Non-elitist Selection Can Improve the Performance of Irace

Furong Ye[1](✉), Diederick Vermetten[1], Carola Doerr[2], and Thomas Bäck[1]

[1] LIACS, Leiden University, Leiden, The Netherlands
{f.ye,d.l.vermetten,t.h.w.baeck}@liacs.leidenuniv.nl
[2] Sorbonne Université, CNRS, LIP6, Paris, France
carola.doerr@lip6.fr

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, making use of iterated statistical tests to discard poorly performing configurations. At the end of the race, a set of *elite* configurations are selected from those *survivor* configurations that were not discarded, using greedy truncation selection. We study two alternative selection methods: one keeps the best survivor and selects the remaining configurations uniformly at random from the set of survivors, while the other applies entropy to maximize the diversity of the elites. These methods are tested for tuning ant colony optimization algorithms for traveling salesperson problems and the quadratic assignment problem and tuning an exact tree search solver for satisfiability problems. The experimental results show improvement on the tested benchmarks compared to the default selection of irace. In addition, the obtained results indicate that non-elitist can obtain diverse algorithm configurations, which encourages us to explore a wider range of solutions to understand the behavior of algorithms.

Keywords: Parameter tuning · Algorithm configuration · Black-box optimization · Evolutionary computation

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 [14], SMAC [13], SPOT [3], MIP-EGO [29], and irace [19] have been applied to problem domains such as combinatorial optimization [19], software engineering [4], and machine learning [15].

Irace, one of the most popular tools, has shown its ability to improve the performance of the algorithms for various optimization problems [2,7,19,27]. However, we can still intuitively expect to improve the performance of irace considering contemporary optimization techniques. Premature convergence is a common problem for optimization methods resulting in being trapped into local optima, which can also present irace from finding the optimal configurations. For example, irace fails to find the optimal configuration of a family of genetic algorithms (GAs) for OneMax in [30]. There exists more than one type of competitive configuration of the GA for OneMax, which is known due to the extensive body of theoretical work [11,28]. However, irace converges to a specific subset of configurations that share similar algorithm characteristics. 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 explore search space around a diverse set of well-performing configurations to avoid converging on one specific type of configuration.

A "soft-restart" mechanism has been introduced for irace to avoid premature convergence in [19], which partially reinitializes the sampling distribution for the configurations that are almost identical to others. However, evaluations can be wasted on testing similar configurations before the restart, and the configuration may converge on the type of configurations that were found before restarting. Therefore, we investigate alternative selection mechanisms which take into account the diversity of the selected elite configurations. In addition, the observations from [30] inspire a discussion on searching for various competitive configurations with different patterns, which is addressed by our discussion that more knowledge can be obtained by searching diverse configurations.

1.1 Our Contributions

In this paper, we show that an alternative random selection of elites can result in performance benefits over the default selection mechanism in irace. Moreover, we propose a selection operator maximizing the entropy of the selected elites. These alternative selection operators are compared to default irace on the tested scenarios.

The alternative approaches are tested on three scenarios: tuning the Ant Colony Optimization (ACO) algorithm for the traveling salesperson problem (TSP) and the quadratic assignment problem (QAP) and minimizing the computational cost of the SPEAR tool (an exact tree search solver for the satisfiability (SAT) problem).

Experimental results show that (1) randomly selecting elites among configurations that survived the racing procedure performs better than the greedy truncation selection, and (2) the irace variant that uses the entropy metric obtains diverse configurations and outperforms the other approaches. Finally, the obtained configurations encourage us to (3) use such a diversity-enhancing approach to find better configurations and understand the relationship between parameter settings and algorithm behavior for future work.

Reproducibility: We provide the full set of logs from the experiments described in this paper in [31]. Additionally, our implementation of the modified irace versions described in this paper is available at https://github.com/FurongYe/irace-1.

2 Related Work

2.1 Algorithm Configuration

Traditionally, the AC problem, as defined below, aims at finding a *single* optimal configuration for solving a set of problem instances [9].

Definition 1 (Algorithm Configuration Problem). *Given a set of problem instances Π, a parametrized algorithm A with parameter configuration space Θ, 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, or mixed-integer. In addition, some parameters can be conditional.

