Non-Elitist Selection among Survivor Configurations can Improve the Performance of Irace

Furong Ye  
LIACS, Leiden University  
Leiden, Netherlands

Carola Doerr  
LIP6, Sorbonne Université  
Paris, France

Diederick Vermetten  
LIACS, Leiden University  
Leiden, Netherlands

Thomas Bäck  
LIACS, Leiden University  
Leiden, Netherlands

ABSTRACT
Modern optimization strategies such as evolutionary algorithms, ant colony algorithms, Bayesian optimization techniques, etc. come with several parameters that steer their behavior during the optimization process. To obtain high-performing algorithm instances, automated algorithm configuration techniques have been developed. One of the most popular tools is irace, which evaluates configurations in sequential races, at the end of which a statistical test is used to determine the set of survivor configurations. It then selects up to five elite configurations from this set, via greedy truncation selection.

We demonstrate that an alternative selection of the elites can improve the performance of irace. Our strategy keeps the best survivor and selects the remaining configurations uniformly at random from the set of survivors. We apply this alternative selection method to tune ant colony optimization algorithms for traveling salesperson problems and to configure an exact tree search solver for satisfiability problems.

We also experiment with two non-elitist selection criteria, based on entropy and Gower’s distance, respectively. Both methods provide more diverse configurations than irace, making them an interesting approach for exploring a wide range of solutions and understanding algorithms’ performance. Moreover, the entropy-based selection performs better on our benchmarks than the default selection of irace.

KEYWORDS
Parameter tuning and algorithm configuration, Combinatorial optimization

1 INTRODUCTION
Algorithm configuration (AC) addresses the issue of determining a well-performing parameter configuration for a given algorithm on a specific set of optimization problems.

Many techniques such as local search, Bayesian optimization, and racing methods have been proposed and applied to solve the AC problem. The corresponding software packages, such as ParamILS [20], SMAC [19], SPOT [5], MIP-EGO [35], and irace [25], have been applied to problem domains such as combinatorial optimization [25], software engineering [6], and machine learning [21].

Irace, one of the most popular tools, has shown its ability to improve the performance of the algorithms for various combinatorial optimization problems. For example, the algorithm configurations tuned by irace outperform the default configuration of the Ant Colony Optimization (ACO) algorithm for the traveling salesman problem (TSP) [25, 32]. Irace has also been used to reduce the runtime of an exact tree search solver (SPEAR) [3] for the satisfiability problem (SAT) [25]. Moreover, a recent study has integrated the elitist iterated racing method of irace with evolutionary operators for traffic light planning [10].

Although irace has been shown to be effective at finding competitive configurations of algorithms on a wide variety of problems, it is often still unclear whether the found configuration is indeed optimal. More generally, the question are there other, competitively well-performing configurations in unexplored areas of the search space? is still an open problem.

A recent study applied irace to the problem of tuning a family of genetic algorithms (GAs) on the OneMax problem [36], in principle, as known due to the extensive body of theoretical work [16, 33] on this problem, there exist more than one type of competitive configurations of the GA. However, irace usually converges to a specific subset these configurations which share similar algorithm characteristics but in doing so fails to find the optimal configuration which is of a considerably different type. In order to avoid issues like this, one could aim to increase the exploration capabilities of irace. However, this does not necessarily address the concern of finding well-performing configurations located in different parts of the space. Instead, we would want to allow irace to automatically focus on finding a diverse set of well-performing configurations, rather than converging on one specific type of configuration.

In this paper, we investigate alternative selection mechanisms for irace, by considering diversity of configurations in the selection process. While this diversity on its own can be beneficial for certain applications, it can also lead to performance improvements by allowing irace to explore more distinct subdomains of the configuration space. In addition, the observations from [36] inspire a discussion on searching for various competitive configurations with different patterns, which is addressed by our discussion on the relationship between parameters of the ACO algorithm and its performance on TSP problem instances.

1.1 Our Contributions
In this paper, we modify the selection procedure of irace by taking diversity into account. We show that this modification is not only
beneficial for the final diversity of the returned configurations, but also improves the performance over the default irace method. This proof of concept motivates further study into usage of diversity in algorithm configuration, for example by changing the AC problem’s objective from finding a “single” optimum to selecting a diverse set of well-performing configurations.

