Towards a Method for Automatic Algorithm Configuration: A Design Evaluation Using Tuners

Elizabeth Montero and María-Cristina Riff[*]

Department of Computer Science
Universidad Técnica Federico Santa María
Valparaíso, Chile
{Elizabeth.Montero,Maria-Cristina.Riff}@inf.utfsm.cl

Abstract. Metaheuristic design is an incremental and difficult task. It is usually iterative and requires several evaluations of the code to obtain an algorithm with good performance. In this work, we analyse the design of metaheuristics by detecting components which are strictly necessary to obtain a good performance (in term of solutions quality). We use a collective strategy where the information generated by a tuner is used to detect the components usefulness. We evaluate this strategy with two well-known tuners EVOCA and I-RACE to analyse which one is more suitable and provides better results to make this components detection. The goal is to help the designer either to evaluate during the design process different options of the code or to simplify her/his final code without a loss in the quality of the solutions.

Keywords: Automated algorithm tuning, automated algorithm configuration, metaheuristics.

1 Introduction

In order to obtain a metaheuristic with good performance, we have to make several design decisions. The design process is usually iterative, and at each step, the involved components must be evaluated. On the other hand, the final code of the metaheuristic can be extremely complex. Thus, analysing and understanding its results becomes difficult, and this is often a very time consuming task. Unexperienced designers usually tend to include more and more components during the iterative design process of a metaheuristic, without evaluating the usefulness of previously incorporated ones. We propose to have intermediate refining steps, during the design process, in order to help the designer to obtain a simpler design with similar performance. Our motivation is to assist the designer to make good decisions in order to produce an efficient metaheuristic (e.g. in terms of the solutions quality). In this paper, we study the information produced by the tuners, and compare their ability in helping the designer. We briefly revise published works related to this subject in section 2.

[*] This work is supported by the Fondecyt project 1120781 and Postdoctoral Fondecyt project 3130754. María Cristina Riff is partially supported by the Centro Científico Tecnológico de Valparaíso (CCTVal) No. FB0821.

In section 3, we introduce the general problem when designing metaheuristics and give a general idea on how design should be achieved. As a particular instance of this idea, we show how we can use the Evolutionary Calibrator (EVOCA) [13] and I-RACE [3] tuners that can work with categorical and numerical parameters [10]. We provide details on how to use a collective strategy based on these tuners in section 4. To evaluate our proposal, we use two well-known metaheuristics: A genetic algorithm that solves the NK-landscapes problems (NK-GA) [11], and a more complex one which is an artificial immune system algorithm that solves multiobjective problems (MOAIS-HV) [12]. These metaheuristics have already shown good performance to solve these kind of problems. Moreover, MOAIS-HV has a quite complex implementation and uses several components with several explicit and implicit parameters (not detailed in its description), which makes it a very interesting case for our work. We give brief description of both NK-GA and MOIAS-HV in section 5, as well as the results of a set of experiments to evaluate our solution. This section also provide a statistical comparison between the results obtained using EVOCA and I-RACE tuners. It is important to remark that our goal is not to find the best solution to the problem to be solved, but to focus on the way to detect the components that are strictly required from the ones that are unnecessary. Finally, we present the conclusions and future work in section 6. The contributions of this paper are: A general iterative method to select the best components and simplify the metaheuristic code, a comparison of the suitability of the EVOCA and I-RACE tuners to help into designing by refining of metaheuristics, and an in-depth analysis of the data generated by both tuners which can be used to make better decisions during the design process.

