Are State-of-the-Art Fine-Tuning Algorithms Able to Detect a Dummy Parameter?*

Elizabeth Montero[1], María-Cristina Riff[1], Leslie Pérez-Caceres[2], and Carlos A. Coello Coello[3]

[1] Departamento de Informática
Universidad Técnica Federico Santa María
Valparaíso, Chile
{elizabeth.montero,maria-cristina.riff}@inf.utfsm.cl
[2] Université Libre de Bruxelles
Bruxelles, Belgium
leslie.perez.caceres@ulb.ac.be
[3] CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07360, Mexico
ccoello@cs.cinvestav.mx

Abstract. Currently, there exist several offline calibration techniques that can be used to fine-tune the parameters of a metaheuristic. Such techniques require, however, to perform a considerable number of independent runs of the metaheuristic in order to obtain meaningful information. Here, we are interested on the use of this information for assisting the algorithm designer to discard components of a metaheuristic (e.g., an evolutionary operator) that do not contribute to improving its performance (we call them "ineffective components"). In our study, we experimentally analyze the information obtained from three offline calibration techniques: F-Race, ParamILS and Revac. Our preliminary results indicate that these three calibration techniques provide different types of information, which makes it necessary to conduct a more in-depth analysis of the data obtained, in order to detect the ineffective components that are of our interest.

Keywords: fine-tuning methods, algorithm design process, ineffective operators.

1 Introduction

We are currently involved in a project whose goal is to propose strategies to assist the decision-making process of designers of metaheuristic algorithms. As designers decide to add new components (e.g., a new evolutionary operator) to a certain metaheuristic, the fine-tuning process gets more complex. This is due to the

* Partially supported by Fondecyt Project no. 1120781, CONACYT/CONICYT Project no. 2010-199 and CONACyT Project no. 103570.

© Springer-Verlag Berlin Heidelberg 2012

highly nonlinear interactions that normally occur among the different parameters of a metaheuristic. Here, we are precisely interested in devising strategies that can help us to detect the components of a metaheuristic that are really crucial for its performance, so that the fine-tuning process can get reduced to a minimum. It is worth noting that other decisions related to the parameters of a metaheuristic algorithm are made during the design process. These decisions are not only concerned with finding the best possible parameter values, but also with deciding which parameters must be empirically tuned, and which ones can take either a fixed value or a value that can be varied or adapted online during the search process [8,6,4,7]. Over the years, there have been several efforts to develop automated fine-tuning methods (see for example [12,3,1,5]). Such methods require thousands of runs (and therefore, large amounts of time) in order to obtain good quality parameter configurations for a metaheuristic. The information that is extracted from this exhaustive process could be, however, very useful for improving the design of the metaheuristic itself. Since that is one of the main goals of this work, we conduct here an in-depth analysis of the information obtained with three well-known fine-tuning methods (ParamILS, F-Race and Revac), aiming to detect ineffective components in the algorithms being fine-tuned. In order to evaluate the output information obtained from the fine-tuning methods being analyzed, we adopted *Ant Solver*[1] [11], which is a well-known ant colony optimization algorithm that has been a popular choice for solving constraint satisfaction problems. It is important to emphasize that our goal here is not to find the best possible solutions to the problems being analyzed, but to detect ineffective components of the algorithm being analyzed, based on the information obtained from its systematic fine-tuning process. For this sake, we include a dummy operator in the code of the Ant Solver. Evidently, such a dummy operator is meant to be ineffective, because it doesn't perform any meaningful task within the algorithm. Its only purpose is to validate our methodology to detect ineffective components of a metaheuristic. The remainder of this paper is organized as follows. The next section provides a short description of the fine-tuning methods adopted for our analysis. Section 3 describes the Ant Solver algorithm adopted for our case study. Section 3.1 discusses the incorporation of a dummy operator into the Ant Solver algorithm. The instances used for our analysis are briefly explained in Section 4. In Section 5, we describe the experiments performed and the results obtained. Finally, Section 6 provides our main conclusions and some possible paths for future research.