We start with applying an alternative randomly selection of elites for irace. Moreover, another two selection operators based on the metrics of entropy and Gower’s distance are proposed for selecting diverse configurations. These alternative approaches are tested on two scenarios of combinatorial optimization that were studied in the initial irace work [25]: tuning for better configurations of the ACOTSP package (the ant colony optimization algorithm to tackle the symmetric TSP) and minimizing the computational cost of the SPEAR tool (an exact tree search solver for SAT problems).

Experimental results show that (1) randomly selecting elites among survivor configurations performs better than the greedy truncation selection, and (2) the irace variant applying which uses entropy during the selection process obtains diverse configurations and outperforms the other approaches. Furthermore, the obtained configurations provide evidence that (3) using such a diversity-enhancing approach can find better configurations and contributes to our understanding of the relationship between parameter settings and algorithm behavior.

Outline: Related work is introduced in Section 2, and we recap the core functionality of irace in Section 3. Results for the proposed random survivor selection method on ACOTSP and SPEAR are reported in Section 4. Section 5 presents the proposed diversity-preserving approaches irace-entropy and irace-Gower, and the results for these configurators on ACOTSP. Section 6 concludes the paper with several ideas for future work.

Reproducibility: Our code, results, and additional figures are available at [1].

2 RELATED WORK

2.1 Algorithm Configuration

Many parameterized algorithms for hard optimization problems need to be properly configured in order to achieve optimal performance. For example, the GA usually asks users to specify population size, mutation rate, crossover probability, etc. Even the self-adaptive algorithms such as Evolution Strategies (ES) [4] depend on settings such as learning rates and population size. Also, for the hybrid algorithms and modular algorithm frameworks such as the modular CMA-ES [13, 34], the selection of operators has significant impact on the algorithms’ performance.

These tuning tasks are commonly considered as instances of the Algorithm Configuration (AC) problem, which aims to find a well-performing configuration for such optimization algorithms automatically.

Traditionally, the AC problem, as defined below, aims at finding a single optimal configuration for solving a set of problem instances [14].

Definition 2.1 (Algorithm Configuration Problem). Given a set of problems instances \( \Pi \), a parametrized algorithm \( A \) with parameter configuration space \( \Theta \), and a cost metric \( c : \Theta \times \Pi \rightarrow \mathbb{R} \) that is subject to minimization, the objective of the AC problem is to find a configuration \( \theta^* \in \arg\min_{\theta \in \Theta} \sum_{\pi \in \Pi} c(\theta, \pi) \).

The parameter space can be continuous, integer, categorical, and mixed-integer. In addition, some parameters can be conditional and dependent on others.

Many configurators have been proposed for the AC problem [5, 19, 20, 23, 25, 35], and they usually follow Definition 2.1 by searching for a single “optimal” solution, although the solvers may apply population-based methods. However, in some cases it can be desirable to find a set of diverse, well-performing solutions to the AC problem, as exemplified by our discussion regarding GA configurations in Section 1. Moreover previous studies [27, 36] found that algorithm configurators can obtain different results when tuning for different objectives (i.e., expected running time, best-found fitness, and anytime performance), which suggests that a bi- or multi-objective approach to algorithm configuration can be a promising research direction. For such multi-objective configuration tasks, having diverse populations of configurations is a necessity to understand the Pareto front.

2.2 Diversity Optimization

To address the objective of obtaining a set of diverse solutions, certain evolutionary algorithms have been designed specifically to converge to more than one solution in a single run. For example, the Niching Genetic Algorithms are applied for solving multimodal functions [8, 17] and searching diverse solutions of association rules [28], chemical structures [12], etc. Diversity optimization also addresses the problem of searching for multiple solutions. Quality-diversity optimization [11] was introduced to aim for a collection of well-performing and diverse solutions. The method proposed in [11] measures the quality of solutions based on their performance (i.e., quality) and distance to other solutions (i.e., novelty) dynamically. The novelty score of solutions is measured by the average distance of the \( k \)-nearest neighbors [22]. Also, to better understand the algorithm’s behavior and possible solutions, feature-based diversity optimization was introduced for problem instance classification [15]. A discrepancy-based diversity optimization was studied on evolving diverse sets of images and TSF instances [29]. The approaches in both studies measure the solutions regarding their features instead of performance. Unfortunately, the AC problem usually deals with mixed-integer search spaces, which are often not considered in the methods described in this section.

