 38], or in terms of attainment surfaces [16, 32]. We chose SQT curves as one of the traditional means of visualising the anytime behaviour of an algorithm, and the most popular view in the literature on combinatorial optimisation algorithm [38].

For each algorithm configuration, we have the best solution quality found f_{rit} on run r on instance i at time t. We compute the relative percentage deviation (RPD) from the optimal solution for each instance as $RPD_{rit} = 100 \cdot f_{rit}/f_i^{\text{opt}}$, where f_i^{opt} is the optimal tour length of instance i. Then, we compute the mean RPD over all 50 instances and over all 15 independent runs of each algorithm as $RPD_t = \frac{1}{50 \cdot 15} \cdot \sum_{i=1}^{50} \sum_{r=1}^{15} RPD_{rit}$. Each line in the plots in Fig. 2 corresponds to the RPD_t of one algorithm configuration, that is, aggregating solution-quality over time.

In the case of the hypervolume computation, we do not aggregate over time, but compute the hypervolume of the (nondominated) performance profile of each run on each instance by normalising both time and solution quality to the interval $[0.0, 0.9]$ and using $(1.0, 1.0)$ as the reference point. Then, for each algorithm configuration we compute its mean hypervolume over all its runs on all test instances, and we give this value in the legend of each plot.

Table 4: Parameter configurations found by a human expert when varying one parameter at a time in MMAS [62].

Configuration	**Parameter settings**
manual var ants	$m_{\text{var}} = \textit{delta}$, $m_{\text{start}} = 1$, $m_{\text{end}} = 25$, $\Delta m = 0.1$
manual var beta	$\beta_{\text{var}} = \textit{switch}$, $\beta_{\text{start}} = 20$, $\beta_{\text{end}} = 3$, $\beta_{\text{switch}} = 50$
manual var rho	$\rho_{\text{var}} = \textit{none}$, $\rho = 0.9$
manual var q0	$q_{0\text{var}} = \textit{delta}$, $q_{0\text{start}} = 0.99$, $q_{0\text{end}} = 0$, $\Delta q_0 = 0.0005$

Table 5: Parameter configurations found by `irace` for MMAS: (auto var *param*) was obtained by tuning, with respect to anytime behaviour, the variation strategy of parameter *param*, while the other parameters are set to their default settings; (auto var ALL) was obtained by tuning, with respect to anytime behaviour, all variation parameters at the same time; (auto fix final) was obtained by tuning the classical MMAS parameters, without any variation, with respect to final quality, that is, the solution quality obtained at 500 seconds; and (auto var final) was obtained by tuning all variation parameters with respect to final quality.

Configuration	**Parameter settings**
auto var ants	$m_{\text{var}} = \textit{delta}$, $m_{\text{delta}} = 0.05$, $m_{\text{start}} = 1$, $m_{\text{end}} = 417$
auto var beta	$\beta_{\text{var}} = \textit{delta}$, $\beta_{\text{delta}} = 0.05$, $\beta_{\text{start}} = 9$, $\beta_{\text{end}} = 4$
auto var q0	$q_{0\text{var}} = \textit{switch}$, $q_{0\text{switch}} = 200$, $q_{0\text{start}} = 0.96$, $q_{0\text{end}} = 0.30$
auto var rho	$\rho_{\text{var}} = \textit{delta}$, $\rho_{\text{delta}} = 0.001$, $\rho_{\text{start}} = 0.82$, $\rho_{\text{end}} = 0.84$
auto var ALL	$m_{\text{var}} = \textit{delta}$, $m_{\text{delta}} = 1$, $m_{\text{start}} = 1$, $m_{\text{end}} = 384$, $\beta_{\text{var}} = \textit{switch}$, $\beta_{\text{switch}} = 79$, $\beta_{\text{start}} = 5$, $\beta_{\text{end}} = 0$, $q_{0\text{var}} = \textit{delta}$, $q_{0\text{delta}} = 0.002$, $q_{0\text{start}} = 0.87$, $q_{0\text{end}} = 0.57$, $\rho_{\text{var}} = \textit{none}$, $\rho = 0.68$
auto fix final	$\beta = 5.9$, $\rho = 0.62$, $m = 84$, $q_0 = 0.099$
auto var final	$m_{\text{var}} = \textit{switch}$, $m_{\text{switch}} = 50$, $m_{\text{start}} = 1$, $m_{\text{end}} = 317$, $\beta_{\text{var}} = \textit{delta}$, $\beta_{\text{delta}} = 0.5$, $\beta_{\text{start}} = 8$, $\beta_{\text{end}} = 2$, $q_{0\text{var}} = \textit{switch}$, $q_{0\text{switch}} = 139$, $q_{0\text{start}} = 0.6241$, $q_{0\text{end}} = 0.2725$, $\rho_{\text{var}} = \textit{switch}$, $\rho_{\text{switch}} = 493$, $\rho_{\text{start}} = 0.338$, $\rho_{\text{end}} = 0.7495$

The first important observation is how the value of the hypervolume matches the quality of the anytime behaviour of the different configurations: A larger hypervolume value indicates a better anytime behaviour. The first four plots (a,b,c,d) in Fig. 2 show a large improvement in the anytime behaviour of the manually tuned configurations with respect to the default configuration of MMAS. Nonetheless, the automatically found configurations are able to match, and in most cases surpass the manually tuned configurations in terms of hypervolume, despite the fact that the manually tuned configurations were found by extensive experimentation under the guidance of human expertise.

Fig. 3 compares the configuration obtained after automatically configuring all parameter variation strategies at once versus the best configurations obtained after automatically configuring the variation strategy of one parameter at a time. In our previous study, the manual tuning and analysis of all parameter strategies at once was ruled out as infeasible, given the extremely large number of potential configurations and interactions among different parameters. Here, we see that automatically configuring all parameters at once leads to an additional improvement in anytime behaviour.

Figures 2(a, b, c, d) and 3 show the mean hypervolume over all runs and all test instances. To assess whether the observed differences are statistically significant, we perform a statistical analysis of the results over the whole set of test instances. The analysis is based on the Friedman test for analysing non-parametric unreplicated complete block designs, and its associated post-test for multiple comparisons [17]. First, we calculate the mean hypervolume of the 15 runs of each algorithm for each instance. Then, we perform a Friedman test using the instances as the blocking factor, and the different configurations of MMAS as the treatment factor. The null hypothesis is that the configurations have identical effect on the ranking according to the hypervolume within each instance. If the Friedman test rejects the null hypothesis given a significance level of $\alpha = 0.05$, we proceed to calculate the minimum difference between the sum of ranks of two configurations that is statistically significant (ΔR_α). In this manner, we identify which configurations are significantly different from the best ranked one, i.e., the one with the lowest sum of ranks.

Table 6 summarises the results of the statistical analysis. It shows the value of ΔR_α for $\alpha = 0.05$, the different configurations of MMAS sorted by increasing sum of ranks, and the difference between the sum of ranks of each configuration and the best configuration (ΔR).

For each parameter considered, the ranking shown in Table 6 always ranks higher the configurations found automatically than their counterparts found by ad-hoc experimentation (auto vs. manual, respectively). More importantly, it shows that the best ranked configuration is the one that automatically configured all parameters at once, and that the difference in ranks between this configuration and the rest is statistically significant.

5.4.2. Hypervolume vs. final quality

Here, we show that the use of the hypervolume as the tuning criterion is the key factor for improving the anytime behaviour. Fig. 4 shows four configurations of MMAS: the default configuration (default); the one resulting from automatically tuning all variation parameters (auto var ALL); a configuration obtained by tuning all variation parameters with respect to final quality, that is, the solution quality obtained at 500 seconds (auto var final); and a configuration obtained by tuning the classical MMAS parameters, without any variation, with respect to final quality (auto fix final). The plot clearly shows that, independently of whether MMAS uses parameter variation or not, the results not tuned with respect to the

Table 6: Various configurations of MMAS ordered according to the sum of ranks with respect to the hypervolume obtained over all test instances. The numbers in parenthesis are the differences of ranks relative to the best ranked configuration. ΔR_α is the statistically significant difference in ranks according to the post-hoc test for the Friedman-test with $\alpha = 0.05$. All configurations are statistically significantly worse than the best one (auto var ALL). For the meaning of the labels, see the caption of Table 5.