2 Fine-Tuning Techniques

The fine-tuning techniques described next are strategies designed to automatically search for the best configuration of parameter values for a stochastic based method. Given a heuristic algorithm with k parameters, a fine-tuning technique searches for the parameter configuration $\theta^* = \{p_1, \ldots, p_k\}$ that provides the best performance of the algorithm. When talking about parameters, we refer to two main sets of elements:

[1] The authors thank Cristine Solnon for kindly providing us the source code of the *Ant Solver*.

- **Categorical parameters:** These are processes or functions that are required in an algorithm but that can be implemented in different ways. For example, the selection mechanism of an evolutionary algorithm.
- **Numerical parameters:** These are parameters expressed with real numbers or integers. For example the population size for an evolutionary algorithm.

The main difference between categorical and numerical parameters is that the latter are searchable (i.e., it is possible to define a distance measure between two different values of the parameter). In contrast, in categorical parameters it is not possible to define the distance between two "values".

2.1 F-Race

The F-Race method was proposed by Birattari et al. [2]. F-Race is a specific racing method specially adapted to fine-tune stochastic search methods. It uses Friedman two-ways analysis of variance by ranks to compare sets of candidate parameter configurations. This is a non-parametric test based on ranking, thus it does not require the formulation of a hypothesis on the distribution of the observations. Moreover, ranking based tests are very useful in fine-tuning problems because they implement a block design, which considers the different problem instances and the random seeds as sources of variation. The performance difference between configurations is analyzed using a hypothesis test. F-Race stops either when there is only one parameter configuration remaining, or when some predefined number of runs has been completed. The F-Race method defines three parameters: the initial number of runs without elimination of calibrations, the confidence level for the tests and the maximum budget. It also requires the range levels for each parameter. The number of levels of all parameters determines the size of the initial set of candidate parameter configurations.

2.2 Revac

The Relevance Estimation and Value Calibration (Revac) of evolutionary algorithms method was proposed by Eiben & Nannen [9]. Revac can be seen as an estimation of distribution algorithm [10]. It works with a set of parameter configurations as its population. For each parameter, it starts the search process with a uniform distribution of values within a given range. As the process advances, Revac performs transformation operations (crossover and mutation) with the aim of reducing each parameter distribution to a range of values that provide the best performance. Revac stops after performing 1000 runs of the fine-tuned algorithm. This approach defines 4 parameters: population size, step size of the crossover operator, step size of the mutation operator and the maximum number of iterations.

2.3 ParamILS

The Parameter Iterated Local Search (ParamILS) strategy was proposed in [5]. It works as an iterated local search algorithm which starts with a default parameter configuration and iteratively improves the configuration performance searching in the neighborhood of the configuration at hand. At each iteration, it performs random perturbations to the configuration at hand, and then applies the local search process and compares the outcome to the performance of the best parameter configuration that has been found so far. There are two well known versions of this approach: BasicILS and FocusedILS. These versions differ in the comparison procedure of parameter configurations they use. The ParamILS method defines four parameters: the amount of random solutions of the first phase, the amount of random solutions of each iteration, a restart probability and the maximum budget.

2.4 Comparison of Fine-Tuning Methods

All the techniques considered here need the definition of an interval of values for each parameter to be fine-tuned. Furthermore, F-Race and ParamILS require the definition of a set of countable values for each parameter (S_i). The number of configurations evaluated by F-Race grows exponentially on the size of each S_i. F-Race evaluates all of these configurations at least r times, whereas ParamILS reduces the number of evaluated configurations by exploring the most promising parameter configurations. Revac and ParamILS are stochastic search methods, then they are sensitive to the random seed adopted for the search process. F-Race is not a stochastic method but it defines a set of random seeds to execute the algorithm to be fine-tuned and these seeds could have an impact on the fine-tuning process. ParamILS provides as its output the best performing configuration, while Revac determines, for each parameter, an interval of values, and F-Race reports the set of the best performing configurations obtained. Revac is not able to search for categorical parameters, because its transformation process is performed on a continuous search space. However, both ParamILS and F-Race are able to tackle both categorical and continuous spaces. Table 1 summarizes the main features of tuning techniques.