3 IRACE

In this section, we describe the outline of irace. Irace is an iterated racing method that has been applied for hyperparameter optimization problems in many domains. It samples configurations (i.e., hyperparameter values) from distributions that evolve along the configuration process. Iteratively, the generated configurations are tested across a set of instances and are selected based on a racing method. The racing is based on statistical tests on configurations’ performance for each instance, and elite configurations are selected from the configurations surviving from the racing. The sampling distributions are updated after selection. The distributions from sampling hyperparameter values are independent unless specific
conditions are defined. As a result, irace returns one or several elite configurations at the end of the configuration process.

### Algorithm 1: Algorithm Outline of irace

1. **Input:** Problem instances \( \Pi = \{\pi_1, \pi_2, \ldots\} \), parameter configuration space \( \Theta \), cost metric \( c \), and tuning budget \( B \);  
2. Generate a set of \( \Theta_1 \) sampling from \( \Theta \) uniformly at random;  
3. \( \Theta_{\text{elite}} = \text{Race}(\Theta_1, B_1) \);  
4. while The budget \( B \) is not used out do  
   5. \( j = j + 1 \);  
   6. \( \Theta_j = \text{Sample}(\Theta, \Theta_{\text{elite}}) \);  
   7. \( \Theta_{\text{elite}} = \text{Race}(\Theta_j \cup \Theta_{\text{elite}}, B_j) \);  
5. **Output:** \( \Theta_{\text{elite}} \)

Algorithm 1 presents the outline of irace [25]. irace determines the number of racing iterations \( N_{\text{iter}} = \lfloor 2 + \log_2(N_{\text{param}}) \rfloor \) before performing the race steps, where \( N_{\text{param}} \) is the number of parameters. For each Race(\( \Theta_j, B_j \)) step, the budget of the number of configuration evaluations \( B_j = (B - B_{\text{used}})/(N_{\text{iter}} - j + 1) \), where \( B_{\text{used}} \) is the used budget, and \( j = \{1, \ldots, N_{\text{iter}}\} \). After sampling a set of new configurations in each iteration, \( \text{Race}(\Theta, B) \) selects a set of elite configurations \( \Theta_{\text{elite}} \) (elites). New configurations are sampled based on the parent selected from elites \( \Theta_{\text{elite}} \) and the corresponding self-adaptive distributions of hyperparameters. Specific strategies have been designed for different types (numerical and categorical) of parameters.

Each race starts with a set of configurations \( \Theta_j \) and performs with a limited computation budget \( B_j \). Precisely, the candidate configurations of \( \Theta_j \) are evaluated on a single instance \( \pi_i \), and the configurations that perform statistically worse than at least another one will be discarded after being evaluated on a number of instances. Note that irace package provides multiple statistical test options for eliminating worse configurations such as F-test and t-test. The race terminates when the remaining budget is not enough for evaluating the surviving configurations on a new problem instance, or when \( N_{\text{min}} \) or fewer configurations survived after the test. At the end of the race, \( N_{\text{surr}} \) configurations remain survival and are ranked based on their performance. irace selects \( \min\{N_{\text{min}}, N_{\text{surr}}\} \) configurations with the best ranks to form \( \Theta_{\text{elite}} \) for the next iteration. Note that irace applies here a greedy elitist mechanism, and this is the essential step where our irace variants alter in this paper.

To avoid confusion, we note that an “elitist iterated racing” is described in the paper introducing the irace package [25]. The “elitist” there indicates preserving the best configurations found so far. The idea is to pretend “elite” configurations from being eliminated due to poor performance on specific problem instances during racing. We apply this “elitist racing” for our experiments in this paper, while non-elitist alternative methods are proposed for the selection of elites.

### 4 RANDOM SURVIVOR SELECTION

In this section, we test a irace variant, instead of using the greedy truncation mechanism, adopting the selection of elites by taking the best-performing configuration and randomly selecting \( N_{\text{iter}} - 1 \) distinct ones from the best \( \sigma N_{\text{min}} \) surviving configurations when \( N_{\text{surr}} > \sigma N_{\text{min}} \), for some \( \sigma > 1 \). The implementation of our variants is built on the default irace package [26].

