Benchmarking discrete optimization heuristics: from building a sound experimental environment to algorithm configuration

Ye, F.

Chapter 6

Automatic Configuration of Genetic Algorithms

We compare in this chapter the results from Section 5.2 with those obtained from three different types of automated algorithm configuration methods, one based on iterated racing (we use Irace [111]), one surrogate-assisted technique (we use the mixed-integer parallel efficient global optimization MIP-EGO [151]), and a classic heuristic optimization method (we use the mixed-integer evolutionary strategies MIES suggested in [109]). Also, we configure the $(\mu + \lambda)$ GA considering two different objectives, i.e., expected running time and the area under empirical cumulative distribution function curve.

6.1 Background

It was discussed in Section 5.2 that there is a long debate about the effectiveness of the two main variators, crossover and mutation, and their combinations [118, 140]. Several works study the synergy of mutation and crossover, both by empirical and by theoretical means, see [34, 63, 85, 121] and [144], 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 scarce, 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* problem.

The AC problem was classically addressed by standard search heuristics such as mixed-integer evolution strategies [4, 71, 109]. More specific AC tools have been developed in recent years, among them surrogate-based models (e.g., SPOT [13], SMAC [82], MIP-EGO [151]), racing-based methods (Irace [111], F-race [18]) and optimization-based methods (ParamILS [83]).

The result in this chapter is built on Section 5.2 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 $p_c \in [0,1]$. Recall that, in Algorithm 8, the framework creates new solution candidates by either applying mutation (with probability $1-p_c$) or by applying 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 Section 5.2 by means of 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.

Note that, to distinguish it from the budget of the configurators, we denote the budget of the GAs as *cutoff time* in the following.

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 travel salesperson problem [20], for MAX-MIN ant systems, and for mixed-integer programming [112]) and for the here-considered function evaluation budgets (see [3] for a recent example). However, we are not aware of any works using anytime measures with the objective to identify configurations that minimize ERT values. On the other hand, previous work has studied the impact of the cutoff time for algorithm configurators [73, 74]. In [74], the authors conclude that considering the best-found fitness is more efficient than considering the expected running time. Our work considers AUC, a measure that takes both the found fitness and running time into account, and we conclude that the cutoff time can influence AUC less, when compared to ERT.

All our data is publicly available at [169]. Section 4.3 describes the configurable $(\mu+\lambda)$ GA (Algorithm 13). The benchmark problems were introduced in Section 2.5, and the cost metrics used to evaluate the algorithms were explained in Section 2.4.

6.2 Configurators

We briefly introduce the three AC methods that are applied for configuring the $(\mu + \lambda)$ GA: Irace [111], a mixed-integer parallel efficient global optimization (MIP-EGO [155]), and the mixed-integer evolution strategy (MIES [109]), 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 [111] is a so-called iterated racing method designed for hyperparameter optimization. The main steps of Irace are (1) sampling values of parameters from particular distributions, after which (2) the algorithms with the corresponding sampled parameter settings are evaluated across a set of instances, then (3) elitist ones will be selected by the racing method and (4) the sampling distributions are updated based on the elitist configurations, to bias the sampling towards the elitist space. The sampling distributions of parameters are independent of each other unless user-defined constraints and conditions exist. For the racing method, the sampled parameter settings are evaluated on instances. After several steps, parameter settings that are statistically worse than others are discarded. 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.

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** [155]) capable of handling mixed-integer search space. EGO starts by randomly sampling solution candidates $\{\theta_1, \theta_2, \ldots\}$ and evaluating their fitness $\{c_1, c_2, \ldots\}$. 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 [67, 137] for an overview. For this work, we use the moment-generating function of improvement (MGFI) [155], which is defined as the weighted combination of all moments of the predictive distribution. For the weights, we took a robust setting in [154], obtained by an extensive empirical study on the BBOB problem set. 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 [109]. MIES starts with a randomly initialized parent population of size μ, and then it generates λ offspring candidates for each generation iteratively. Offspring are generated by recombining two randomly selected parents and then mutating the solution resulting from the 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.

6.3 Experimental Results

6.3.1 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 10 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 [32]. MIP-EGO starts with 10 initial candidates by the default setting of the package [155]. We use a $(4, 28)$ MIES, following the parameter settings suggested in [109].

