Towards transferring algorithm configurations across problems

Alberto Franzin
IRIDIA, CoDE
Université Libre de Bruxelles
Brussels, Belgium
afranzin@ulb.ac.be

Thomas Stützle
IRIDIA, CoDE
Université Libre de Bruxelles
Brussels, Belgium
stuetzle@ulb.ac.be

Abstract

Automatic approaches for algorithm configuration and design have received significant attention in the last years, thanks to both the potential to obtain high performing algorithms, and the ease for algorithm designers and practitioners. One limitation of current methods is the need to repeat the task for every new scenario encountered. We show how the observation of problem-independent features of the solution landscape can enable the use of past experiments to infer good configurations for unseen scenarios, both in case of new instances and new problems. As a proof of concept, we report preliminary experiments obtained when configuring a metaheuristic with two parameters.

1 Introduction

Algorithm configuration (AC) or (hyper-)parameter tuning is a crucial task in many practical settings, where a parameterized algorithm is used to obtain the best possible results in a given scenario. Given a (without loss of generalization) minimization problem $\mathcal{P}$, a set of instances $\mathcal{I}$, an algorithm $\mathcal{A}_{\mathcal{P}}^{\Theta}$ for $\mathcal{P}$ with parameters $\Theta = \theta_1, \theta_2, \ldots, \theta_k$, and a measure of the cost $c(\cdot)$ of $\mathcal{A}_{\mathcal{P}}^{\Theta}$, the goal is to find Θ^* such that $c(\mathcal{A}_{\mathcal{P}}^{\Theta^*})$ is minimized over $\mathcal{I}$.

Recently, the development of automatic methods for AC has been the focus of various lines of research in areas such as Operations Research (OR) and Machine Learning (ML), where the performance of the algorithms used is crucial. Automatic algorithm configuration strategies can not only improve over final solution qualities or running times, but they also unburden developers and practitioners of the tedious task of manual configuration, and can be useful in understanding the problem at hand. Alongside automatic algorithm selection, with which it shares several similarities, automatic configuration is one of the key tasks of Automated Machine Learning (AutoML) pipelines. Configurators are, in a nutshell, machine learning algorithms that learn the best configurations for a training set, to be applied on a test set or in a production setting. When features of the instances are included in the learning process in order to adapt the final configuration to each specific instance of the test set, we talk about per-instance algorithm configuration.

A large collection of methods have been proposed to tackle the AC problem, using various strategies such as random search [5], model-based methods [11, 4], Bayesian optimization [18, 15], racing methods [6, 16], and local search algorithms [12, 2, 1].

One main limitation of current automatic AC methods is that a new tuning is usually required for every new given task or scenario. To obtain good results, configurations have to be obtained on a training set of instances that comes from the same distribution of the test set. The extent to which a certain configuration will generalize to different instance distributions is in principle unknown, and in general we can expect to observe a decrease in performance. One consequence of this is the difficulty of

1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020, Vancouver, Canada.

using configurations, and thus knowledge, obtained in past experiments to new, unseen scenarios. In this work, we explore the possibility of combining data obtained in previous configuration tasks with instance features to obtain instance-specific configurations, and apply this to metaheuristic algorithms for combinatorial optimization problems. In particular, our goal is to infer algorithm configurations for a certain problem, starting from configurations obtained on other problems. While a similar approach has been successfully applied to adjust hyperparameter configurations in machine learning tasks [8], to the best of our knowledge this is the first case of transfer learning in OR algorithms.

2 Materials and methods

We consider the fixed-temperature variant of Simulated Annealing (SA), for which we configure two parameters: (i) the initial temperature value (a real-valued scaling coefficient in the range $[0, 1]$), and (ii) the neighbourhood exploration (binary, to choose a neighbourhood move to be evaluated either randomly or following some ordering), identified as one of the main SA components [17, 14, 7, 9]. Conversely to the traditional "cooling" strategy of SA, the fixed-temperature variant (FT) keeps the same temperature value throughout the whole search.