#### 4.1 Tuning Scenario: ACOTSP

ACOTSP [32] is a package implementing ACO algorithms for the symmetric TSP. The package is configured with three categorical, four continuous, and four integer variables (11 in total). We apply irace in this paper to tune for configurations that obtain lower solution costs (fitness). The experimental results reported in the following are from 20 independent runs of irace. Each run of irace is assigned with a budget of 5000 runs of ACOTSP, and ACOTSP executes 20s of CPU-time per run following the suggestion in [25]. We set \( \sigma N_{\text{min}} = N_{\text{surr}} \) indicating that irace randomly selects survivor configurations to form elites (irace-rand). Other irace settings remain default: the “the elitist iterated racing” is applied, and \( N_{\text{min}} = 5 \). We apply the benchmark set of Euclidean TSP instances of size 2000 with 200 train and test instances, respectively.

We observe in Figure 1 that the performance of irace-rand is comparable to that of irace: We plot here the deviations of the best configurations obtained by each run from the best-found configuration obtained by 20 (80 in total) runs of the irace variants, regarding the average solution cost across 200 problem instances. The deviation median and mean of irace-rand is smaller than irace, while the performance variance among 20 runs increases.

Though irace is initially proposed for searching configurations that generally perform well across a whole set of problem instances, we are nevertheless interested in the performance of the obtained configurations on individual instances. By zooming into the instances, we expect to gain insights into the behavior of both configurators and the obtained configurations. Therefore, we plot in Figure 2 the performance of all obtained configurations on nine instances. The instances are generated using different random seeds, so the subset selected is unbiased here. Still, we observe comparable performance between irace and irace-rand. It is not surprising that the performance of irace-rand obtains larger variance because the configurations that do not perform the best get a chance to be selected. Interestingly, we spot significant improvement on instances “2000-6” and “2000-7”, where the configurations obtained by irace-rand generally perform closer to the optima, compared to irace. Moreover, we provide in Table 1 how often each configurator recommended the best-found configuration for each TSP instance.

|        | irace-rand | irace-entropy | irace-Gower |
|--------|------------|---------------|-------------|
| # configs | 95         | 73            | 82          |
| # best   | 1          | 12            | 172         |

Table 1: Total number of elite configurations that were obtained at the end of 20 independent runs of each of the four configurators on ACOTSP (100 max, since we use \( N_{\text{min}} = 5 \)) and number of test instances (out of 200) on which the best-performing configuration was suggested by each irace variant.
Figure 1: Deviations of the best obtained configurations from the optimum. Each dot indicates the result of a run of irace, which plots the average fitness of the best-found configurations across 200 TSP instances. Configurations are measured by the average result of 10 validation runs per instance. The “optimum” is the best-found configuration obtained by 20 (80 in total) runs of the plotted methods.

Figure 2: Performance deviation of the 314 obtained configurations from the best configuration found for the same instance.

4.2 Tuning Scenario: SPEAR

SPEAR [3] is a custom-made SAT solver configurable with 26 categorical parameters, of which nine are conditional, i.e., their activation depends on the values of one or several of the other parameters. Our goal here is to minimize the mean runtime of SPEAR. We run each irace variant 20 independent times. Each run of irace is assigned with a budget of 10,000 runs of SPEAR, and the maximal runtime of SPEAR is 30s CPU-time per run. Other irace settings remain default: the “elitist iterated racing” is applied, and $N_{\text{min}} = 6$. The training and test set are 302 different SAT instances, respectively [2]. Note that the number of survivor configurations is large (~250) during racing, and experimental results show that randomly selecting with such a large population deteriorates the performance of irace. Therefore, in this section we cap the size of survivor candidates by $2N_{\text{min}}$ to select from a relatively well-performing population.

We report here that both irace and irace-rand obtain 100 elite configurations in total at the end of 20 independents runs. The average runtime of the best-found configurations obtained by irace across 302 SAT instances is 0.902 ($sd = 1.54$), while the value for the configurations obtained by irace-rand is 1.18 ($sd = 2.13$). Moreover, irace obtains the best-performing configuration for 161 SAT instances, and irace-rand obtains the best ones for the remaining 141 instances. Though the averaged result of irace-rand is outperformed by irace, it achieves advantages on a decent amount of instances.