Many configurators have been proposed for the AC problem [3,13,14,17,19,29], and they usually follow Definition 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. For example, previous studies [21,30] 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 multimodular functions [6,12] and searching diverse solutions of association rules [24], chemical structures [23], etc. Diversity optimization also addresses the problem of searching for multiple solutions. Quality-diversity optimization [8] was introduced to aim for a collection of well-performing and diverse solutions. The method proposed in [8] 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 [16]. Also, to better understand the algorithm's behavior and possible solutions, feature-based diversity optimization was introduced for problem instance classification [10]. A discrepancy-based diversity optimization was studied on evolving diverse sets of images and TSP instances [25]. The approaches in both studies measure the solutions regarding their features instead of performance. Unfortunately, the AC problem usually deals with a mixed-integer search space, which is 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 $X$,
  cost metric $c$, and tuning budget $B$;
2 Generate a set of $\Theta_1$ sampling from $X$ 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     $\Theta_j = \text{Sample}(X, \Theta^{\text{elite}})$;
7     $\Theta^{\text{elite}} = \text{Race}(\Theta_j \cup \Theta^{\text{elite}}, B_j)$;
8 Output: $\Theta^{\text{elite}}$
```

Algorithm 1 presents the outline of irace [19]. 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^{param} is the number of parameters. For each Race(Θ_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_{used} is the used budget, and $j = \{1, \ldots, N^{\text{iter}}\}$. After sampling a set of new configurations in each iteration, Race(Θ, B) selects a set of elite configurations Θ^{elite} (elites). New configurations are sampled based on the parent selected from elites Θ^{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 Θ_j and performs with a limited computation budget B_j. Precisely, each candidate configuration of Θ_j is evaluated on a single instance π_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 the irace package provides multiple statistical test options for eliminating worse configurations such as the F-test and the t-test. The race terminates when the remaining budget is not enough for evaluating the surviving configurations on a new problem instance, or when $N^{\min}$ or fewer configurations survived after the test. At the end of the race, N_j^{surv} configurations survive and are ranked based on their performance. Irace selects $\min\{N^{\min}, N_j^{\text{surv}}\}$ configurations with the best ranks to form Θ^{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 [19]. The "elitist" there indicates preserving the best configurations found so far. The idea is to prevent "elite" configurations from being eliminated due to poor performance on specific problem instances during racing, and the best surviving "elite" configurations are selected to form Θ^{elite}. We apply this "elitist racing" for our experiments in this paper, while the alternative methods select diverse surviving "elite" configurations instead of the best ones.

4 Random Survivor Selection

To investigate the efficacy of the greedy truncation selection mechanism used by default within irace, we compare the baseline version of irace to a version of irace that uses a random selection process. In particular, we adopt the selection of elites by *taking the best-performing configuration and randomly selecting the remaining* $N_j^{\text{surv}} - N^{\min} - 1$ distinct ones from the best $\sigma N^{\min}$ surviving configurations when $N^{\text{surv}} \geq \sigma N^{\min}$, for some $\sigma \geq 1$. The implementation of our variants is built on the default irace package [20].

4.1 Tuning Scenario: ACOTSP

