HAL
open science

Automated Configuration of Genetic Algorithms by Tuning for Anytime Performance

Furong Ye, Carola Doerr, Hao Wang, Thomas Back

To cite this version:

Furong Ye, Carola Doerr, Hao Wang, Thomas Back. Automated Configuration of Genetic Algorithms by Tuning for Anytime Performance. IEEE Transactions on Evolutionary Computation, 2022, 26 (6), pp.1526 - 1538. 10.1109/TEVC.2022.3159087 . hal-03614648

HAL Id: hal-03614648

https://hal.sorbonne-universite.fr/hal-03614648v1

Submitted on 21 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Automated Configuration of Genetic Algorithms by Tuning for Anytime Performance

Furong Ye, Carola Doerr, Hao Wang, and Thomas Bäck, *Fellow, IEEE*

arXiv:2106.06304v2 [cs.NE] 17 Mar 2022

***Abstract*—Finding the best configuration of algorithms' hyperparameters for a given optimization problem is an important task in evolutionary computation. We compare in this work the results of four different hyperparameter optimization approaches for a family of genetic algorithms on 25 diverse pseudo-Boolean optimization problems. More precisely, we compare previously obtained results from a grid search with those obtained from three automated configuration techniques: iterated racing, mixed-integer parallel efficient global optimization, and mixed-integer evolutionary strategies.**

Using two different cost metrics, expected running time and the area under the empirical cumulative distribution function curve, we find that in several cases the best configurations with respect to expected running time are obtained when using the area under the empirical cumulative distribution function curve as the cost metric during the configuration process. Our results suggest that even when interested in expected running time performance, it might be preferable to use anytime performance measures for the configuration task. We also observe that tuning for expected running time is much more sensitive with respect to the budget that is allocated to the target algorithms.

***Index Terms*—Evolutionary Computation, Algorithm Configuration, Black-Box Optimization, Performance Measures**

I. Introduction

Mutation and crossover are two key operators of genetic algorithms (GAs), and there is a long debate about the effectiveness of the two operators and their combinations [33], [39].

Mutation can maintain the population diversity of the GA, and it allows the GA to exploit promising search areas by using small mutation rates. In the early years of evolutionary computation, *evolution strategies* [37] concentrated on mutation and showed their power with adaptive mutation rates for optimization [4].

Classic GAs, on the other hand, used *crossover* as their main variation operator [3], [12], [20]. Intuitively speaking, the key idea of crossover is to create offspring by recombining information from two or more "parents". As is the case for mutation operators [14], [53], novel types of crossover keep being developed [44], [45], [46].

Several works study the synergy of mutation and crossover, both by empirical and by theoretical means, see [12], [18], [26], [34] and [42], respectively, as well as references mentioned therein. However, most of these results focus on specific algorithms and problems. Widely accepted guidelines for their deployment are sparse, leading to a situation in which users often rely on their own experience. To reduce the bias inherent to such manual decisions, a number of *automated algorithm configuration* techniques have been developed, to assist the user with data-driven suggestions. To deploy these techniques, one formulates the operator choice and/or their intensity as a meta-optimization problem, widely referred to as the *algorithm configuration (AC)* or the *hyperparameter optimization (HPO)* problem.

The AC problem was classically addressed by standard search heuristics such as mixed-integer evolution strategies [2], [21], [30]. More specific AC tools have been developed in recent years, among them surrogate-based models (e.g., SPOT [5], SMAC [24], MIP-EGO [47]), racing-based methods (Irace [31], F-race [6]) and optimization-based methods (ParamILS [25]).

Summary of Our Contributions. In this work, we study the effects of automated algorithm configuration on a genetic algorithm (GA) framework, applied to 25 diverse pseudo-Boolean optimization problems. We compare the results of four different configuration techniques. Our main interest is in analyzing the influence of the cost metric that is used to score different configurations on the quality of the configuration suggested by the AC methods. We consider two different cost metrics: *minimizing the expected running time (ERT)* and *maximizing the area under the empirical cumulative distribution function curve of running times (AUC)*. While minimizing ERT favors the average first hitting time of a single *fixed-target*, maximizing the AUC metric aims at optimizing *anytime performance*, which is measured across a whole set of (budget, target value) pairs. We show that in several cases tuning for AUC yields configurations that have smaller ERT values than those that were obtained when directly tuning for ERT.

Concretely, we build on our previous work [55] in which we analyzed a configurable framework of $(\mu + \lambda)$ GAs that scales the relevance of mutation and crossover by means of the crossover probability $p_c \in [0, 1]$. The framework creates new solution candidates by applying either mutation (with probability $1-p_c$) or crossover (with probability p_c). This way, it can separate the influence of these operators from each other. While we have studied several operator choices in [55] via plain grid search, we consider here only one type of crossover (uniform crossover) and one type of mutation (standard bit mutation), to keep the search space manageable and to better highlight our key findings.

We compare in this work the results from [55] with those

Furong Ye (E-mail: f.ye@liacs.leidenuniv.nl), Hao Wang (E-Mail: h.wang@liacs.leidenuniv.nl), and Thomas Bäck (E-Mail: t.h.w.baeck@liacs.leidenuniv.nl) are with Leiden Institute of Advanced Computer Science, Leiden University, Leiden, Netherlands.

Carola Doerr is with Sorbonne Université, CNRS, LIP6, Paris, France. E-mail: Carola.Doerr@lip6.fr.

obtained from three different types of automated algorithm configuration methods, one based on iterated racing (we use Irace [31]), one surrogate-assisted technique (we use the mixed-integer parallel efficient global optimization MIP-EGO [47]), and a classic heuristic optimization method (we use the mixed-integer evolutionary strategies MIES suggested in [30]). Our testbed are the 25 functions from the pseudo-Boolean optimization (PBO) test suite, suggested in [16], [55] and available in the IOHprofiler benchmarking environment [15].

Compared to the $(1+1)$ EA and to the configurations obtained by the grid search in [55], we observe that $(1+\lambda)$ mutation-only GAs perform well on ONEMAX, LEADINGONES, and most of their so-called W-model extensions (see Sec. 3.7 in [16]), and on Ising-Models, whereas the $(\mu+\lambda)$ GA benefits from using crossover and different mutation rates on the more complex optimization problems. Thanks to its ability to handle conditional configuration spaces, Irace is the only method that finds configurations of $(1+\lambda)$ mutation-only GAs for problems on which these are efficient. We also observe that, on some problems, the automatic configurators cannot obtain hyperparameter settings that are as good as those provided by a simple grid search. However, our key finding is that the configuration methods can find better configurations in terms of the ERT by using AUC as the cost metric, compared to directly using ERT instead. This advantage is particularly pronounced when the budget of the GAs is small compared to the ERT obtained by the best possible configuration. In this case, ERT cannot reflect the differences among the configurations, and the anytime performance provides more information to guide the configuration process.

Related Work. Automated algorithm configuration for improving the anytime performance of algorithms has been applied in several works, both with respect to classical CPU time (e.g., for the traveling salesman problem [7], for a MAX-MIN ant system [32], and for mixed-integer programming ibid) and for the here-considered function evaluation budgets (see [1] for a recent example). Sensitivity of algorithm configuration approaches with respect to the cutoff time (i.e., the budget of function evaluations for which algorithms are evaluated) has been studied in [23], [22]. In particular, [22] provides an example in which tuning for fixed-budget performance can be preferable under small cutoff times than tuning for optimization time, even if the latter is the original objective. We are not aware of any works using anytime performance measures with the objective to identify configurations that minimize ERT values.

Availability of Data and Source Code. All our data is publicly available at [54] and at the public repository of IOHanalyzer, accessible via https://iohanalyzer.liacs.nl/. Users interested in an interactive evaluation of our data can find our data by selecting 'PBO' as dataset source and '2021-mlga' as dataset. After loading the data from the repository, various statistics and visualizations are available for further investigation, see [50] for an overview. The source code of our experiments is available at https://github.com/FurongYe/Configuration-of-Genetic-Algorithms.

For readers' convenience, we provide in the supplementary material fixed-target charts for each of the 25 functions, similar to the one presented in Fig. 1.

Paper Organization. Sec. II describes the configurable $(\mu+\lambda)$ GA, the benchmark problems, and the cost metrics used to evaluate the algorithms. In Sec. III we present results for the three selected AC methods, evaluate their performance against the $(1+1)$ EA, and discuss the impact of the cost metrics for the AC methods. We conclude our work and discuss directions for future research in Sec. IV.

II. PRELIMINARIES

In this section, we describe our $(\mu+\lambda)$ GA framework (Sec. II-A), the benchmark problems (Sec. II-B), and the performance measures used to evaluate the algorithms (Sec. II-C).

Notational conventions: Throughout this paper, we study the maximization of so-called pseudo-Boolean functions, i.e., functions of the type $f : \{0,1\}^n \to \mathbb{R}$, mapping length-$n$ bit strings to real numbers. In particular, we always denote by n the dimension of the problem. For each positive integer k, we abbreviate $[k] = \{j \in \mathbb{N} \mid 1 \le j \le k\}$ and $[0..k] = [k] \cup \{0\}$.

A. A Family of $(\mu+\lambda)$ Genetic Algorithms

To study how mutation, crossover, and their combinations can be beneficial for GAs, we investigate a configurable genetic algorithm framework that can switch from mutation-only GAs towards crossover-only GAs by tuning the crossover probability p_c. Details of the framework can be found in Alg. 1, and we refer to it as the $(\mu+\lambda)$ GA in the following.

The $(\mu+\lambda)$ GA initializes the population uniformly at random (u.a.r.) and terminates when the optimum is found or when the cutoff time (i.e., the maximal number of fitness evaluations that the algorithm may perform) is reached. For each iteration, the $(\mu+\lambda)$ GA creates λ offspring. Each offspring is created by applying *either* mutation *or* crossover. More precisely, crossover is applied with probability p_c and mutation is used otherwise. Consequently, the $(\mu+\lambda)$ GA with $p_c = 0$ refers to a mutation-only GA, and the $(\mu+\lambda)$ GA with $p_c = 1$ refers to a crossover-only GA. After each iteration, the best μ points out of the combined set of μ parents and λ offspring form the next set of parents ("plus selection").

For this work, we consider *uniform crossover* and *standard bit mutation* with mutation rate p_m. These two variation operators are briefly described as follows. Note that the framework can easily be extended to include other operators.