We compute problem-independent instance features to use the same set of features across different problems. This contrasts with previous instance-specific configuration approaches for combinatorial optimization problems, where problem-specific instance features are usually considered [13]. The choice of a set of features that represents the main characteristics of an instance is an open problem in itself. We choose to use a set of features that represents the conditions encountered by the FT algorithm during the search. As the FT (and SA) are built by introducing a diversification mechanism on top of standard hill climbing strategies, we compute a total of forty probing landscape features by executing first and best improvement runs on each instance, and computing a series of statistics about the convergence and the neighbourhoods traversed. The most relevant features include statistics on the number of moves to reach a local optimum and on the amount of improving and plateau moves in the neighbourhoods traversed, and the slope of a linear model fit on the sequence of the average gaps from the best solution in the neighbourhood traversed.

Following a recent interpretation of algorithm configuration as an inference task, in this work we investigate if a generic inference tool can be an effective alternative in practice to classic AC packages [10]. As configurator we use the irace package, an R implementation of the racing algorithm [6, 16]. For computational reasons we perform approximate inference of parameter values from the landscape features using the MERCS package, which implements a forest of multi-directional decision trees [19].

The workflow for each experiment is as follows. First we tune a FT algorithm for a given problem and instance class, all instance sizes considered, using irace. Each tuning consists of 1000 evaluations of FT configurations for 10 seconds on an instance. We perform 15 tunings for each scenario, and the resulting configurations are then benchmarked on the test set. We then compute the forty landscape features for all the instances of the training set seen in the different tunings, associate each set of features with the results obtained by all the configurations evaluated on the relative instance, and select for each instance the configuration that obtained the best results. We finally compute the same set of landscape features for the instances of the test set, use MERCS to infer the best configuration for each test instance, and benchmark the outcome.

3 Per-instance algorithm configuration of metaheuristics

We start by exploring the possibility of performing per-instance algorithm configuration of metaheuristics using problem-independent instance features. We use the Quadratic Assignment Problem (QAP) with two instance classes (random and structured), each containing sizes 60, 80 and 100. Each instance class is partitioned into a training and a test set, each one with 50 instances per size.

In the first experiment, we consider separately the two instance classes. We have therefore one tuning for the random instances, and one for the structured instances. We also use only data obtained on the random training instances to infer good configurations on the random test instances, and do likewise for the structured instances.

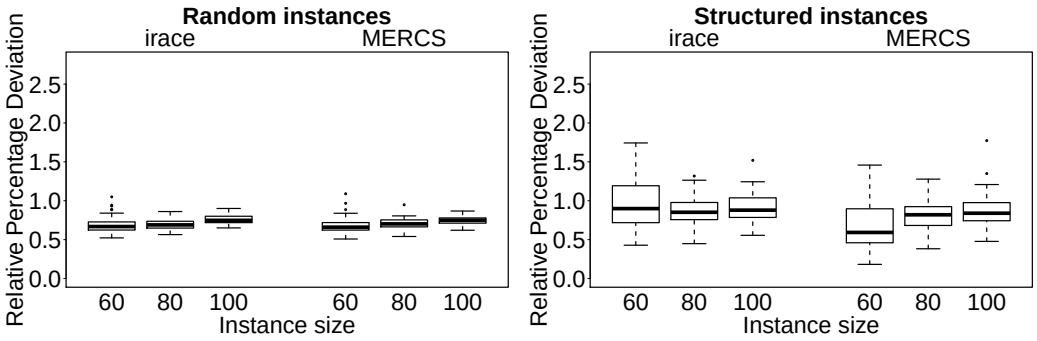

Figure 1: Results obtained in terms of relative percentage deviation from the best known solutions on the random and structured QAP instances by irace (separate tunings for each instance class) and MERCS; lower boxplots represent better results.

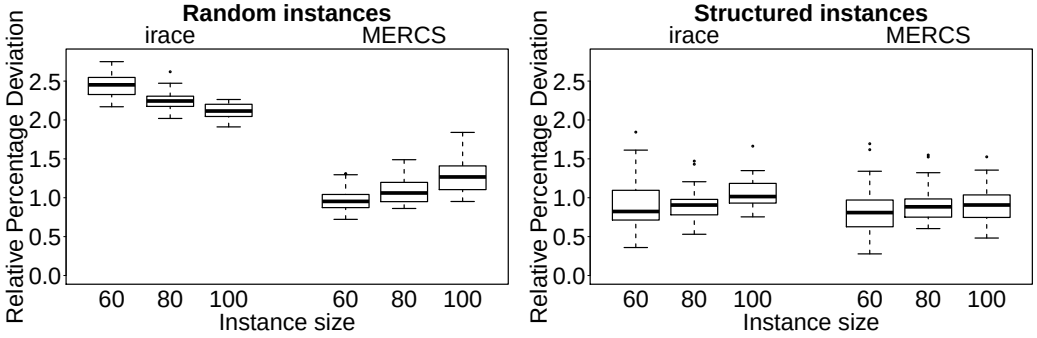

Figure 2: Results obtained in terms of relative percentage deviation from the best known solutions on the random and structured QAP instances by irace (one single tunings for both instance classes) and MERCS; lower boxplots represent better results.