2 Related Work

Our approach shares much of its motivation with existing work on automated parameter tuning and algorithm configuration [5]. In automated parameter tuning the design space is defined by an algorithm whose behavior is controlled by a set of parameters, and the task is to find performance-optimizing settings of these parameters. For this, various methods can be used ranging from well-known numerical optimization procedures such as gradient-free CMA-ES algorithm [7] to discrete approaches based on experimental design methods [2], response-surface models [1] or stochastic local search procedures [8]. In automated algorithm configuration, the design space is defined by an algorithm scheme that contains a number of instantiable components, along with a discrete set of concrete choices for each of these. It can be mapped as a tuning problem, in which categorical parameters are used to select a set of components to instantiate the given scheme. However, only ParamILS and a genetic programming procedure applied to the configuration of local search algorithms for SAT [6] have been obtained promising results. Our approach is also related to the hyperheuristic methods [4]. When designing hyperheuristics, the goal is to find a good design with sufficient quality, and not to find the best algorithm. It is very useful to tackle real-world problems which must be quickly solved.

3 The Problem When Designing Metaheuristics

We can identify two problems when designing metaheuristics: The On-the-fly design problem and the post-design problem or refining problem. The first one occurs during the design process and is about the decisions to make when building efficient metaheuristics. Some of these decisions are related to the components or procedures to append to the algorithm to improve its performance. The On-the-fly Metaheuristic Design problem (OMD) can be stated as follows: given an intermediate design step, the current code of a metaheuristic M, and a set of candidate components $S = \{C_1, \ldots, C_n\}$ to be included in M. The OMD consists in finding a code M' for the metaheuristic using the selected components among the already included and the new candidates, in order to improve the performance of M. Unlike parameter control strategies, OMD problem is focused on how solution tool is constructed by selecting useful components from a set of possible options, like in a hyperheuristic approach.

The second problem occurs during post-design of the metaheuristic when we are interested in simplifying its code without performance loss.

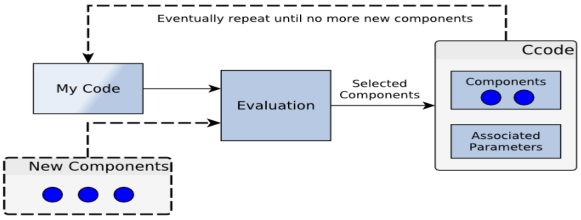

Fig. 1. Illustration of the designing problems

The Refining Metaheuristic Design problem (RMD) can be stated as follows: given M, a target algorithm or metaheuristic, a set of parameters for the algorithm and a set of input data. The RMD consists in finding a reduced code for the metaheuristic which gives, at least, the same performance with the same input data than M. Figure 1 illustrates both problems. For the OMD problem (doted lines), when we are building My Code we evaluate the inclusion of new components to improve its performance. After this evaluation a current code Ccode is obtained, which can follows a new procedure of inclusion of new components. For the RMD problem when the code is finished it follows an evaluation step in order to identify, when this is possible, an alternative code which uses a reduced number of its components but has at least the same performance.

4 Strategy to Use Tuners for Designing Metaheuristics

Collective Strategy : The key idea in the collective strategy is to generate a competition between the components that we are evaluating. The role of the tuner is to help us in the decision process. In this work we map RMD and OMD to a configuration problem, where new parameters are introduced to the algorithm in order to identify alternative designs for M. More formally,

Definition 1. *Given a metaheuristic code M, an instance of the problems consists in a 6-tuple $P = (M, S, \Theta, \Pi, \kappa_{max}, M')$, where S is the set of new binary parameters introduced in M that allows to either turn on or off some components. M' is the modified version of M that includes S. Θ is the configurations space for M'. Π is the set of input problem instances, $g(\theta, \Pi)$ is a function that computes the expected gain (e.g., the quality of the solutions) of running M' using instance $\pi \in \Pi$ when using configuration θ. κ_{max} is a time out after which all instances of M' will be terminated if they are still running.*
Any configuration $\theta \in \Theta$ is a candidate configuration of P. The gain of a candidate configuration θ is given by:

$$G_P(\theta) = mean_{\pi \in \Pi}(g(\theta, \pi)) \tag{1}$$

This definition considers the mean of gain induced by $g(\theta, \pi)$, but any other statistic could be used instead (e.g. median, variance). Given G^M, the gain of metaheuristic M using its best parameter configuration, an alternative code defined by the configuration θ^* has a value $G_P(\theta)$ such that:

$$G_P(\theta^*) \geq G^M \tag{2}$$

In this definition for the OMD problem M is the current code and M' also includes the alternative components to be evaluated. For the RMD M is the final metaheuristic code. We define the **collective strategy** as the evaluation of the metaheuristic M' using different parameter configurations. For this strategy, we use the tuner to obtain the $P_1, P_2, \ldots, P_k$ binary values that indicate which of the k components must be turned on in the M' code. The evaluation of the performance of the algorithm with its best set of parameters values is used to decide whether or not the algorithm can be modified.