ΔR_α	**Configurations** (ΔR)
9.87	**auto var ALL (0)**, auto var q0 (34), manual q0 (94), auto var rho (125), manual rho (189), auto var ants (236), auto var beta (294), manual ants (345), manual beta (380), default (433)

Table 7: Configurations of MMAS ordered according to the sum of ranks with respect to the final solution quality obtained after 500 seconds. The numbers in parenthesis are the difference of ranks relative to the best configuration. ΔR_α is the statistically significant difference in ranks according to the post-hoc test for the Friedman-test with $\alpha = 0.05$. Configurations that are not significantly different from the best one are indicated in bold face. For the meaning of the labels, see the caption of Table 5.

ΔR_α	**Configurations** (ΔR)
13.68	**auto var final (0)**, auto var ALL (48), auto fix final (52), default (132)

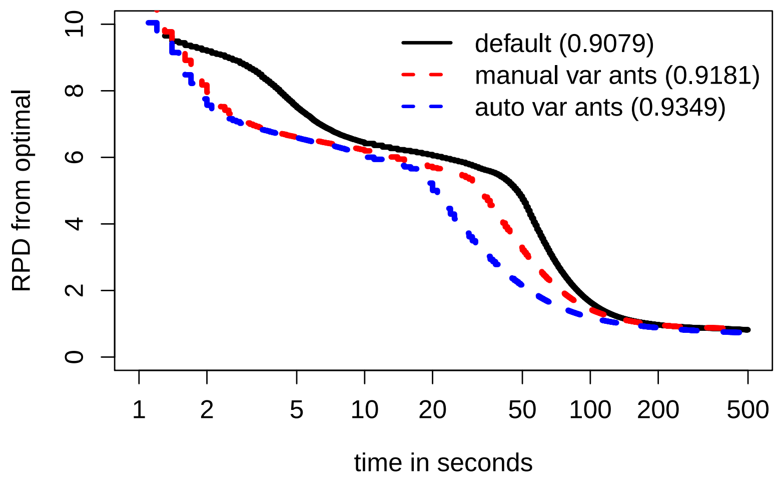
hypervolume have worse anytime behaviour.

A possible concern of tuning for anytime behaviour is a significant loss of final quality. Hence, we examine the final quality achieved by these four variants of MMAS in Fig. 5. According to the boxplots, there is an important improvement in the final quality achieved in comparison with the default configuration of MMAS, even for the configuration tuned for anytime behaviour. The boxplot does not show a large difference between the three automatically configured variants. Nonetheless, the Friedman test indicates that the final quality obtained by the variant tuned for anytime behaviour is statistically worse than the variants tuned for final quality (Table 7). In order to assess the loss of final quality, we compute the 95% confidence interval on the mean difference in final quality between the configuration tuned for anytime behaviour and the best ranked configuration, which is $[0.0244, 0.0480]$, measured in RPD.[1] Although the loss in final quality when tuning for anytime behaviour is small in this case, an anytime algorithm should aim to match the best possible final quality in the ideal case.

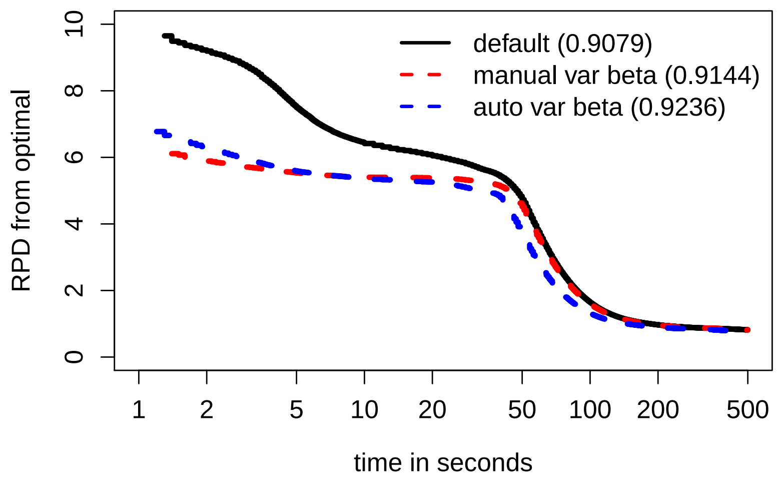
6. Articulation of preferences in automatic configuration of anytime algorithms

The use of the hypervolume for automatic tuning of anytime algorithms has an additional advantage compared to other unary measures, that is, the possibility of specifying the decision-maker's preferences. A recent proposal extends the hypervolume indicator by a weight func-

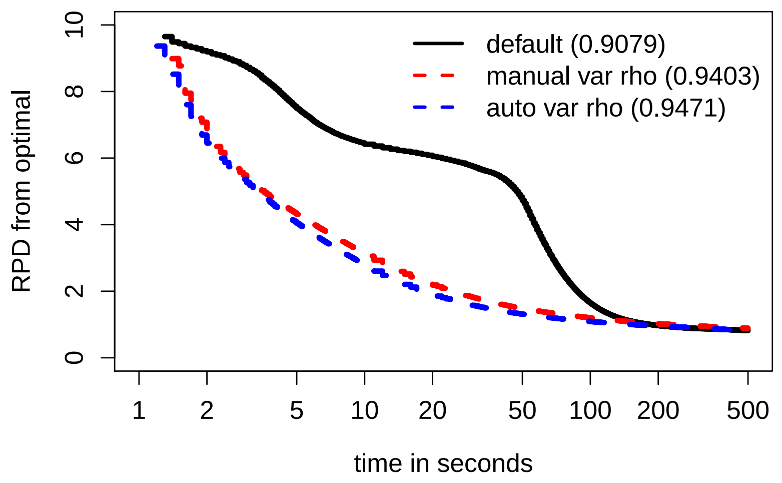
[1]The confidence interval is computed using the Welch's t statistic for two paired samples, which assumes that the samples follow a normal distribution. Nonetheless, for large sample sizes, as used here, the method is robust against deviations from normality.



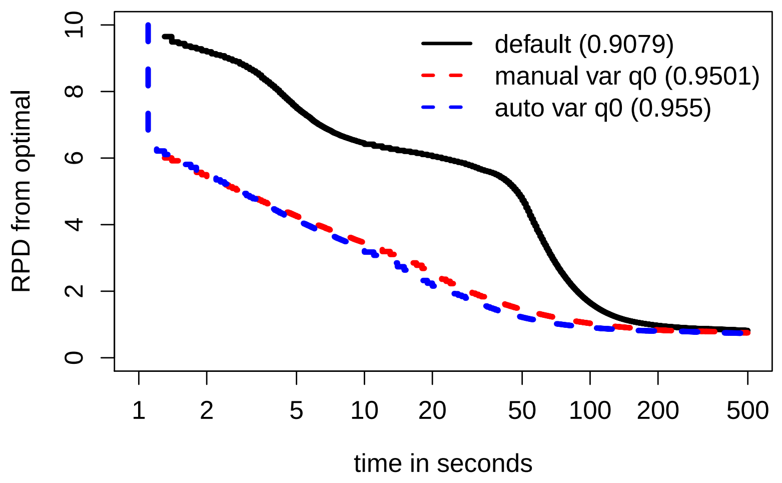
(a) Parameter m (ants)



(b) Parameter β (beta)



(c) Parameter ρ (rho)



(d) Parameter q_0

Figure 2: Anytime behaviour of manually tuned vs. automatically tuned configurations of MMAS. The number in the legend is the mean hypervolume of each configuration over all runs. For the meaning of the labels, see the caption of Table 5.

