An Analysis of Parameters of irace

Leslie Pérez Cáceres, Manuel López-Ibáñez, and Thomas Stützle

IRIDIA, CoDE, Université libre de Bruxelles, Belgium
{leslie.perez.caceres,manuel.lopez-ibanez,stuetzle}@ulb.ac.be

Abstract. The irace package implements a flexible tool for the automatic configuration of algorithms. However, irace itself has specific parameters to customize the search process according to the tuning scenario. In this paper, we analyze five parameters of irace: the number of iterations, the number of instances seen before the first elimination test, the maximum number of elite configurations, the statistical test and the confidence level of the statistical test. These parameters define some key aspects of the way irace identifies good configurations. Originally, their values have been set based on rules of thumb and an intuitive understanding of the configuration process. This work aims at giving insights about the sensitivity of irace to these parameters in order to guide their setting and further improvement of irace.

1 Introduction

Algorithm configuration [5, 9] is the task of finding a setting of the categorical, ordinal, and numerical parameters of a target algorithm that exhibit good empirical performance on a class of problem instances. Currently, few tools are available for configuring algorithms automatically [1, 4, 10–12]. The irace package [12] implements an iterated racing framework for the automatic configuration of algorithms [3, 6]. irace is currently available as an R package and the details of its implementation and a tutorial on how to use it can be found in [12]. The implementation of irace is flexible, allowing the user to adjust the configuration process according to the configuration scenario at hand. As a flexible tool, irace itself has parameters. The default parameter settings of irace have been defined by rules of thumb based on intuition of how the configuration process may work [3, 6, 12]. So far, experimental analysis of these parameters have focused on a single race, studying the effect of the number of initial configurations, the particular statistical test or the confidence level [5, 7]. This paper is the first to empirically study the impact that specific settings of irace parameters have on the effectiveness of the configuration process. Section 2 describes details of irace and its default settings, Section 3 describes the experimental setup and Section 4 presents the experiments results. We conclude in Section 5.

2 The irace Procedure

Automatic algorithm configuration tools, henceforth called *configurators*, are algorithms that tackle expensive, stochastic nonlinear mixed-variable optimization

C. Blum and G. Ochoa (Eds.): EvoCOP 2014, LNCS 8600, pp. 37–48, 2014.

problems. The problem tackled by a configurator is called a *configuration scenario* and it is given as a target algorithm to be configured, a set of training instances representative of the problem to be solved by the target algorithm, and the configuration budget, which is the maximum computational effort (e.g., number of runs of the target algorithm) that the configurator has available. In addition, configurators have themselves parameters that affect their search.The search of `irace` consists of a number of iterations. In each iteration, a set of candidate algorithm configurations is generated and the best configurations of the iteration are identified by racing. Within a race, configurations are tested on a sequence of problem instances and, at each step, all surviving configurations are tested on a new instance. Candidate configurations are eliminated from the race if they are found to be poor performing according to some criterion. In `irace`, this criterion is implemented by means of statistical testing. For `irace` a minimum number of iterations (N^{iter}) is defined as $N^{iter} = \lfloor 2 + \log_2(N^{param}) \rfloor$, where N^{param} is the number of parameters of the target algorithm. The user-defined budget (B, the maximum number of target algorithm runs) is distributed across the iterations by setting the budget B_i available for iteration i to

$$B_i = \frac{(B - B_i^{used})}{N^{iter} - i + 1} \tag{1}$$

where B_i^{used} is the budget that has already been used before iteration i. An iteration is stopped as soon as the budget is spent or the number of candidates in the race reaches N^{max}, defined by $N^{max} = \lfloor 2 + \log_2(N^{param}) \rfloor$. If an iteration is stopped due to this latter condition, the iteration budget may not be used completely, and thus there may be enough budget to do more than N^{iter} iterations. Each iteration starts with a set of C_i configurations:

$$C_i = \left\lfloor \frac{B_i}{\max(\mu, T^{first}) + \min(5, i)} \right\rfloor \tag{2}$$