Therefore, we plot the runtime of irace and irace-rand for 30 (out of the 131) randomly picked instances of “hsat” in Figure 3 and for the 27 “winegcc” instances in Figure 4. According to Figure 4, the variance and median of the runtime obtained by irace-rand are large on instances such as “hsat_vc3632” and “hsat_vc3652”, while its performance remains comparable to irace on some instances such as “hsat_vc3641” and “hsat_vc3556”. Whereas, we observe in Figure 4 significant improvement using irace-rand for most of “winegcc” instances, especially “winegcc_vc1032,” “winegcc_vc1046,” “winegcc_vc1081,” etc.

Figure 3: Boxplots of the runtime (in log scale) of the SPEAR configurations for 30 “hsat” SAT instances.

Figure 4: Boxplots of the runtime (in log scale) of the SPEAR configurations for 27 “winegcc” SAT instances.

5 SELECTING DIVERSE ELITES

According to the results introduced in Section 4, we find that, while keeping the best configuration, randomly selecting from
well-performing survivor configurations to form elites has positive impacts on the performance of irace. irace-rand obtains better configurations for ACOTSP and shows improvements on subsets of tested SAT instances for SPEAR. An intuitive explanation is that irace-rand allows exploring search space around those non-elitist configurations, which matches our expectation following the motivation introduced in Section 1.

The encouraging performance of irace-rand motivates us to investigate if further improvements can be achieved through an explicit consideration of the diversity of the survivors in the selection of the elite configurations. To this end, we study in this section two alternative selection strategies, one building on entropy as diversity measure [7] and the other using Gower’s distance [18] as selection criteria.

### 5.1 Maximizing Population Entropy

In information theory, entropy represents random variables’ information and uncertainty level [31]. The larger the entropy, the more information the variables deliver, i.e., the more diverse the solutions are. Our irace-entropy configurator makes use of this idea, by using the Shannon entropy as criterion for selecting elites.

For a random variable \(X \) with distribution \( P(X) \), the normalized entropy of \(X \) is defined as:

\[
H(X) = \sum_{i=1}^{n} P(X_i) \log P(X_i)/\log(n),
\]

In this paper, we estimate the entropy of integer and categorical variables from the frequencies of each value. For continuous variables, the values are discretized into bins, and entropy is estimated based on the counts of each bin. The domain of a continuous variable is equally divided into \(n \) bins, where \(n \) is the number of observations (i.e., configurations). Finally, we calculate the diversity level \(D(\Theta)\) of a set of configuration \(\Theta\) using the mean entropy across \(p\) variables (i.e., parameters), which is defined as:

\[
D(\Theta) = \frac{\sum_{j=1}^{p} H(\Theta^j)}{p}, \quad \Theta^j = \{\theta^j_1, \theta^j_2, \ldots, \theta^j_p\}
\]

We introduce a variant of irace (irace-entropy) maximizing \(D(\Theta^{\text{elite}})\) for each race step. Recall that \(N^{\text{surv}}\) configurations survive at the end of race, and the \(N^{\min}\) best-ranked configurations are selected to form \(\Theta^{\text{elite}}\) in Algorithm 1 (line 7). irace-entropy adapts this step by selecting a subset of configurations \(\Theta\) with the maximal \(D'(\Theta)\), where \(|\Theta| = N^{\min}\) and the best-ranked configuration \(\Theta^*\) ∈ \(\Theta\). In practice, we adjust the greedy truncation selection in Algorithm 1 (line 7) by Algorithm 2.

### 5.2 Maximizing Gower’s Distance

This section introduces a heuristic of another alternative selection for irace (irace-Gower). irace-Gower also starts from a set of survivor configurations, which iteratively adds the furthest configurations until the elite population reaches the limited size. We apply here Gower’s distance [18] to measure the distance between individual configurations and populations.