- *Uniform crossover*: Each bit of the offspring is copied either from the first or from second parent, chosen independently and u.a.r. for each position.
- *Standard bit mutation*: The offspring is created by first copying the parent and then flipping ℓ distinct bits that are chosen u.a.r. The value ℓ is called the *mutation strength*. It is sampled from a conditional binomial distribution $\mathrm{Bin}_{>0}(n, p_m)$, which assigns to each $k \in [n]$ a probability of $\binom{n}{k} p_m^k (1-p_m)^{n-k} / (1-(1-p_m)^n)$. Compared to the classic textbook description of standard bit mutation, this version avoids sampling copies of the parents, by allocating the probability of sampling $\ell = 0$ proportionally to all positive values $\ell \in [n]$; see [9] for a discussion of this "resampling" strategy.

Algorithm 1: A Family of $(\mu+\lambda)$ Genetic Algorithms

1 **Input:** Population sizes μ, λ, crossover probability p_c, mutation rate p_m, cutoff time B;
2 **Initialization: for** $i = 1, \ldots, \mu$ **do**
3 Sample $x^{(i)} \in \{0,1\}^n$ u.a.r.;
4 Evaluate $f(x^{(i)})$
5 Set $P = \{x^{(1)}, x^{(2)}, ..., x^{(\mu)}\}$;
6 **Optimization: for** $t = 1, 2, 3, \ldots$ **do**
7 $P' \leftarrow \emptyset$;
8 **for** $i = 1, \ldots, \lambda$ **do**
9 Sample $r \in [0,1]$ u.a.r.;
10 **if** $r \leq p_c$ **then**
11 Select two individuals $x, y \in P$ u.a.r. (with replacement);
12 $z^{(i)} \leftarrow$ Crossover(x, y);
13 **if** $z^{(i)} \notin \{x, y\}$ **then** evaluate $f(z^{(i)})$
14 **else** Infer $f(z^{(i)})$ from parent;
15 **else**
16 Select an individual $x \in P$ u.a.r.;
17 $z^{(i)} \leftarrow$ Mutation(x);
18 **if** $z^{(i)} \neq x$ **then** evaluate $f(z^{(i)})$
19 **else** Infer $f(z^{(i)})$ from parent;
20 $P' \leftarrow P' \cup \{z^{(i)}\}$;
21 P is updated by the best μ points in $P \cup P'$ (ties broken u.a.r.);
22 **Terminate** the algorithm if the optimum is found or if the number of evaluations (at line 13 and line 18) exceeds the given cutoff time.

B. The IOHprofiler Problem Suite PBO

The PBO problem suite suggested in [16], [55] and available in the IOHprofiler benchmarking environment [15] contains 25 pseudo-Boolean optimization problems. The definitions of these problems is available in the supplementary material. We provide here only a short summary: **F1** is OneMax, **F2** is LeadingOnes, **F3** is a linear function with harmonic weights, **F4-10** and **F11-17** are so-called W-model extensions of F1 and F2, respectively,[1] **F18** is the Low Autocorrelation Binary Sequences (LABS) problem, **F19-21** are Ising Model problems, **F22** is an instance of the Maximum Independent Vertex Set (MIVS) problem, **F23** is the N-Queens problem (NQP), **F24** is a Concatenated Trap (CT) problem, and **F25** is a randomly chosen NK Landscape instance.

Our algorithms are unbiased in the sense of Lehre and Witt [29], i.e., they are invariant under reordering of the bit string and under XOR of the solution candidates with arbitrary strings. They are also comparison-based, making them invariant with respect to strictly motononic transformations of the function values. We can therefore restrict our experiments to a single problem instance, and do not make use of IOHprofiler's option to transform the instances into problems with isomorphic fitness landscape and/or translated fitness values. More precisely, we tune on the first instance of each problem in the PBO suite.

[1]The W-model transformations were suggested in [52], [51] and later extended in [16]. In a nutshell, these transformations introduce dummy variables that do not impact the quality of a solution ("dummy" transformation), perturb the fitness values ("ruggedness"), reduce the string size by partitioning the bit string and applying a majority vote within each substring ("neutrality"), or map substrings to different ones ("epistasis").

C. Algorithm Performance Measures

As mentioned in the introduction, we use two main performance criteria: *expected running time (ERT)* and the *area under the curve representing the empirical cumulative distribution function (AUC)*. While the ERT concerns only the hitting time for a single "final" target, AUC considers a set of targets. The formal definitions are as follows.

Definition 2.1 (ERT: Expected Running Time): Given a target value ϕ for a problem P, the ERT of algorithm A for hitting ϕ is

$$\mathrm{ERT}(A, P, \phi) = \frac{\sum_{i=1}^{r} \min\{t_i(A, P, \phi), B\}}{\sum_{i=1}^{r} \mathbb{1}\{t_i(A, P, \phi) < \infty\}}\,, \qquad (1)$$

where r is the number of independent runs of A, B is the cutoff time (i.e., the maximal number of solution candidates that algorithm A may evaluate in one run), $t_i(A, P, \phi) \in \mathbb{N} \cup \{\infty\}$ is the running time (for finite values, the running time is the number of function evaluations that the i-th run of A on problem P uses to hit the target ϕ and $t_i(A, P, \phi) = \infty$ is used if none of the solutions is better than ϕ), and $\mathbb{1}(\mathcal{E})$ is the indicator function returning 1 if event $\mathcal{E}$ is true and 0, otherwise. If the algorithm hits the target ϕ in all r runs, the ERT is equal to the average hitting time (AHT).

Definition 2.2 (ECDF: empirical cumulative distribution function of the running time): Given a set of targets $\Phi = \{\phi_i \in \mathbb{R} \mid i \in [m]\}$ for a real-valued problem P and a set of budgets $T = \{t_j \in [B] \mid j \in [z]\}$ for an algorithm A, the ECDF value of A at budget t_j is the fraction of (run, target)-pairs (s, ϕ_i) that satisfy that in run s of algorithm A at least one of the first t_j evaluated solutions has fitness at least as good as ϕ_i.

Definition 2.3 (AUC: area under the ECDF running time curve): Given a set of targets $\Phi = \{\phi_i \in \mathbb{R} \mid i \in [m]\}$ and a set of budgets $T = \{t_j \in [B] \mid j \in [z]\}$, the AUC $\in [0,1]$ (normalized over B) of algorithm A on problem P is the area under the ECDF curve of the running time over multiple targets. For maximization, it reads,

$$\mathrm{AUC}(A, P, \Phi, T) = \frac{\sum_{h=1}^{r} \sum_{i=1}^{m} \sum_{j=1}^{z} \mathbb{1}\{\phi_h(A, P, t_j) \geq \phi_i\}}{rmz}\,,$$

where r is the number of independent runs of A and $\phi_h(A, P, t)$ denotes the value of the best solution that A evaluated within its first t evaluations of the run h.

III. Algorithm Configuration for Benchmarking

To explore promising configurations of the $(\mu+\lambda)$ GA, we study its associated algorithm configuration problem:

Definition 3.1 (Algorithm Configuration (AC) Problem): Given a problem P, a parametrized algorithm A with parameter space Θ, and a cost metric $c : \Theta \times P \to \mathcal{R}$ that is subject

TABLE I
Target values used to compute the ERT metric on each problem.

funcId	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Target	100	100	5 050	50	90	33	100	51	100	100	50	90	33	7	51	100	100	4.216	98	180	260	42	9	17.196	-0.297

to minimization, the objective of the AC problem is to find a configuration $\theta^* \in \arg\min_{\theta \in \Theta} c(\theta, P)$.
Note that this definition is adjusted to our setting in which we tune for a single problem instance. It therefore ignores aggregation over multiple problem instances, hence the deviation from other common definitions as, for example, available in [17], [40].

A. Algorithm Configurators Utilized

We apply three AC methods to configure the $(\mu + \lambda)$ GA: Irace [31], a mixed-integer parallel efficient global optimization (MIP-EGO [49]), and the mixed-integer evolution strategy (MIES [30]), which we briefly describe in the following paragraphs. All configurators work with a user-defined *configuration budget*, which is the maximal number of algorithm runs that the AC method is allowed to perform before recommending its final outcome.

Irace [31] is a so-called iterated racing method designed for hyperparameter optimization. The main steps of Irace are (1) sampling hyperparameter values from distributions that evolve over time, (2) evaluating the configurations across a set of instances, (3) applying a racing method (i.e., statistical tests) to select the configurations that are not further evaluated in the subsequent rounds, and (4) updating the sampling distributions. At the end of the configuration process, one or several *elite* configurations are returned to the user, along with their performance observed during the configuration. The distributions from which the hyperparameters are sampled are independent of each other, unless the user specifies constraints and/or other requirements.

Efficient global optimization (EGO), also known as Bayesian optimization, is designed to solve costly-to-evaluate global optimization problems. For our configuration problem, we use an EGO-variant called mixed-integer parallel EGO (MIP-EGO [49]) capable of handling mixed-integer search space. EGO starts with an initial set of solution candidates $\{\theta_1, \theta_2, ...\}$ and evaluating their fitness $\{c_1, c_2, ...\}$. From these observations, EGO learns a predictive distribution of the fitness value for each unseen configuration using stochastic models, e.g., random forests or Gaussian processes. Aiming at balancing the trade-off between the accuracy and uncertainty of this predictive distribution, EGO uses a so-called *acquisition function* to decide which solution candidates to sample next. Common acquisition functions are *expected improvement* and *probability of improvement*; see [38], [19] for an overview. For this work, we use the moment-generating function of improvement (MGFI) [49], which is defined as the weighted combination of all moments of the predictive distribution. For the weights, we used the setting recommended in [48]. To learn the predictive distribution, we choose a random forest model as it deals with the mixed-integer/categorical search space more naturally than Gaussian processes.

The mixed-integer evolution strategy (MIES) uses principles from evolution strategies for handling continuous, discrete, and nominal parameters by using self-adaptive mutation operators for all three parameter types [30]. MIES starts with a randomly initialized parent population of size μ. In each iteration it generates λ offspring, by recombining two randomly selected parents and then mutating the solution resulting from their recombination, after which (μ, λ) selection is applied to the offspring population, i.e., the μ best of the λ offspring form the parent population of the next iteration.

B. Experimental Setup

Each AC method is granted a budget of $5\,000$ *target runs*, where each target run corresponds to ten independent runs of the $(\mu + \lambda)$ GA using the configuration that the AC method wishes to evaluate. As previously mentioned, we use ERT and AUC as performance metric, and these values are computed from the ten independent runs.

