Motivations for the Development of a Multi-objective Algorithm Configurator

Nguyen Thi Thanh Dang and Patrick De Causmaecker
K.U. Leuven KULAK, CODeS, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium

Keywords: Automated (Automatic) Algorithm Configuration, Multi-objective Optimization.

Abstract: In the single-objective automated algorithm configuration problem, given an algorithm with a set of parameters that need to be configured and a distribution of problem instances, the automated algorithm configurator will try to search for a good parameter configuration based on a pre-defined performance measure. In this paper, we point out two motivations for the development of a multi-objective algorithm configurator, in which more than one performance measure are considered at the same time. The first motivation is a parameter configuration case study for a deterministic single machine scheduling algorithm with two performance measures: minimization of the average running time and maximization of the total number of optimal solutions. The second one is the configuration problem for non-exact multi-objective optimization algorithms. In addition, a discussion of solving approach for the first motivating problem is also presented.

1 INTRODUCTION

The *automated algorithm configuration* problem, sometimes called *parameter tuning*, can be informally stated as follows: given an algorithm A (*target algorithm*) with a list of parameters, a set of problem instances B and a performance measure for the evaluation of A's quality on B; the automated algorithm configurator (or simply *configurator*) will try to find a good parameter configuration of A, i.e., an assignment of A's parameters into specific values, in such a way that the performance measure's value on B is optimized. The problem instance set B could be generated from a problem's distribution and its size is usually chosen to be large to well represent the problem. Since this set is large, the exact evaluation of the performance measure over B often incurs very high computational costs. Therefore, the configurator normally deals with a subset of B, called *training instance set*, during the configuring procedure. To avoid over-fitting, this training set certainly should be sufficiently large and widely spread over the problem distribution, then, the final parameter configuration obtained is tested on another problem instance set, named *test instance set*, to evaluate the efficiency of the configurator. The test set could be a subset of B, or even the same as B.

The automated algorithm configuration problem could be considered as an optimization problem, in which the search space is the set of possible parameter configurations and the objective function is the performance measure over the training instance set. This optimization problem presents three challenges. The evaluation of each solution's quality is often very expensive. This evaluation could be stochastic if the target algorithm is not deterministic. And finally, different types of variables (target algorithm's parameters) could exist in the same configuration problem. To be more specific, an algorithm's parameter could be continuous, integer or categorical. A parameter could also be conditional, i.e., its activation could depend on specific values of some other parameter.

The automated algorithm configuration problem has received large attention in the last decade. A vast number of solving approaches, originating from various fields, have been proposed in recent years. According to Stutzle et al., 2013, they could be classified into four groups: approaches from the experimental design community, sequential statistical testing techniques, model-based optimization approaches and metaheuristics. Several other works have also been devoted to a literature review of the algorithm configuration topic, such as Hoos, 2012. Among these methods, irace (López-

Dang N. and De Causmaecker P..
Motivations for the Development of a Multi-objective Algorithm Configurator.
DOI: 10.5220/0004925203280333
In *Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems* (ICORES-2014), pages 328-333
ISBN: 978-989-758-017-8

Ibánez et al., 2011, a statistical testing technique), Gender-based Genetic Algorithm (Ansótegui et al., 2009), ParamILS ((Hutter et al., 2009), a metaheuristic based on Iterated Local Search) and SMAC (Hutter et al 2011, a model-based optimization method) have shown remarkable results on several hard algorithm configuration problems with large numbers of parameters. As an example, ParamILS has been used to configure the CPLEX solver with 76 parameters and has achieved a speedup ratio up to 50 over CPLEX's default parameter configuration as well as configurations got from CPLEX's automatic tuning tool (Hutter et al 2010). Several other academic applications of ParamILS have also been reported on the website of its authors: *http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/applications.html*.

Solving approaches for the automated algorithm configuration problem so far are single-objective, i.e., only one performance measure is considered during the configuring stage. In this paper, we describe two motivations for the development of a multi-objective algorithm configurator, in which multiple performance measures are considered at the same time. The first one is an algorithm configuration case study for a single machine scheduling algorithm, with the requirement of balancing between target algorithm's robustness and speed, the second one is the case when the target algorithm is a non-exact multi-objective optimization algorithm.