Definition 2. *Given M a metaheuristic with a performance G^M and $M_1, \ldots, M_l$ alternative algorithms with performances $G_{M_1}(\theta_1), \ldots, G_{M_l}(\theta_l)$ for a maximization problem. M_i belongs to the* **set of alternative designs** *S_d if and only if $G_{M_i}(\theta_i) \geq G^M$. We define M_i as the* **best alternative design** *such that $G_{M_i}(\theta_i) \geq G_{M_j}(\theta_j), \forall M_i, M_j \in S_d, i \neq j$*

When two or more best alternative designs are identified, the decision will be to use the simplest one. When we use the collective strategy during the post-design we are looking for a possible alternative code that allows the metaheuristic M' to solve the problems as M does with at least the same performance, but using a reduced number of its initial components. Given its stochastic nature, it is noteworthy to mention that the refined algorithm could show better performance than the initial one. When we use the collective strategy during the design we are looking for a code M' which has better performance than M and is composed by a new set of components.

4.1 How to Use Tuners during the Design Process

The designer usually follows an incremental procedure for the components selection. At the beginning, the designer has a first set of components that he/she believes to be some good candidates to include in the code. The typical question is therefore: Which of these components are more suitable to be included in my code?. In other words, which of these components do have the best performance to solve my problem. Using the collective strategy, the designer can determine which are the best code alternatives according to the information provided by the calibrator when using these components. Let's call *CCode* the current code, which corresponds to the best one obtained by this evaluation. Then, the designer can add to the *CCode* a set of new components that he/she thinks they could improve the metaheuristic. A new evaluation is made with the collective strategy to determine which components among the previously selected and the new ones allows the algorithm to have the best performance, that is which is the best alternative for the new *CCode*. Note that all the components can be turn on or off during the evaluation and therefore, previous selected ones may be discarded at this step. This is mainly because some new components could do the same search than previous selected ones, but more efficiently. The same process is repeated until the designer does not have more components to add. The final *CCode* is then the best one determined by the collective strategy. Figure 2 shows an example where 4 components (C_1, C_2, C_3, C_4) are initially considered to be included into the code. The information provided by the calibrator allows to select a *CCode* that includes (C_1, C_2) as the best alternative. Then, in a second step, the designer considers to include components (C_5, C_6, C_7) into the code. The evaluation of the tuner suggests to include components (C_6, C_7), but in this case to discard C_1 which has been selected in the previous step. The design process continues until obtaining a Final Code of high quality. Many other design options can be considered. For instance, the inclusion of a component associated to a particular method could be easily evaluated together. Moreover, it is also possible to consider the inclusion of a component that was previously discarded before and to evaluate its inclusion at the current time of the design process.

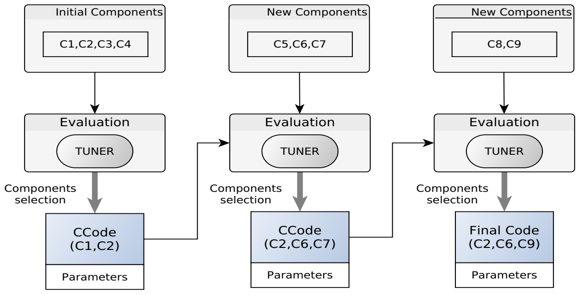

Fig. 2. Example for using the Collective Strategy

5 Experiments

The purpose of these experiments is to analyse different scenarios of using the collective strategy. Any tuning method able to calibrate categorical parameters can be used by our framework: sampling, model-based, screening or meta-evolutionary methods [5]. In our study we consider the following I-RACE [2] and the EVOCA [13] tuners.