Irace requires a set of instances for the tuning process. We imitate these *instances* by the independent runs of the (stochastic) solvers. This is in line with previous approaches, suggested, for example, in [11]. MIP-EGO starts with 10 initial candidates by the default setting of the package [49]. We use a $(4, 28)$-MIES, following the parameter settings suggested in [30].

To obtain a useful baseline against which we can compare other algorithms, we configure the $(\mu + \lambda)$ GA on each PBO problem separately. We focus on the problems in dimension $n = 100$.

Once the optimized parameter settings are obtained from the algorithm configuration, we perform 100 *independent validation runs* with the suggested parameter settings. This is to have a fair comparison, as the number of algorithm evaluations may differ quite drastically between the configurators and between different runs of the same configurator. All performance comparisons discussed in this paper are made with respect to these validation runs.

For evaluating GA candidates during the configuration process, the ERT values are calculated with respect to the targets listed in Tab. I and to the cutoff time of $50\,000$ function evaluations. This is half the budget used in [55], but, according to the results presented there, our cutoff time is still larger than the number of function evaluations needed to hit the corresponding targets – except for F18, which is a very challenging problem.

For the AUC, the set of targets are 100 values, equally spaced in the interval $[\phi_{\min}, \phi_{\max}]$, where $\phi_{\max}$ is equal to the ERT targets listed in Tab. I and $\phi_{\min}$ is equal to 0 except for the following functions: $\phi_{\min} = -19\,590$ for function F22,

$\phi_{\min} = -3\,950\,000$ for function F23, and $\phi_{\min} = -1$ for function F25. We evaluate the AUC at each budget, i.e., in the notation of Definition 2.3 we use $T = [50\,000]$.

The configuration space of the AC problem is $\Theta = \{\mu, \lambda, p_c, p_m\}$, where $\mu \in [100]$ is the parent population size, $\lambda \in [100]$ is the offspring population size, $p_c \in [0, 1]$ is the crossover probability, and $p_m \in [0.005, 0.5]$ is the mutation rate. A positive crossover probability $p_c > 0$ requires $\mu > 1$; the configuration is considered infeasible otherwise. The results for the grid search are based on our work [55], where we used $\mu \in \{10, 50, 100\}$, $\lambda \in \{1, \mu/2, \mu\}$, $p_c \in \{0, 0.5\}$, and $p_m = 0.01$. Note that this is a considerably smaller search space, whose full enumeration requires only 18 different configurations, which is much less than the budget allocated to the automated configuration techniques. We will nevertheless observe that for some problems none of the automated configurators could find hyperparameter settings that are equally good as those provided by this small grid search.

C. Results Obtained by Automated Configuration

The $(1 + 1)$ EA with $p_m = 1/n$ has shown competitive results in [16] for the PBO problems, so we use it as the baseline against which we compare the GAs obtained by the configurators. Note that this algorithm is part of the GA framework in Alg. 1 and could therefore be identified by the configuration methods.

Tab. II lists the configurations of the $(\mu+\lambda)$ GA obtained by the grid search and by the three AC methods. Tab. III compares the performance of these configurations, by listing the ERT and AUC values of the $(1 + 1)$ EA and the corresponding relative deviations of the configured GAs. More precisely, the ERT and AUC values of the AC methods result from using ERT and AUC as the cost metric, respectively. The relative improvement of ERT is computed as ($\text{ERT}_{(1+1)\text{ EA}}$ - ERT) / $\text{ERT}_{(1+1)\text{ EA}}$, and the relative improvement of AUC is computed as (AUC - $\text{AUC}_{(1+1)\text{ EA}}$) / $\text{AUC}_{(1+1)\text{ EA}}$.

1) ERT Results: For the ONEMAX-based problems F1 and F4-F6, configurations of the $(\mu + \lambda)$ GA using crossover outperform the mutation-only GA with $\mu \geq 10$ [55]. However, according to the values in Tab. III, the $(1+1)$ EA outperforms the configurations with $p_c > 0$, relatively large μ, and also relatively large λ. This observation matches our expectation because our previous study has shown that the $(1 + 1)$ EA is efficient on ONEMAX. Meanwhile, we observe an interesting configuration with $p_m < 0.01$, $\mu = \lambda = 2$, and $p_c > 0$ that achieves competitive ERT values against the $(1 + 1)$ EA for F1. This configuration ties well with the results on different $(\mu + 1)$ GAs that were shown to outperform the $(1 + 1)$ EA (and any mutation-based algorithm, in fact) in a series of recent works [9], [10], [41].

On F4, F6, F7, F13, F15-F17, and F19-22, none of the configurations returned by the AC methods was able to outperform the $(1 + 1)$ EA, whereas on F9-10, F14, F18, F24, all AC methods find configurations that perform much better than the $(1 + 1)$ EA.

On LEADINGONES, a slightly better result compared to the ERT value $5\,574$ of the $(1 + 1)$ EA is found by MIES. This result is quite sensitive with respect to the mutation rate. When changing it from the MIES-suggestion of $p_m = 0.005$ to $p_m = 0.01$ we obtain an ERT value of $5\,829$. MIES also obtains an improvement on F11, which corresponds to a $(2 + 2)$ GA with $p_m = 0.0245$. On F14, we already observed in [55] that mutation-only GAs with $p_m = 1/n$ are inferior to other GA configurations with a larger offspring population size and higher mutation rate. As expected, all three methods easily suggest configurations that outperform the $(1 + 1)$ EA by a great margin.

On F18 and F24, all configurators unanimously suggest fairly small values for the crossover probabilities. For F25, however, GAs with $p_c > 0.8$ show the (by far) best performance.

2) AUC results: Since we evaluate the AUC at each budget $[50\,000]$, the AUC values tend to be very close to 1, especially for the GAs that require much fewer than $50\,000$ evaluations to find an optimal solution.

On 13 out of the 25 problems, none of the configuration methods is able to identify a hyperparameter setting that yields better AUC values than those of the $(1 + 1)$ EA. In some, but clearly not all of these cases, the achieved AUC values are not much worse than those of the $(1 + 1)$ EA. The largest improvements are obtained on functions F10, F14, F17, F18, F24, and F7 (in this order). In most of these cases all four configuration techniques found improvements over the $(1 + 1)$ EA except for F17, where Irace is the only method finding an improvement and except for F7, where all three automated configuration techniques find an improvement but not the grid search (the inverse is true on F23 but the advantage of the grid search is fairly small, not statistically significant, and the algorithm a mutation-only (10+1) GA with $p_m = 1/n$, which is very close to the $(1 + 1)$ EA).

We next discuss the results for the functions for which large improvements over the $(1 + 1)$ EA were obtained.

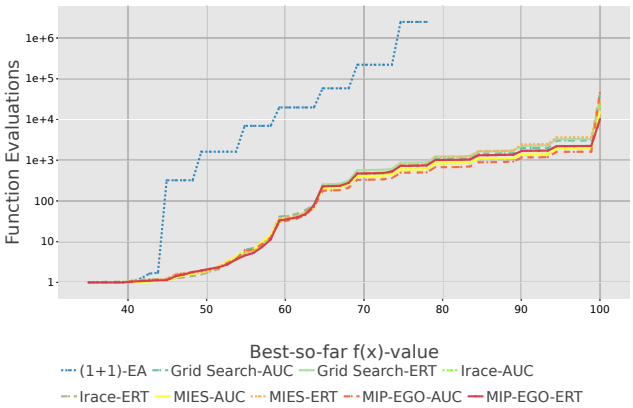


Fig. 1. Fixed-target ERT values of the configurations suggested for F10. The suffix "-ERT/AUC" indicates which cost metric was used during the tuning.

On **F10**, large crossover probabilities seem beneficial; all three automated configuration techniques return settings with $p_c > 0.98$. The mutation rate seems to have less impact on the results, and the suggested settings vary from $8.5/n$ (Irace) to $41.3/n$ (MIES, which achieves the best AUC value).

TABLE II
Configurations of the $(\mu + \lambda)$ GA obtained by grid search, Irace, MIP-EGO, and MIES. C indicates the cost metric used by the configurators. Results for maximizing AUC are obtained independently from those obtained for minimizing ERT.