ACOTSP [27] is a package implementing ACO for the symmetric TSP. We apply irace variants in this paper to configure 11 parameters (three categorical, four continuous, and four integer variables) of ACO for lower solution costs (fitness). The experimental results reported in the following are from 20 independent runs of each irace variant. Each run is assigned with a budget of $5,000$ runs of ACOTSP, and ACOTSP executes 20s of CPU-time per run following the suggestion in [19]. We set $\sigma N^{\min} = N^{\text{surv}}$ indicating that the irace variant, irace-rand, randomly selects survivor configurations to form elites. Other settings remain as default: the "elitist iterated racing" is applied, and $N^{\min} = 5$. We apply the benchmark set of Euclidean TSP instances of size $2,000$ with 200 train and 200 test instances.

Figure 1 plots the deviations of the best configurations, which are obtained by each run, from the best-found (optimum) configuration obtained by 20 (60 in total) runs of the irace variants. The results are averaged across 200 TSP instances. We observe that the median and mean of irace-rand results are smaller than those of irace, but the performance variance among these 20 irace-rand runs is significantly larger.

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. Therefore, we plot in Fig. 2 the performance of all obtained configurations on nine instances. Still, we observe comparable performance between irace and irace-rand. It is not surprising that the performance of irace-rand presents larger variance because the configurations that do not perform the best get a chance to be selected. Moreover, we spot significant improvement on instances "2000-6" and "2000-9", on which the configurations obtained by irace-rand generally perform closer to the optima, compared to irace.

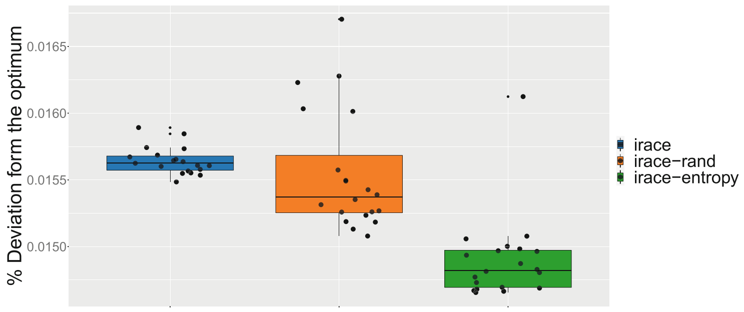


Fig. 1. Average deviation from the optimum of the best obtained configurations. Each dot corresponds to the best final elite obtained by a run of irace, which plots the average deviation from the best-found fitness across 200 TSP instances. Configurations are measured by the average result of 10 validation runs per instance. The "optimum" for each instance is the best-found configuration obtained by 20 (60 in total) runs of the plotted methods.

4.2 Tuning Scenario: ACOQAP

We apply the same ACO implementation in [19] for solving QAP [22]. ACOQAP executes 60s CPU-time per run following the default setting of the package, and we apply the benchmark set of 50 train and test instances, respectively. The other settings remain the same with the ACOTSP scenario.

Unfortunately, we do not observe similar improvement of using irace-rand for ACOQAP. Irace-rand present worse performance than irace, comparing the

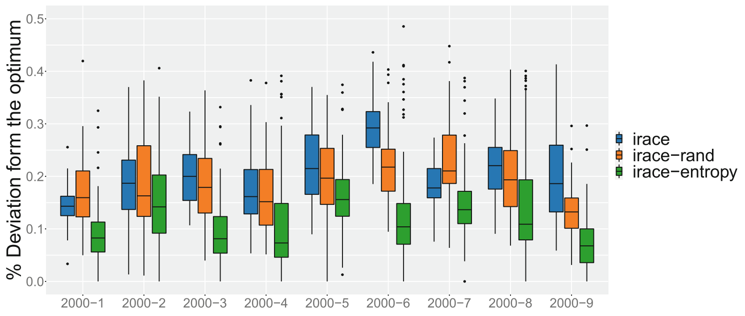


Fig. 2. Boxplots of the deviation from the optimum of the obtained configurations for TSP instances. Results are from the average fitness of 10 validation runs for each obtained configuration.

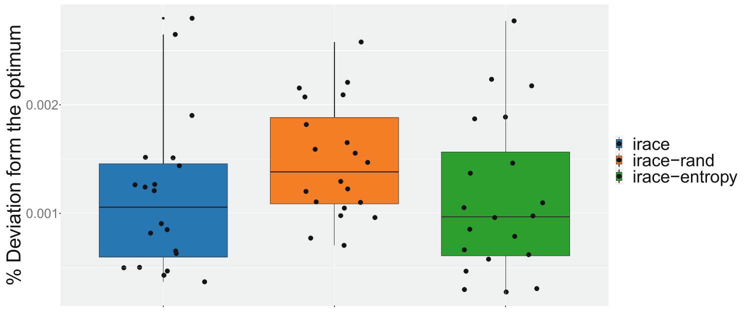


Fig. 3. Average deviation from the optimum of the best obtained configurations. Each dot corresponds to the best final elite obtained by a run of irace, which plots the average deviation from the best-found fitness across 50 QAP instances. Configurations are measured by the average result of 10 validation runs per instance. The "optimum" is the best-found configuration obtained by 20 (60 in total) runs of the plotted methods.

average results across 50 instances in Fig. 3. While looking at Fig. 4, which plots the results on ten randomly picked instances, we do not observe improvement using irace-rand on ACOQAP, either. These observations indicate that using this random selection to select elite configurations may deteriorate the performance of irace for ACOQAP, though it does not necessarily mean that diverse configurations are not helpful for the configuring process. We will discuss this topic in more detail in Sect. 5.

4.3 Tuning Scenario: SPEAR

SPEAR [2] 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 runtime of SPEAR. We run each irace variant 20 independent times. Each run of irace

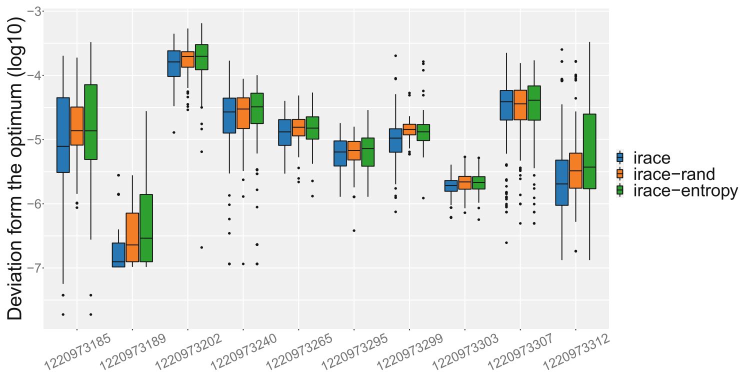


Fig. 4. Boxplots of the deviation from the optimum of the obtained configurations for ACOQAP instances. Results are from the average fitness of 10 validation runs for each obtained configuration.