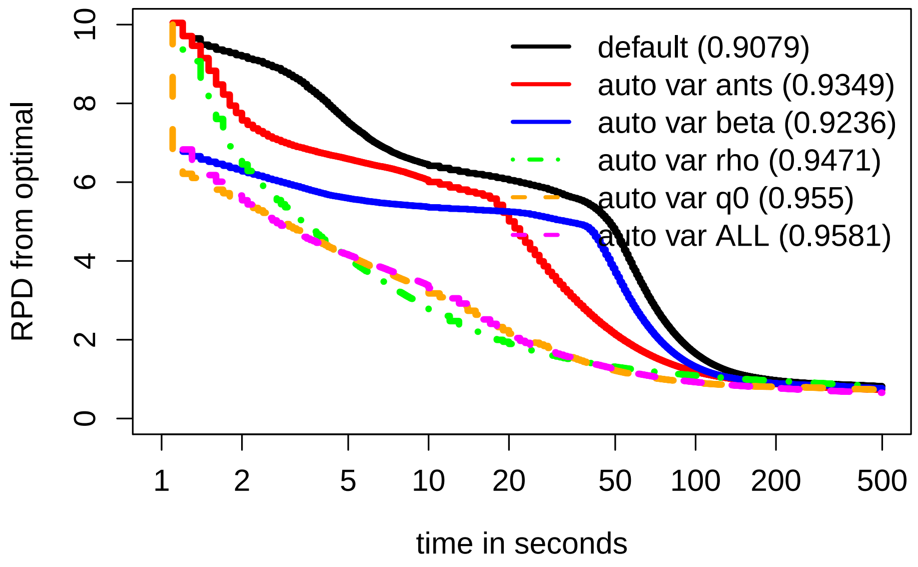


Figure 3: Anytime behaviour of automatically tuned configurations of MMAS vs. the default configuration. The number in the legend is the mean hypervolume of each configuration over all runs. For the meaning of the labels, see the caption of Table 5.

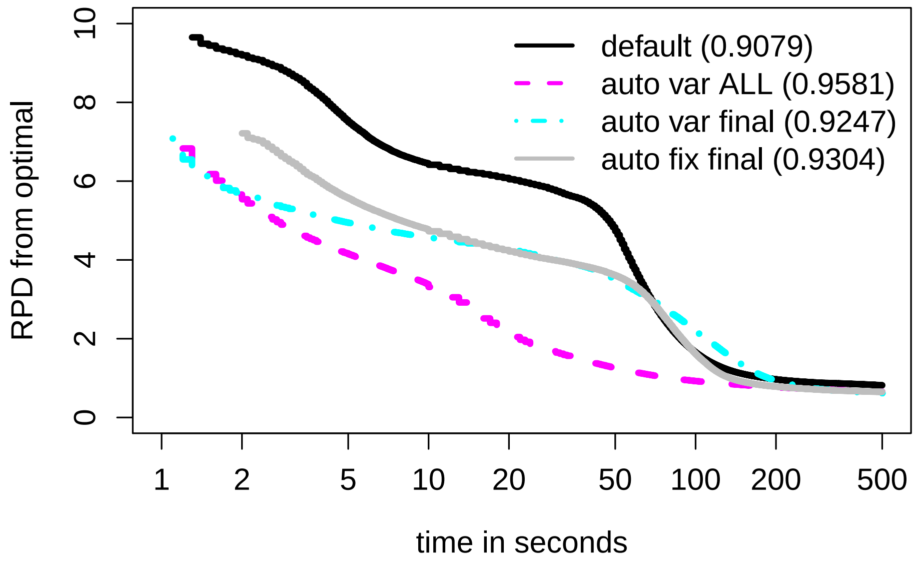


Figure 4: Anytime behaviour of automatically tuned configurations of MMAS vs. the default configuration. The number in the legend is the mean hypervolume of each variant over all runs. For the meaning of the labels, see the caption of Table 5.

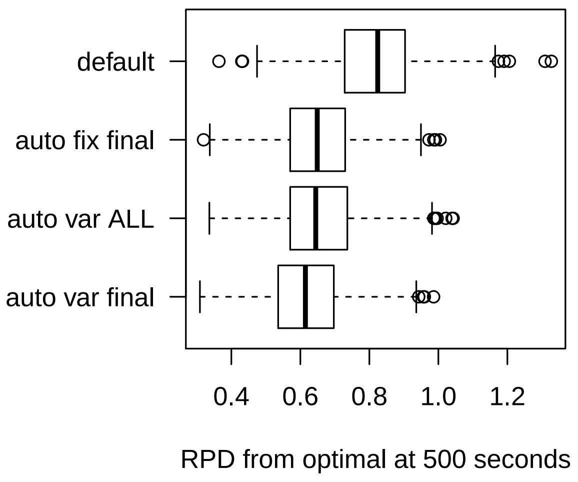


Figure 5: Final quality achieved by several variants of MMAS. For the meaning of the labels, see the caption of Table 5.

tion over the objective space [6, 69]. A weight function that assigns a larger value to a certain region of the objective space will bias the hypervolume indicator to favour nondominated sets that dominate that region. We show here that this formulation can straightforwardly be used to introduce a bias in the anytime behaviour produced by automatic configuration.

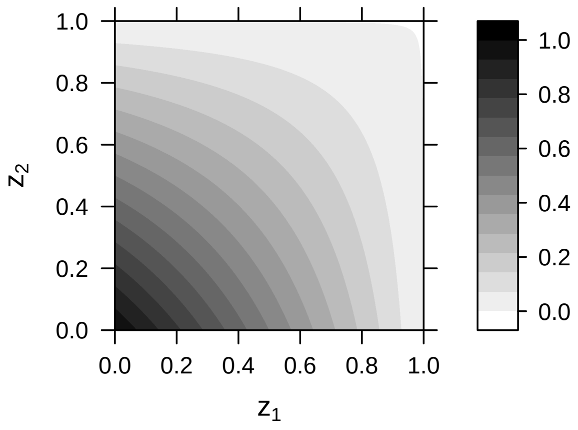
As an example, let us assume that the decision maker's preference is to obtain as good final solution quality as possible, while still giving some minor importance to achieving a good anytime behaviour. In other words, the decision maker prefers configurations that generate solution-quality curves that are better towards minimising the solution quality (in our case, the second objective). Zitzler et al. [69] suggest to model this preference by considering the following weight function (adapted here to minimisation):

$$w^{\text{qual}}(z) = e^{20\cdot(1-z_2)}/e^{20} \tag{6}$$

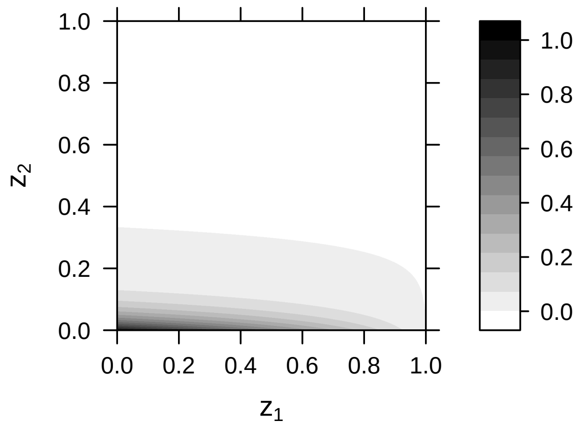
where $z = (z_1, z_2) \in Z$ is an objective vector, with z_1 representing time and z_2 representing solution quality, and $Z = [0,1] \times [0,1]$ represents the normalised bi-objective space of time$\times$quality.

The weighted hypervolume is computed as the integral of the weight function over the region dominated by a set of nondominated points and bounded above by a reference point. To give a rough idea of this integral when using the weighted function w^{qual}, Figure 6(b) shows the value of the weighted hypervolume for each individual vector in the normalised objective space Z and with reference point $(1,1)$. The plot shows that vectors with very small values of z_2 are assigned a high hypervolume, but vectors with values of z_2 larger than 0.2 are assigned a hypervolume close to zero. By comparison, Figure 6(a) shows the value of the non-weighted hypervolume, which is symmetric around the diagonal, that is, without a preference for either objective.