Gower’s distance is a metric to measure the dissimilarity of two items. It has been widely applied in pattern configuration domain due to its advantages on mixed-type (numeric and non-numeric) data [9, 30], perfectly matching our requirement of measuring distance among configurations with mixed-integer variables. The Gower’s distance of two items (i.e., configurations) is defined as below:

\[
S(\theta_i, \theta_k) = \frac{\sum_{j=1}^{p} s^j_{ik}}{\sum_{j=1}^{p} s^j_{ik}}, \quad s^j_{ik} = \frac{|\theta^j_i - \theta^j_k|}{R^j}
\]

where \(p\) is the number of features of the items (i.e., parameters of the configurations), \(s^j_{ik}\) defines the score of two features according to their types. For categorical variables, \(s^j_{ik} = 1\) if \(\theta^j_i = \theta^j_k\), otherwise, \(s^j_{ik} = 0\). For continuous and integer variables, \(s^j_{ik} = |\theta^j_i - \theta^j_k|/R^j\), where \(R^j\) is the range of the \(j\)-th feature. \(s^j_{ik} = 1\) if \(\theta^j_i\) and \(\theta^j_k\) are comparable along with the \(j\)-th feature. \(s^j_{ik} = 0\) for the uncomparable cases, for example when “Null” values are present, or when the conditional parameters are not activated.

In practice, irace-Gower replaces the greedy truncation selection in Algorithm 1 (line 7) by the one defined in Algorithm 3. First, the best-ranked configuration \(\Theta^*\) is added to the elite configurations \(\Theta^{\text{elite}}\). Then the furthest configurations \(\Theta^\prime\) are added to \(\Theta^{\text{elite}}\) until \(N^{\min}\) configurations are grouped. We measure the distance \(S(\theta, \Theta)\) between a population and a configuration by the Gower’s distance \(S(\theta^{\text{mean}}, \Theta)\) between the mean configuration of \(\Theta\) and \(\theta\). We generate the mean configuration \(\theta^{\text{mean}}\) as follows: the values of categorical variables are the modes of the population. If multiple values appear most frequently, one of the values is sampled uniformly at random; the values of continuous and integer variables are the mean values of the population.

### 5.3 Experimental Results

Due to the promising results of irace-rand introduced in Section 4.1, we continue testing irace-entropy and irace-Gower for tuning ACOTSP in this section. All the settings remain the same as reported in Section 4.1 except the introduced alternative selection methods.

#### 5.3.1 Performance analyses

Figure 1 compares the best configurations until the elite population reaches the limited size. We continue testing irace-entropy and irace-Gower for tuning ACOTSP and shows improvements on subsets of tested SAT instances for SPEAR. An intuitive explanation is that irace-rand allows exploring search space around those non-elitist configurations, which matches our expectation following the motivation introduced in Section 1.

**Algorithm 2: Entropy-maximization selection**

1. **Input:** A set of ranked configurations \(\Theta^{\text{surv}}\), the maximal size \(N^{\min}\) of \(\Theta^{\text{elite}}\).
2. If \(|\Theta^{\text{surv}}| \leq N^{\min}\) then
3. \(\Theta^{\text{elite}} = \Theta^{\text{surv}}\)
4. Else
5. \(\Theta^{\text{elite}} = \{\Theta^*\}, \Theta^{\text{surv}} = \Theta^{\text{surv}}\setminus\{\Theta^*\}\), where \(\Theta^*\) ∈ \(\Theta^{\text{surv}}\) is the best-ranked;
6. \(\Theta^{\text{elite}} = \Theta^{\text{elite}} \cup S^\star\), where
   \[
   S^\star = \arg \max_{S \subset \Theta^{\text{surv}}, |S| = N^{\min} - 1} D(\Theta^{\text{elite}} \cup S)
   \]
7. **Output:** \(\Theta^{\text{elite}}\)
Algorithm 3: An alternative selection for elites using the largest Gower’s distance

1. **Input**: A set of ranked configurations $\Theta^{\text{surv}}$, the maximal size $N^{\text{min}}$ of $\Theta^{\text{elite}}$
2. **if** $|\Theta^{\text{surv}}| \leq N^{\text{min}}$ **then**
3. $\Theta^{\text{elite}} = \Theta^{\text{surv}}$
4. **else**
5. $\Theta^{\text{elite}} = \{0^*\}, \Theta^{\text{surv}} = \Theta^{\text{surv}} \setminus \{0^*\}$, where $0^* \in \Theta^{\text{surv}}$ is the best-ranked;
6. **while** $|\Theta^{\text{elite}}| < N^{\text{min}}$ **do**
7. $\Theta^{\text{elite}} = \Theta^{\text{elite}} \cup \{0'\}$, where $0' = \arg \max_{\theta \in \Theta^{\text{surv}}} S(\Theta^{\text{elite}}, \theta)$
8. **Output**: $\Theta^{\text{elite}}$

By using the heuristic in Algorithm 3 to select the configurations with the largest Gower’s distances, irace-Gower obtains better results in many runs, whereas the performance of some runs deteriorates. However, we observe improvement for irace-entropy achieving minor deviations from the optimum and more robust results than irace-rand.