F	C	Grid Search				Irace				MIP-EGO				MIES			
		μ	λ	p_m	p_c	μ	λ	p_m	p_c	μ	λ	p_m	p_c	μ	λ	p_m	p_c
1	ERT	10	1	0.01	0.5	15	5	0.007	0.782	3	4	0.024	0.774	2	2	0.006	0.778
	AUC	10	1	0.01	0.5	1	1	0.006	0	3	11	0.006	0.208	2	2	0.005	0.521
2	ERT	10	1	0.01	0.5	1	39	0.006	0	2	9	0.006	0.008	2	3	0.005	0.026
	AUC	10	1	0.01	0	1	34	0.013	0	22	5	0.009	0.461	2	1	0.008	0.005
3	ERT	10	1	0.01	0.5	4	32	0.006	0.867	3	24	0.009	0.391	2	3	0.005	0.357
	AUC	10	1	0.01	0.5	4	23	0.011	0.550	2	3	0.007	0.511	2	1	0.006	0.203
4	ERT	10	1	0.01	0.5	1	1	0.013	0	2	3	0.022	0.655	2	2	0.024	0.198
	AUC	10	1	0.01	0.5	1	1	0.013	0	12	9	0.052	0.668	2	1	0.055	0.614
5	ERT	10	5	0.01	0.5	3	17	0.005	0.415	4	2	0.026	0.294	2	3	0.006	0.321
	AUC	10	1	0.01	0.5	8	8	0.012	0.882	11	20	0.008	0.318	2	2	0.006	0.358
6	ERT	10	10	0.01	0	30	23	0.104	0.849	3	11	0.031	0.033	24	12	0.135	0.908
	AUC	10	10	0.01	0.5	1	1	0.033	0	2	1	0.032	0.424	2	3	0.059	0.236
7	ERT	50	50	0.01	0	45	22	0.460	0.887	95	46	0.018	0.971	54	18	0.268	0.907
	AUC	10	10	0.01	0	1	58	0.028	0	2	20	0.016	0.082	23	27	0.025	0.672
8	ERT	10	10	0.01	0.5	25	48	0.014	0.643	78	7	0.087	0.991	5	9	0.009	0.507
	AUC	10	10	0.01	0.5	3	6	0.015	0.441	11	12	0.007	0.499	21	25	0.011	0.800
9	ERT	100	50	0.01	0.5	64	90	0.056	0.938	76	10	0.488	0.977	54	32	0.013	0.643
	AUC	50	25	0.01	0.5	46	40	0.029	0.815	6	7	0.020	0.158	18	25	0.025	0.505
10	ERT	100	50	0.01	0.5	95	69	0.325	0.978	87	84	0.478	0.962	79	17	0.056	0.320
	AUC	100	1	0.01	0.5	99	13	0.085	0.982	76	16	0.321	0.986	92	32	0.413	0.989
11	ERT	10	1	0.01	0	1	72	0.031	0	4	57	0.031	0.004	2	2	0.025	0.000
	AUC	10	1	0.01	0	1	13	0.035	0	2	1	0.040	0.003	2	1	0.042	0.016
12	ERT	10	1	0.01	0	1	32	0.006	0	4	1	0.012	0.016	2	1	0.005	0.000
	AUC	10	1	0.01	0	1	29	0.010	0	2	1	0.017	0.016	2	1	0.014	0.005
13	ERT	10	10	0.01	0	1	10	0.044	0	10	19	0.032	0.282	2	5	0.032	0.008
	AUC	10	10	0.01	0	1	1	0.044	0	2	4	0.038	0.023	3	8	0.047	0.036
14	ERT	100	100	0.01	0.5	1	60	0.343	0	57	34	0.077	0.988	57	23	0.028	0.012
	AUC	100	50	0.01	0	48	83	0.409	0.812	5	80	0.456	0.287	54	9	0.481	0.480
15	ERT	10	10	0.01	0	5	64	0.014	0.042	5	15	0.008	0.018	6	67	0.011	0.023
	AUC	10	10	0.01	0	21	71	0.020	0.165	3	9	0.016	0.078	2	3	0.011	0.024
16	ERT	10	10	0.01	0	6	34	0.011	0.025	2	2	0.006	0.069	2	4	0.005	0.002
	AUC	10	10	0.01	0	1	1	0.010	0.000	22	66	0.009	0.138	2	3	0.007	0.001
17	ERT	10	10	0.01	0.5	60	57	0.133	0.512	79	62	0.345	0.430	2	3	0.006	0.003
	AUC	10	10	0.01	0	1	1	0.019	0	2	5	0.012	0.007	2	4	0.019	0.034
18	ERT	100	50	0.01	0	50	44	0.006	0.010	13	77	0.019	0.002	26	21	0.006	0.002
	AUC	10	5	0.01	0	20	10	0.005	0.031	2	22	0.006	0.011	23	83	0.006	0.010
19	ERT	10	10	0.01	0.5	98	94	0.174	0.324	2	5	0.014	0.286	2	12	0.006	0.061
	AUC	50	50	0.01	0	1	5	0.005	0	7	18	0.014	0.834	2	7	0.005	0.133
20	ERT	50	50	0.01	0	1	32	0.008	0	6	53	0.007	0.012	5	66	0.005	0.033
	AUC	10	10	0.01	0	1	30	0.013	0	5	7	0.011	0.483	6	75	0.006	0.095
21	ERT	10	5	0.01	0	1	86	0.014	0	7	52	0.009	0.144	5	66	0.009	0.077
	AUC	10	10	0.01	0	4	33	0.006	0.412	7	9	0.006	0.833	2	3	0.005	0.219
22	ERT	100	100	0.01	0	16	48	0.025	0.783	18	12	0.019	0.820	2	3	0.005	0.270
	AUC	10	10	0.01	0.5	1	1	0.020	0	2	3	0.007	0.250	2	2	0.010	0.088
23	ERT	10	1	0.01	0	62	39	0.006	0.252	10	40	0.013	0.010	5	1	0.006	0.006
	AUC	10	5	0.01	0	29	25	0.005	0.016	4	4	0.006	0.170	8	16	0.005	0.038
24	ERT	100	50	0.01	0	53	85	0.025	0.004	58	45	0.031	0.077	21	75	0.060	0.014
	AUC	10	5	0.01	0	7	86	0.026	0.776	8	3	0.032	0.277	8	33	0.045	0.273
25	ERT	50	1	0.01	0	13	52	0.296	0.170	68	91	0.024	0.801	98	40	0.009	0.941
	AUC	10	10	0.01	0	26	53	0.028	0.790	4	3	0.023	0.107	9	1	0.021	0.535

Interestingly, also two of the three configurations tuned for minimizing ERT have large crossover probabilities, with the exception of the one returned by MIES ($p_c = 0.32$). The performances of all these configurations are very similar, as we can see in Fig. 1.

On **F14**, the best improvement is obtained by a mutation-only (100+50) GA using $p_m = 1/n$, closely followed by the Irace result, which is a (48+83) GA using crossover probability $p_c = 0.812$ and $p_m \approx 41/n$. We cannot observe any clear pattern in the results, and the four suggested configurations all differ quite a bit.

For **F17**, as mentioned, only Irace finds a better configuration, which is also a (1+1) EA, but using a slightly larger mutation rate of $p_m = 1.9/n$ instead of $1/n$.

For **F18**, all configurators return settings with small crossover probability $p_c < 0.031$ and small mutation rate $p_m < 0.01$, which indicates that local search methods may be more suitable for this problem than globally searching GAs. Interestingly, the performance of randomized local search is not very good (see [16] for details), which suggests that a positive probability for escaping local optima via small jumps in the search space or via crossover are needed to be efficient on this problem.

For **F24**, no clear pattern can be observed in the suggested configurations, and also the crossover probabilities differ widely, from 0 for the grid search, values around 0.27 for

TABLE III

Absolute ERT and AUC values for the $(1+1)$ EA and relative improvement of ERT and AUC for the configurations suggested by the four configuration methods, in comparison against the $(1+1)$ EA values. Compared to the ERT and AUC values of the $(1+1)$ EA on each problem (indicated by "EA"), the relative improvement obtained from the automated configuration are shown for each AC method, where both measures are calculated from hitting times of 100 validation runs for $(1+1)$ EA and AC methods. We also indicate the statistical significance in the empirical distributions of the hitting time (**** for $p < 0.001$, *** for $p < 0.01$, ** for < 0.01, and * for $p < 0.05$) for each pair of the AC result and that of the $(1+1)$ EA. The Mann–Whitney U test is applied with the Benjamini and Hochberg method for all 120 pairwise comparisons to control the false discovery rate. Runs are cut off at 50 000 evaluations if it does not hit the final target. The significant comparisons are colour-coded with respect to the relative improvement, where a darker colour signifies a more considerable improvement.

F	Relative improvement of ERT using ERT as the cost metric					Relative improvement of AUC using AUC as the cost metric				
	EA	**GridSearch**	**Irace**	**MIP-EGO**	**MIES**	**EA**	**GridSearch**	**Irace**	**MIP-EGO**	**MIES**
1	665	-0.54****	-0.47****	-0.34****	0.01	0.9987	-0.11****	-0.49**	-1.21****	-0.70
2	5574	-0.64****	-0.12****	-0.03	0.05	0.9514	-1.37****	-0.51****	-4.64****	0.25
3	694	-0.62****	-0.53****	-0.78****	0.03	0.9989	-0.11****	-1.12****	-0.57	-0.50
4	344	-0.87****	-0.04	-0.30****	-0.11*	0.999	-0.06****	-0.66	-1.20****	-0.73****
5	598	-0.50****	-0.41****	-0.55****	0.04	0.9987	-0.11****	-1.17****	-1.90****	-0.68**
6	271	-1.96****	-0.91****	-0.58****	-7.66****	0.9993	-0.11****	-0.43	-1.32****	-0.54***
7	Inf	—	—	—	Inf	0.8995	-0.33	1.69	0.70	0.24
8	7926	0.78****	0.77****	-0.01****	0.86****	0.9938	0.25****	-0.37****	-0.65****	-0.75****
9	22670	0.85****	0.85****	0.15****	0.77****	0.9877	0.60****	0.42****	0.50****	0.41****
10	Inf	Inf****	Inf****	Inf****	Inf****	0.6362	54.86****	55.60****	55.50****	55.73****
11	2071	-0.29****	-0.35****	-0.45****	0.12**	0.9807	-0.36****	-1.05	-0.89	-0.93
12	4691	-0.45****	-0.11***	-0.12***	0.06*	0.9601	-1.51****	-0.64***	-0.15*	0.13
13	997	-0.83****	-0.14*	-0.70****	-0.05	0.9902	-0.57****	-0.8	-0.99	-1.26****
14	8171	0.98****	0.99****	0.98****	0.96****	0.596	47.69****	47.29****	36.75****	41.05****
15	6668	-1.46****	-1.17****	-0.50****	-1.23****	0.9404	-7.04****	-8.98****	-2.05****	-0.88****
16	9520	-2.05****	-0.97****	-0.43****	-0.23****	0.9174	-17.54****	0.41	-12.97****	-1.78****
17	45964	-Inf****	-Inf****	-Inf****	-0.96****	0.6698	-48.43****	2.45***	-14.40****	-7.93****
18	130863	0.85****	0.80****	0.69****	0.85****	0.9432	1.86***	1.32	0.06	1.26****
19	9467	-525.05****	-Inf****	-1.72****	-0.72****	0.9899	-6.19****	0.00	-4.96****	-0.37**
20	1460	-9.77****	-0.83****	-2.16****	-2.47****	0.9956	-1.26****	-0.20****	-0.25****	-0.68****
21	948	-12.75****	-3.37****	-3.66****	-3.95****	0.9965	-0.40****	-0.56****	-0.42****	-0.04****
22	3366	-2.60****	-0.11****	-0.57****	-0.15	0.9993	-0.02****	-0.05****	-0.06	-0.02****
23	3066	0.08***	-1.00****	-0.68****	-1.03*	0.9993	0.01**	-0.12****	-0.40***	-0.23****
24	Inf	Inf****	Inf****	Inf****	Inf****	0.9545	1.11**	1.42****	0.90	1.71****
25	48946	0.22*	-Inf****	0.66****	0.56****	0.6953	0.07**	-0.00****	0.04	-0.03**
#improvements		8	6	6	13	-	8	9	7	8

MIP-EGO and MIES, to 0.7 for Irace.