To obtain a useful baseline against which we can compare other algorithms, we configure the $(\mu+\lambda)$ GA *on each PBO problem separately.* We consider $n = 100$ for all problems and take ERT or AUC (see Section 2.4) calculated from 10 independent GA runs as the cost metric, respectively. Also, we perform 100 independent validation runs with the suggested parameter settings from a configurator, which is meant to mitigate the randomness of the cost metric, thereby producing a fair empirical comparison. All experimental data discussed later are obtained using this validation procedure.

For evaluating GA candidates during the configuration process, the ERT values are calculated with respect to the targets listed in Table 5.2 and to the cutoff time of $50,000$ function evaluations. This is half the budget used in Section 5.2, 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 Table 5.2 and $\phi_{\min}$ is 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 for each point in $[50,000]$, i.e., in the notation of Definition 2.5 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 $\mu > 1$, which otherwise renders a configuration infeasible. The results for the grid search are based on our work in Section 5.2, 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.

Table 6.1: Configurations of the $(\mu + \lambda)$ GA obtained by grid search, Irace, MIP-EGO, and MIES. 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

Table 6.2: 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 the 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	**ERT**					**AUC**				
	EA	**GS**	**Irace**	**MIP-EGO**	**MIES**	**EA**	**GS**	**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	5,574	-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	7,926	0.78****	0.77****	-0.01****	0.86****	0.9938	0.25****	-0.37****	-0.65****	-0.75****
9	22,670	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	2,071	-0.29****	-0.35****	-0.45****	0.12**	0.9807	-0.36****	-1.05	-0.89	-0.93
12	4,691	-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	8,171	0.98****	0.99****	0.98****	0.96****	0.596	47.69****	47.29****	36.75****	41.05****
15	6,668	-1.46****	-1.17****	-0.50****	-1.23****	0.9404	-7.04****	-8.98****	-2.05****	-0.88****

16	9,520	-2.05****	-0.97****	-0.43****	-0.23****	0.9174	-17.54****	0.41	-12.97****	-1.78****
17	45,964	-Inf****	-Inf****	-Inf****	-0.96****	0.6698	-48.43****	2.45***	-14.40****	-7.93****
18	130,863	0.85****	0.80****	0.69****	0.85****	0.9432	1.86***	1.32	0.06	1.26****
19	9,467	-525.05****	-Inf****	-1.72****	-0.72****	0.9899	-6.19****	0.00	-4.96****	-0.37**
20	1,460	-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	3,366	-2.60****	-0.11****	-0.57****	-0.15	0.9993	-0.02****	-0.05****	-0.06	-0.02****
23	3,066	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	48,946	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

6.3.2 Results Obtained by Automated Configuration

The $(1+1)$ EA with $p_m = 1/n$ has shown competitive results in [55] for the PBO problems, so we use it as the baseline against which we compare the GAs obtained by the configurators. Although it is part of the GA framework in Algorithm 8 and could therefore be identified by the configuration methods, we still manually add it as a baseline and we compare the results obtained by the three configurators to it. Table 6.1 lists the configurations of the $(\mu+\lambda)$ GA obtained by the grid search and by the three AC methods. Table 6.2 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 - $\text{AUC}_{(1+1)-\text{EA}}$) / $\text{AUC}_{(1+1)-\text{EA}}$. Grey tiles indicate that the result is better than the corresponding result of the $(1+1)$ EA, and the degree of gray represents the degree of improvement. We use in this table the Mann-Whitney U test to compare the average running times of the (1+1) EA and the GAs suggested by the configuration methods (pairwise comparisons). Runs that did not hit the final target within the cutoff time of $50,000$ evaluations are capped at this value. Asterisks in Table 6.2 indicate that the average hitting time of the GAs suggested by the corresponding configuration methods is significantly different from the average hitting time of the $(1+1)$ EA.

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 $\mu \geq 10$ [170]. However, according to the values in Table 6.2, 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 [25, 29, 143].