The results are reported in Figure 1 in terms of percentage deviation $(z - \hat{z})/\hat{z} \times 100$ from the best known solutions $\hat{z}$, divided by instance size. In both instance classes the results are very similar. There is statistically significant difference only on the structured instances of sizes 60, where the inference with MERCS performs better than the regular tuning. This happens because, for each instance class, there is a relatively narrow interval of values that can obtain those results in the runtime allowed, and both irace and MERCS can identify it.

In the second experiment we consider the two instance classes together. We therefore perform one single tuning task, and use data from both instance classes to infer configuration on the whole test set. In this case, the configurations that globally obtain the best results across the two instance classes are suboptimal for each single class. The inference tasks is performed as in the previous case. The results are reported in Figure 2 in terms of percentage deviation from the best known solutions, divided by instance size. In this case, as expected, the outcome of the tuning task is not of the same quality as in the case of separate tunings. MERCS can, however, make use of the features to identify better parameter values in many cases, in particular on the random instances: only on the structured instances of sizes 60 and 80 the results are not statistically significantly different. The reason why the results of MERCS are not as good as in the previous experiment is the lack of better starting data, since irace in this case was not able to exploit the good areas of the parameter space.

4 Transferring configurations of metaheuristics across problems

Having seen how a generic inference tool can be successfully used for per-instance configuration, in this experiment we use the same approach based on problem-independent features to compute configurations across different problems using a generic inference tool. We apply the inference task as described in Section 2, but this time using data obtained on three problems to infer the best settings for a fourth, different problem.

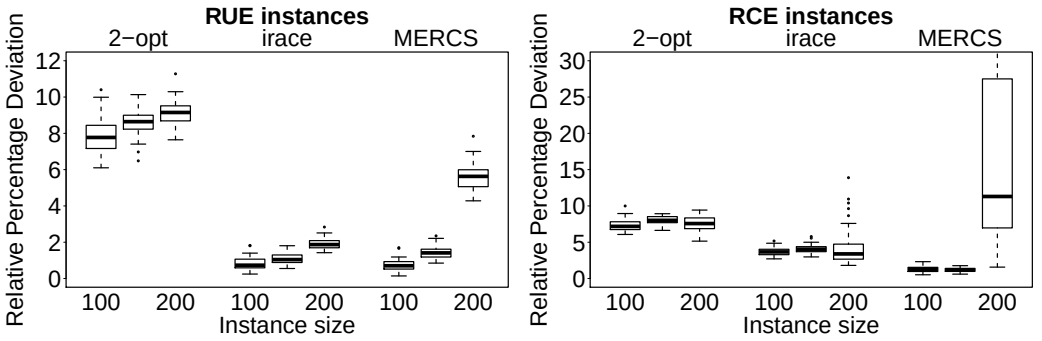

Figure 3: Results obtained in terms of relative percentage deviation from the optimal solutions on the random and clustered TSP instances by irace (separate tunings for each instance class) and MERCS (inference from QAP and PFSP experiments); lower boxplots represent better results.

The target problem is the Traveling Salesperson Problem (TSP), for which we have a set of random instances (RUE) and a set of clustered instances (RCE). Each instance class contains instances of sizes 100, 150 and 200 cities. The instances are equally divided in a training and a test set. The three problems we start from are the QAP and two objectives of the Permutation Flow Shop Problem (PSFP), the makespan objective and the total completion time objective. For each problem we have two instance classes (one random and one structured) of various sizes. The starting point for the inference is the data collected in six set of tunings, one for each objective function and instance class. So, in total, we have six different families of landscapes that we use as our training set. Analogously to the previous experiments, we compute the landscape features on the TSP test instances, and use MERCS to infer FT configurations starting from our training set. Using irace, we tune a FT on the training set of TSP instances, and benchmark the resulting configurations on the relative test set. As in the first experiment, we tune separately per instance class, considering all instance sizes together.