Please note that there are some works in the evolutionary computation community that are dedicated to a Multi-Objective Tuning Problem (MOTP), such as the tuner BONESA proposed in Smit et al., 2011. However, MOTP is different from the multi-objective algorithm configuration considered in this paper. In MOTP, the problem instance set is just a limited and small set of continuous optimization functions, so that it does not have to deal with the problem of configuration's expensive evaluation. The main goal of MOTP is trying to balance algorithm configurations' performance over all optimized functions, i.e., finding robust algorithm configurations, while the multi-objective algorithm configuration problem focus on solving the problem of over-fitting: trying to deal with a quite large training instance set.

This paper is organized as follows: section 2 describes the first motivation and a proposal of solving approach, section 3 presents the second motivation and section 4 draws some conclusions and future works.

2 THE FIRST MOTIVATION

2.1 Problem Description

The target algorithm considered in this problem is SiPSi, a single machine scheduling algorithm proposed in Tanaka et al., 2012. The algorithm and relevant problem instances are downloadable from Tanaka's website (*http://turbine.kuee.kyoto-u.ac.jp/~tanaka/index-e.html*). SiPSi is applied on different single machine scheduling instances with various sizes (the number of jobs). This algorithm is single-objective, exact and deterministic. It has 25 parameters, including 14 continuous parameters, 7 integer parameters and 4 categorical parameters. Among them, 11 parameters are conditional.

In this case study, a pre-defined cut-off time is specified for each problem size. The running of SiPSi on a problem instance is stopped when either an optimal solution is found or the running time reaches the relevant cut-off time. The goal of the algorithm configuration task is to find parameter configurations that have small running time and large number of optimally solved instances over the whole instance set. The specification of cut-off time for each problem size is two-fold: first, when some researcher develops an algorithm for solving a well-known problem, some so-far-state-of-the-art method for that problem usually exists and the researcher wants to "beat" the performance of this state-of-the-art algorithm, then problem-size-dependent cut-off time is an appropriate choice; second, if we are interested in configuring algorithm for very large problem's sizes, e.g., from 500 to thousands of jobs, a configuring procedure directly deals with a training instance set of such large sizes is impossible, in that case, problem instance set with smaller sizes in which cut-off time is gradually increased according to problem size can be helpful, since we can try the configuring procedure on this alternative problem instance set, in the hope that the obtained parameter configuration is robust enough to work well on even unseen and larger problem instance sizes.

During Tanaka's manual tuning procedure, he observed a possible trade-off between the two mentioned performance measures: "A safe parameter configuration performs well for many types of instances, but the algorithm becomes slow, on the other hand, a tuned set of parameters improves the speed of the algorithm, but the performance deteriorates considerably (and sometimes the algorithm fails to find an optimal solution) for some specific instances due to

parameter sensitivity". Such an example is shown in table 1. In this experimental example, the first parameter configuration, dubbed $c_{default}$, is the currently default configuration of SiPSi. The second one, denoted by c_{tuned}, is a configuration obtained by using ParamILS (Hutter et al., 2009), with a weighted aggregation function of the two performance measures. Results are tested on 5975 problem instances. From table 1, we can see the trade-off between the total average running time over all problem sizes and the total number of optimally solved instances. The total number of optimal solutions gained from $c_{default}$ is smaller than c_{tuned}, but the average running time results of $c_{default}$ are better and also statistically different throughout the non-parametric pairwise Wilcoxon test with significance level 0.05.

Table 1: Summarized performance of two configurations.

	#optimal solutions	#total average runtime (s)
$c_{default}$	5963	**603.2**
c_{tuned}	**5970**	693.1

From this analysis, the motivation for a multi-objective automated algorithm configurator becomes obvious. As being seen in table 1, when we consider the two performance measures' values over the whole test set, incomparable parameter configurations do exist. Therefore, result obtained from such a multi-objective algorithm configurator is a Pareto-approximation set of parameter configurations. The algorithm designer will do a manually deeper analysis on the resulting configurations, such as consideration of average running time on each problem sizes, the minimum and maximum running time, the average optimality gap, etc, to choose the final best algorithm configuration for future usage.

2.2 Solving Approach

In this part, we discuss our initial idea for a solution approach. Taking advantage of remarkable techniques currently available for single-objective algorithm configuration problem, we try to adapt them into the multi-objective context. Among the four groups of single-objective configurators, we decide to focus on the group of metaheuristics, due to their significant results on several hard algorithm configuration problems.