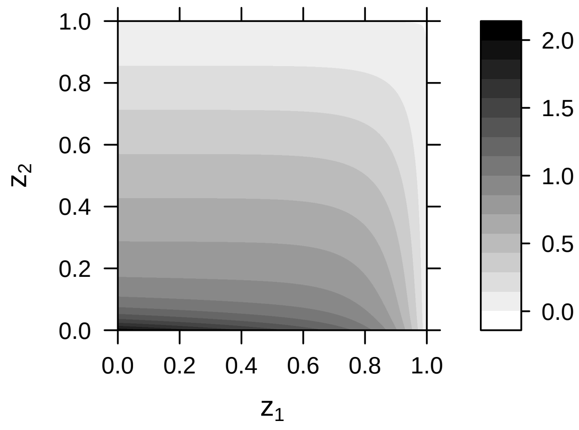
As shown in Figure 6(b), when using the weighted function w^{qual}, the gradient of the hypervolume values is very steep and most of the objective space has a hypervolume close to zero. We can make the gradient gentler by weighting also the z_1 component (corresponding to time), but then we have to increase the exponent associated to z_2 in order to keep a strong



(a) classical hypervolume



(b) weighted hypervolume with w^{qual}



(c) weighted hypervolume with w^{xqual}

Figure 6: Value of the classical hypervolume (a) and the weighted hypervolume when using weight functions w^{qual} (b) and w^{xqual} (c) on the normalised objective space. The grey level at each point gives the (weighted) hypervolume of that individual point, which is computed as the integral of the weight function over the region dominated by the point and bounded above by the reference point $(1, 1)$.

preference for low solution quality. This is done with the following weight function:

$$w^{\text{xqual}} = e^{10\cdot z_1}/e^{10} + e^{100\cdot(1-z_2)}/e^{100} \tag{7}$$

The weighted hypervolume using this weight function for each individual vector in the objective space Z is shown in Fig. 6(c). In this case, there is a gentler gradient of the hypervolume value than in Fig. 6(b). Moreover, the value of the hypervolume increases exponentially in the direction of decreasing z_2 (solution quality), while it stays roughly constant along z_1 (except for very high values of z_1).

We illustrate the differences between the original hypervolume and the two weighted variants above with an example. Figure 7 shows five performance profiles (not aggregated over fixed run-time or over fixed quality-targets) in the normalised objective space Z. The plot shows the region $z_2 \in [0.0, 0.15]$, where we can see that the performance profiles are ordered according to the final quality achieved, with profile a being the best and profile e being the worst. The legend provides three numbers for each profile, which correspond to evaluating the profile with the classical hypervolume, the hypervolume weighted by w^{qual} and the hypervolume weighted by w^{xqual}, respectively. Table 8 gives the profiles in increasing order of preference according to each measure.

Table 8: Ranking of the performance profiles in Fig. 7 according to various preferences

Preference	**Ranking (best to worst)**
final quality	a b c d e
hypervolume	c b d a e
w^{qual}	c b a d e
w^{xqual}	a c b d e

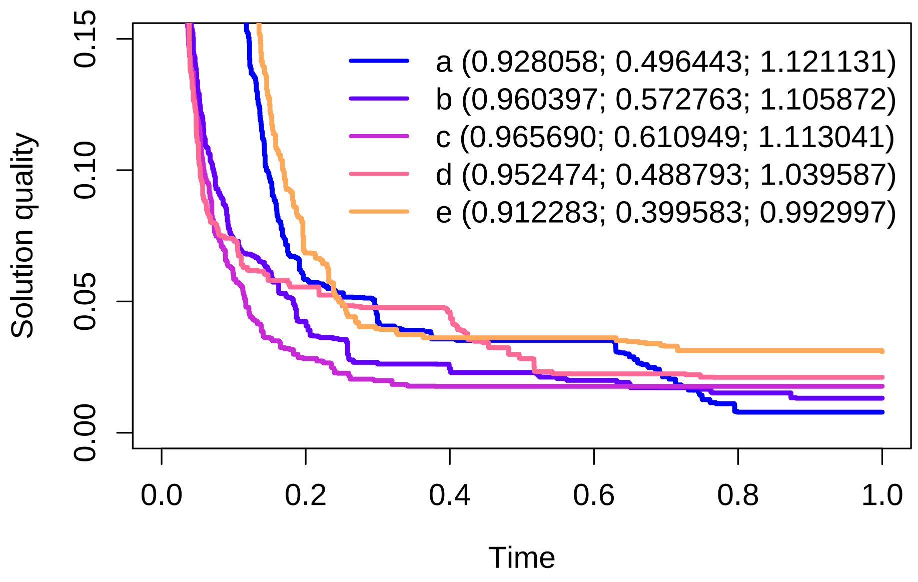


Figure 7: For each performance profile, the legend shows the classical hypervolume, and the weighted hypervolume variants w^{qual} and w^{xqual}.

In this example, the classical hypervolume ranks profile a, which is the profile with the best final quality, worse than other three profiles. The weighted hypervolume functions increase the preference for profile a, and our proposed variant w^{xqual} gives it the highest rank.

Next, we test the effect of these two weighted hypervolume functions on the automatic configuration procedure. In particular, we carry out additional runs of `irace` using the weighted hypervolume variants described above, i.e., w^{qual} (Eq. 6) and w^{xqual} (Eq. 7). We run `irace` with the same setup as for tuning all parameter variations at once in Section 5.2, in particular, with a budget of 10 000 runs of ACOTSP. These additional tuning runs produce two new configurations of MMAS, which we ran 25 times with different random seed on each test instance.

Figure 8 plots the mean RPD over all runs of the resulting four configurations of MMAS: the default configuration (default); the one resulting from automatically tuning all variation parameters at once using the classical hypervolume (auto var ALL); the configuration obtained with the same tuning setup but using the weighted hypervolume with w^{qual} (whv qual); and the

Table 9: Configurations of MMAS ordered according to the sum of ranks with respect to the final solution quality obtained. The numbers in parenthesis are the difference of ranks relative to the best configuration. ΔR_α is the statistically significant difference in ranks according to the post-hoc test for the Friedman-test with $\alpha = 0.05$. Configurations that are not significantly different from the best one are indicated in bold face. For the meaning of the labels, see the caption of Table 5.

ΔR_α	**Configurations (ΔR)**
23.16	**whv (xqual) (0), auto var final (11)**, whv (qual) (47), auto var ALL (94), auto fix final (99), default (199)

configuration obtained using the weighted hypervolume with w^{xqual} (whv xqual). In addition, the legend provides three numbers for each profile, which correspond to evaluating the results with the classical hypervolume, the hypervolume weighted by w^{qual} and the hypervolume weighted by w^{xqual}, each of them averaged over all runs.

In terms of final quality, the two configurations tuned with the weight functions (w^{qual} and w^{xqual}) are slightly better than the one tuned with the classical hypervolume, as indicated by the boxplots given in Fig. 9. Moreover, the configurations tuned with the weight functions obtain the lowest final quality in most instances. In fact, according to the Friedman test, these configurations are significantly better than configurations obtained by tuning for the classical hypervolume and for final quality (Table 9). The main conclusion of these experiments is that the weighted hypervolume allows us to set preferences on the trade-off between quality and time. For example, the weighted function w^{xqual} imposes a strong preference for good final quality.