where μ ensures a minimum number of instances seen per iteration (five by default) and T^{first} is explained below. Using this definition of C_i, the number of candidates sampled decreases with the iteration number. This was done to account for the effect that candidate configurations become more similar as the configuration process progresses and more problem instances are needed to discriminate between them. In the first iteration of `irace`, an initial set of candidate configurations may be specified, the other candidates are generated uniformly at random. In the following iterations, the set of candidates is formed by the best candidates of the previous iteration and by new candidates that are sampled around these best candidates. For numerical (integer or continuous) parameters a truncated normal distribution is used and for categorical parameters a discrete one. Ordinal parameters are treated as integers. The distributions are updated every iteration, biasing the sampling towards the best candidates found. Each race evaluates the current set of candidates on a sequence of problem instances. Candidates are discarded from the race as soon as they show statistically worse performance than the best candidate so far. In the current `irace` implementation, either the Friedman test with its associated post-test [8] or a Student

t-test can be used as statistical test. The first statistical test of an iteration is performed after seeing T^{first} instances ($T^{first} = 5$, by default). The survivors are evaluated on the next instances and every T^{each} instances a statistical test is applied ($T^{each} = 1$, by default). A race finishes when at most N^{max} survivors remain in the race or the available budget B_i is exhausted. At the end of an iteration, the best candidates are selected from the survivors; these candidates are called elite candidates. The number of elite candidates in an iteration is given by $N_i^{elite} = \min\{N_i^{surv}, N^{max}\}$, where N_i^{surv} is the number of candidates that remain in the race when iteration i is finished. The selection of the elite candidates is done by ranking them (according to the sum of ranks for the Friedman test or the mean quality for t-test), and selecting from the lowest ranked. The elite candidates are then used to generate new candidates; to do so, they are selected to become parents with probability;

$$p_z = \frac{N_i^{elite} - r_z + 1}{N_i^{elite} \cdot (N_i^{elite} + 1)/2} \tag{3}$$

where N_i^{elite} is the actual number of elite candidates in iteration i and r_z is the ranking of elite candidate z. New candidates are sampled according to the distribution associated to each parameter in the selected parent. Once C_i candidates are obtained (including the N_i^{elite} ones and newly sampled ones), a new race begins. `irace` terminates when the total budget is exhausted or when the remaining budget is not enough to perform a new iteration. Finally, the best configuration found is returned.

3 Experimental Setup

In this section, we detail the configuration scenarios used for analyzing `irace`. Each scenario has a target algorithm, a set of training and test instances and a budget. The configuration scenarios are available at the supplementary information page (`http://iridia.ulb.ac.be/supp/IridiaSupp2013-008/`).

3.1 Configuration Scenarios

ACOTSP is a software package that implements various ant colony optimization (ACO) algorithms [17] for solving the Traveling Salesman Problem (TSP). The ACOTSP scenario requires the configuration of 11 parameters of ACOTSP, three categorical, four integer and four continuous. The training set is composed of ten random Euclidean TSP instances of each of 1000, 1500, 2000, 2500 and 3000 cities; the test set has 50 instances of each of the previous sizes. All instances and their optimal solutions are available from the supplementary pages. The goal is to minimize tour length. The maximum execution time of a run of ACOTSP is set to 20 seconds and the total configuration budget to 5000 runs.

SPEAR is a tree search solver for SAT problems [2]. The SPEAR scenario requires the configuration of 26 parameters of SPEAR, all of them categorical.

The training and the test set are composed of 302 SAT instances each, which belong to the SAT configuration benchmark "Spear-swv". The goal is to minimize mean algorithm runtime. The maximum execution time for each run of SPEAR is set to 300 seconds and the total configuration budget is 10000 runs.

MOACO is a framework of multi-objective ACO algorithms [13]. The MOACO scenario requires the configuration of 16 parameters: 11 categorical, one integer and four real. The training and the test set are composed each of 10 instances of 500, 600, 700, 800, 900, 1000 cities. The goal is to optimize the quality of the Pareto-front approximation as measured by the hypervolume quality measure [18]. The hypervolume is to be maximized, however, for consistency with the other scenarios, we plot the negative normalized hypervolume, which is to be minimized. The maximum execution time of each run of MOACO is defined by $4 \cdot (instance\_size/100)^2$. The total configuration budget is 5000 runs.