On **F7**, the configuration obtained by Irace (which achieves the best improvement) is a $(1+\lambda)$ EA with $p_m \approx 3/n$, the one obtained by MIP-EGO is a $(2+20)$ GA with small crossover probability $p_c = 0.082$, and the one obtained by MIES is a (23+27) GA with crossover probability $p_c = 0.672$. These results indicate that the fraction of configurations achieving better AUC value than the $(1+1)$ EA may be fairly large.

On the LEADINGONES problem F2, the $(2+1)$ GA with $p_m = 0.008$ and $p_c = 0.005$, found by MIES, yields a (small) improvement over the $(1+1)$ EA, whereas the configurations found by the other methods perform worse.

All in all, we find that on several problems the suggested configurations differ widely, far more than we would have expected and this across all four parameters. Analyzing the landscape of the AC problem suggests itself as an interesting follow-up study, e.g., using similar approaches to those suggested in [36], [43].

D. Discussions on the Configurators' Performance

We now compare the performance of the three automated AC methods. The last row of Tab. III summarizes for how many settings each method was able to find configurations that outperform the $(1+1)$ EA. These numbers are rather balanced between the different methods, with the notable exception of the minimizing ERT objective, for which MIES suggested 13 improvements, compared to 6-8 improvements found by the other methods. MIES also suggested the best configurations in most of the cases, but, for several of these, the improvements over the $(1+1)$ EA are rather minor. This shows that barely counting such cases does not do justice to the complex behavior observed in Tab. III, from which we cannot derive a clear winning configurator. We can nevertheless make a few observations.

1) Handling conditional parameter spaces: We easily see from Tab. II that Irace is the only method that obtains mutation-only GAs, and in all of these cases it returns a $(1+\lambda)$ EA. We recall that setting $\mu = 1$ requires to set $p_c = 0$; the configuration is infeasible otherwise. This advantage of Irace lies in its handling of conditional parameters: Irace samples non-conditional parameters first, and samples conditional parameters only if the condition is satisfied. MIP-EGO and MIES, in contrast, sample parameter values from independent distributions and give penalties to infeasible settings. With this strategy, the two methods can avoid infeasible candidates, but the probability of sampling feasible conditional candidates may be too small. For example, MIP-EGO can find a configuration with $\mu = 2$ and $p_c = 0.0065$ on F16 in Tab. II, but it cannot obtain the competitive configuration of $(1+\lambda)$ mutation-only GA because the probability of sampling $\mu = 1$ and $p_c = 0$ simultaneously is too small.

2) Impact of the cost metric: We have already observed that MIES obtains better configurations for more problems when using ERT as the cost metric. For AUC, in contrast, Irace

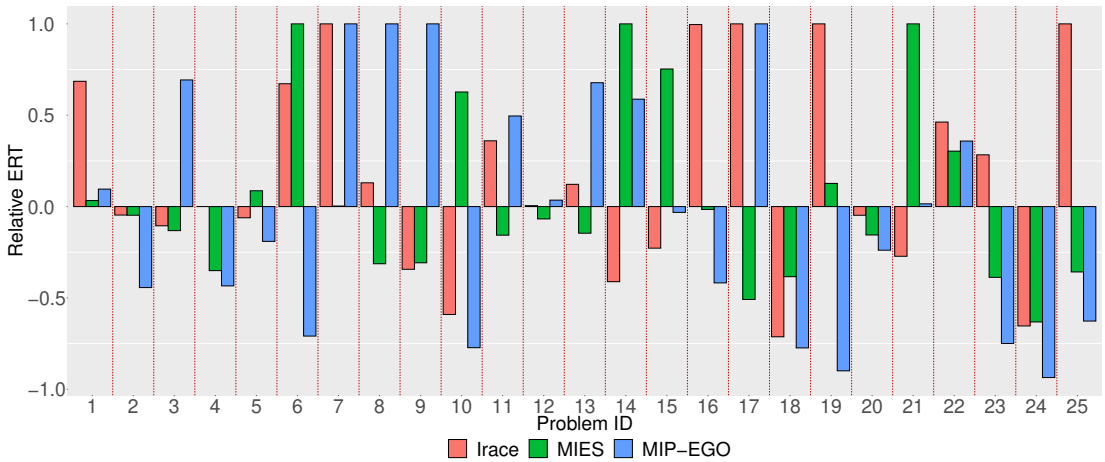


Fig. 2. Relative ERT values of the GAs obtained by Irace, MIES, and MIP-EGO using ERT as the cost metric compared to the ERT of the GAs obtained by the same method when using AUC as the cost metric during the configuration process. Plotted values are $(\text{ERT}_{\text{using ERT}}\text{-ERT}_{\text{using AUC}})/\text{ERT}_{\text{using AUC}}$, capped at -1 and 1. Positive values therefore indicate that configurator obtains better results when configuring AUC.

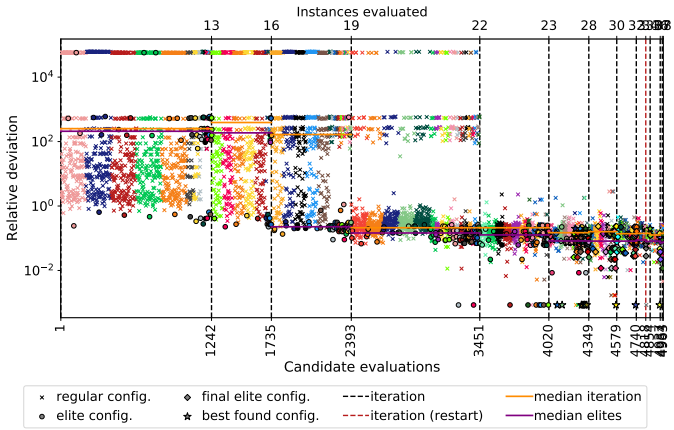


Fig. 3. The relative deviation from the best-known ERT value of the GAs obtained during the configuration process of Irace for tuning the $(\mu+\lambda)$ GA for ONEMAX in dimension $n = 100$, with the objective to minimize the ERT for the optimum $f(x) = 100$. The maximal number of configurations that can be tested by Irace is set to 5 000. The figure is produced by the acviz tool, and illustrations for the details of plot representation can be found in Sec. 3 of [13].

finds more configurations that improve over the $(1+1)$ EA, which can be explained as follows. In the first few iterations, AUC is able to differentiate the performance of two poor configurations if both fail to find the final target, whereas the ERT value will be infinite and thereby incomparable in this case. Hence, using AUC as the cost metric, Irace could learn to avoid evaluating those poor configurations in the following iterations. It is worth noting that such an observation is also supported by a case study of Irace [35], in which the authors discovered that Irace would spend too much time on poor configurations if the mean running time is taken as the cost metric. As a solution, the adaptive capping strategy [25] is introduced to Irace in this work. Interestingly, this discussion connotes that the AUC metric realizes a similar effect as with adaptive capping for minimizing the running time of an optimization algorithm. This behavior also indicates that the choice of the cost metric might be a factor to consider when choosing which AC technique to apply.

For an algorithm that cannot hit the target in all runs, the variance of its ERT values can be high due to the uncertain success rate. Besides, ERT cannot distinguish algorithms that cannot hit the target in any runs, even though their performance may differ in terms of results for other targets. This shortcoming is mitigated when tuning for large AUC, since this performance metric also takes into account the hitting times for easier targets.

Fig. 3 plots the relative deviations from the best-known ERT value of the configurations obtained during one run of Irace when using ERT as the cost metric. We observe that many configurations show large relative deviation values, which stem from GAs that cannot hit the optimum within the given budget. These configurations do not provide much useful information, since they all look equally bad for the configurator.

E. The Choice of the Cost Metric

We now evaluate how well configurations that are obtained by tuning for the AUC cost metric perform in terms of ERT. Fig. 2 summarizes these result, by plotting the relative advantage of the configurations tuned for AUC, compared to those that were explicitly tuned for ERT. More precisely, we plot $(\text{ERT}_{\text{using ERT}}\text{-ERT}_{\text{using AUC}})/\text{ERT}_{\text{using AUC}}$, so that positive values indicate that tuning for AUC gave better ERT values than the configurations obtained when tuning for ERT. We see that this is the case for 13, 12, and 9 out of the 25 problems when using Irace, MIP-EGO, and MIES, respectively.

We now zoom into the results obtained by Irace. We abbreviate by "Irace-ERT" ("Irace-AUC") the configurations obtained when using ERT (AUC) as the cost metric. For the problems on which the ERT of Irace-AUC was worse than that of Irace-ERT, we plot in Fig. 4 violin plots for the running times of the 100 validation runs. We observe that F15 is the only problem where Irace-ERT significantly outperforms Irace-AUC. On F2-3, F5, and F20, we observe that the result of most runs of Irace-AUC and Irace-ERT are

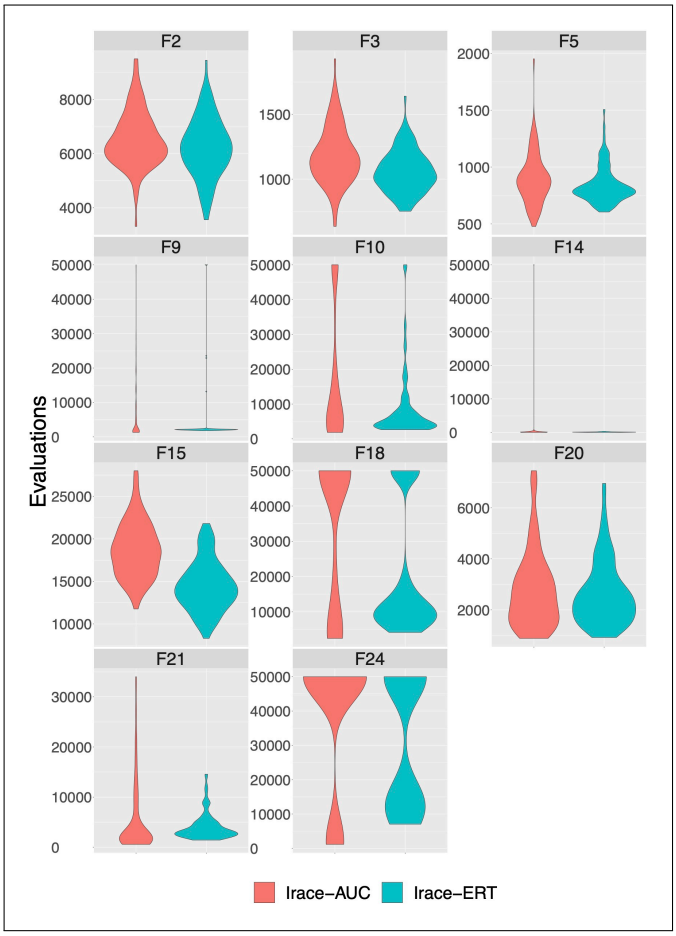


Fig. 4. Violin plots of first hitting times for the configurations found by Irace when tuning for ERT and AUC, respectively. Only showing results for problems on which **Irace-ERT outperforms Irace-AUC**. Results are from the 100 independent validation runs. Targets are listed in Tab. I, and the configurations of the GAs can be found in Tab. II. For each run, values are capped at the budget 50 000 if the algorithm cannot find the target.