is assigned with a budget of 10 000 runs of SPEAR, and the maximal runtime of SPEAR is 30 s CPU-time per run. Other irace settings remain default: the "elitist iterated racing" is applied, and $N^{\min} = 6$. The training and test set are 302 different SAT instances, respectively [1]. 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, for this scenario, we cap the size of survivor candidates by $2N^{\min}(\sigma = 2)$ to select from a relatively well-performing population.

Overall, we observe that the performance difference between the two methods is tiny for most instances, though irace-rand can not obtain better average results of runtime across all tested instances than irace. Note that the obtained configurations may use much runtime (∼30 s) for a few instances, resulting in the comparison among the average runtime (∼3 s) across all instances can be significantly affected by the results on those particular instances. Therefore, we plot only the runtime for the first two instances of each class of instances in Fig. 5. Compared to irace, though the performance of irace-rand deteriorates on "itox" instances, significant improvements using irace-rand can be observed on more instances such as "dspam_vc9400", "winedump" instances, and "xinetd_vc56633".

5 Selecting Diverse Elites

The optimistic results of ACOTSP and SPEAR scenarios introduced in Sect. 4 indicate that, while keeping the best configuration, randomly selecting from well-performing survivor configurations to form elites can have positive impacts on the performance of irace. An intuitive explanation is that irace-rand allows exploring search space around those *non-elitist* configurations to avoid premature convergence on specific types of configurations, which matches our expectation following the motivation introduced in Sect. 1. However, the failure to

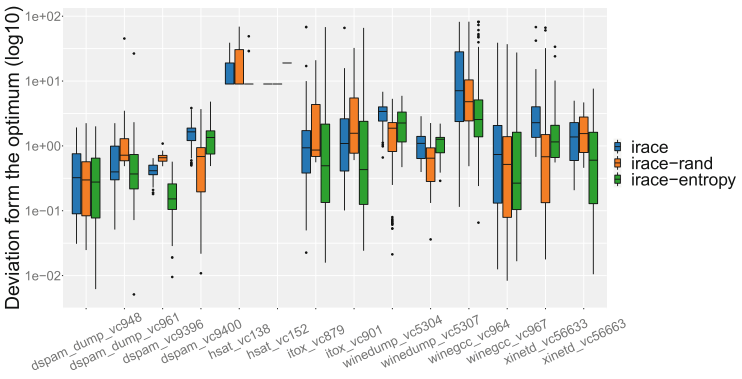


Fig. 5. Boxplots of the deviation from the optimum of the obtained configurations for SPEAR instances. Results are from the average fitness of 10 validation runs for each obtained configuration. Results of "gzip" class is omitted because runtime of all obtained configurations are identical.

achieve improvements for ACOQAP requires us to consider explicitly controlling the selected elite configurations' diversity. To this end, we study an alternative selection strategy based on entropy [5] as a diversity measure.

5.1 Maximizing Population Entropy

In information theory, entropy represents random variables' information and uncertainty level [26]. The larger the entropy, the more information the variables deliver, e.g., the more diverse the solutions are. Our irace-entropy configurator makes use of this idea, by using the Shannon entropy as criterion for selecting survivor configurations to form 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 probability of each value. For continuous variables, the values are discretized into bins, and entropy is estimated based on the counts of each bin. Precisely, 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 Θ using the mean entropy across p variables (i.e., parameters), which is defined as:

$$D(\Theta) = \frac{\sum_{j=1}^{p} H(\Theta^j)}{p}, \quad \Theta^j = \{\theta_1^j, \theta_2^j, \ldots, \theta_n^j\}$$

We introduce a variant of irace (irace-entropy) maximizing $D(\Theta^{\text{elite}})$ for each race step. Recall that N^{surv} configurations survive at the end of race, and the

$N^{\min}$ best-ranked configurations are selected to form Θ^{elite} in Algorithm 1 (line 7). Irace-entropy adapts this step by selecting a subset of configurations Θ with the maximal $D^*(\Theta)$, where $|\Theta| = N^{\min}$ and the best-ranked configuration $\theta^* \in \Theta$. In practice, we replace the greedy truncation selection in Algorithm 1 (line 7) with Algorithm 2. Note that we do not explicitly handle conditional parameters.

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 $|\Theta^{surv}| \leq N^{min}$ then
3     $\Theta^{\text{elite}} = \Theta^{\text{surv}}$
4 else
5     $\Theta^{\text{elite}} = \{\theta^*\}$, $\Theta^{\text{surv}} = \Theta^{\text{surv}} \setminus \{\theta^*\}$, where $\theta^* \in \Theta^{\text{surv}}$ is the best-ranked;
6     $\Theta^{\text{elite}} = \Theta^{\text{elite}} \cup S^*$, where $S^* = \underset{S \subset \Theta^{\text{surv}}, |S| = N^{\min}-1}{\arg\max} D(\Theta^{\text{elite}} \cup S)$
7 Output: $\Theta^{\text{elite}}$
```