3.2 Training Set Analysis

The homogeneity of the training set with respect to algorithm performance is conjectured to have a high impact on the configuration process and possibly on the parameter settings of configurators. Homogeneity refers to the correlation between algorithm performance across instance sets: highly homogeneous instance sets maintain the same relative ranking of algorithms; highly heterogeneous instance sets lead to strongly different rankings depending on the particular problem instance to be tackled. Consequently, highly heterogeneous sets hinder the progress of the configuration process as candidate algorithm configurations may have inconsistent performance. A parametric measure of instance set homogeneity was proposed in [14], suggesting that the Friedman test statistic may be useful. As this latter statistic is not normalized and, thus, depends on the number of instances and configurations, we use the Kendall concordance coefficient (W) [15], which is a normalization of the Friedman test statistic. For each training set we generate 100 candidates uniformly distributed in the configuration parameter space. These candidates are evaluated on the instance set and Kendall's W statistic is calculated using instances as blocks and candidates as groups. The statistic of this test can be interpreted as a measure of how similar is the relative performance of candidates (that is, their ranking) across the instance set. A value close to one indicates high homogeneity, a value close to zero high heterogeneity. We performed the test using the complete instance sets and subsets grouped by instance size (ACOTSP and MOACO) or instance type (SPEAR). As shown by Table 1, the instances used in the ACOTSP and MOACO scenarios are much more homogeneous than those in the SPEAR scenario.

3.3 Experimental Setup

In the following sections, each experiment consists of 20 trials of `irace`, resulting in 20 final best configurations for each configuration scenario and each parameter setting of `irace`. For each configuration obtained, the average performance on

Table 1. Kendall's W statistic measured across 100 algorithm configurations on the training sets (*all*) and subsets grouped by size or type. The *Set* column is the set or subset of instances, the *Size* column is the number of instances in the set.

ACOTSP			SPEAR			MOACO		
Set	Size	W	Set	Size	W	Set	Size	W
all	50	0.96974	all	302	0.16017	all	60	0.99049
1000	10	0.98227	dspam	49	0.15446	500	10	0.99152
1500	10	0.98125	gzip	37	0.38442	600	10	0.99206
2000	10	0.98250	hsat	148	0.15510	700	10	0.99322
2500	10	0.98493	itox	26	0.61934	800	10	0.99256
3000	10	0.98089	winedump	17	0.29974	900	10	0.99311
			winegcc	22	0.62083	1000	10	0.99096
			xinetd	3	0.35308			

Table 2. Wilcoxon signed-rank test p-values comparing the mean performance over the test instances of configurations obtained by irace using $N^{iter} \in \{1, 3, \text{default}, \text{large}\}$

	default vs. large			default vs. 3			default vs. 1		
	ACOTSP	SPEAR	MOACO	ACOTSP	SPEAR	MOACO	ACOTSP	SPEAR	MOACO
F-test	0.33	0.4304	0.8695	0.7562	0.7562	0.0003948	$1.907e^{-5}$	0.7285	$1.907e^{-6}$
t-test	0.2943	0.498	0.0007076	0.7285	0.4304	$1.907e^{-6}$	0.0002098	0.5459	$1.907e^{-6}$

the test set is computed. We repeat each experiment using, as elimination test either the F-test (and its associated post-hoc tests) or the Student t-test without multiple test correction.[1] The experiments were executed on a cluster running Cluster Rocks GNU/Linux 6.0. The experiments involving the ACOTSP scenario were executed on an AMD Opteron 6128 with 8 cores of 2GHz and 16GB RAM. The ones involving the SPEAR and MOACO scenarios were executed on an AMD Opteron 6272 with 16 cores of 2.1GHz and 64GB RAM.

4 Experiments

In this section, we examine the impact of five parameters of irace on the performance of the final algorithm configuration found in the configuration process.