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,

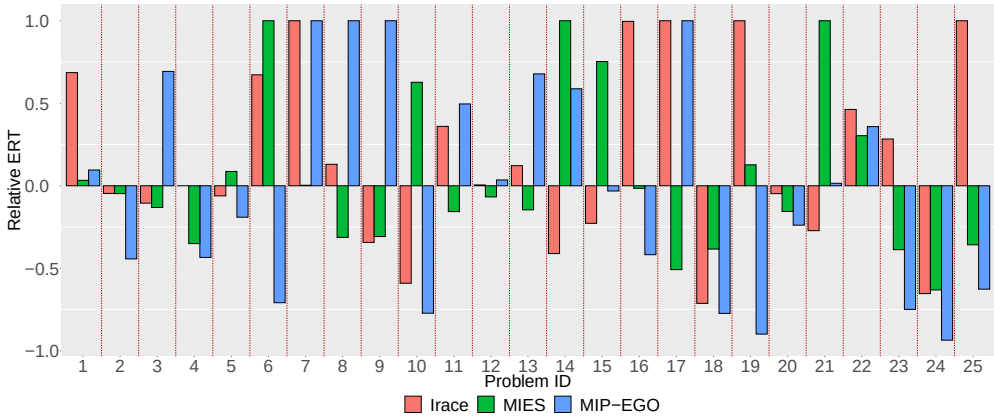

Figure 6.1: Relative ERT values of the GAs obtained by Irace, MIES, and MIP-EGO using ERT as the cost metric compared to the ERT of the GAs obtained by the same method when using AUC as the cost metric during the configuration process. Plotted values are $(\text{ERT}_{\text{using ERT}}\text{-}\text{ERT}_{\text{using AUC}})/\text{ERT}_{\text{using AUC}}$, capped at -1 and 1. Positive values therefore indicate that configurator obtains better results when configuring AUC.

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 [170] 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.

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.

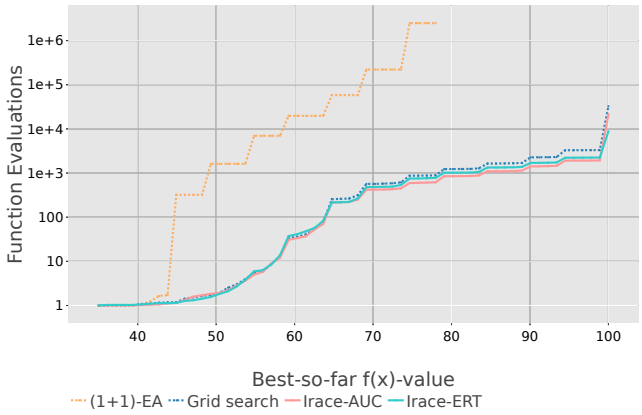

Figure 6.2: Fixed-target ERT values of the configurations suggested for F10. The suffix "-ERT/AUC" indicates which cost metric is used

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 that of the (1+1) EA. In some, but clearly not all of these cases, the achieved AUC values are not much worse than that of the $(1+1)$ EA. The largest improvements are obtained on functions F7, F10, F14, F17, F18, and F24. 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 obtained algorithm is 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 improvement over the $(1+1)$ EA were obtained.

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 large 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 **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, the best AUC value). 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 Figure 6.2.

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. Interestingly, the performance of randomized local search is not very good (see [55] 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 MIP-EGO and MIES, to 0.7 for Irace.

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, which would require an effort that goes beyond the scope of this work. However, we have seen related work being conducted in [129, 146]

6.3.3 Discussions on the Configurators' Performance

We now compare the performance of the three automated AC methods. The last row of Table 6.2 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 the improvements over the $(1 + 1)$ EA are, however, rather minor in several of these cases, so that barely counting them does not give justice to the complex behavior observed in Table 6.2, from which we cannot derive a clear winning configurator. We can nevertheless make a few observations.

Handling Conditional Parameter Spaces

We easily see from Table 6.1 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. In contrast, the two other AC methods, MIP-EGO and MIES, 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 Table 6.1, 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. We observe a similar performance of MIES on F17.