5.2 Experimental Results

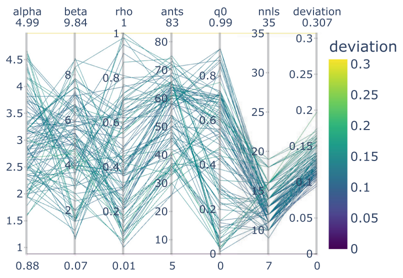
We present the results of irace-entropy in this section. All the settings remain the same as reported in Sect. 4 while applying the introduced alternative selection method.

For ACOTSP, we observe in Fig. 1 that irace-entropy performs better than irace and irace-rand, obtaining significantly smaller deviations from the optimum than those of irace for 19 out of 20 runs. Regarding the results on individual problem instances, irace-entropy also shows in Fig. 2 significant advantages against irace and irace-rand across all the plotted instances.

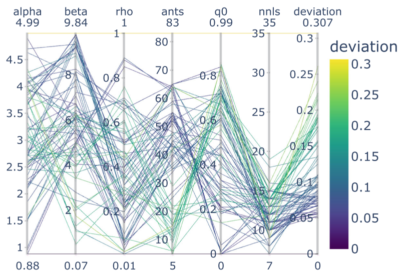
Recall that, for ACOQAP, the performance of irace-rand deteriorates compared to irace by randomly selecting survivor configurations to form elites. However, through using entropy as the metric to control the diversity explicitly, irace-entropy shows comparable results to irace in Fig. 3, obtaining a smaller median of deviations from the optimum for 20 runs. We also observe that the performance of irace-entropy is comparable to irace for individual instances in Fig. 4. In addition, irace-entropy can obtain the best-found configurations for some instances such as “1220973202” and “1220973265”.

For SPEAR, we observe in Fig. 5 that irace-entropy outperforms irace. For 12 out of the 14 plotted instances, irace-entropy obtains better median results than irace. Moreover, irace-entropy achieves improvements compared to irace-rand for most instances. Especially for the “itox” instances, in which irace-rand does not perform as well as irace, irace-entropy obtains better results while also keeping the advantages over irace on other instances.