Number of Iterations. The number of iterations (N^{iter}) strongly modifies the search behavior of irace. With more iterations, fewer configurations are used in each iteration. The number of newly sampled configurations is also reduced as the number of elite configurations remains the same. Overall, this leads to an

[1] Using multiple test corrections in the Student t-test results in a search process that does not effectively eliminate poor candidates [5]. Avoiding multiple test corrections makes the process more heuristic, but proves to be effective.

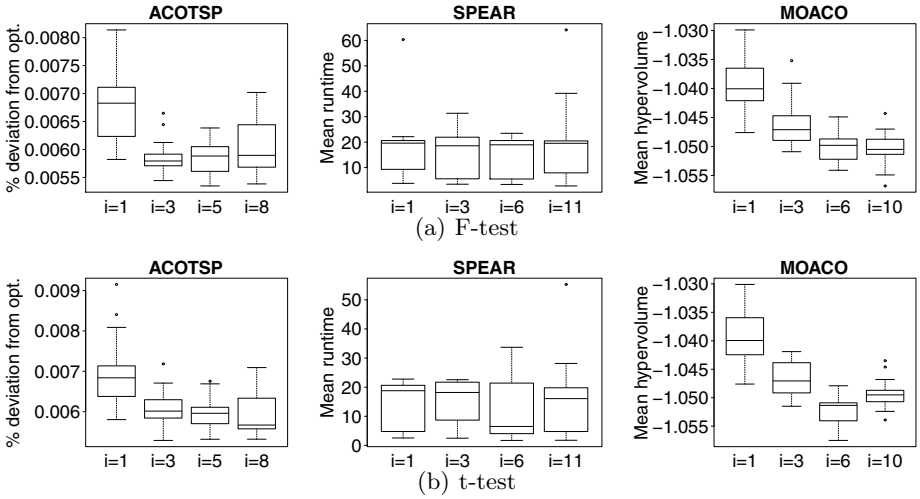


Fig. 1. Box plots of the mean performace over the test instances of 20 configurations obtained by `irace` using $N^{iter} \in \{1, 3, \text{default}, \text{large}\}$

intensification of the search by splitting the budget in short races. Less iterations, on the other hand, lead to a stronger diversification of the search. The default number of iterations of `irace` depends on the number of parameters. We increase this value to $N^{iter} = \lfloor 2 + 2 \cdot \log_2(N^{param}) \rfloor$ and we refer to this setting as "large" in the following. Additionally, we use two fixed settings: $N^{iter} = 3$ and $N^{iter} = 1$. The latter actually corresponds to a single race using configurations sampled uniformly at random [3]. In Fig. 1, we present the results of the 20 independent executions of `irace` on the three configuration scenarios and the results of the Wilcoxon test are shown in Table 2. In the SPEAR scenario, none of the differences is statistically significant, confirming the observation from the box-plots that no clear differences arise. Surprisingly, even a race based on a single random sample of configurations ($N^{iter} = 1$) obtains reasonable performance here. This is different from the MOACO and ACOTSP scenarios, where `irace` with $N^{iter} = 1$ performs significantly worse than the other settings, confirming earlier results [6]. Other differences in the ACOTSP scenario are, however, not statistically significant. In the MOACO scenario, the default setting performs significantly better than $N^{iter} = 3$, while the large setting performs significantly worse than the default only when using t-test. The results indicate that the default setting is overall reasonably robust. Nonetheless, the number of iterations has an impact on the quality of the final configurations and the adaptation of the number of iterations to the configuration scenario may be useful to improve `irace` performance.

