Towards transferring algorithm configurations across problems

Anonymous Author(s)
Affiliation
Address
email

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 characteristics (or 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.

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 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

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

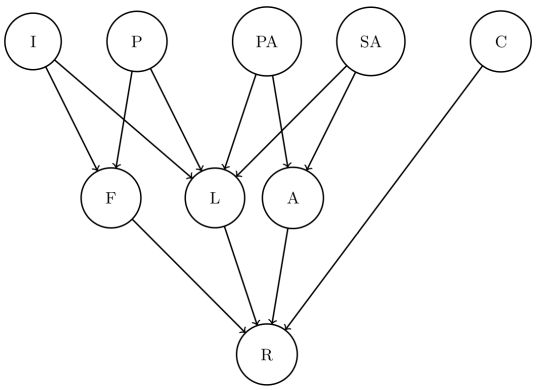

Figure 1: The causal framework proposed in [3] to relate the elements involved in the solution of an optimization problem.

configurations for a certain problem, starting from configurations obtained on other problems. To the best of our knowledge, this is the first case of transfer learning in OR algorithms.

2 A causal framework for optimization algorithms

In a recent work a causal framework has been proposed as a way of representing the interaction of the several entities that are involved in the solution of an optimization problem [3]. The framework is based on four working hypotheses: (i) optimization algorithms can be considered a combination of basic components and parameters; (ii) those basic components can be partitioned into problem-specific ones, and algorithm-specific ones, that define the algorithm behaviour; (iii) an algorithm can be considered as operating a traversal of the search space; and (iv) the performance of an algorithm can be measured. Based on these hypotheses, the interplay between problems, instances, algorithms and results is modeled in a causal fashion, as represented in Figure 1. In particular, problems (node P in the framework), instances (I), components and parameters (PA for the problem-specific ones, and SA for the algorithmic specific ones), and the computational environment (C) are the causes, and the results (R) obtained the effects. Problem features (F), landscape features (L), and fully-configured algorithms (A) compose a mediator level between the causes and the effects.

Under the framework of [3] it is possible to represent both the algorithm selection and configuration tasks as inference tasks on the same set of variables. This makes it possible to use any inference tool to perform configuration, allowing us to choose the ones we deem the most appropriate in each particular case, based on the data we start from. For these experiments we use MERCS [7].

3 Per-instance algorithm configuration

We use a fixed-temperature variant of Simulated Annealing (SA), for which we configure two parameters: (i) the initial temperature value, and (ii) the neighbourhood exploration, identified as one of the main SA components [6, 4, 1, 2]. The fixed-temperature algorithm (FT) uses the traditional Metropolis acceptance criterion, and can be seen as a variant of a hill climbing algorithm that can accept worsening moves with a probability that depends only on the relative worsening of the move evaluated, being the temperature value the same throughout the whole search. Consequently, we compute 40 probing landscape statistics using a first improvement and a best improvement algorithm. These features are problem independent, but relevant to represent the conditions under which the FT operates.

As a first experiment, we report the results obtained by performing a per-instance algorithm configuration as an inference task on two instance sets of the Quadratic Assignment Problem (QAP). The experiments comprise two QAP instance classes (random and structured), divided in three sizes, 60, 80 and 100. Our workflow is the following. We divide each instance class into a training set and a test set, each one with 50 instances for each size. We perform two classic tuning task for the FT algorithm using irace [5], one for the random and one for the structured instances, with a budget of

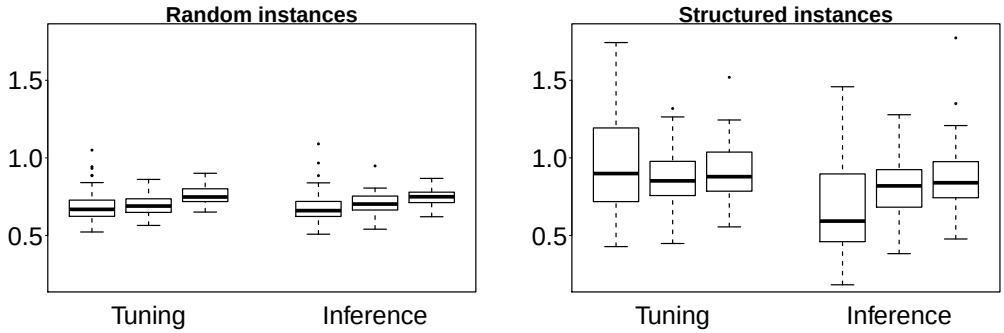

Figure 2: Results obtained in terms of relative percentage deviation from the best known solutions on the random and structured QAP instances by irace (separate tunings for each instance class) and MERCS.