Table 1. Features of the fine-tuning methods adopted

Method	**F-Race**	**Revac**	**ParamILS**
Type	Experimental Design	Search Based	Search Based
Initial input	Set of values	Interval/precision	Set of values
Expected output	Set of best configurations	An interval of values for each parameter	Best configuration
# parameters	3	4	4
Scope	Categorical/Numerical	Numerical	Categorical/Numerical
Stop criterion	Max runs or one configuration left	Max iterations	Max runs/time

3 Ant Solver

For our experiments we used an Ant Colony Optimization (ACO) based approach called *Ant Solver* [11]. This algorithm was proposed to solve constraint satisfaction problems (CSP). Ant Solver searches for a solution that minimizes the number of violated constraints. At each step of Ant Solver, each ant constructs a complete assignment for the CSP, and the pheromone trails are updated at the end of each cycle as usually done in ACO algorithms. The pheromone is laid on a binary graph whose vertices (X_i, v) represent the assignment of value v to variable X_i and the edges between two vertices represent those simultaneous assignment of values. Ant solver includes *pre-* and *post-* processing features that use a min-conflicts based local search procedure. The pre-processing phase performs local search repeatedly to collect information that is used to initialize pheromone trails and the post-processing phase performs local search after each ant has constructed a complete assignment. In our experiments, the pre- and post-processing procedures were disabled in order to analyze the behavior of the ACO algorithm alone. Ant Solver has four parameters: α, β, ρ and $nAnts$. α and β determine, respectively, the weight of the pheromone and the weight of heuristics in the computation of transition probabilities, ρ represents the level of pheromone evaporation and $nAnts$ corresponds to the number of ants used. We added an operator (which is described next) and, consequently, a new parameter for testing the fine-tuning techniques previously indicated.

3.1 Dummy Operator

Since our hypothesis was that a fine-tuning technique can provide information about ineffective components of an algorithm, we decided to add a dummy operator to the Ant Solver in order to validate it. This dummy operator takes an assignment and returns it without making any further changes. Its execution is controlled by a parameter δ that indicates its execution probability. In a real scenario, the operators that do not help in the search process use resources and spend time. Therefore, in order to simulate this behavior, the dummy operator is set to consume 1% of the constraint checks allowed in the execution in order to represent these costs.

4 Instances

The CSP instances used in this paper correspond to 3-coloring problems. Such instances were generated using a simple heuristic that generates instances that have at least one solution. The construction heuristic works as follows: first, the problem variables are separated in three disjoint sets, and then the variables are iteratively connected exclusively with variables in different sets, forbidding the intra set connections. This process continues until a given average connection level is met and the problem instance is obtained. The instances generated for these experiments have 400 and 500 variables and a connection average of 3.

5 Experiments

For our experiments we used a public domain implementation of ParamILS available at: `www.cs.ubc.ca/labs/beta/Projects/ParamILS`. We also adopted the authors' implementation of F-Race and our own implementation of Revac. The source code of F-Race, Revac and the Ant Solver are available at:

`www.inf.utfsm.cl/~emontero`. For all our experiments we tuned the probability of application of the dummy operator. Each tuning process was executed 5 times in order to obtain representative results. The rest of the parameter were set as follows: $nAnts = 15$, $\alpha = 2.00$, $\beta = 10.00$, and $\rho = 0.01$ according to recommendations of author in [11].

The hardware platform adopted for the experiments was a PC with an Intel Corei7-920, having 4GB of RAM, and using the Linux Mandriva 2010 operating system. Two sets of experiments were conducted:

– An analysis of the information obtained by the three fine-tuners when solving problems with 400 variables from the *Test Suite 1*.
– An analysis of the information obtained by the three fine-tuners when solving problems with 500 variables from the *Test Suite 2*.