6.1. On the relative difficulty of tuning for different weighted hypervolume variants

The numbers provided in the legend of Fig. 8 correspond to evaluating each tuned configuration with the classical hypervolume, the hypervolume weighted by w^{qual} and the hypervolume weighted by w^{xqual}, respectively. Interestingly, the numbers indicate that the configuration tuned using w^{xqual} as the anytime criterion obtains a better hypervolume weighted by w^{qual} than the configuration tuned using w^{qual} as the anytime criterion. To assess whether this is due to the stochastic nature of the algorithms, we repeat each tuning run five times with different random seed, thus producing five different MMAS configurations tuned using w^{qual} and another five using w^{xqual}. The configurations are evaluated on the test instances and we measure for each of them the mean hypervolume weighted by w^{qual} and the mean hypervolume weighted by w^{xqual}, as above.

Results are summarised on Table 10. In particular, the 95% confidence interval of the difference between the two tuning approaches does not indicate a statistically significant difference between the w^{qual} values on the test set. On the other hand, it does suggest that higher (i.e., better) w^{xqual} values are obtained on the test set when tuning with w^{xqual} than when tuning with w^{qual}. A preliminary conclusion from these results would be that maximising w^{qual} is equally difficult when tuning with w^{qual} or w^{xqual}. Conversely, maximising w^{xqual} appears to be easier when tuning with w^{xqual} than with w^{qual}.

Taking into account Fig. 6(b) and (c), we can observe that both weight functions are strongly correlated and also that w^{qual} is "flatter" than w^{xqual}, that is, there are more *plateau* regions with almost the same value for different points. Our conjecture is that the strong

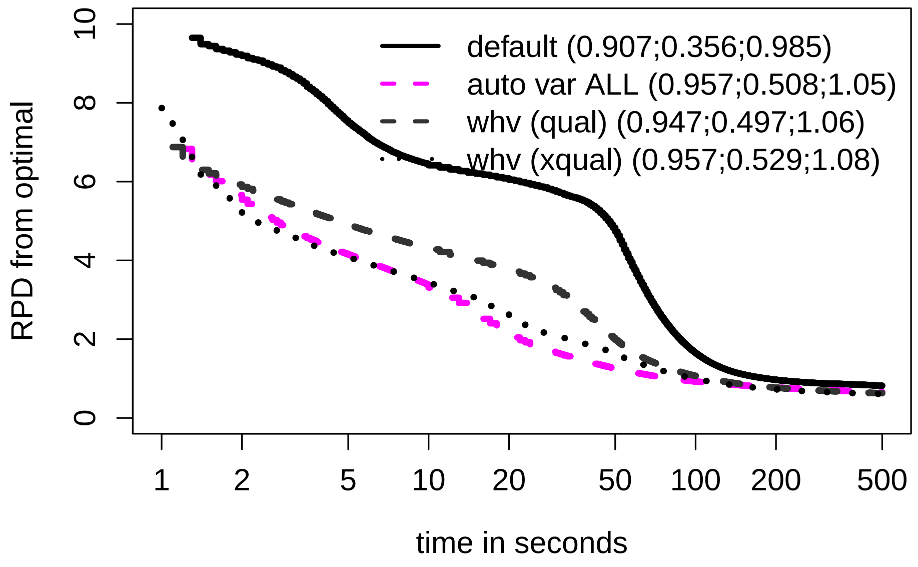


Figure 8: Automatically tuned configurations of MMAS vs. the default configuration. For each configuration, the legend shows the classical hypervolume, and the weighted hypervolume variants w^{qual} and w^{xqual}. For the meaning of the labels, see the caption of Table 5.

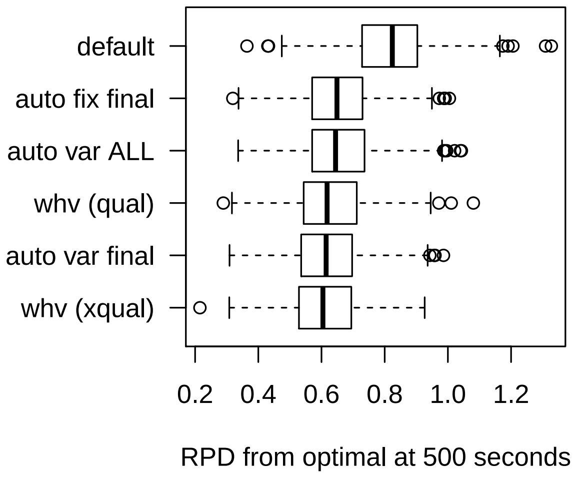


Figure 9: Final quality achieved by several variants of MMAS. For the meaning of the labels, see the caption of Table 5.

Table 10: Comparison of independent runs of `irace` using either the hypervolume weighted by w^{qual} or by w^{xqual} as the anytime criterion. Each cell reports the mean hypervolume, weighted by w^{qual} or by w^{xqual}, obtained on the test set by the configuration generated by each `irace` run. We also report the 95% confidence interval of the difference between values that correspond to w^{qual} as the tuning criterion minus the values that correspond to w^{xqual} as the tuning criterion.

		Test criterion	
Tuning criterion	Run	w^{qual}	w^{xqual}
w^{qual}	#1	0.497	1.060
	#2	0.465	1.020
	#3	0.457	1.042
	#4	0.489	1.043
	#5	0.486	1.043
w^{xqual}	#1	0.529	1.080
	#2	0.387	1.031
	#3	0.505	1.070
	#4	0.494	1.062
	#5	0.490	1.061
Paired 95% CI of the difference		$(-0.0625, 0.0581)$	$(-0.0267, -0.0117)$

correlation makes possible to tune for one weight function and maximise the other. At the same time, the relative flatness of w^{qual} makes it a harder optimisation criterion for tuning than w^{xqual}.

7. Case Study: Automatic configuration of an anytime MIP solver

7.1. Experimental setup

In this second scenario, we apply our proposed approach to a very different problem with a large number of parameters. In particular, we tune 207 parameters of SCIP [1], a mixed integer programming (MIP) solver. The number of parameters is too large to be detailed here, but details can be found in the supplementary page [47].

The benchmark set is composed of 2 000 MIP-encoded instances (200 goods, 1000 bids) of the NP-hard winner determination problem for combinatorial auctions [45] previously used by Hutter et al. [39]. The benchmark set is split in two disjoint sets of 1 000 instances each, one is used for training and the other for testing. In a combinatorial auction, bids are placed for subsets of goods. The goal in the winner determination problem is to find an assignment of goods to bids that maximises the total value of the winning bids.

For our experiments here, we use SCIP version 2.0.2 linked with the linear programming solver SoPlex 1.5.0. We set the maximum memory limit of SCIP to 350 MB. During our experiments, we discovered that some parameter configurations produced an incorrect behaviour of SCIP, and we assign those configurations the worst possible hypervolume. We give SCIP a time limit of 300 seconds, and we allow 5 000 runs of SCIP for each run of `irace`. We carry out the tuning as before, that is, we combine `irace` with the hypervolume measure in order

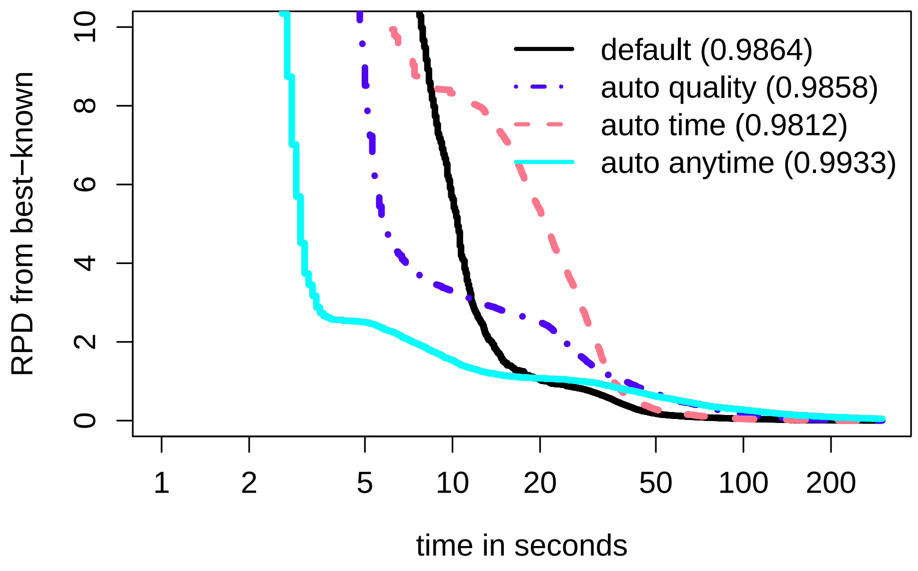


Figure 10: Mean RPD over all test instances for different configurations of SCIP. The number in parentheses is the mean hypervolume corresponding to that configuration.

to improve the anytime behaviour of SCIP. We seed the automatic configuration procedure with the default configuration of SCIP.