First Elimination Test. The elimination of candidates during the race allows `irace` to focus the search around the best configurations. Here, we analyze the

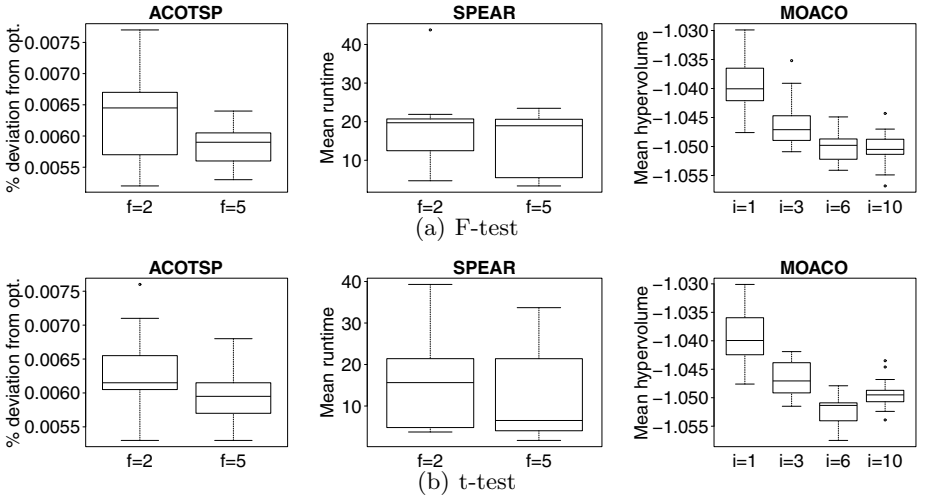


Fig. 2. Box plots of the mean performace over the test instances of 20 configurations obtained by `irace` using $T^{first} \in \{2, 5\}$

Table 3. Wilcoxon signed-rank test p-values comparing the mean performance over the test instances of configurations obtained by `irace` using $T^{first} = 2$ vs. $T^{first} = 5$

	ACOTSP	SPEAR	MOACO
F-test	0.01362	0.1231	0.1231
t-test	0.03623	0.5958	0.6477

sensibility of `irace` to the number of instances evaluated before performing the first elimination test (T^{first}). We performed experiments using the default setting of ($T^{first} = 5$) and a reduced value of $T^{first} = 2$. Reducing the value of T^{first} allows `irace` to more aggressively eliminate configurations. The budget saved in this way may be used later to sample a higher number of configurations. However, good configurations may erroneously be lost more easily. The experimental results are shown in Fig. 2. In the ACOTSP scenario a setting of $T^{first} = 2$ seems to worsen performance, while on the SPEAR and MOACO scenarios no clear differences are visible. The Wilcoxon paired test in Table 3 supports this analysis. Our hypothesis was that with a setting of $T^{first} = 2$, poor candidates are eliminated earlier and in later iterations more candidates may be sampled. In order to corroborate this hypothesis, we plot the development of the number of surviving configurations during the search process of `irace` (Fig. 3). The plots show one run of `irace` that is representative for the general behavior.

Maximum Number of Elite Configurations. The maximum number of elite configurations (N^{max}) influences the exploration / exploitation trade-off in the search process. In the extreme case of $N^{max} = 1$, `irace` samples new

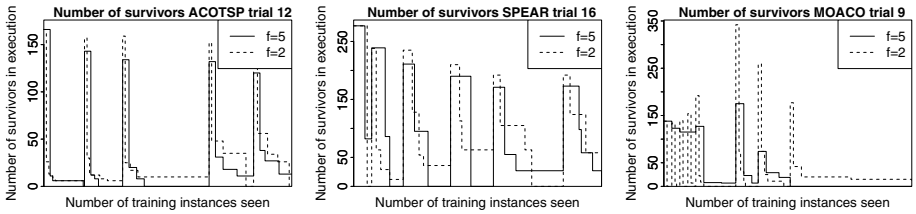


Fig. 3. Number of surviving candidates in `irace` using $T^{first} \in \{2, 5\}$ and F-test

configurations only around the best configuration found so far. A larger value of N^{max} (potentially) induces a more diverse search. In this section, we examine the possible differences that are incurred by setting $N^{max} = 1$ and compare it to the default setting. The results of these experiments are shown in Fig. 4 and the Wilcoxon test p-values in Table 4. While in the ACOTSP scenario using only one elite configuration worsens significantly performance, in the MOACO and SPEAR scenarios performance it is not significantly worse. Intensifying the search by strongly reducing the number of elite candidates does not seem to improve the performance of the final configurations in any of the configuration scenarios. These results indicate that the default setting is reasonably adequate.