In Figure 3 we show the results obtained, in terms of relative percentage deviation from the optimal solutions computed using Concorde [3]. As a comparison, we report also the results that are obtained on our test set by a 2-opt heuristic.

On the RUE instances, the configurations obtained using MERCS performs similarly to those obtained by irace on sizes 100 and 150, while they perform significantly worse on size 200. They are still outperforming the solutions that can be found with a 2-opt local search. On the RCE instances, instead, per-instance configuration with MERCS outperforms the classical tuning on instance of sizes 100 and 150, but obtains very poor results on instances of size 200. In this latter case, there is a huge variance in the solution qualities that can be obtained, and in many instances MERCS is outperformed even by the 2-opt heuristic.

These results can be explained with the fact that, while the landscape shape generated by the three different sizes is the same for each instance class, the numerical values involved differ. The proper set of values for the size 200 TSP instances has not been observed in any of the experiments on the other three scenarios. The set of features we use are not sufficient to fully capture the relationships between the numerical values of the instances and the proper temperature values needed on the 200-size instances, especially on the RCE instances. On the other hand, while the landscapes of the four problems considered differ, sometimes significantly, in shape, the set of features observed is representative enough to infer good parameter values for the sizes 100 and 150, for both instance classes. This however suggests which kind of features would properly complement the set used in these experiments.

5 Conclusions

We have presented some preliminary results showing how, starting from an archive of experiments, generic conference tools can be used to perform per-instance algorithm configuration based on a representative set of problem-independent features, also on unseen problems. To the best of our knowledge, this is the first time transfer learning has been used in the context of algorithm configuration and optimization algorithms.

Acknowledgments and Disclosure of Funding

Alberto Franzin acknowledges support from the Innoviris project 2018-SHAPE-25a. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Director.

References

[1] Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15), pp. 733–739. IJCAI/AAAI Press, Menlo Park, CA (2015)

[2] Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming, CP 2009, LNCS, vol. 5732, pp. 142–157. Springer (2009). DOI: 10.1007/978-3-642-04244-7_14

[3] Applegate, D., Bixby, R.E., Chvátal, V., Cook, W.J.: Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html (2014), version visited last on 15 April 2014

[4] Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems (NIPS 24), pp. 2546–2554. Curran Associates, Red Hook, NY (2011), http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[5] Bergstra, J.S., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**, 281–305 (2012)

[6] Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Langdon, W.B., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, pp. 11–18. Morgan Kaufmann Publishers, San Francisco, CA (2002)

[7] Cohn, H., Fielding, M.J.: Simulated annealing: Searching for an optimal temperature. SIAM Journal on Optimization **9**(3), 779–802 (1999)

[8] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems (NIPS 28). pp. 2962–2970 (2015), http://papers.nips.cc/book/advances-in-neural-information-processing-systems-28-2015

[9] Franzin, A., Stützle, T.: Revisiting simulated annealing: A component-based analysis. Computers & Operations Research **104**, 191–206 (2019). DOI: 10.1016/j.cor.2018.12.015

[10] Franzin, A., Stützle, T.: A causal framework to understand optimisation algorithms. In: Foundations of Trustworthy AI - Integrating Learning, Optimization and Reasoning (2020)

[11] Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello Coello, C.A. (ed.) Learning and Intelligent Optimization, 5th International Conference, LION 5, LNCS, vol. 6683, pp. 507–523. Springer (2011)

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

[13] Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC: Instance-specific algorithm configuration. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) Proceedings of the 19th European Conference on Artificial Intelligence. pp. 751–756. IOS Press (2010)

[14] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)

[15] Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian optimization of machine learning hyperparameters on large datasets. In: Artificial Intelligence and Statistics. pp. 528–536. PMLR (2017)

[16] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016). DOI: 10.1016/j.orp.2016.09.002

[17] Mitra, D., Romeo, F., Sangiovanni-Vincentelli, A.: Convergence and finite-time behavior of simulated annealing. In: Decision and Control, 1985 24th IEEE Conference on. pp. 761–767. IEEE (1985)

[18] Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems (NIPS 25), pp. 2960–2968. Curran Associates, Red Hook, NY (2012)

[19] Van Wolputte, E., Korneva, E., Blockeel, H.: MERCS: multi-directional ensembles of regression and classification trees. AAAI pp. 4276–4283 (2018)