For the purposes of comparison, we perform two additional tuning runs with two different objectives: (1) minimising the run time to find the optimal solution, and (2) maximising the final objective value obtained after 300 seconds. This way we obtained two additional configurations of SCIP, which we label as auto time and auto quality, respectively. We use these configurations to asses the potential loss of either run time or final solution quality, when tuning for improving the anytime behaviour.

Finally, we run all configurations of SCIP obtained from the various tuning setups plus the default configuration one time on each instance from the test set.

7.2. Analysis of SCIP configurations

As a first step in our analysis, we graphically examine the solution quality over time. For each configuration, we compute the mean RPD over the 1 000 test instances at each time step. Next, we plot the mean RPD over time in Fig. 10. We also give in the legend the mean hypervolume value corresponding to each configuration of SCIP. The plot uses a logarithmic scale for the x-axis (time), since the largest differences appear on the first half of the computation time limit.

The plot shows that the configuration tuned with the hypervolume (auto anytime) obtains a better anytime behaviour (and a higher hypervolume) than the rest. Moreover, both the configuration tuned for final quality and the one tuned for solving time show worse anytime behaviour (and lower hypervolume) than the default configuration of SCIP. The differences observed in the hypervolume values (and, hence, in the anytime behaviour) of each SCIP configuration are more evident in Fig 11(a), which shows that the hypervolume values corresponding to auto anytime are much larger than those corresponding to the other configurations of SCIP.

Improving the anytime behaviour does not necessarily mean that instances are solved faster to optimality. Fig. 11(b) shows the time required by each configuration to solve each of the 1 000 test instances. The best configurations of SCIP according to this criterion are the `default` configuration and the configuration tuned specifically for this criterion (`auto time`). This result is not surprising, since this is the most popular evaluation criterion in mixed-integer programming, and, hence, we presume that SCIP is by default tuned for it.

We also examine the potential loss of final quality. Fig. 11(c) shows the RPD from the optimal at the cut-off time of 300 seconds. All configurations solve most of the instances to optimality (or very close to it). However, the configuration tuned for anytime (`auto anytime`) is the one that diverges most often from near-optimality. Hence, there is some loss of final quality when tuning using the hypervolume.

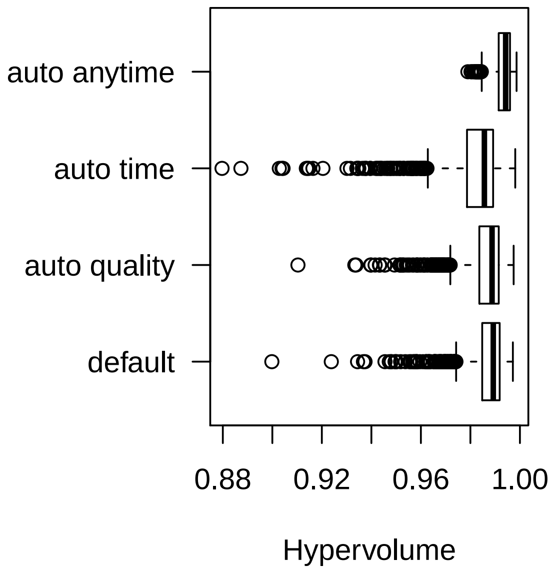
If we look at the solution quality up to a different cut-off time, the situation is certainly different. For example, if we consider solution quality up to 10 seconds (Fig. 11(d)), there is a large difference between the configurations. While the `auto anytime` configuration obtains an RPD value much lower than 10% in most cases, the RPD values of the `default` configuration are frequently larger than 10%.

The observations above are further confirmed by statistical analysis. We carry out four independent Friedman tests (as described in Sec. 5.4.1), one for each evaluation criterion shown in Fig. 11. The results of the four tests are reported in Table 11. As expected, the best configuration in terms of hypervolume is the one tuned for that criterion (`auto anytime`), which is significantly better than the rest by a large margin. The `auto anytime` configuration is also the clear winner in terms of the solution quality obtained if stopped after 10 seconds. Moreover, in terms of final quality, the differences between the strategies are not statistically significant. In fact, the difference in the sum of ranks between `auto anytime` and `default` is only 44.5.

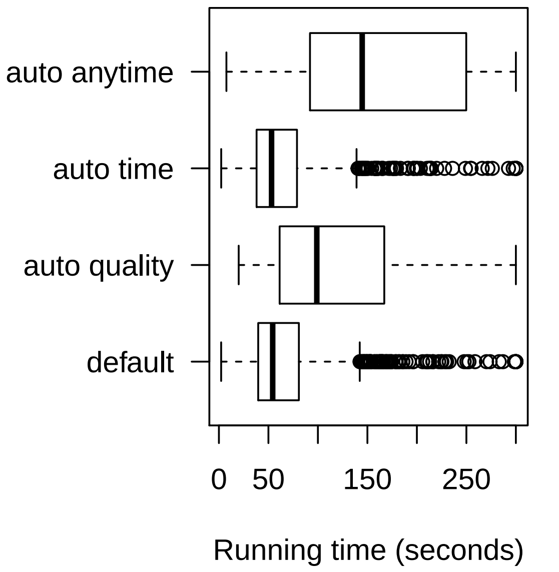
Finally, by using the weighted hypervolume as explained in Section 6, we are able to find a configuration of SCIP with good anytime behaviour and that ranks better than `default` according to final quality. However, the differences in ranks are still not statistically significant. Hence, for conciseness, we do not discuss the results of using the weighted hypervolume for tuning SCIP here, but we provide the results as supplementary material [47]. The results provided here are sufficient to conclude that the proposed method was able to find a configuration of SCIP that has better anytime behaviour than the default, without a significant loss of final quality.

8. Conclusions

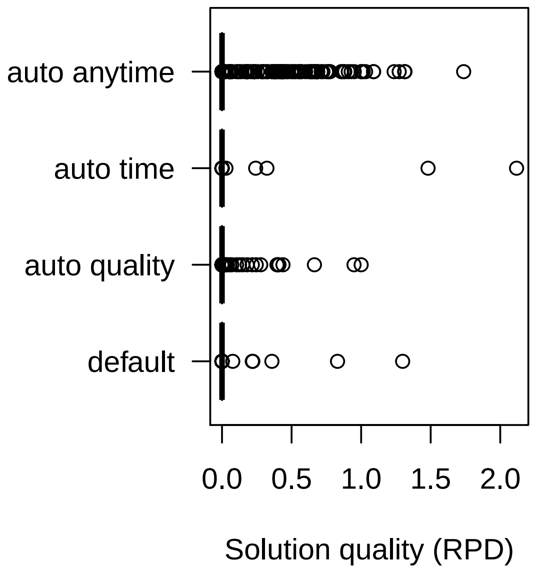
In this paper, we have shown that the combination of `irace` and the hypervolume quality measure is effective at improving the anytime behaviour of optimisation algorithms. We have presented two representative and challenging case studies. The first case study compared the results obtained automatically against those obtained by a human expert for the task of designing parameter variation strategies that show good anytime behaviour. Our results show that the automatic configuration method is able to match the anytime behaviour obtained by the parameter variation strategies designed by a human expert. Moreover, the automatic method allows exploring a much larger design space, potentially leading to configurations with better anytime behaviour. These are expected results when using automatic configuration tools for tuning with a fixed termination criterion. However, this is the first time that such results have been obtained when automatically designing anytime algorithms. Recently, we