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 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 [127], 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 [83] 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 behaviour also indicates that the choice of the cost metric might be a factor to consider when choosing which AC

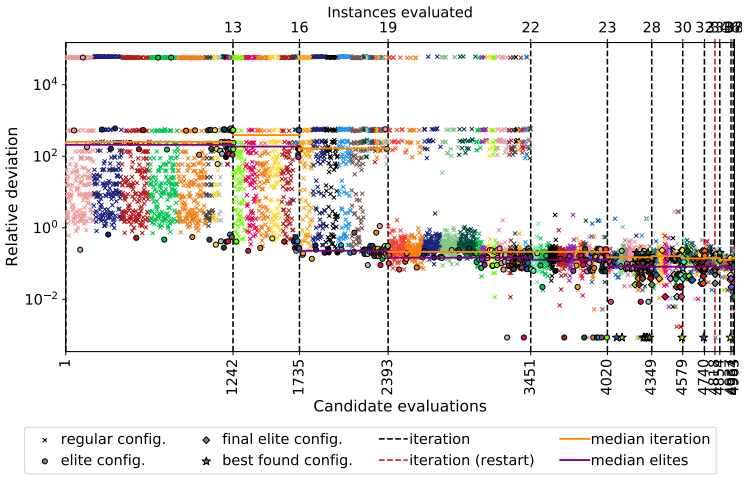

Figure 6.3: The relative deviation from the best-known ERT value of the GAs obtained during the configuration process of Irace for tuning the $(\mu + \lambda)$ GA for OneMax in dimension $n = 100$, with the objective to minimize the ERT for the optimum $f(x) = 100$. The maximal number of configurations that can be tested by Irace is set to 5,000. The figure is produced by the acviz tool [36].

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 can not distinguish algorithms that can not 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.

Figure 6.3 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.

6.3.4 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. Figure 6.1 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 give 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 abbreviates "Irace-ERT" ("Irace-AUC") the configurations obtained when using ERT (AUC) as cost metric. For the problems on which the ERT of Irace-AUC was worse than that of Irace-ERT, we plot in Figure 6.6 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 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. Figure 6.4 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. This observation indicates that configuring AUC can provide novel instances to investigate how the GA performs during the optimization process.

We plot in Figure 6.7 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 can not 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.

Figure 6.5 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. Figure 6.8 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 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.

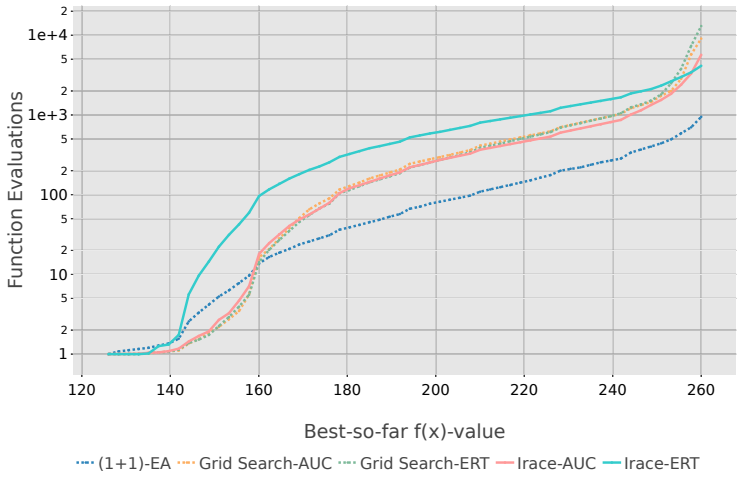

Figure 6.4: Fixed-target ERT values of the GAs listed in Table 6.1 for F21.

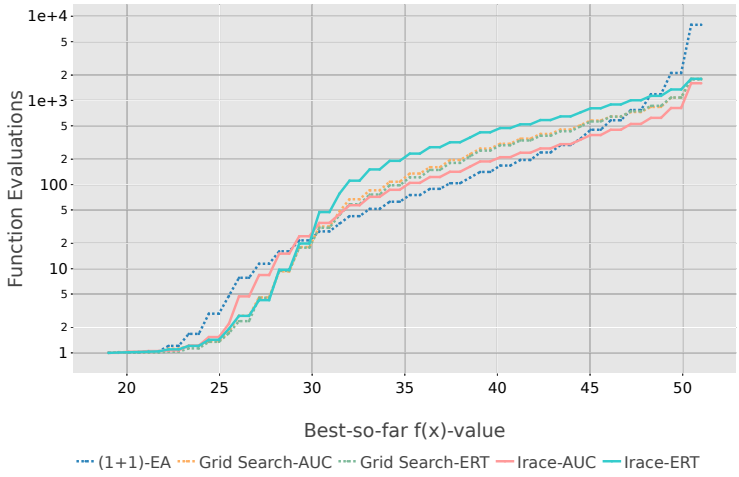

Figure 6.5: Fixed-target ERT values of the GAs listed in Table 6.1 for F8.