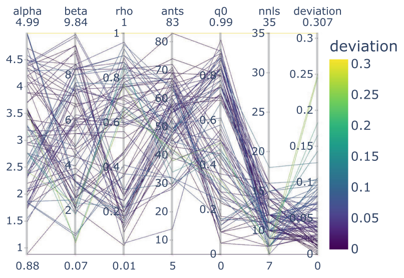
According to these results, we conclude that non-elitist selection can help improve the performance of irace. By using entropy as the metric to maximize the diversity of the selected elite configurations, irace-entropy achieves improvements compared to irace. However, irace-entropy does not obtain significant advantages against irace for ACOQAP and performs worse than irace-rand on some SPEAR instances, indicating potential improvements for non-elitist selection through further enhancements in regards to controlling the diversity of elites for specific problem instances.



(a) irace



(b) irace-rand



(c) irace-entropy

Fig. 6. Parameter values and deviations from the optimum of the ACOTSP configurations obtained by the irace variants. Each configuration is represented as a polyline with vertices on the parallel axes. The point of the vertex on each x-axis corresponds to the parameter value. We process "Null" values of the conditional parameter as 0. The color of the lines indicates the deviation of their average solution costs across all tested instances from that of the best-found one. Darker lines indicate better configurations.

5.3 Benefits from Diverse Configurations

While the irace variants, i.e., irace-rand and irace-entropy, achieve improvements by using non-elitist selection, significant variances are noticeable in Figs. 2, 4, and 5 for results of the obtained configurations. Recall that AC techniques have been applied in [30] for exploring promising configurations of the GA on diverse problems. Apart from analyzing a single optimal configuration, such benchmarking studies can also benefit from diverse configurations to investigate algorithms' performance with specific parameter settings. Therefore, we illustrate in this section that non-elitist selection can not only improve the performance of irace but also help understand the behavior of algorithms.

Using ACOTSP as an example, we show the configurations obtained by each irace variant in Fig. 6. 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.

We observe that irace-entropy obtains most of the competitive configurations while covering a wider range of performance with deviations from 0 to 0.25. However, the performance of the configurations obtained by the irace cluster in a range of deviations from 0.1 to 0.2. Moreover, 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.

We will not investigate how the parameter values practically affect the performance of ACOTSP since it is beyond the scope of this paper. Nonetheless, Fig. 6 provides evidence that irace-entropy can provide more knowledge concerning the distribution of obtained performance (i.e., fitness) and parameter values, which is helpful for understanding algorithms' behavior.

6 Conclusions and Discussions

In this paper, we have demonstrated that randomly selecting survivor configurations can improve the performance of irace, as illustrated on the cases of tuning ACO on TSP and tuning SPEAR to minimize the runtime of a SAT solver. Moreover, we have proposed an alternative selection method to form diverse elite configurations, using Shannon entropy as the diversity metric. Experimental results show significant advantages of maximizing entropy in this way.

While the irace-entropy presents improvement in the performance of irace via exploring diverse configurations in all the tested scenarios, irace-rand obtain better configurations for specific SPEAR instances. Therefore, there is still room for further study of incorporating diversity into the selection operators. More in-depth analysis on a wider set of algorithm configuration problems can help us better understand the benefits of considering diversity in selection. In addition, 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 to find a diverse portfolio of well-performing algorithm configurations. In the context of algorithm selection, such approaches are studied under the notion of *algorithm portfolio selection* [18].