In order to design a solving approach for this case study, we firstly discuss the question of which metaheuristic should be chosen. An analysis of challenges when applying that metaheuristic to our multi-objective algorithm configuration problem as well as our proposed solution approach is presented. Finally, addition of preference information into our configurator is mentioned.

2.2.1 Choosing a Metaheuristic

Since we are solving an algorithm configuration problem, chosen metaheuristic should be parameter-independent as much as possible. We keep this condition in mind during the search for an appropriate multi-objective metaheuristic approach.

Multi-objective metaheuristics can be divided into three major groups: aggregation-based, Pareto-dominance-based and indicator-based methods (Basseur et al., 2012). Among them, we decide to focus on the aggregation-based group. The reason for our choice is the impossibility of using adaptive capping strategies in the search procedure when the two later groups are used. We explain capping strategies in the next sub-section.

In aggregation-based approaches, all objective-functions are integrated into a single function throughout some scalarization technique, in which various sets of weight for each objective function are considered sequentially, each set will guide the search towards a point of optimal Pareto front. The main advantage of these approaches is that if solving methods for single-objective version of the considered problem are already available, we can re-use them to solve the multi-objective problem in a fairly straight forward manner. However, these techniques have a major drawback when the single-objective algorithm requires expensive computational cost, since each run of this algorithm just aims to only one point in the true Pareto front. Algorithms in this group usually require parameters for specifying running time amount for each scalarization. Algorithms' performance is influenced by these parameters. To avoid such a parameter sensitivity, we choose the Adaptive Anytime Two-Phase Local Search (AA-TPLS) proposed in Dubois-Lacoste et al 2011, in which these running time amount values are adaptively chosen based on information collected from previous runs.

The underlying single-objective algorithm configurator used in AA-TPLS could be either ParamILS or GGA, since such a choice is independent of the general framework of AA-TPLS.

2.2.2 Adaptation: Challenges and Solutions

In this subsection, we firstly describe a group of techniques, named adaptive capping, for solving the

expensive evaluation problem; then, we discuss challenges faced in adaptations of the chosen metaheuristics into our configuration problem, as well as our initial ideas for solving them.

a) *Capping Techniques.* The most difficult challenge in solving algorithm configuration problems is the expensive evaluation of an algorithm configuration's quality, due to the usual large size of the training instance set. In ParamILS by Hutter et al 2009, special techniques, named adaptive capping methods, have been developed to reduce the computational effort required for this task. Given two parameter configuration c_1 and c_2, a problem instance set S and a performance measure f that needs to be minimized, assume that c_1 has been run on S already and its performance value is known. Now we want to compare c_1 and c_2 and discard the worse one. The idea of adaptive capping is to try to not run c_2 on the whole set S, but stopping earlier at some instance according to information collected before.

Here we briefly describe the two adaptive capping techniques proposed by Hutter et al 2009, namely trajectory-preserving and aggressive capping, and propose some additional adaptive capping methods to improve the reduction of computational cost.

- Trajectory-preserving: Let us assume that the performance measure f is the average running time over the whole instance set, N is the total number of problem instances in S, t_1 and t_2 are the total running time of c_1 and c_2, respectively, on S ($t_1 = f_1 \times N$). When we run c_2 on S, if the total running time value so far is larger than t_1, we can stop the run immediately and conclude that c_2 is worse than c_1, instead of running c_2 on the whole set S.
- Aggressive capping: this is a heuristic version of trajectory-preserving capping, in which the total running time t_{best} of the best configuration so far c_{best} is saved and $T \times t_{best}$ will serve as an upper bound for the running of other configurations. Here T is a predefined ratio ($T = 2$ in ParamILS's setting). When a configuration c_i is run on S, if t_i got so far exceeds the upper bound, c_i's run is stopped.
- Additional capping methods: In our case study, the target algorithm is deterministic. We do not have to deal with the difficulty of stochastic evaluation. Hence, another capping idea is that for each parameter configuration c and each problem instance p, the performance of c on p is saved when its running time exceeds a pre-defined threshold (we do not save all cases for reasons of memory size). This information is useful when c is revisited. Moreover, from this information, we can also roughly predict performance of a configuration c_i on p if c_i's neighbors have been run on p so far. The predicted value could be a weighted sum of c_i's neighbors' performance values on p, whereas the weight is smaller for larger dissimilarity between p and the neighbor. For a good prediction, it is reasonable to have an upper bound for the dissimilarity between c_i and its considered neighbors.