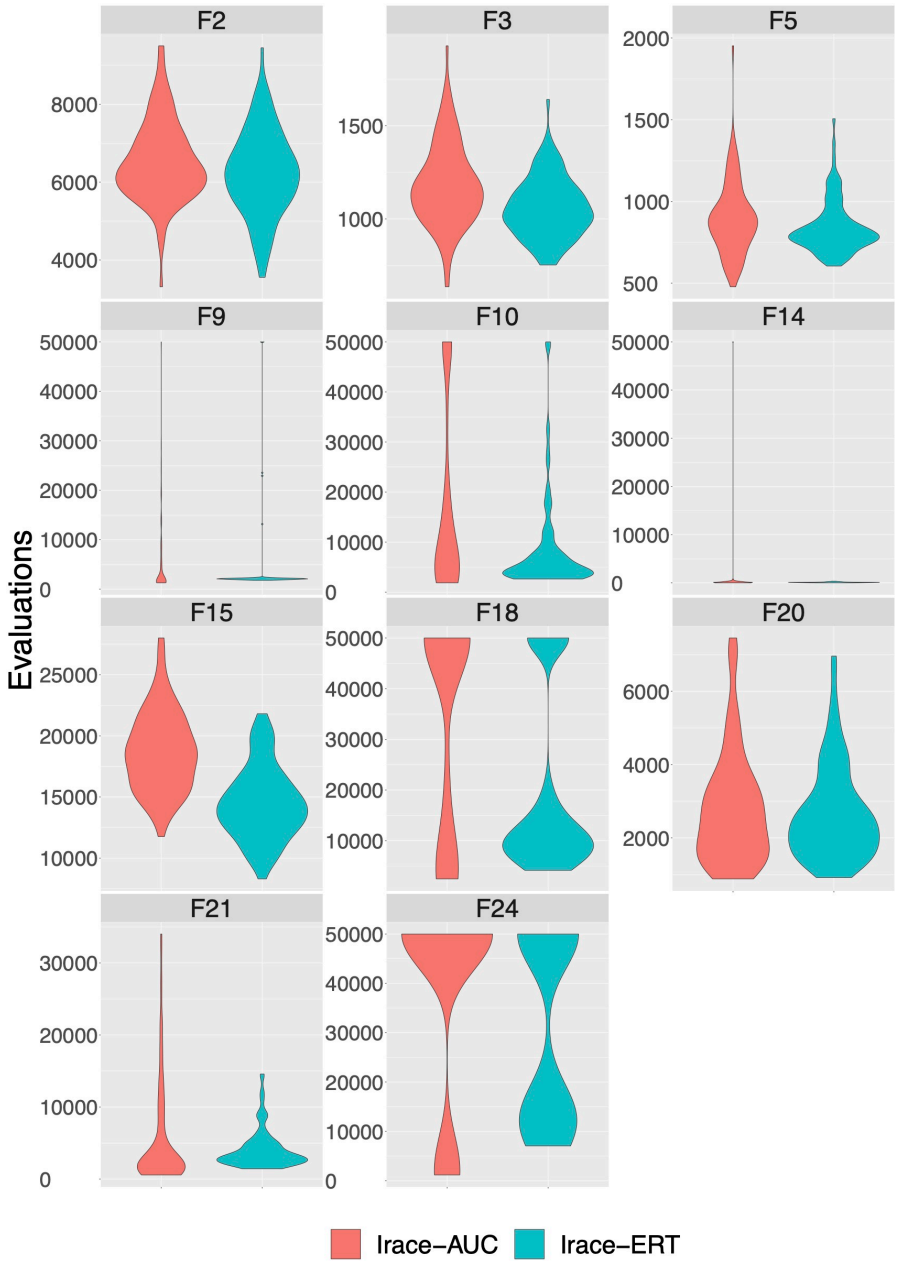

Figure 6.6: Violin plots of first runtimes hitting the targets for the configurations found by Irace when tuning for ERT and AUC, respectively. Only showing results for problems on which Irace-ERT outperforms Irace-AUC. Results are from the 100 independent validation runs. Targets are listed in Table 5.2, and the configurations of the GAs can be found in Table 6.1. For each run, values are capped at the budget $50,000$ if the algorithm can not find the target.