References

1. Babić, D., Hu, A.J.: Structural abstraction of software verification conditions. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 366–378. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_41
2. Babic, D., Hutter, F.: Spear theorem prover. Solver description, SAT competition 2007 (2007)
3. Bartz-Beielstein, T.: SPOT: an R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. CoRR abs/1006.4645 (2010)
4. Basmer, M., Kehrer, T.: Encoding adaptability of software engineering tools as algorithm configuration problem: a case study. In: Proceedings of International Conference on Automated Software Engineering Workshop (ASEW 2019), pp. 86–89. IEEE (2019)
5. Bromiley, P., Thacker, N., Bouhova-Thacker, E.: Shannon entropy, Renyi entropy, and information. In: Statistics and Information Series, pp. 1–8 (2004)
6. Cavicchio, D.: Adaptive search using simulated evolution. Ph.D. thesis, University of Michigan (1970)
7. Cintrano, C., Ferrer, J., López-Ibáñez, M., Alba, E.: Hybridization of racing methods with evolutionary operators for simulation optimization of traffic lights programs. In: Zarges, C., Verel, S. (eds.) EvoCOP 2021. LNCS, vol. 12692, pp. 17–33. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72904-2_2
8. Cully, A., Demiris, Y.: Quality and diversity optimization: a unifying modular framework. IEEE Trans. Evol. Comput. **22**(2), 245–259 (2017)
9. Eggensperger, K., Lindauer, M., Hutter, F.: Pitfalls and best practices in algorithm configuration. J. Artif. Intell. Res. **64**, 861–893 (2019)
10. Gao, W., Nallaperuma, S., Neumann, F.: Feature-based diversity optimization for problem instance classification. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 869–879. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_81
11. Gießen, C., Witt, C.: The interplay of population size and mutation probability in the $(1+\lambda)$ EA on OneMax. Algorithmica **78**(2), 587–609 (2017)
12. Goldberg, D.E., Richardson, J., et al.: Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of International Conference on Genetic Algorithms (ICGA 1987), vol. 4149 (1987)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
14. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
15. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA: automatic model selection and hyperparameter optimization in WEKA. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Automated Machine Learning. TSSCML, pp. 81–95. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5_4
16. Lehman, J., Stanley, K.O.: Abandoning objectives: evolution through the search for novelty alone. Evol. Comput. **19**(2), 189–223 (2011)
17. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. **18**(1), 6765–6816 (2017)

18. Lindauer, M., Hoos, H., Hutter, F., Leyton-Brown, K.: Selection and Configuration of Parallel Portfolios. In: Handbook of Parallel Constraint Reasoning, pp. 583–615. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63516-3_15
19. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The Irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
20. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: Irace: iterated racing for automatic algorithm configuration. github.com/cran/irace (2020). commit: bae6ae86f2ee0fab9e3270801343482600f095e7
21. López-Ibánez, M., Stützle, T.: Automatically improving the anytime behaviour of optimisation algorithms. Eur. J. Oper. Res. **235**(3), 569–582 (2014)
22. López-Ibáñez, M., Stützle, T., Dorigo, M.: Ant colony optimization: a component-wise overview. In: Marti, R., Panos, P., Resende, M. (eds.) Handbook of Heuristics, pp. 1–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-07153-4_21-1
23. de Magalhães, C.S., Almeida, D.M., Barbosa, H.J.C., Dardenne, L.E.: A dynamic niching genetic algorithm strategy for docking highly flexible ligands. Inf. Sci. **289**, 206–224 (2014)
24. Martín, D., Alcalá-Fdez, J., Rosete, A., Herrera, F.: NICGAR: a niching genetic algorithm to mine a diverse set of interesting quantitative association rules. Inf. Sci. **355–356**, 208–228 (2016)
25. Neumann, A., Gao, W., Doerr, C., Neumann, F., Wagner, M.: Discrepancy-based evolutionary diversity optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2018), pp. 991–998. ACM (2018)
26. Shannon, C.E.: A mathematical theory of communication. ACM SIGMOBILE: Mob. Comput. Commun. Rev. **5**(1), 3–55 (2001)
27. Stützle, T.: ACOTSP: a software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem (2002). www.aco-metaheuristic.org/aco-code
28. Sudholt, D.: Crossover speeds up building-block assembly. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2012), pp. 689–702. ACM (2012)
29. Wang, H., van Stein, B., Emmerich, M., Bäck, T.: A new acquisition function for bayesian optimization based on the moment-generating function. In: Proceedings of International Conference on Systems, Man, and Cybernetics (SMC 2017), pp. 507–512. IEEE (2017)
30. Ye, F., Doerr, C., Wang, H., Bäck, T.: Automated configuration of genetic algorithms by tuning for anytime performance. IEEE Trans. Evol. Comput. (2022). https://doi.org/10.1109/TEVC.2022.3159087
31. Ye, F., Vermetten, D., Doerr, C., Bäck, T.: Data Sets for the study "Non-Elitist Selection Can Improve the Performance of Irace" (2022). https://doi.org/10.5281/zenodo.6457959