I-RACE or Iterated F-Race: Iterated F-Race[1]. is an iterative version of F-Race algorithm [2]. At each iteration, Iterated F-Race uses a number of surviving candidate parameter configurations to bias the sample of new candidate configurations. Iterated F-Race follows the framework of model-based search: (1) construct a candidate solution based on some probability model; (2) evaluate all candidates and (3) update the probability model of biasing the next sample.

The EVOlutionary CAlibrator: EVOCA[2] is itself an evolutionary algorithm that works with a population of parameter configurations. It uses two operators. Wheel-crossover constructs one child from the whole population. The child replaces the worst individual in the current population. The mutation operator is a hill climbing procedure. The child generated by mutation replaces the second worst individual in the current population, when finding a better individual.

5.1 Experiments with NK-GA

For these experiments we use a genetic algorithm that solves unrestricted NK landscape problems [9], proposed in [11].

NK-GA: This genetic algorithm (GA) evolves a population of fixed-length binary strings. New solutions are created by applying variation operators to the population of selected solutions. The algorithm has three genetic operators bit-flip mutation, uniform crossover and two-point crossover.

Test Suite: We perform our experiments using a set of unrestricted NK landscape instances with k ranging from $k = 2$ to $k = 6$. A total number of 15 problem categories are considered. The minimization of the number of evaluations required by the genetic algorithm to solve all the instances is considered as a criteria to evaluate the parameter configuration. We consider a budget of 3500 runs for both tuners. We focus here on the use of the three genetic operators: Uniform crossover(U), two-points crossover (T) and bit-flip mutation (M). We run each tuner 20 times and we show the best 5 results as well as the execution time in table 1. For both tuners, the set of alternative designs S_d includes combinations that use less components than the original algorithm. They solve all the instances. Thus, using both tuners we can identify that both crossover components are not necessary to the algorithm for solving the 15 categories of the problem. We can also remark that using only one or two components instead of the three initial ones neither implies a significant increase of the number of evaluations nor of the execution time. In conclusion, both the uniform crossover and the two-points crossover are not strictly necessary to the algorithm performance. Similar results were obtained using a reduced number of NK-GA evaluations.

[1] I-RACE is available from *CRAN*, `http://cran.r-project.org`

[2] EVOCA is available in our website `comet.informaticae.org`

Table 1. Algorithms selected by EVOCA and I-RACE

EVOCA					**I-RACE**				
	solved	Average success	Average Evaluations	Average Time [s]		solved	Average success	Average Evaluations	Average Time [s]
UM	15	100	116.2	0.01	UTM	15	100	106.3	0.01
M	15	100	121.0	0.01	UTM	15	100	113.1	0.01
UTM	15	100	125.6	0.01	UTM	15	100	122.5	0.01
TM	15	100	131.4	0.01	M	15	100	137.4	0.01
UM	15	100	144.5	0.01	UM	15	100	141.1	0.01

5.2 Experiments with MOAIS-HV

We now use the MOAIS-HV algorithm proposed in [12] to solve multiobjective optimization problems, to show the effectiveness of our proposal.

MOAIS-HV: The main idea of MOAIS-HV is to maintain an online population of antigens and antibodies. In this case antigens are considered to be good quality solutions and antibodies are the bad ones. The antigens are cloned and a mutation operator is applied. The mutated clones and the best antigens found are merged and the size of the main population is maintained by discarding individuals that contribute the least to maximize the hypervolume.

Test Suite: In this case, we want to analyze the design of the MOAIS-HV's hypermutation process. In MOAIS-HV, hypermutation is applied to each variable according to a probability value. Each time a variable has to be mutated, the probability is recomputed. This probability indicates a trade-off between the Global Gaussian (G) and the Local Gaussian (L). The probability changes as the algorithm goes on and the hypermutation will perform more/less local searches. The changing probability criteria is based on the computation of a complex formula that uses explicit and some implicit values which are obtained by other procedures on the code.