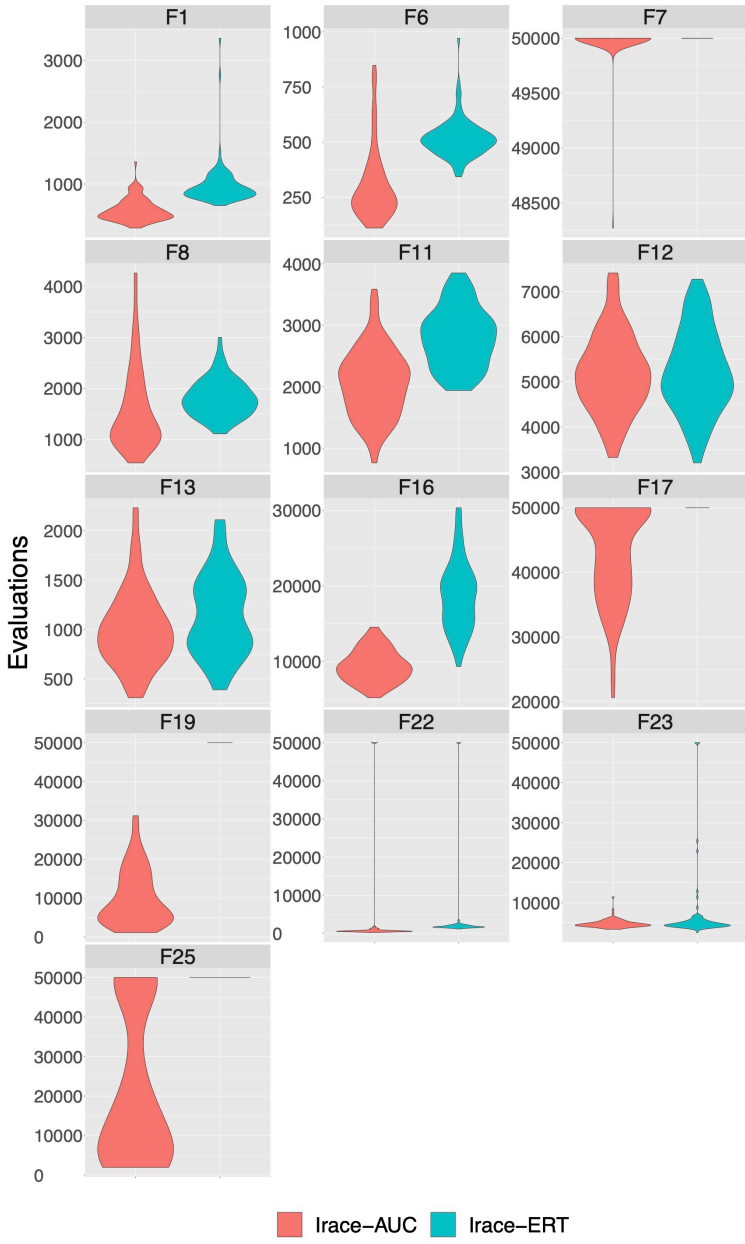

Figure 6.7: Violin plots of first runtimes hitting the targets for the configurations found by Irace when tuning for ERT and AUC, respectively, for problems on which Irace-AUC outperforms Irace-ERT. Results are from the 100 independent validation runs. Targets are listed in Table 5.2, and the configurations of the GAs can be found in Table 6.1. For each run, values are capped at the budget $50,000$ if the algorithm can not find the target.

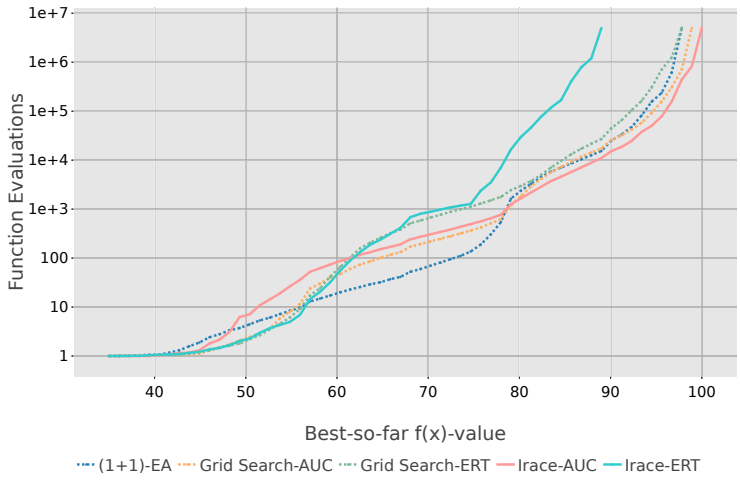

Figure 6.8: Fixed-target ERT values of the GAs listed in Table 6.1 for F7.