Table 2. Performance measure for instances from the Test Suite 1

p_dummy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
fitness	0	0	11.3	18.9	21.8	25.6	26.6	27.7	32.4	32.7	32.7
conflict checks E+008	0.34	3.17	5.40	5.76	5.78	5.78	5.79	5.80	5.80	5.82	5.78

Fine-tuning methods use the number of violated constraints of the best solution found as the evaluation criteria to assess the performance of the Ant Solver.

5.1 Ineffective Operator

The objective of this experiment is to assess if the fine-tuning techniques adopted are able to provide information that allows us to infer that the algorithm design includes an ineffective operator. The expected result is that the probability assigned to the operator is zero. As indicated before, we included a dummy operator for this experiment. This operator receives a candidate solution and does not perform any change to it.

5.2 Performance Analysis

Here, we present a set of experiments which aim to understand the noticeable effect that can have an ineffective operator in the algorithm. For this purpose, we measure the quality of the solutions found and the number of conflict checks performed for different values of the dummy operator rate. Table 2 shows these values for instances of 400 variables and Table 3 shows them for instances of 500 variables. We can observe in Table 2 that the average performance of Ant Solver increases as the dummy rate decreases. It is important to note that the performance for rates 0.0 and 0.1 is the same. This is because in both cases, Ant Solver has a sufficient budget of evaluations to search until the best solution is found. However, the number of conflict checks performed by the second case is almost 10 times the number of checks performed by the first one. The higher the values of the dummy probability, the larger becomes the number of resources consumed by the dummy operator and, consequently, the problem instances can no longer be solved. As the value of the dummy operator gets larger, the worse is the performance of the Ant Solver. In Table 3, we can observe the performance of the Ant Solver when dealing with instances of 500 variables. In this case, the algorithm clearly shows that using the dummy operator strongly increases the number of constraints checks. The higher the dummy probability, the worse becomes the performance of the Ant Solver.

Table 3. Performance measure for instances in Test Suite 2

p_dummy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
fitness	0	2.5	31.9	46.1	49.5	50.6	52.2	55.2	58.8	61.4	62.6
conflict checks E+008	0.59	6.53	8.98	9.04	9.03	9.04	9.05	9.06	9.07	9.07	9.14

5.3 Test Suite 1

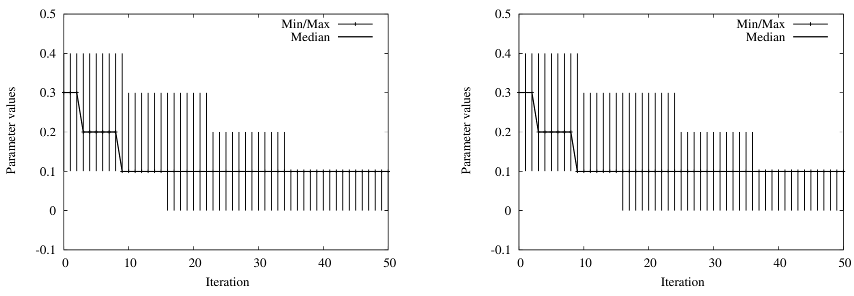
Here, we detail the results obtained using the Test Suite 1 composed by 10 instances of 400 variables as described in Section 4.

Analysis of Results for F-Race. The F-Race algorithm started the fine-tuning process with a set of 11 parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$. After finishing the first phase of 5 runs without elimination of configurations, F-Race discarded 9 configurations and kept only two of them $S' = \{0.0, 0.1\}$. Then, it ended with the set S' containing both configurations. This means that both parameter configurations are equivalent (i.e., the incorporation of the dummy operator at very low rates (lower or equal than 0.1) does not affect the performance of Ant Solver). F-Race required 550 Ant Solver runs in order to detect the ineffective operator.