The goal of our experiments is to analyse if the algorithm really needs this complex hypermutation process. We evaluate the use of the Global and the Local mutation to obtain a good performance. We also evaluate if it is really required to change the probability of using L during the search. For our experiments we use well-known 2-objectives standard functions: zdt1-zdt4, zdt6 and 3 objectives: dtlz1-dtlz7. These are the same functions used by the authors to introduce MOAIS-HV. For the 2-objectives functions, we consider a population of size 100 and a maximum of 200 iterations. For the 3-objectives functions, the population size is 200, and there is a maximum of 500 iterations. The test consists in the maximization of the hypervolume of the previously mentioned functions. This maximization is used to evaluate each configuration. Source code of MOAIS-HV is also available in our website[3]. In this experiment, we want to evaluate both the inclusion of the three components in the code: Global mutation(G),

[3] `comet.informaticae.org`

Table 2. Algorithms and their performance found by EVOCA and I-RACE

EVOCA			**I-RACE**		
Algorithm	Average Hypervolume	Average Time [s]	*Algorithm*	Average Hypervolume	Average Time [s]
LR	0.790047	1.29	GR	0.777034	1.73
GL	0.775806	1.32	GLR	0.789478	1.81
LR	0.786416	1.71	GLR	0.787906	1.72
LR	0.781908	1.82	GLR	0.789524	1.80
GLR	0.781233	1.80	GLR	0.784471	1.43
GLR	0.784627	1.69	GLR	0.789200	1.64
GLR	0.783951	1.68	GLR	0.789115	1.67
GLR	0.785932	1.69	GLR	0.789502	1.63
GLR	0.784209	1.68	GLR	0.785856	1.64
LR	0.787449	1.65	GLR	0.789323	1.68
GLR	0.785943	1.69	G	0.780221	1.65
GLR	0.784256	1.67	GLR	0.780044	1.67
LR	0.790048	1.66	GLR	0.786530	1.67
GL	0.784752	1.67	GLR	0.788129	1.64
LR	0.790048	1.66	GLR	0.788683	1.67

Local mutation(L) and Random mutation(R) and the method to apply them. Thus, we initially consider a code where the mutation components have the same fixed and equally probability to be applied. The tuning process considers all the functions together. When using EVOCA, the relative difference to the best solution found was used to compare the performance of different test functions. For both tuners, 12000 runs are set as maximum budget.

Analysis: Table 2 shows the algorithms found by EVOCA and I-RACE, their execution times and the performance measured in their 15 executions. The performance is the average hypervolume of 50 runs with different seeds. Differences in performance when the same algorithms were selected are due to differences in the parameters values tuned at each execution. For EVOCA, the best code uses the Local and Random Mutation, and a fixed probability value. For I-RACE, the best option uses the three mutations, also with a fixed probability value. This code also improves the average performance of the MOAIS-HV, but less significantly than the algorithm selected by EVOCA.

Statistical Evaluation: Table 3 shows the statistical analysis indicators obtained using Wilcoxon test. Considering p-value=0.05, the algorithms identified by both, EVOCA and I-RACE, outperform the original algorithm. Moreover, EVOCA's algorithm in 364 cases is better than original MOAIS-HV and I-RACE's algorithm in 336. Table 4 shows the performance obtained by the algorithm when solving each function. We observe that both, the original algorithm and the algorithm found by I-RACE, obtained the best performance in 3 functions, and the algorithm defined by EVOCA shows a better performance in 6 functions. In terms of the execution time, the algorithm selected using EVOCA and the one using I-RACE are similar. In both cases the code obtained is much simpler than the

Table 3. Wilcoxon ranks

	MOAIS-HV - EVOCA			MOAIS-HV - I-RACE		
Ranks	N	Mean Rank	Sum of Ranks	N	Mean Rank	Sum of Ranks
Negative Ranks	364	322.80	117497.5	336	346.90	116559.0
Positive Ranks	179	168.71	30198.50	208	152.31	31681.00
Ties	57			56		
Total	600			600		
Statistics						
Z			-11.93			-11.57
Asymp. Sig. (2-tailed)			0.00			0.00