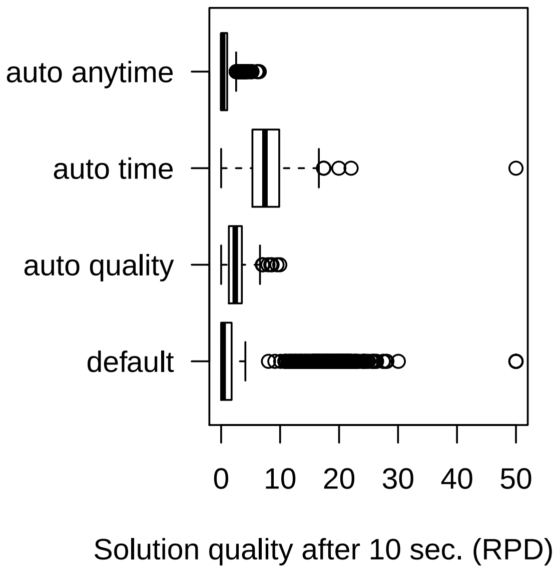
(a)



(b)



(c)



(d)

Figure 11: Boxplots of the results obtained by four different parameter configurations of SCIP according to various evaluation criteria.

Table 11: Configurations of SCIP ordered according to the sum of ranks with respect to four different evaluation criteria. The numbers in parenthesis are the difference of ranks relative to the best ranked configuration. ΔR_α is the statistically significant difference in ranks according to the post-hoc test for the Friedman-test with $\alpha = 0.05$. Configurations that are not significantly different from the best one according to the Friedman test are indicated in bold face.

ΔR_α	**Configurations** (ΔR)	**Evaluation criterion**
75.1	**auto anytime** (0), default (1183), auto quality (1490), auto time (2335)	*Hypervolume*
53.52	**auto time** (0), default (192), auto quality (1603), auto anytime (2353)	*Time to best found*
83.01	**auto anytime** (0), default (460), auto quality (1019.5), auto time (2024.5)	*Quality after 10 seconds*
∞	**default** (0), **auto time** (8.5), **auto quality** (19), **auto anytime** (44.5)	*Final quality (300 seconds)*

have applied the approach proposed here to improve the anytime behaviour of a state of the art optimiser for black-box continuous optimisation [51]. Our results there show that even for such state of the art optimisers, the default parameter settings are not well-suited for scenarios where the termination criterion is unknown in advance. Although the results presented here focus on single-objective optimisers, our approach is applicable to multi-objective optimisers as well. In recent work, we have applied it to automatically configure the parameters of multi-objective evolutionary algorithms in order to improve their anytime behaviour [57].

In the second case study presented here, we apply our approach to an off-the-shelf optimisation solver, with a very large number of parameters. In this case, the optimisation solver is already tuned to solve problems to optimality as fast as possible. However, we show that if stopped before reaching optimality, the results may be very poor. Our proposed approach helps to tune such solvers in order to be more robust in case of earlier termination, without specifying in advance when the algorithm could be terminated. Our results show that important improvements can be obtained, specially for very early termination, without sacrificing much of the final quality. Moreover, comparing the configurations that produce better anytime behaviour versus those that produce better final quality (or shorter time to optimality) may lead to improvements in the solvers themselves.

The choice of the hypervolume measure also allows us to incorporate preference information into the automatic configuration process by means of the weighted hypervolume. We propose a weighted formulation that emphasises a good final quality but still takes into account the overall anytime behaviour of the algorithms. We show that by adding such preferences, it is possible to effectively bias the configurations selected by the automatic configuration tool. This allows customising optimisation algorithms to very specific anytime scenarios, where an exact termination criterion is not known, but there is some a priori knowledge of what is expected.

An open question is how to extend the results to longer termination criteria than the ones that are feasible to test during automatic configuration. A problem that may arise is that configurations produce good results up until the tested termination criterion, but the performance becomes unsatisfactory for longer runs. Woodruff et al. [65] have studied how to dynamically set a termination criterion. Survival analysis techniques may help to estimate the behaviour of the algorithms for longer runtime [29]. These techniques could be incorporated into our approach in order to dynamically adjust the maximum cut-off time while tuning the anytime behaviour.

Finally, we are convinced that our approach contributes towards the final goal of designing algorithms that are more robust to different termination criteria and, hence, applicable to a wider range of scenarios, without sacrificing solution quality.

Acknowledgements We are grateful to the SCIP developers, in particular, Timo Berthold, Stefan Heinz and Ambros Gleixner, for kindly providing information about SCIP parameters. The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n^o 246939. This work was also supported by the Meta-X project, funded by the Scientific Research Directorate of the French Community of Belgium. Manuel López-Ibáñez and Thomas Stützle acknowledge support of the F.R.S.-FNRS of which they are a post-doctoral researcher and a research associate, respectively.

References

[1] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

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

[3] S. Aine, R. Kumar, and P. P. Chakrabarti. Adaptive parameter control of evolutionary algorithms to improve quality-time trade-off. *Applied Soft Computing*, 9(2):527–540, 2009. doi: 10.1016/j.asoc.2008.07.001.

[4] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In I. P. Gent, editor, *Principles and Practice of Constraint Programming, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, Heidelberg, Germany, 2009. doi: 10.1007/978-3-642-04244-7_14.

[5] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*, pages 1769–1776. IEEE Press, Piscataway, NJ, Sept. 2005. doi: 10.1109/CEC.2005.1554902.

[6] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Investigating and exploiting the bias of the weighted hypervolume to articulate user preferences. In F. Rothlauf, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009*, pages 563–570. ACM Press, New York, NY, 2009.

[7] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. Hypervolume-based multiobjective optimization: Theoretical foundations and practical implications. *Theoretical Computer Science*, 425:75–103, 2012. doi: 10.1016/j.tcs.2011.03.012.

[8] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer, Heidelberg, Germany, 2007.

[9] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism.* Springer, Berlin, Germany, 2006.

[10] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The sequential parameter optimization toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Germany, 2010.

[11] T. Bartz-Beielstein, J. Ziegenhirt, W. Konen, O. Flasch, P. Koch, and M. Zaefferer. *SPOT: Sequential Parameter Optimization*, 2011. URL `http://cran.r-project.org/package=SPOT`. R package.

[12] M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence.* Springer, Berlin/Heidelberg, Germany, 2009. doi: 10.1007/978-3-642-00483-4.

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

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

[15] J. Branke and J. Elomari. Simultaneous tuning of metaheuristic parameters for various computing budgets. In N. Krasnogor and P. L. Lanzi, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 263–264. ACM Press, New York, NY, 2011. doi: 10.1145/2001858.2002006.

[16] M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems.* PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2005.

[17] W. J. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, NY, third edition, 1999.

[18] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

[19] T. Dean and M. S. Boddy. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI-88*, pages 49–54. AAAI Press, 1988.

[20] M. den Besten. *Simple Metaheuristics for Scheduling.* PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2004. URL `http://tuprints.ulb.tu-darmstadt.de/516/`.

[21] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1 (1):53–66, 1997.

[22] J. Dréo. Using performance fronts for parameter setting of stochastic metaheuristics. In F. Rothlauf, editor, *GECCO (Companion)*, pages 2197–2200. ACM Press, New York, NY, 2009. doi: 10.1145/1570256.1570301.

[23] J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In N. Krasnogor and P. L. Lanzi, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2019–2026. ACM Press, New York, NY, 2011. doi: 10.1145/2001576.2001847.

[24] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011. doi: 10.1016/j.swevo.2011.02.001.

[25] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In F. Lobo, C. F. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, pages 19–46. Springer, Berlin, Germany, 2007.