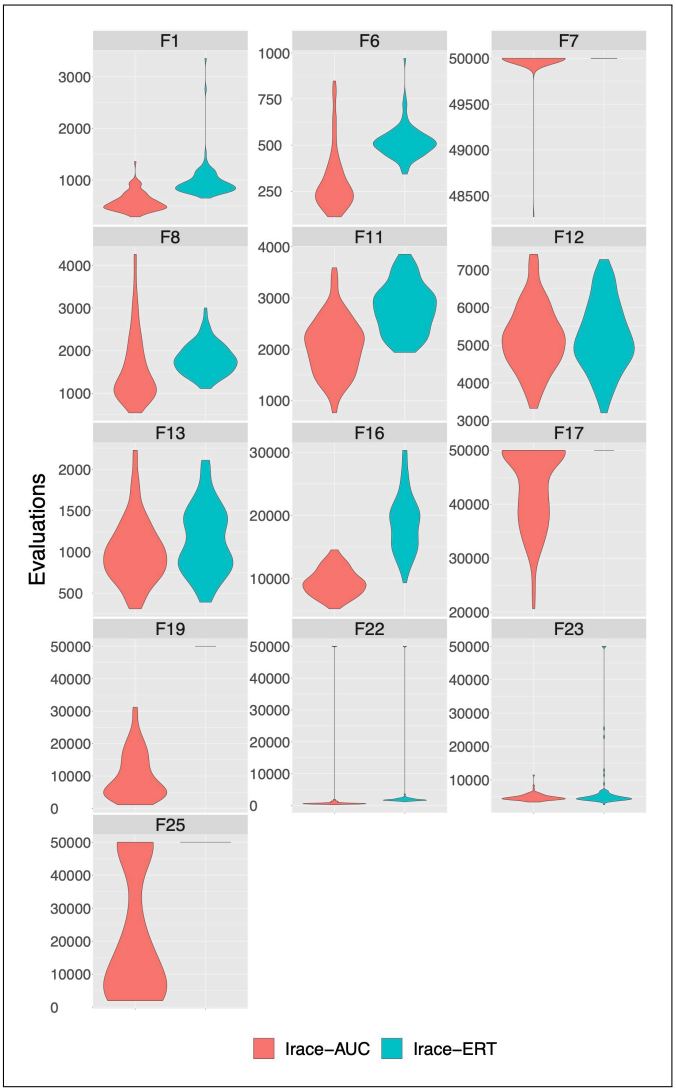


Fig. 5. Violin plots of first hitting times for the configurations found by Irace when tuning for ERT and AUC, respectively, for problems on which **Irace-AUC outperforms Irace-ERT**. Results are from the 100 independent validation runs. Targets are listed in Tab. I, and the configurations of the GAs can be found in Tab. II. For each run, values are capped at the budget 50 000 if the algorithm cannot find the target.

close, but the variances of the results of Irace-AUC are higher than for Irace-ERT. On the remaining problems, we observe high variances for the result of both Irace-ERT and Irace-AUC. Irace-ERT finds the configurations with fewer *unsuccessful* runs, which makes sense because the number of *unsuccessful* runs significantly affects the ERT value. However, AUC does not only consider the evaluations needed to hit the final target, so we observe more *unsuccessful* runs and *competitive* partial runs for Irace-AUC, i.e., in cases of F21 and F24.

Although Irace-AUC does not obtain better ERT values than Irace-ERT, it can still provide valuable insights concerning the resulting configurations and performance profiles. Fig. 6 plots the fixed-target ERT values of the GAs obtained by Irace-ERT and Irace-AUC for F21. We observe that the result of Irace-ERT outperforms the result of Irace-AUC for the final target $f(x) = 260$. However, for the long period when $f(x) < 258$, Irace-AUC performs better.

We plot in Fig. 5 the violin plots of the running times for the problems where Irace-AUC obtains better ERT values than Irace-ERT. The advantage of Irace-AUC is significant on several problems, i.e., F1, F6, F11, F16, and F22. Moreover, Irace-ERT cannot find the final targets of F7, F17, F19, and F25 within the cutoff time, whereas Irace-AUC hits the targets in some (F7, F17, and F25) or all (F19) of the runs.

Fig. 7 plots the fixed-target result of different GAs on F8. Compared to Irace-ERT, we observe that Irace-AUC outperforms the other algorithm at the final target and also exhibits advantages over other algorithms in most of the optimization process. Fig. 8 plots the fixed-target result of different GAs on F7. The figure shows that Irace-AUC is the only one that hits the optimum $f(x) = 100$, and the best-found fitness of Irace-ERT is less than 90. We observe that none of the GAs shown in the figure hits the optimum in all runs (because the ERT values are larger than the cutoff time of 50 000 function evaluations).

The results of Irace-AUC and Irace-ERT on the PBO problems reveal the questions of how the cost metric affects

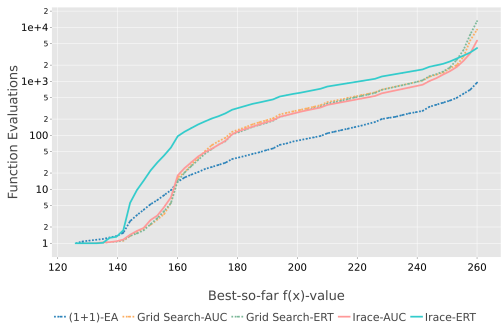


Fig. 6. Fixed-target ERT values of the GAs listed in Tab. II for F21.

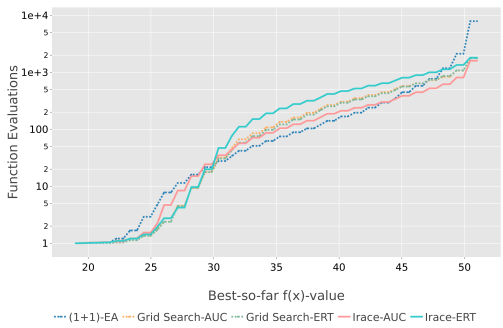


Fig. 7. Fixed-target ERT values of the GAs listed in Tab. II for F8.

the performance of Irace for different configuration tasks for future study. We study in the following the impact of the cutoff time concerning the behavior of Irace on ONEMAX and LEADINGONES.

1) Sensitivity with respect to the cutoff time: Inspired by the result in Fig. 8, we study the sensitivity of ERT and AUC with respect to the cutoff time of the GAs. To this end, we consider the set $\{(0.5 + 0.1t) \times \text{ERT}_{(1+1)\text{ EA}} \mid t \in [0..15]\}$ of 16 different cutoff times. For each of these cutoff time, for each of the two cost metrics (AUC and ERT), and for each of **F1** and **F2**, we run Irace 20 independent times with the same configuration budget of 5 000 target runs (where each target run corresponds again to ten independent runs of the respective $(\mu+\lambda)$ GA configuration). Fig. 9 plots ERT values of the GAs obtained this way (as before, each ERT value is based on 100 independent validation runs). For comparison, the red line indicates the performance of the $(1+1)$ EA.

On ONEMAX, we observe that Irace-ERT cannot find promising configurations when the cutoff time of the GAs is too small to hit the optimum. This is the case for cutoff times smaller than 665. However, Irace-AUC can work with small budgets that are not sufficient to hit the optimum. Even with cutoff times larger than 665, Irace-AUC still obtains better ERT values than Irace-ERT. Similarly, the result for LEADINGONES shows that Irace-ERT cannot find promising configurations with insufficient cutoff times. Still, Irace-AUC performs well across all cutoff times.

Overall, we thus see that tuning with respect to AUC is much less sensitive with respect to the cutoff time.

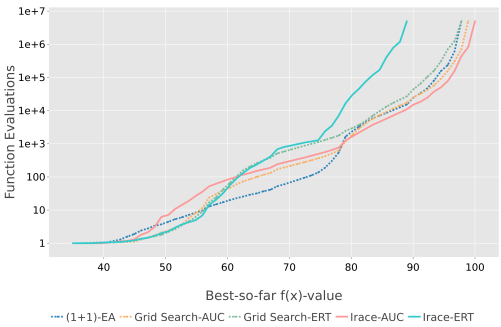


Fig. 8. Fixed-target ERT values of the GAs listed in Tab. II for F7.