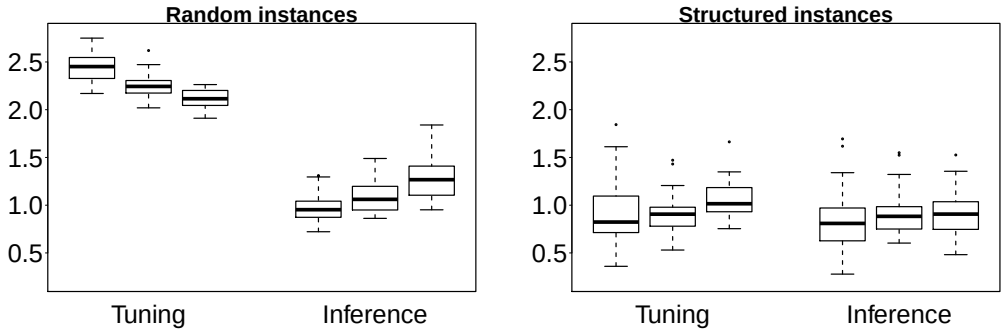

Figure 3: Results obtained in terms of relative percentage deviation from the best known solutions on the random and structured QAP instances by irace (one single tunings for both instance classes) and MERCS.

1000 experiments per tuning. Each experiment runs a FT algorithm on an instance for ten seconds. The resulting configurations are then benchmarked on the test set. Each tuning and testing is repeated 15 times. To perform configuration as an inference task, we compute the 40 landscape features for all the instances seen in the training, and associate each set of feature with the results obtained by the various configurations evaluated on the relative instance, selecting for each instance the configuration that obtained the best results. We then compute the same set of features on the test set, and use MERCS to infer the best parameter values for each instance of the test set. The results are reported in Figure 2 in terms of percentage deviation from the best known solutions, 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.

The results change when we perform one single tuning task, including both instance classes. In this case, we have one single configuration to apply to both the random and the structured instance test sets. The inference tasks is performed as in the previous case. The results are reported in Figure 3 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. Inference using MERCS can, however, identify better parameter values in many cases: only on the structured instances of sizes 60 and 80 the results are not statistically significantly different.

4 Transferring configurations across problems

Having seen how viewing algorithm configuration as an inference task is useful for per-instance configuration, in this second experiment we use the same approach based on problem-independent features to compute configurations across different problems using a generic inference tool. We apply

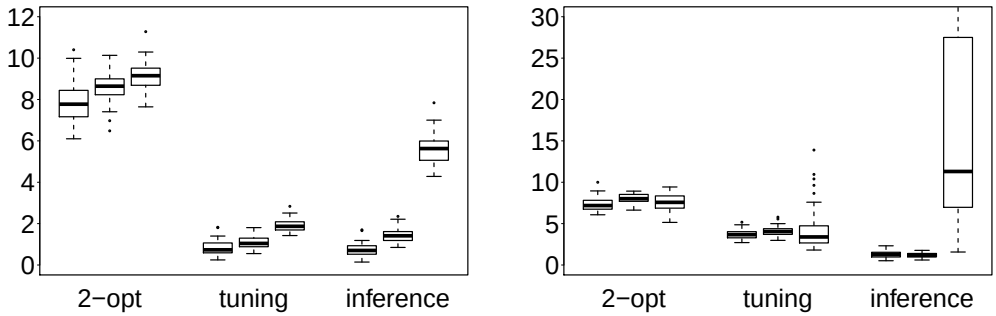

Figure 4: Results obtained in terms of relative percentage deviation from the optimal solutions on the random and clustered TSP instances by irace (separate tunings for each instance class) and MERCS (inference from QAP and PFSP experiments).

the inference task as described in the previous Section, using data obtained on three problems to infer the best settings for a fourth, different problem.

The three problems we start from are the Quadratic Assignment Problem 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 (random and structured) of various sizes. For each objective function and instance class we perform fifteen tunings, with a budget of 1000 experiments per tuning. In addition to the QAP instances used in the previous Section, also for the two PFSP objectives we use a set of random instances and a set of structured instances of various sizes. So, in total, we have six different families of landscapes that we use as our training set. 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. In Figure 4 we show the results obtained on the test set by the configurations found by a classic tuning task using irace and those found for each instance using inference with MERCS; as a comparison, we report also the results that obtained on our test set by the 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 via inference 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 it is often 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 set of features we use are not sufficient to fully capture the relationships between the numerical values and the proper temperature values needed on the 200-size instances. On the other hand, while the landscape shapes of the four problems considered differ, the set of features observed (and their values) is representative enough to infer good parameter values for the sizes 100 and 150, for both instance classes.

5 Conclusions

Building on a recently proposed causal framework to relate the elements involved in the solution of an optimization problem, 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, 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.

References

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

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

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

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

[5] 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). https://doi.org/10.1016/j.orp.2016.09.002

[6] 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)

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