Sensitivity with respect to the cutoff time

Inspired by the result in Figure 6.8, we study the sensitivity of ERT and AUC with respect to the cutoff time of the GAs. To this end, we consider the set $\{(0.5+0.1t) \times \mathrm{ERT}_{(1+1)\ \mathrm{EA}} \mid t \in [0..15]\}$ of 16 different cutoff times. For each of these cutoff times, 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). Figure 6.9 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 can not find promising configurations when the cutoff time of the GAs is too small to hit the optimum. This is the case for cutoff time of the budgets smaller than 665. However, Irace-AUC can work with small budgets that are not sufficient to hit the optimum. Even with the cutoff time of the budgets 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 time of GAs. Still, Irace-AUC performs well across all given cutoff time.

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

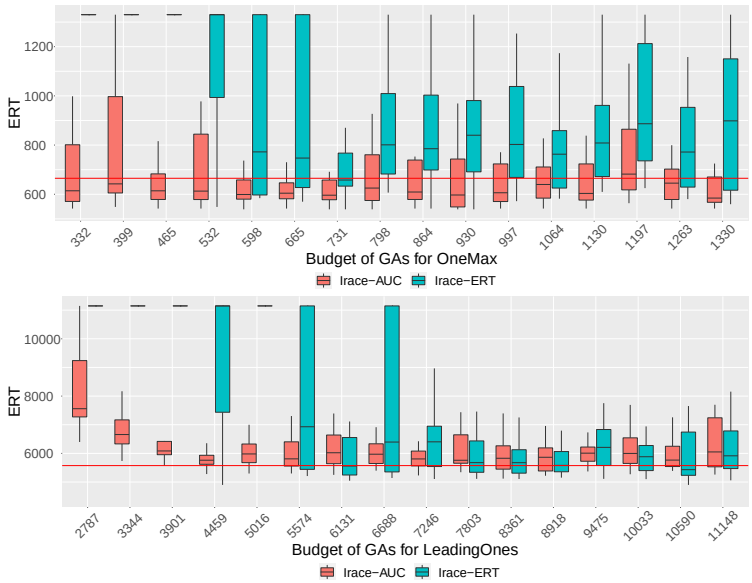

Figure 6.9: ERT values (y-axis) of the GAs obtained by Irace for OneMax and LeadingOnes in dimension $n = 100$, for different budgets B that the GAs can spend to find the optimum (x-axis). Showing results for $B \in \{(0.5 + 0.1t)\text{ERT}_{(1+1)\text{ EA}} \mid t \in [15]\}$. For comparison, the ERT values of the $(1 + 1)$ EA are plotted by horizontal red lines. Results are for the best found configurations obtained from 20 independent runs of Irace, and each of the ERT values is with respect to 100 independent validation runs.

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. Figure 6.10 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.

6.4 Summary

In this chapter, we extended the analysis on the performance of a family of $(\mu + \lambda)$ GAs, based on the work of Section 5.2. Four different configuration methods have

been applied for finding promising configurations of GAs: the grid search and three automated techniques.

The experimental results showed that mutation-only GAs usually benefit from small parent population size. On the contrary, crossover-based GAs require sufficient population sizes. On the 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 AC methods: Irace, MIP-EGO, and MIES. Irace is the only method that has found conditional configurations of $(1 + \lambda)$ mutation-only GAs. It handles the non-conditional parameters first and samples the conditional parameters when the condition is satisfied, but the other two automated methods sample all parameters independently, leading to worse results in cases where mutation-only is beneficial.

We also observed that the cost metric used as tuning objective has a major impact on the performance of 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 have also demonstrated that none of the configuration methods clearly outperforms all others, suggesting to either combine them or to develop guidelines that can help users select a most suitable configuration technique for their concrete problem at hand. Finally, we also observe 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.

Figure 6.10: 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 $B_T \in \{(0.5 + 0.25t)5,000 \mid t \in [4]\}$. 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.