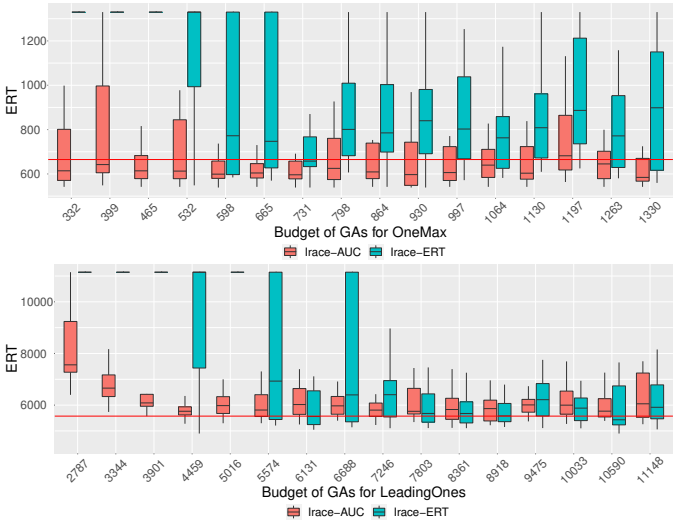
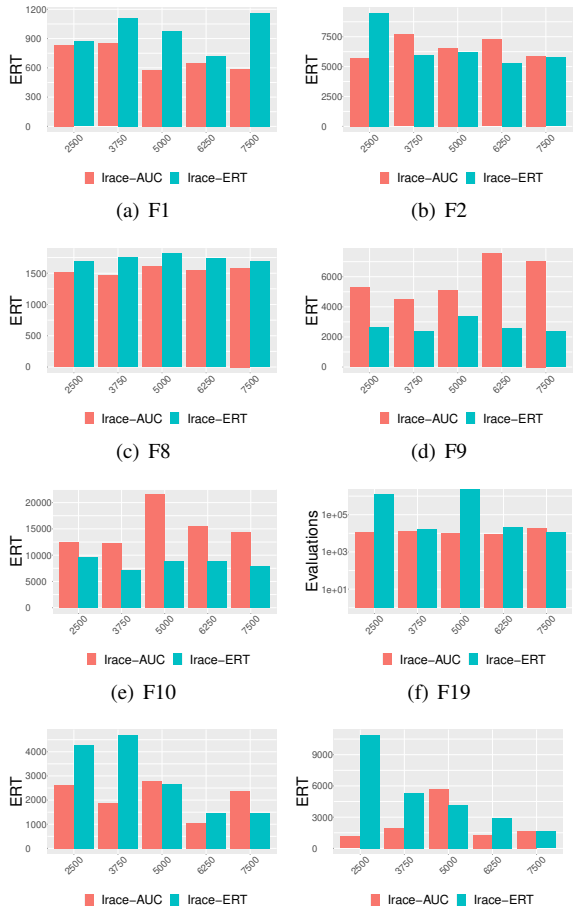


Fig. 9. ERT values (y-axis) of the GAs obtained by Irace for ONEMAX and LEADINGONES in dimension $n = 100$, for different cutoff time B that the GAs can spend to find the optimum (x-axis). Showing results for $B \in \{(0.5 + 0.1t)\text{ERT}_{(1+1)\text{ EA}} \mid t \in [0..15]\}$. For comparison, the ERT values of the $(1+1)$ EA are plotted by horizontal red lines. Results are for the best found configurations obtained from $r = 20$ independent runs of Irace, and each of the ERT values is with respect to 100 independent validation runs.

2) Sensitivity with respect to the configuration budget of Irace: We also analyze the sensitivity of the results with respect to the configuration budget, i.e., the number of target runs that the configurator can perform before it suggests a configuration. We use Irace for this purpose. Fig. 10 plots the ERT values of the configurations suggested by Irace, for 8 selected problems from the PBO suite. Interestingly, the ERT values are not monotonically decreasing, as one might have expected, at least for the configurations that are explicitly tuned for small ERT. Tuning for AUC gave the best ERT values for F1, 8, 19, 20, and 21.

IV. DISCUSSION AND CONCLUSIONS

In this paper, we have analyzed the performance of a family of $(\mu+\lambda)$ GAs in which offspring are generated by applying either crossover (with probability p_c) or mutation (probability $1-p_c$). Four different configuration methods have been applied for finding promising configurations of GAs: grid search and three automated techniques.



(a) F1 (b) F2 (c) F8 (d) F9 (e) F10 (f) F19 (g) F20 (h) F21

Fig. 10. ERT values (y-axis) of the GAs obtained by Irace with different configuration budgets B_T (the number of configurations that Irace can test, x-axis). Results are for $B_T \in \{(0.5 + 0.25t)5\,000 \mid t \in [0..4]\}$. Each ERT value is for the 100 validation runs of the configuration suggested by Irace after a single run, i.e., one for each budget.

The experimental results show that mutation-only GAs usually benefit from small parent population size. On the contrary, crossover-based GAs require sufficient population sizes. On our PBO problem set, the $(1+1)$ EA outperforms the other tested GAs on ONEMAX, LEADINGONES, and some of their W-model extensions. However, crossover can be beneficial for the W-model extensions with epistasis and ruggedness, concatenated trap, and NK-landscapes.

We have also investigated the performance of the AC methods, Irace, MIP-EGO, and MIES. Irace is the only method that could find $(1+\lambda)$ mutation-only GAs, highlighting the importance of handling conditional parameter spaces appropriately.

We also observed that the cost metric used as tuning objective has a major impact on the performance of the AC methods. When using ERT, the AC methods cannot obtain useful information from configurations that cannot hit the optimum. But not only for these cases we observed that tuning for AUC gave better ERT values than when directly tuning for ERT.

Our results raise several interesting topics for future research. Apart from analyzing more closely the configurations obtained for each problem, and the sensitivity of the performance with respect to the different parameters, we also plan to study in what sense our observation that tuning for AUC can help finding better configurations for ERT generalizes to other algorithm families and/or problems. A bi-objective (or even multi-objective, if considering different performance measures) optimization process might be able to balance advantages of the different cost metrics. Such an approach would tie well with recent works in the context of algorithm selection [8], [27], [28].

Our results have also demonstrated that none of the AC methods clearly outperforms all others, suggesting to either combine them or to develop guidelines that can help users select the most suitable configuration technique for their concrete problem at hand. Finally, we have also observed that in several cases none of the techniques could find configurations that outperform or perform on par with the $(1+1)$ EA, which may indicate improvement potential for these configuration methods.

ACKNOWLEDGMENTS

We thank Manuel López-Ibáñez for pointing us to the acviz tool [13]. We thank Thomas Weise for pointing out a mistake in the previous definition of the Ising model problems F19-21.

Our work was supported by the Chinese scholarship council (CSC No. 201706310143), by the Paris Ile-de-France Region, by ANR-11-LABX-0056-LMH (LabEx LMH), and by COST Action CA15140.

REFERENCES

[1] A. Aziz-Alaoui, C. Doerr, and J. Dréo. Towards large scale automated algorithm design by integrating modular benchmarking frameworks. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'21)*, pages 1365–1374. ACM, 2021.

[2] T. Bäck. Parallel optimization of evolutionary algorithms. In *In Proc. of Parallel Problem Solving from Nature (PPSN'94)*, pages 418–427. Springer, 1994.

[3] T. Bäck. *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Inc., USA, 1996.

[4] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *Proc. of International Conference on Genetic Algorithms (ICGA'91)*. Citeseer, 1991.

[5] T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuß. Sequential parameter optimization. In *Proc. of Congress on Evolutionary Computation (CEC'05)*, pages 773–780. IEEE, 2005.

[6] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, 2010.

[7] J. Bossek, P. Kerschke, and H. Trautmann. Anytime behavior of inexact TSP solvers and perspectives for automated algorithm selection. In *Proc. of Congress on Evolutionary Computation (CEC'20)*, pages 1–8. IEEE, 2020.

[8] J. Bossek, P. Kerschke, and H. Trautmann. A multi-objective perspective on performance assessment and automated selection of single-objective optimization algorithms. *Applied Soft Computing*, 88:105901, 2020.

[9] E. Carvalho Pinto and C. Doerr. A simple proof for the usefulness of crossover in black-box optimization. In *Proc. of Parallel Problem Solving from Nature (PPSN'18)*, pages 29–41. Springer, 2018.

[10] D. Corus and P. S. Oliveto. Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 22(5):720–732, 2018.

[11] N. Dang and C. Doerr. Hyper-parameter tuning for the (1 + (λ, λ)) GA. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*, pages 889–897. ACM, 2019.

[12] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[13] M. de Souza, M. Ritt, M. López-Ibáñez, and L. P. Cáceres. Acviz: A tool for the visual analysis of the configuration of algorithms with irace. *Operations Research Perspectives*, 8:100186, 2021.

[14] B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen. Fast genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 777–784. ACM, 2017.

[15] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck. IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv e-prints:1810.05281*, Oct. 2018.

[16] C. Doerr, F. Ye, N. Horesh, H. Wang, O. M. Shir, and T. Bäck. Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, 88:106027, 2020.

[17] K. Eggensperger, M. Lindauer, and F. Hutter. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893, 2019.

[18] S. M. Elsayed, R. A. Sarker, and D. L. Essam. Multi-operator based evolutionary algorithms for solving constrained optimization problems. *Computers & Operations Research*, 38(12):1877 – 1896, 2011.

[19] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[20] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[21] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[22] G. T. Hall, P. S. Oliveto, and D. Sudholt. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 823–831, 2020.

[23] G. T. Hall, P. S. Oliveto, and D. Sudholt. On the impact of the performance metric on efficient algorithm configuration. *Artif. Intell.*, 303:103629, 2022.

[24] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of Learning and Intelligent Optimization (LION'11)*, pages 507–523. Springer, 2011.

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

[26] Hyun-Sook Yoon and Byung-Ro Moon. An empirical study on the synergy of multiple crossover operators. *IEEE Transactions on Evolutionary Computation*, 6(2):212–223, April 2002.

[27] A. D. Jesus, A. Liefooghe, B. Derbel, and L. Paquete. Algorithm selection of anytime algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, pages 850–858. ACM, 2020.

[28] A. D. Jesus, L. Paquete, and A. Liefooghe. A model of anytime algorithm performance for bi-objective optimization. *J. Glob. Optim.*, 79(2):329–350, 2021.

[29] P. K. Lehre and C. Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

[30] R. Li, M. T. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64, 2013.

[31] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[32] M. López-Ibánez and T. Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014.

[33] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In *Proc. of Neural Information Processing Systems Conference (NIPS'93)*, pages 51–58, 1993.

[34] T. Murata and H. Ishibuchi. Positive and negative combination effects of crossover and mutation operators in sequencing problems. In *Proc. of Congress on Evolutionary Computation (CEC'96)*, pages 170–175, May 1996.

[35] L. Pérez Cáceres, M. López-Ibáñez, H. Hoos, and T. Stützle. An experimental study of adaptive capping in irace. In *Proc. of International Conference on Learning and Intelligent Optimization (LION'17)*, pages 235–250. Springer, 2017.

[36] Y. Pushak and H. Hoos. Algorithm configuration landscapes. In *Prof. of Parallel Problem Solving from Nature (PPSN'18)*, pages 271–283. Springer, 2018.

[37] I. Rechenberg. Evolution strategy: Nature's way of optimization. In *Optimization: Methods and applications, possibilities and limitations*, pages 106–126. Springer, 1989.

[38] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the Human Out of the Loop: A review of Bayesian optimization. *Proc. of the IEEE*, 104(1):148–175, 2016.

[39] W. M. Spears. Crossover or mutation? In *Proc. of Foundations of genetic algorithms (FOGA'93)*, pages 221–237. Elsevier, 1993.