[26] C. M. Fonseca, V. Grunert da Fonseca, and L. Paquete. Exploring the performance of stochastic multiobjective optimisers with the second-order attainment function. In C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, editors, *Evolutionary Multi-criterion Optimization (EMO 2005)*, volume 3410 of *Lecture Notes in Computer Science*, pages 250–264. Springer, Heidelberg, Germany, 2005.

[27] C. M. Fonseca, L. Paquete, and M. López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*, pages 1157–1163. IEEE Press, Piscataway, NJ, July 2006. doi: 10.1109/CEC.2006.1688440.

[28] C. M. Fonseca, A. P. Guerreiro, M. López-Ibáñez, and L. Paquete. On the computation of the empirical attainment function. In R. H. C. Takahashi et al., editors, *Evolutionary Multi-criterion Optimization (EMO 2011)*, volume 6576 of *Lecture Notes in Computer Science*, pages 106–120. Springer, Heidelberg, Germany, 2011. doi: 10.1007/978-3-642-19893-9_8.

[29] M. Gagliolo and C. Legrand. Algorithm survival analysis. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 161–184. Springer, Berlin, Germany, 2010. doi: 10.1007/978-3-642-02538-9_7.

[30] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Boston, MA, USA, 1989.

[31] V. Grunert da Fonseca and C. M. Fonseca. A link between the multivariate cumulative distribution function and the hitting function for random closed sets. *Statistics & Probability Letters*, 57(2):179–182, 2002.

[32] V. Grunert da Fonseca and C. M. Fonseca. The attainment-function approach to stochastic multiobjective optimizer assessment and comparison. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, Berlin, Germany, 2010.

[33] V. Grunert da Fonseca, C. M. Fonseca, and A. O. Hall. Inferential performance assessment of stochastic optimisers and the attainment function. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Evolutionary Multi-criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 213–225. Springer, Heidelberg, Germany, 2001.

[34] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, France, 2009.

[35] J. N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2): 201–212, 1994.

[36] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1996.

[37] H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, Feb. 2012. doi: 10.1145/2076450.2076469.

[38] H. H. Hoos and T. Stützle. *Stochastic Local Search—Foundations and Applications.* Morgan Kaufmann Publishers, San Francisco, CA, 2005.

[39] 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, Oct. 2009.

[40] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization, 5th International Conference, LION 5*, Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2011.

[41] D. S. Johnson, L. A. McGeoch, C. Rego, and F. Glover. 8th DIMACS implementation challenge. `http://www.research.att.com/~dsj/chtsp/`, 2001.

[42] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. In C. Boutilier, editor, *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 517–524. AAAI Press, Menlo Park, CA, 2009. URL `http://ijcai.org/papers09/Papers/IJCAI09-093.pdf`.

[43] J. D. Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. In *Proceedings of the 5th International*

Conference on Intelligent Systems Design and Applications, pages 552–557, 2005. doi: 10.1109/ISDA.2005.15.

[44] J. D. Knowles, L. Thiele, and E. Zitzler. A tutorial on the performance assessment of stochastive multiobjective optimizers. TIK-Report 214, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, Feb. 2006. Revised version.

[45] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In A. Jhingran et al., editors, *ACM Conference on Electronic Commerce (EC-00)*, pages 66–76. ACM Press, New York, NY, 2000. doi: 10.1145/352871.352879.

[46] M. López-Ibáñez and T. Stützle. Automatic configuration of multi-objective ACO algorithms. In M. Dorigo et al., editors, *Swarm Intelligence, 7th International Conference, ANTS 2010*, volume 6234 of *Lecture Notes in Computer Science*, pages 95–106. Springer, Heidelberg, Germany, 2010. doi: 10.1007/978-3-642-15461-4_9.

[47] M. López-Ibáñez and T. Stützle. Automatically Improving the Anytime Behaviour of Optimisation Algorithms: Supplementary material. `http://iridia.ulb.ac.be/supp/IridiaSupp2012-011/`, 2012.

[48] M. López-Ibáñez and T. Stützle. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012. doi: 10.1109/TEVC.2011.2182651.

[49] M. López-Ibáñez, L. Paquete, and T. Stützle. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin, Germany, 2010. doi: 10.1007/978-3-642-02538-9_9.

[50] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

[51] M. López-Ibáñez, T. Liao, and T. Stützle. On the anytime behavior of IPOP-CMA-ES. In C. A. Coello Coello et al., editors, *PPSN 2012, Part I*, volume 7491 of *Lecture Notes in Computer Science*, pages 357–366. Springer, Heidelberg, Germany, 2012. doi: 10.1007/978-3-642-32937-1_36.

[52] S. Loudni and P. Boizumault. Combining VNS with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191:705–735, 2008. doi: 10.1016/j.ejor.2006.12.062.

[53] O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Research*, 11(1–5):193–225, 1997.

[54] M. Maur, M. López-Ibáñez, and T. Stützle. Pre-scheduled and adaptive parameter variation in MAX-MIN Ant System. In H. Ishibuchi et al., editors, *Proceedings of the*

2010 Congress on Evolutionary Computation (CEC 2010), pages 3823–3830. IEEE Press, Piscataway, NJ, 2010. doi: 10.1109/CEC.2010.5586332.

[55] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo. Frankenstein's PSO: A composite particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 13(5):1120–1132, 2009. doi: 10.1109/TEVC.2009.2021465.

[56] V. Nannen and A. E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In M. Cattolico et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006*, pages 183–190. ACM Press, New York, NY, 2006. doi: 10.1145/1143997.1144029.

[57] A. Radulescu, M. López-Ibáñez, and T. Stützle. Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. In R. C. Purshouse, P. J. Fleming, C. M. Fonseca, S. Greco, and J. Shaw, editors, *Evolutionary Multi-criterion Optimization (EMO 2013)*, volume 7811 of *Lecture Notes in Computer Science*, pages 825–840. Springer, Heidelberg, Germany, 2013. ISBN 978-3-642-37139-4. doi: 10.1007/978-3-642-37140-0_61.

[58] T. Stützle. MAX-MIN Ant System for the quadratic assignment problem. Technical Report AIDA–97–4, FG Intellektik, FB Informatik, TU Darmstadt, Germany, July 1997.

[59] T. Stützle. ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem, 2002. URL `http://www.aco-metaheuristic.org/aco-code/`.

[60] T. Stützle and H. H. Hoos. MAX-MIN Ant System and local search for combinatorial optimization problems. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 137–154. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[61] T. Stützle and H. H. Hoos. MAX-MIN Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[62] T. Stützle, M. López-Ibáñez, P. Pellegrini, M. Maur, M. A. Montes de Oca, M. Birattari, and M. Dorigo. Parameter adaptation in ant colony optimization. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 191–215. Springer, Berlin, Germany, 2012. doi: 10.1007/978-3-642-21434-9_8.

[63] B. W. Wah and Y. X. Chen. Optimal anytime constrained simulated annealing for constrained global optimization. In R. Dechter, editor, *Principles and Practice of Constraint Programming, CP 2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 425–440. Springer, Heidelberg, Germany, 2000. doi: 10.1007/3-540-45349-0_31.

[64] S. Wessing, N. Beume, G. Rudolph, and B. Naujoks. Parameter tuning boosts performance of variation operators in multiobjective optimization. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 728–737. Springer, Heidelberg, Germany, 2010. doi: 10.1007/978-3-642-15844-5_73.

[65] D. L. Woodruff, U. Ritzinger, and J. Oppen. Research note: the point of diminishing returns in heuristic search. *International Journal of Metaheuristics*, 1(3):222–231, 2011. doi: 10.1504/IJMHeur.2011.041195.

[66] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3): 73–83, 1996.

[67] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto evolutionary algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[68] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

[69] E. Zitzler, D. Brockhoff, and L. Thiele. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In S. Obayashi et al., editors, *Evolutionary Multi-criterion Optimization (EMO 2007)*, volume 4403 of *Lecture Notes in Computer Science*, pages 862–876. Springer, Heidelberg, Germany, 2007. doi: 10.1007/978-3-540-70928-2_64.