Regarding the results on individual problem instances in Figure 2, irace-entropy shows significant advantages compared to the other three methods. Irace-Gower shows comparable performance and obtains better solutions for some instances. The variance of solution costs obtained by irace-Gower is significant for some instances such as “2000-6”, which matches our expectation because it is designed as exploring configurations with more considerable distances. In addition, we observe significant advantages of irace-entropy compared to the other methods across all the plotted instances.

Overall, these results indicate that the non-elitist survivor selection can be a meaningful alternative to the elitist selection used by irace.

5.3.3 Parameter values of the obtained configurations. We recall that the motivation of this work is to avoid premature convergence of irace via diversity-preserving selection rules. In the previous sections, we have observed that the performance of irace was improved on some ACOTSP instances, but not on all. The overall impact on performance is rather mild across all instances. However, the diversity-based selection may bear an additional advantage in that it may recommend more diverse solutions at the end of the entire configuration run. Such diversity could be beneficial for an a posteriori investigation into the impact of different variables on the performance. For example, one may conjecture that a variable with high diversity of the final elites has less impact on the overall performance than variables where all configurations are very similar. To analyze this claim, we investigate how diverse the obtained solutions are when applying the different selection mechanisms.

From the 20 runs of each of the four irace variants we obtain a total number of 314 configurations for ASOTSP. Note here that $N^{\text{min}} = 5$, but we do not always obtain five configurations for each run because for any given race we could have $N^{\text{surv}} < N^{\text{min}}$. Figure 5 plots the smoothed histograms of seven numeric parameters from the 314 ACOTSP configurations. Within these 314 configurations, we find some clear commonalities in their categorical parameters: for localsearch, option “3” is always selected, while dbi is always “1”. For the choice of algorithm, a significant majority of the configurations (297) use ascs, followed by 14 ras and 3 with mmas. Because of this, these parameters are excluded in Figure 5.

According to Figure 5, all the irace variants share similar distributions for nnls. Irace-Gower is the only method that obtains algorithm = ras and meets the condition of rsrcs. Therefore, a line is drawn for rsrcs. Irace shows notable peaks for all parameters except rsrcs. However, the other methods show relatively flat for some parameters, which supplies evidence for diverse configurations, for examples, the distributions of alpha, beta, and q0 for irace-entropy and the distribution of rho for irace-Gower. Another interesting observation is that the peaks locate differently for the irace variants. For example, the mode of beta is around 2.5 for irace-Gower but around 5 for irace and irace-rand. Also, the mode of rho is around 0.35 for irace, but the peaks present in the distribution curve of irace-entropy.

Many differences can be observed in Figure 5 for the distributions of the configurations obtained by the irace variants. However, the dissimilarity can be explained by either diverse configurations obtained for each run or different behavior in individual runs of the irace variants. In other words, we need to confirm this is not the case that (1) the obtained configurations converge to different values in individual runs and (2) the distributions plotted in Figure 5 reflect the dissimilarity of each run’s result instead of the general distributions for independent runs. Therefore, we also present configurations obtained by four individual runs in Figure 6. We observe in Figure 6 that the obtained configurations do not converge to particular values. On the contrary, the observations of the configurations distribute following the curves plotted in Figure 5.

Based on Figures 5 and 6, we conclude that the distributions of the configurations obtained by irace alter by using different selection methods of constructing elites, consequently delivering different performances. Moreover, we observe diverse configurations (e.g., flat curves) obtained by the three variants compared to irace.

5.3.3 Learning from Algorithm Configuration. We have gained improvement by applying non-elitist selections and investigated the dissimilarity among the distributions of the configurations obtained by the irace variants. The following question is: What is the relationship between algorithms’ parameters and behavior? Fortunately, we can apply the AC problem to learn the answer to this question.