Table 4. Performance comparison

Function	MOAIS-HV	EVOCA's Algorithm	I-RACE's Algorithm
zdt1	**0.871372**	0.868736	0.870774
zdt2	**0.538028**	0.535536	0.537482
zdt3	**1.328516**	1.326314	1.327164
zdt4	0.814616	**0.825334**	0.813102
zdt6	0.504312	**0.504320**	0.504300
dtlz1	0.314374	0.316040	**0.316274**
dtlz2	0.744758	**0.748010**	0.747234
dtlz3	0.670666	0.735176	**0.741356**
dtlz4	0.741330	0.749178	**0.749238**
dtlz5	0.434040	**0.434300**	0.431530
dtlz6	0.431980	**0.437506**	0.436822
dtlz7	1.987018	**2.000120**	1.999014

original one with a same or best level of performance. There is especially no more complex formula to compute the dynamic probability for the mutations.

6 Conclusions and Future Work

In this work, we have analysed two problems about the metaheuristics design. The On-the-fly metaheuristics design problem (OMD) occurs during the design process and the Refining Metaheuristics Design problem (RMD) which is a post-design problem. The OMD problem, which concerns the selection of the components and methods to include in the code is always present when designing metaheuristics, but the RMD problem does not always necessary and strongly depends on the designer goals. We have compared two well-known tuners to help the designer during the evaluation process, but any tuning method able to work with categorical parameters can be used. We have evaluated the code selected by EVOCA and I-RACE for two metaheuristics: a genetic algorithm (NK-GA) and an artificial immune algorithm (MOAIS-HV). Both tuners have shown to be able to detect the most suitable components, and to produce simpler and efficient algorithms. Moreover, in terms of the performance, there is not a statistical significant difference between their identified algorithms. The results obtained indicate the suitability of both tuners for helping the designer evaluation task and could be used in the future for including on a framework for automatic configuration algorithms or hyperheuristics.

Acknowledgment. We thank Dr. Carlos Coello for MOAIS-HV code and Mrs. Leslie Pérez for her support with I-RACE implementation.

References

1. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation—The New Experimentalism. Natural Computing Series. Springer (2006)
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, USA (2002)
3. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and Iterated F-Race: An Overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Heidelberg (2010)
4. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 57, pp. 457–474. Springer, US (2003)
5. Eiben, A.E., Smit, S.K.: Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms. Swarm and Evolutionary Computation 1(1), 19–31 (2011)
6. Fukunaga, A.: Automated Discovery of Composite SAT Variable Selection Heuristics. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 641–648 (2002)
7. Hansen, N., Kern, S.: Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 282–291. Springer, Heidelberg (2004)
8. 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)
9. Kauffman, S.A.: Adaptation on Rugged Fitness Landscapes. Lecture Notes in the Sciences of Complexity 1, 527–618 (1989)
10. Montero, E., Riff, M.C., Neveu, B.: A Beginner's Guide to Tuning Methods. Applied Soft Computing 17(0), 39–51 (2014)
11. Pelikan, M.: Analysis of Estimation of Distribution Algorithms and Genetic Algorithms on NK landscapes. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO, pp. 1033–1040. ACM, USA (2008)
12. Pierrard, T., Coello Coello, C.A.: A Multi-Objective Artificial Immune System Based on Hypervolume. In: Coello Coello, C.A., Greensmith, J., Krasnogor, N., Liò, P., Nicosia, G., Pavone, M. (eds.) ICARIS 2012. LNCS, vol. 7597, pp. 14–27. Springer, Heidelberg (2012)
13. Riff, M.C., Montero, E.: A New Algorithm for Reducing Metaheuristic Design Effort. In: IEEE Congress on Evolutionary Computation (CEC 2013), Cancún, México, pp. 3283–3290 (June 2013)