Statistical Test. The main difference between the F-test (plus post-test) and the Student t-test is that the latter uses the raw quality values returned by the target algorithm, while the former transforms the values into ranks. Hence, the F-test can detect minimal but consistent differences between the performance of the configurations but it is insensitive to large sporadic differences, whereas the t-test is influenced by such outliers. Figure 5 shows box-plots comparing the configurations obtained using both statistical tests and Table 5 provides the Wilcoxon test p-values. The first set of plots show the average performance of the candidates on the test set and the second set of plots compares the average performance of the candidates per instance. The results of the Wilcoxon test indicate significant differences only for the MOACO case, where the usage of the t-test leads to better performance. It is interesting, however, to analyze in more detail the SPEAR configuration scenario. While there is no significant difference w.r.t. to the average performance (mean runtime), the F-test leads to shorter runtimes on more instances than the t-test; however, the t-test performs much better than the configurations obtained by the F-test on the subset of the *hsat* instances. Configurations obtained by using the F-test solve a majority of instances faster than configurations obtained by using the t-test; this difference is statistically significant. This corresponds to the fact that the F-test prefers to improve the mean ranking by performing well on a majority of instances while the t-test improves the mean performance and tends to reduce worst case performance, which in the SPEAR configuration scenario are very long runtimes. In this sense, these results confirm earlier observations for different configurators [11, 16].

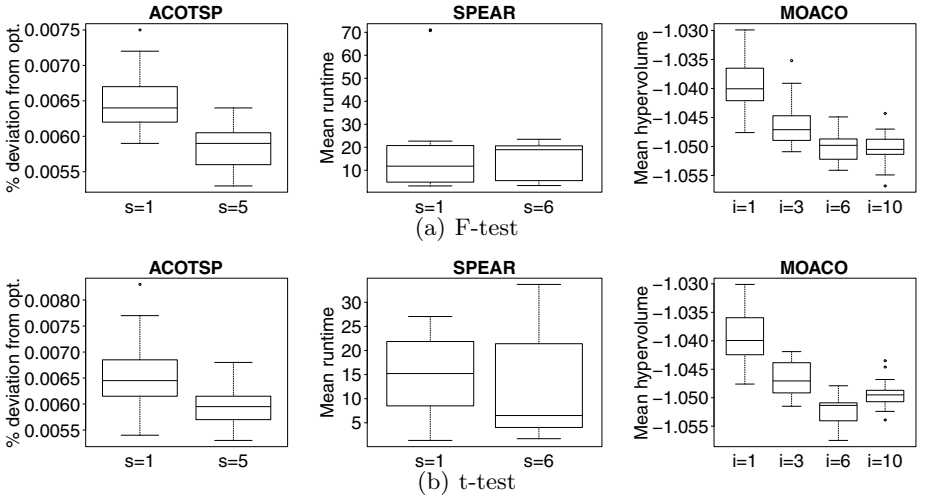


Fig. 4. Box plots of the mean performace over the test instances of 20 configurations obtained by `irace` using $N^{max} \in \{1, \text{default}\}$

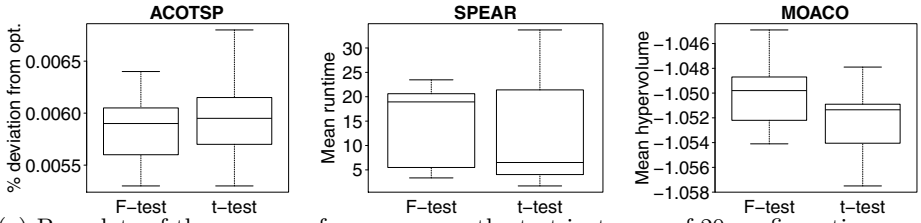
Table 4. Wilcoxon signed-rank test p-values comparing configurations obtained by `irace` using the default setting of N^{max} vs. $N^{max} = 1$, over the test set

	ACOTSP	SPEAR	MOACO
F-test	$3.624e^{-5}$	0.7285	0.4304
t-test	0.0005856	0.5958	0.4304