[40] T. Stützle and M. López-Ibáñez. *Automated Design of Metaheuristic Algorithms*, pages 541–579. Springer, Cham, 2019.

[41] D. Sudholt. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary computation*, 25(2):237–274, 2017.

[42] D. Sudholt. The benefits of population diversity in evolutionary algorithms: A survey of rigorous runtime analyses. In B. Doerr and F. Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 359–404. Springer, 2020.

[43] R. Tanabe. Analyzing adaptive parameter landscapes in parameter adaptation methods for differential evolution. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 20')*, pages 645–653, 2020.

[44] D. Thierens and P. A. Bosman. Optimal mixing evolutionary algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 617–624, 2011.

[45] R. Tinós, D. Whitley, F. Chicano, and G. Ochoa. Partition crossover for continuous optimization: ePX. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'21)*, pages 627–635. ACM, 2021.

[46] R. Tinós, L. D. Whitley, and F. Chicano. Partition crossover for pseudo-boolean optimization. In *Proc. of Foundations of Genetic Algorithms (FOGA'15)*, pages 137–149. ACM, 2015.

[47] B. van Stein, H. Wang, and T. Bäck. Automatic configuration of deep neural networks with parallel efficient global optimization. In *Proc. of International Joint Conference on Neural Networks (IJCNN'19)*, pages 1–7. IEEE, 2019.

[48] H. Wang, M. Emmerich, and T. Bäck. Cooling Strategies for the Moment-Generating Function in Bayesian Global Optimization. In *Proc. of Congress on Evolutionary Computation (CEC'18)*, pages 1–8. IEEE, 2018.

[49] H. Wang, B. van Stein, M. Emmerich, and T. Bäck. A new acquisition function for Bayesian optimization based on the moment-generating function. In *Proc. of International Conference on Systems, Man, and Cybernetics (SMC'17)*, pages 507–512. IEEE, 2017.

[50] H. Wang, D. Vermetten, F. Ye, C. Doerr, and T. Bäck. IOHanalyzer: Performance analysis for iterative optimization heuristic. *ACM Transactions on Evolutionary Learning and Optimization*, 2022.

[51] T. Weise, Y. Chen, X. Li, and Z. Wu. Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing*, 92:106269, 2020.

[52] T. Weise and Z. Wu. Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18, Companion Material)*, pages 1769–1776. ACM, 2018.

[53] F. Ye, C. Doerr, and T. Bäck. Interpolating local and global search by controlling the variance of standard bit mutation. In *Proc. of IEEE Congress on Evolutionary Computation (CEC'19)*, pages 2292–2299. IEEE, 2019.

[54] F. Ye, C. Doerr, H. Wang, and T. Bäck. Data sets for the study "Automated Configuration of Genetic Algorithms by Tuning for Anytime Performance". https://doi.org/10.5281/zenodo.4823492, May 2021.

[55] F. Ye, H. Wang, C. Doerr, and T. Bäck. Benchmarking a $(\mu + \lambda)$ genetic algorithm with configurable crossover probability. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, pages 699–713. Springer, 2020.

Furong Ye, is a PostDoc at the Leiden Institute of Advanced Computer Science (LIACS), after finishing his PhD study at LIACS. His PhD topic is "Benchmarking discrete optimization heuristics: From building a sound experimental environment to algorithm configuration". He is part of the core development team of IOHprofiler, with a focus on the IOHexperimenter. His research interests are the empirical analysis of algorithm performance and (dynamic) algorithm configuration.

Carola Doerr, formerly Winzen, is since 2013 a permanent CNRS researcher at Sorbonne Université in Paris, France. Carola's main research activities are in the mathematical analysis of randomized algorithms, with a strong focus on evolutionary algorithms and other sampling-based optimization heuristics. She is also interested in all aspects of benchmarking these algorithms. Carola obtained her PhD in Computer Science from Saarland University and the Max Planck Institute for Informatics in 2011 and she successfully defended her habilitation (HDR) at Sorbonne Université in 2020.

She is associate editor of IEEE Transactions on Evolutionary Computation, of ACM Transactions on Evolutionary Learning and Optimization, editorial board member of the Evolutionary Computation journal, advisory board member of the Springer Natural Computing Book Series, and guest editor of a special issue in TEVC and two special issues in Algorithmica. She is a founding and coordinating member of the Benchmarking Network, an initiative created to consolidate and to stimulate activities on benchmarking sampling-based optimization heuristics.

Hao Wang is an assistant professor of computer science in Leiden University since September 2020, focusing black-box optimization, Bayesian optimization, multi-objective optimization, and variational quantum algorithms. Hao obtained his Ph.D. (cum laude) at Leiden University in 2018, followed by two postdoctoral appointments: at Leiden University (2018.05 – 2019.12) and LIP6, Sorbonne Université, Paris, France (2020.01 – 2020.08). He served as the proceedings chair for the PPSN 2020 conference and will be organizing the EMO (Evolutionary Multi-Objective Optimization) 2023 international conference as one of the general co-chairs. He received the best paper award at the PPSN (Parallel Problem Solving from Nature) 2016 conference for proposing new measures to understand the difficulties of multi-objective optimization problems.

Thomas Bäck, is Full Professor of Computer Science at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands, since 2002. He received the Diploma degree in computer science in 1990 and the PhD degree in 1994, both from the University of Dortmund, Germany. His research interests include evolutionary computation, machine learning, and their real-world applications, especially in sustainable smart industry and health.

In 2021, Thomas has been elected as member of the Royal Netherlands Academy of Aarts and Sciences (KNAW). He was a recipient of the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Pioneer Award in 2015, was elected as Fellow of the International Society of Genetic and Evolutionary Computation in 2003, and received the best PhD thesis award from the German society of Computer Science (Gesellschaft für Informatik, GI) in 1995.

He currently serves as an Associate Editor for the IEEE Transactions on Evolutionary Computation and Artificial Intelligence Review journals, and as area editor for the ACM Transactions on Evolutionary Learning and Optimization. He is co-editor of the Natural Computing Book Series, Handbook of Evolutionary Computation, Handbook of Natural Computing, and author of Evolutionary Computation in Theory and Practice.

FIGURE CAPTIONS AND TABLE CAPTIONS

Fig. 1. Fixed-target ERT values of the configurations suggested for F10. The suffix "-ERT/AUC" indicates which cost metric was used during the tuning.

Fig. 2. Relative ERT values of the GAs obtained by Irace, MIES, and MIP-EGO using ERT as the cost metric compared to the ERT of the GAs obtained by the same method when using AUC as the cost metric during the configuration process. Plotted values are ($\text{ERT}_{\text{using ERT}}$-$\text{ERT}_{\text{using AUC}}$)/$\text{ERT}_{\text{using AUC}}$, capped at -1 and 1. Positive values therefore indicate that configurator obtains better results when configuring AUC.

Fig. 3. The relative deviation from the best-known ERT value of the GAs obtained during the configuration process of Irace for tuning the $(\mu + \lambda)$ GA for ONEMAX in dimension $n = 100$, with the objective to minimize the ERT for the optimum $f(x) = 100$. The maximal number of configurations that can be tested by Irace is set to 5 000. The figure is produced by the acviz tool, and illustrations for the details of plot representation can be found in Sec. 3 of [13].

Fig. 4. Violin plots of first hitting times for the configurations found by Irace when tuning for ERT and AUC, respectively. Only showing results for problems on which Irace-ERT outperforms Irace-AUC. Results are from the 100 independent validation runs. Targets are listed in Tab. I, and the configurations of the GAs can be found in Tab. II. For each run, values are capped at the budget 50 000 if the algorithm cannot find the target.

Fig. 5. Violin plots of first hitting times for the configurations found by Irace when tuning for ERT and AUC, respectively, for problems on which Irace-AUC outperforms Irace-ERT. Results are from the 100 independent validation runs. Targets are listed in Tab. I, and the configurations of the GAs can be found in Tab. II. For each run, values are capped at the budget 50 000 if the algorithm cannot find the target.

Fig. 6. Fixed-target ERT values of the GAs listed in Tab. II for F21.

Fig. 7. Fixed-target ERT values of the GAs listed in Tab. II for F8.

Fig. 8. Fixed-target ERT values of the GAs listed in Tab. II for F7.

Fig. 9. ERT values (y-axis) of the GAs obtained by Irace for ONEMAX and LEADINGONES in dimension $n = 100$, for different cutoff time B that the GAs can spend to find the optimum (x-axis). Showing results for $B \in \{(0.5 + 0.1t)\text{ERT}_{(1+1)\text{ EA}} \mid t \in [0..15]\}$. For comparison, the ERT values of the $(1+1)$ EA are plotted by horizontal red lines. Results are for the best found configurations obtained from $r = 20$ independent runs of Irace, and each of the ERT values is with respect to 100 independent validation runs.

Fig. 10. ERT values (y-axis) of the GAs obtained by Irace with different configuration budgets B_T (the number of configurations that Irace can test, x-axis). Results are for $B_T \in \{(0.5 + 0.25t)5\,000 \mid t \in [0..4]\}$. Each ERT value is for the 100 validation runs of the configuration suggested by Irace after a single run, i.e., one for each budget.

Table. 1. Target values used to compute the ERT metric on each problem.

Table. 2. Configurations of the $(\mu + \lambda)$ GA obtained by grid search, Irace, MIP-EGO, and MIES. C indicates the cost metric used by the configurators. Results for maximizing AUC are obtained independently from those obtained for minimizing ERT.

Table. 3. Absolute ERT and AUC values for the $(1+1)$ EA and relative improvement of ERT and AUC for the configurations suggested by the four configuration methods, in comparison against the $(1 + 1)$ EA values. Compared to the ERT and AUC values of the $(1 + 1)$ EA on each problem (indicated by "EA"), the relative improvement obtained from the automated configuration are shown for each AC method, where both measures are calculated from hitting times of 100 validation runs for $(1 + 1)$ EA and AC methods. We also indicate the statistical significance in the empirical distributions of the hitting time (**** for $p < 0.001$, *** for $p < 0.01$, ** for < 0.01, and * for $p < 0.05$) for each pair of the AC result and that of the $(1 + 1)$ EA. The Mann–Whitney U test is applied with the Benjamini and Hochberg method for all 120 pairwise comparisons to control the false discovery rate. Runs are cut off at 50 000 evaluations if it does not hit the final target. The significant comparisons are colour-coded with respect to the relative improvement, where a darker colour signifies a more considerable improvement.