Figure 7 presents the obtained configurations by polyline with vertices corresponding to the parameter values on the x-axis lines. The color of the configuration lines are scaled by the deviation $\frac{f^* - f}{f^*}$ from the optimum, where $f$ is the average solution cost across 200 problem instances. Green lines indicate better configurations than red ones. We process “Null” values of the conditional parameter as 0.

Firstly, we observe that irace-entropy performs the best with the most competitive configurations in green lines while the performance of the configurations obtained by irace cluster together. Irace-Gower and irace-rand obtain wide ranged configurations in terms of deviations. This observation matches our expectation that they are designed for exploring diverse configurations that do not
Figure 5: Probability density functions of the four continuous (alpha, beta, rho, and q0) and three integer (ants, nnls, and rasrank) parameters of ACOTSP obtained by the irace variants. q0 is activated with algorithm = acs, while rasrank is activated with algorithm = ras. Results are from 314 configurations (95, 73, 82, and 64 for irace, irace-rand, irace-entropy, and irace-Gower, respectively).

Figure 6: Plot grids for distributions of the maximal five configurations obtained by individual runs of the irace variants. The configurations obtained by each run are presented by row (labeled by 101-104), and distributions for each parameter are listed by line. Subplots in each grid plot the distribution for the corresponding parameter by dividing the x-axis (parameter domain) into 20 bins and counting the number of observations (i.e., configurations) in each bin (y-axis).

necessarily perform the best. As a result, this design helps them find new well-performing configurations. Secondly, regarding the parameters of the obtained configurations, the range of beta and q0 are narrower for irace compared to the other methods. However, the configurations with beta > 8 and q0 > 0.9, outside the range obtained by irace, generally perform well. Recall that q0 = 0 indicates "Null" values not reflecting scales, which indicates the configurations do not apply algorithm = acs. The results of irace-Gower indicates algorithm = acs is not a good option. We will not research how the parameter values practically affect the performance of ACOTSP since it is beyond the topic of this paper.

Nonetheless, Figure 7 still provides evidence of applying the AC problem to investigate the behavior of algorithms, especially when diverse configurations are obtained.

6 CONCLUSIONS AND FUTURE WORK
We have demonstrated in this paper that randomly selecting survivor configurations can improve the performance of irace, both for tuning ACO on TSP and for tuning SPEAR to minimize the runtime of a SAT solver. Moreover, we have proposed two alternative selection methods to form diverse elites, applying Shannon entropy as the diversity metric and Gower’s distance to measure the
dissimilarity between configurations. Experimental results show significant advantages of maximizing the entropy for selecting elites. Finally, benefiting from the obtained configurations, we discussed the promising parameter settings that lead to better performance of ACOTSP.

Since we have shown that even a completely random selection procedure can potentially lead to irace returning better performing configurations than the elitist selection, there seems to be a lot of room for further study. While the two non-elitist selection methods present improvement in the performance of irace via exploring diverse configurations, we did not modify the procedure of sampling new configurations. Nevertheless, we believe effectively generating diverse configurations can be beneficial and shall be studied for future work.

Apart from boosting the performance of irace via focusing more on diversity, we can find a diverse portfolio of well-performing algorithm configurations while keeping the benefits of the iterated racing approach, by changing the objective of the tuning from finding the best performing configuration. In the context of algorithm selection, such approaches are studied under the notion of algorithm portfolio selection [24].

REFERENCES
[1] Anonym, N. Data Sets for the study “Non-Elitist Selection among Survivor Configurations can Improve the Performance of Irace”. Link blinded for double-blind reviewing, Feb. 2022.
[2] Babic, D., and Hu, A. J. Structural abstraction of software verification conditions. In Proc. of International Conference on Computer Aided Verification (2007), Springer, pp. 366–378.
[3] Babic, D., and Hutter, F. Spear theorem prover. Solver description, SAT competition 2007 (2007).
[4] Back, T., Fonseca, C., and Kaushik, P. Contemporary Evolution Strategies. Natural Computing Series. Springer, 2013.
[5] Bartcz-Bieleski, T. SPOT: an R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. CoRR abs/1006.6045 (2010).
[6] Basmer, M., and Kehrer, T. Encoding adaptability of software engineering...