Analysis of Results for Revac. Revac started the fine-tuning process with an initial interval of values in the range $[0.0, 1.0]$. The convergence process performed by Revac to fine-tune the dummy parameter is shown in Figure 1(a). This plot shows the median, the minimum and the maximum values of the ranges of values for the parameter at each iteration. Here, we can see that at the first iteration, the range of parameter values has already been reduced to $[0.1, 0.4]$, but it is still required to perform more iterations to refine this range of values. As shown in Figure 1(a), Revac required 35 iterations (around 1350 runs of Ant Solver) to converge to the range of values that performs the best for the dummy probability $[0.0, 0.1]$.

Analysis of Results for ParamILS. For ParamILS, we also considered 11 parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$ and the initial configuration was set to 0.5. After 100 runs of Ant Solver, ParamILS was able to change the value from 0.5 to 0.1. ParamILS ended with a value of 0.1, because the performance of Ant Solver with a probability of 0.1 for the dummy operator is equivalent to the performance obtained with a probability of 0.0.

Discussion. It is important to remark that the three fine-tuning methods adopted here were able to minimize the effect of an ineffective operator included



(a) Test Suite 1 (b) Test Suite 2

Fig. 1. Convergence of Revac for Test Suite 1 and 2

in the algorithm. ParamILS was clearly the most efficient, because both Revac and F-Race require the execution of an initialization phase. In this particular set of instances, we observed in Section 5.2 that the performance of Ant Solver is equivalent when using a probability of either 0.0 or 0.1 for the dummy operator. This is because the algorithm is able to solve the problem even with a dummy probability of 0.1. Considering this specific problem, we suggest that when the algorithm designer doubts about the effectiveness of a component, he can use ParamILS giving zero as the first value for the parameter that controls such a component. ParamILS will then be able to change this value to another one if it can obtain a significantly better result. In our tests, we began by assigning 0.5 to this parameter value, and then, when ParamILS decreases this value and it finds out that using 0.1 produces good results, it stops searching.

5.4 Test Suite 2

Here, we summarize the results obtained during the fine-tuning processes of the instances from the Test Suite 2. This test suite is composed by 10 problem instances of 3-coloring problem of 500 variables each.

Analysis of Results for F-Race. F-Race started the fine-tuning process with a set of 11 parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$. After finishing the first phase of 5 runs without elimination of configurations, F-Race discarded 9 configurations and kept only two of them $S' = \{0.0, 0.1\}$. At the next race, these two configurations were compared and the dummy value 0.0 showed a better performance than the configuration 0.1. In this case, the process ended after 570 Ant Solver runs.

Analysis of Results for Revac. Revac started the fine-tuning process with an interval of values in the range $[0.0, 1.0]$. The convergence process of Revac for fine-tuning the dummy parameter for the instances in Test Suite 2 is shown in Figure 1(b). Here, we can see that at the first iteration the range of parameter values has been reduced to the same range as that for Test Suite 1. The entire convergence process is very similar to the process corresponding to the Test Suite 1. This is because both processes were performed considering the same random seed. There are, however, some small differences. For example, in this case, the final range reduction took place at iteration 37. This means that Revac required around 1370 runs of Ant Solver to converge to the range of values $[0.0, 0.1]$.

Analysis of Results for ParamILS. In this case, we also considered 11 possible parameter configurations $S = \{0.0, 0.1, ..., 1.0\}$ and the initial value for the dummy rate was set to 0.5. The parameter changed to the value 0.1 after 100 Ant Solver runs, and it changed to 0.0 after 590 Ant Solver runs. The parameter value did not change during the rest of the fine-tuning process.

Discussion. In this case, the three fine-tuning methods were able to detect the ineffective operator included in the algorithm. ParamILS and F-Race were both the most efficient. The initialization phase of Revac was again detrimental in its competitiveness, but not for F-Race. This set of instances constitutes a more typical example of the scenarios that fine-tuning methods could face when searching for ineffective operators. In this case, the performance of the algorithm can be considered as the only indicator of the quality of the search that the algorithm is performing.