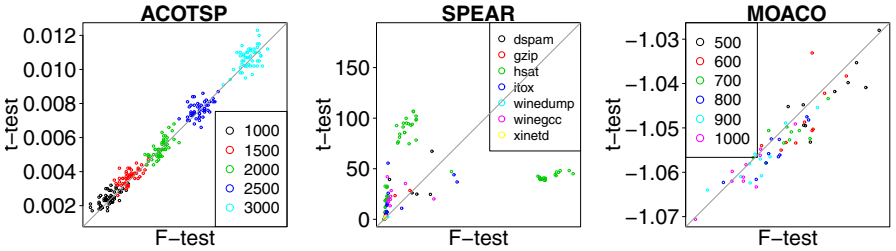
Table 5. Wilcoxon signed-rank test p-values comparing configurations obtained by `irace` using F-test vs. t-test

ACOTSP	SPEAR	MOACO
0.2943	0.5958	0.03277

Statistical Test Confidence Level. The confidence level of the `irace` elimination test is set by default to 0.95. Larger values mean that the test is more strict, so it takes more evaluations (or clearer differences) to eliminate configurations; lower values allow eliminating configurations faster, save budget, but risk removing good configurations based on few unlucky runs. We assess the effect of this parameter on the configuration process by experimenting with confidence levels $\in \{0.75, 0.95, 0.99\}$. Results are summarized in Fig. 6 and Table 6. For the ACOTSP configuration scenario, a setting of 0.99 is clearly worse than the default setting. Even if on the MOACO configuration scenario the 0.99 confidence level is significantly better than the default, the absolute difference is small and we would recommend using the default 0.95 level. Considering a smaller confidence level such as 0.75 may be an option. In fact, in two cases this setting is



(a) Box plots of the mean performance over the test instances of 20 configurations obtained by `irace`.



(b) Scatter plots of the mean performace per instance over the 20 configurations obtained by `irace`.

Fig. 5. Comparison of the mean performance over the test instances of 20 configurations obtained by `irace` using F-test and t-test

statistically better than the default setting while in one it is worse. However, the results also indicate that the behavior of `irace` is affected differently by the confidence level used depending on the statistical test used (see, e.g. MOACO configuration scenario). This is different from the other experiments, where the impact of `irace` parameter settings was similar for both elimination tests.

5 Final Remarks and Future Work

In this paper, we analyse the impact of five `irace` parameters on the final configuration performance. The experiments were performed on three configuration scenarios. The ACOTSP and the MOACO scenarios show a fairly homogeneous training set, while the SPEAR configuration scenario has a highly heterogeneous set of instances and a large variability of the quality values (runtime), making the configuration process more variable. The default settings of the number of iterations and the number of elite configurations proved to be reasonably robust. Reducing the setting of the first elimination test did not improve the performance of `irace`, although the results obtained suggest that lower values for this parameter could be used with highly homogeneous sets of instances. Larger differences were observed when altering the type of statistical test. However, in this case, the best setting depends on the goal of the configuration process. If the goal should take into account outliers, then `irace` should use the t-test rather than the F-test. Finally, the confidence level had a strong effect on the results.