b) *Adaptation of AA-TPLS.* First, we briefly describe general ideas of the AA-TPLS in Dubois-Lacoste et al 2011. The AA-TPLS algorithm has two phases. In the first phase, the available single-objective algorithm is separately applied on a randomly chosen objective function to conduct an initial solution for the next phase. (Application on all objective functions is also possible. In this case, the best solution according to the first scalarization in phase 2 will be taken.) Then, in the second phase, a sequence of scalarizations will be called; the single-objective algorithm is used to solve the problem at each scalarization. Solution obtained from a scalarization will be used as the initial solution for its succeeding scalarization. Non-dominated solutions are also saved in an archive. The set of weight for each objective function in each scalarization is adaptively determined based on solutions from previous scalarization to guarantee the coverage the true-Pareto front.

The challenge when applying AA-TPLS to our problem lies in the algorithm's sequential implementation. Each step in phase 2 just provides only one solution, so that if the single-objective algorithm requires too large running time, it might be impossible to get a reasonable number of Pareto-optimal solutions. In the literature of algorithm configuration techniques, for problem with large number of parameters (more than 24), the appropriate amount of each configurator's run should be at least 10 hours. Therefore, the sequential implementation of AA-TPLS should be modified in order to deal with such an expensive computational running time. Here we propose a parallelization of AA-TPLS's phase 2 as depicted in Figure 1. We still keep a part of the sequential property due to its important role: adapting weight values in scalarizations based on information from previous steps to ensure the diversity of the obtained approximation front. The new scheme of phase 2 has

k sequential steps. At each step, n different scalarizations are run in parallel. In order to guide the search better, we can let them cooperate with each other: after some time interval, the best solutions obtained so far of every parallel scalarization are put into a global archive, then each scalarization will check this archive for a better solution than the best one found by itself, based on its currently used set of weight, if such a solution exists, it will be updated into the scalarization searching process.

It is worth noting that besides the usage of capping strategies mentioned in previous part, performance value (for each objective function) of the best solution found so far on every run problem instances will also be saved, and the order of training instances considered in every parallel scalarization at the same step are fixed. Thanks to that, when the best configuration c from some scalarization s_i^r is considered for entering some other scalarization s_j^r, the computational cost of evaluating c according to the s_j^r's set of weight will be lower, since we just need to run c on problem instances in s_i^r that have not been saved yet.

Besides, order of instances in the training set also influences ParamILS's or GGA's results. Since at each iteration, just a subset of the training set is used for configuring in order to save computational cost, this subset is extended gradually after time, so that the more iterations the algorithm runs, the more exact the estimation of performance measure value on the whole training set is. Therefore, in this AA-TPLS, to deal with such a variance, order of training instances are randomly shuffled after each step of phase 2.

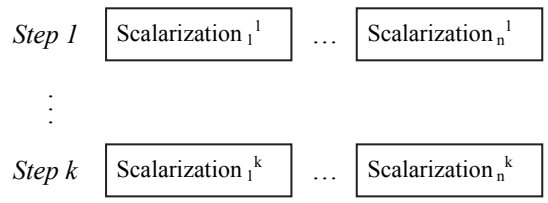

Figure 1: Parallel scheme of AA-TPLS's phase 2.

2.2.3 Integration of Preference Information

Trade-off between two considered performance measures is possible. For example, given two configurations c_1 and c_2, if the percentage of corresponding optimal solutions are respectively 90% and 70%, while the total average running time of c_1 is just some second less than c_2, c_1 is certainly preferred. Consequently, extreme points towards the average running time axis are not interesting and could be ignored. Moreover, configurations with small optimality gap for hard unsolved instances are also preferable. Integration of such preference information into the configuration algorithm is useful to reduce the computational effort and make a better matching with algorithm designer's desire. We are in presently collaborating with Tanaka, the author of the target algorithm SiPSi, to identify characteristics of this integration.

3 THE SECOND MOTIVATION

In this section, we explain the second algorithm configuration problem supporting the need for a multi-objective algorithm configurator: the case when the target algorithm is a non-exact Multi-Objective Optimization Algorithm (MOOA).