5.5 Final Remarks

Considering the two fine-tuning scenarios analyzed here, it is important to notice that the first one is more complex, since in that case the performance of the algorithm is not enough to categorically determine the elimination of the dummy operator. For the first scenario, a multistage procedure could be performed in order to analyze different quality measures of the search process in order to debug the design of the algorithm. Some good practices could also be considered for the application of fine-tuning methods to identify ineffective operators. For example, let's consider as our initial solution a configuration that could discard an operator (operator rate set to 0.0) in ParamILS. Only if such operator shows to be useful for the algorithm, either isolated or combined with other operators, ParamILS will incorporate it. For Revac and F-Race, their initial phases could be oriented to better analyze configurations discarding the use of the operator analyzed. Only when the fine-tuning method decided that these operators are useful for the algorithm, their operator rates would be fine-tuned.

6 Conclusions and Future Work

In this paper we have proposed the use of the information obtained by fine-tuning techniques for assisting the design process of metaheuristics. We have shown the way in which this information can be used to identify ineffective components (an operator, in this case).

Our experiments indicated that ParamILS was the best technique at identifying this situation in a more efficient way. Revac and F-race required more resources because of their expensive initialization phases. In our experiments the three fine-tuners were allowed to execute a fixed maximum of executions of the Ant Solver. The three fine-tuners studied here can be stopped at any time before reaching the maximum number of evaluations. However, in our experiments we waited until completing all the executions, so that a fair comparison of the tuning approaches could be done.

As part of our future work, we would like to study more recent fine-tuners such as sequential parameter optimization and the irace methods. Moreover, we aim to develop mechanisms that allow the collaboration of different fine-tuning techniques as a way of assisting the design of heuristic algorithms. In this case, however, the aim would be to detect effective components, instead of ineffective ones. We would also like to study other interesting aspects related to the algorithm design process, such as the identification of more than one ineffective operator, and the identification of opposite behavior of operators in the presence of noise and also considering continuous fitness landscapes.

References

1. Bartz-Beielstein, T., Lasarczyk, C.W.G., Preuss, M.: Sequential Parameter Optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, pp. 773–780 (2005)
2. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 11–18. Morgan Kaufmann, New York (2002)
3. Eiben, A.E., Schut, M.C.: New Ways to Calibrate Evolutionary Algorithms. In: Advances in Metaheuristics for Hard Optimization, pp. 153–177. Springer, Heidelberg (2008)
4. Fialho, Á., Da Costa, L., Schoenauer, M., Sebag, M.: Analyzing bandit-based adaptive operator selection mechanisms. Annals of Mathematics and Artificial Intelligence – Special Issue on Learning and Intelligent Optimization (2010)
5. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the Twenty-Second Conference on Artifical Intelligence, pp. 1152–1157 (2007)
6. Maturana, J., Lardeux, F., Saubion, F.: Autonomous operator management for evolutionary algorithms. Journal of Heuristics 16, 881–909 (2010)
7. Montero, E., Riff, M.C., Neveu, B.: C-strategy: A Dynamic Adaptive Strategy for the CLONALG Algorithm. Transactions on Computational Sciences, Special Issue 8, 41–55 (2010)
8. Montero, E., Riff, M.C.: On-the-fly calibrating strategies for evolutionary algorithms. Information Sciences 181(3), 552–566 (2011)
9. Nannen, V., Eiben, A.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Joint International Conference for Artificial Intelligence (IJCAI), pp. 975–980 (2007)
10. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A Survey of Optimization by Building and Using Probabilistic Models. Computational Optimization and Applications 21(1), 5–20 (2002)
11. Solnon, C.: Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 6(4), 347–357 (2002)
12. Yuan, Z., de Oca, M.A.M., Stützle, T., Birattari, M.: Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. IRIDIA - Technical Report Serie TR/IRIDIA/2011-017, Université Libre de Bruxelles (Agosto 2011)