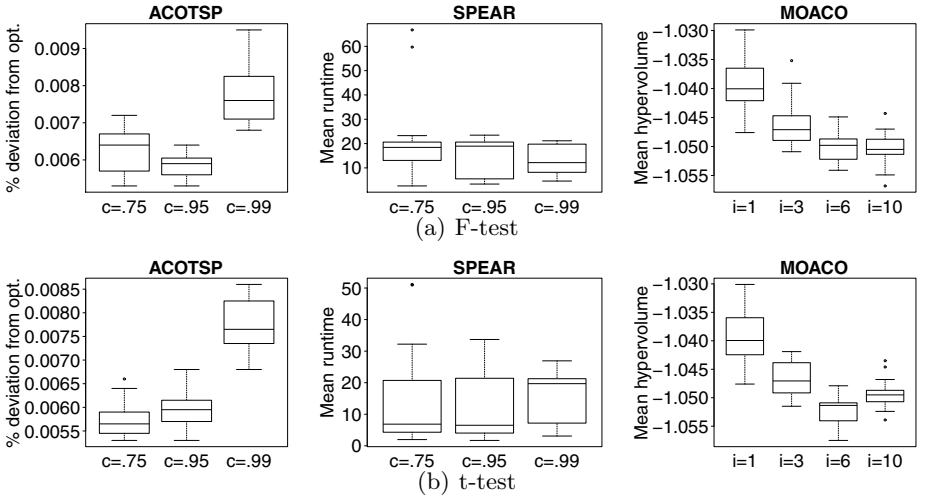


Fig. 6. Box plots of the mean performace over the test instances of 20 configurations obtained by `irace` using confidence level in $\{0.75, 0.95, 0.99\}$

Table 6. Wilcoxon signed-rank test p-values comparing configurations obtained by `irace` using confidence level in $\{0.75, 0.95, 0.99\}$

	0.75 vs. 0.95			0.99 vs. 0.95		
	ACOTSP	SPEAR	MOACO	ACOTSP	SPEAR	MOACO
F-test	0.01531	0.4091	$1.907e^{-6}$	$1.907e^{-6}$	0.7841	0.002325
t-test	0.02148	0.9563	0.1429	$1.907e^{-6}$	0.3683	0.2455

Large values were consistently worse, whereas lower values were sometimes better depending on the scenario and the type of statistical test. Further work will extend and complement the current experimental analysis in order to account for more parameter settings and their possible interactions. Additional scenarios may help to identify clearer trends or use an automatic configuration process to configure improved default settings of `irace`. Nonetheless, the insights obtained in this work are helping us to design future improvements to `irace`.

Acknowledgments. This work received support from the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium, the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office, and the EU FP7 ICT Project COLOMBO, Cooperative Self-Organizing System for Low Carbon Mobility at Low Penetration Rates (agreement no. 318622). Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a postdoctoral researcher and a senior research associate, respectively.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009)
2. Babić, D., Hutter, F.: Spear theorem prover. In: SAT 2008: Proceedings of the SAT 2008 Race (2008)
3. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HM 2007. LNCS, vol. 4771, pp. 108–122. Springer, Heidelberg (2007)
4. Bartz-Beielstein, T., Lasarczyk, C., Preuss, M.: Sequential parameter optimization. In: Proceedings of CEC 2005, pp. 773–780. IEEE Press (2005)
5. Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective. SCI, vol. 197. Springer, Heidelberg (2009)
6. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated F-race: An overview. In: Bartz-Beielstein, T., et al. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Berlin (2010)
7. Branke, J., Elomari, J.: Racing with a fixed budget and a self-adaptive significance level. In: Nicosia, G., Pardalos, P. (eds.) LION 7. LNCS, vol. 7997, pp. 272–280. Springer, Heidelberg (2013)
8. Conover, W.J.: Practical Nonparametric Statistics. John Wiley & Sons (1999)
9. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Hamadi, Y., et al. (eds.) Autonomous Search, pp. 37–71. Springer, Berlin (2012)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello Coello, C.A. (ed.) LION 5. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
12. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
13. López-Ibáñez, M., Stützle, T.: The automatic design of multi-objective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 16(6), 861–875 (2012)
14. Schneider, M., Hoos, H.H.: Quantifying homogeneity of instance sets for algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) LION 6. LNCS, vol. 7219, pp. 190–204. Springer, Heidelberg (2012)
15. Siegel, S., Castellan Jr., N.J.: Non Parametric Statistics for the Behavioral Sciences, 2nd edn. McGraw Hill (1988)
16. Smit, S.K., Eiben, A.E.: Beating the "world champion" evolutionary algorithm via REVAC tuning. In: Ishibuchi, H., et al. (eds.) Proceedings of CEC 2010, pp. 1–8. IEEE Press (2010)
17. Stützle, T.: ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem (2002), `http://www.aco-metaheuristic.org/aco-code/`
18. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation 7(2), 117–132 (2003)