Given a MOOA configuration c and problem instance p, the result obtained from a run of c on p is an approximation set of the optimal Pareto front. In order to compare two MOOA configurations based on p, we can use different performance assessment methods, including unary indicators, binary indicators, statistical testing on multiple-run and attainment function analysis. We refer to Ziztler et al., 2008 for a comprehensive description. Among these methods, the unary indicator approaches, in which a real-value is assigned to each approximation set to define a total order in the objective function space, are the ones that usually require the lowest computational cost. However, each indicator has its own preference and might bias towards some specific parts the true Pareto front. As suggested in Knowles et at., 2006: "Each unary indicator is based on different preference information - therefore using them all will provide more information than using just one". Indeed, different unary indicators could give inconsistent conclusion, as observed in the experimental results of Jaeggi et al 2008. Moreover preference of some unary indicators have been theoretically shown, as e.g. in the analysis of hypervolume indicator in Auger et al., 2012 and of the R2 indicator in Brockhoff et al., 2012. Hence, the configuration task for a MOOA could be done as follows: first, a multi-objective algorithm configurator is used to find a set of good parameter configurations based on a set of unary indicators as performance measures; then, a post-processing step starts, in which a final best algorithm configuration is decided based on more expensive assessment methods, e.g., statistical testing over several runs with different indicators.

4 CONCLUSIONS

In this paper, we present two algorithm configuration problems in which more than one performance measures should be considered at the same time, leading to a multi-objective algorithm configurator. Development of the solving approach is still in progress. We are currently focusing on the first problem: balancing between algorithm configuration's robustness and speed, while the second one is reserved for our future work. Although several single-objective configurators with significant performance have been proposed in the literature, the application in a multi-objective context is quite challenging. However, we believe that such an effort is worthwhile. The consideration of more than one algorithm performance measure during the automated configuration process and the postponement of the final choice to a deeper analysis in a post-processing phase will give more flexibility to the algorithm designer. Indeed, the definition of a good algorithm configuration in practice usually depends on various performance perspectives.

ACKNOWLEDGEMENTS

The authors would like to thank Professor Thomas Stutzle for his valuable comments. This work is supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX. (*http://comex.ulb.ac.be*) and by Research Foundation Flanders (FWO).

REFERENCES

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In *Principles and Practice of Constraint Programming-CP 2009* (pp. 142-157). Springer Berlin Heidelberg.

Basseur, M., Zeng, R. Q., & Hao, J. K. (2012). Hypervolume-based multi-objective local search. *Neural Computing and Applications*, *21*(8), 1917-1929.

Brockhoff, D., Wagner, T., & Trautmann, H. (2012). On the Properties of the R2 Indicator. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference* (pp. 465-472). ACM.

Dubois-Lacoste, J., López-Ibáñez, M., & Stützle, T. (2011). Improving the anytime behavior of two-phase local search. *Annals of mathematics and artificial intelligence*, *61*(2), 125-154.

Hoos, H. H. (2012). Automated algorithm configuration and parameter tuning. In *Autonomous Search* (pp. 37-71). Springer Berlin Heidelberg.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*(1), 267-306.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (pp. 186-202). Springer Berlin Heidelberg.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization* (pp. 507-523). Springer Berlin Heidelberg.

Jaeggi, D. M., Parks, G. T., Kipouros, T., & Clarkson, P. J. (2008). The development of a multi-objective tabu search algorithm for continuous optimisation problems. *European Journal of Operational Research*, 185(3), 1192-121

Knowles, J., Thiele, L., & Zitzler, E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. Tik report, 214, 327-332.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004*.

Smit, S. K., & Eiben, A. E. (2011). Multi-problem parameter tuning using BONESA. In Artificial Evolution (pp. 222-233).

Stützle, T., & López-Ibáñez, M. (2013, July). Automatic (offline) configuration of algorithms. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion* (pp. 893-918). ACM.

Tanaka, S., & Fujikuma, S. (2012). A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. *Journal of Scheduling*, *15*(3), 347-361.

Zitzler, E., Knowles, J., & Thiele, L. (2008). Quality assessment of pareto set approximations. In *Multiobjective Optimization* (pp. 373-404). Springer Berlin Heidelberg.