The algorithm configuration problem

Gabriele Iommazzo[1,2], Claudia D'Ambrosio[1], Antonio Frangioni[2], and Leo Liberti[1]

1 Introduction

Automatic configuration of algorithmic parameters is an area of active research [15, 26], going back to the foundational work [31], published in 1976. The Algorithm Configuration Problem (ACP) focuses on configurable algorithms, deployed to solve instances of a given decision or optimization problem. It concerns the issue of how to identify the setup of algorithmic parameters delivering the best algorithmic performance, when the algorithm is run on its input.

Formally, we let $\mathcal{A}$ be a parametrized target algorithm. Its input consists of an instance of the problem being solved and an array of parameters. We call the latter "algorithmic configuration".

The inputs of the ACP are:

- Π: the decision/optimization problem to be solved by $\mathcal{A}$, consisting of a (potentially) infinite set of instances. Each instance is an input string for the problem; among the many encodings available for instance data, the most common one is a vector of discrete/continuous values, containing the most important attributes of the instance. In the following, we assume that several encodings can be transformed into each other efficiently (i.e., without too much loss of information) and we refer to Π as the set of *encoded* instances;
- $C_{\mathcal{A}}$: the set of parameter configurations of $\mathcal{A}$, i.e., an array of data of different types (boolean, numeric, categorical), usually encoded by vectors of q continuous and/or discrete/categorical values. Not all possible parameter values may be admissible, due to logical conditions concerning multiple parameters. Thus, for simplicity, we assume that $C_{\mathcal{A}}$ only contains feasible algorithmic configurations;
- $p_{\mathcal{A}}$: the performance function of $\mathcal{A}$

LIX CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France, e-mail: {dambrosio,giommazz,liberti}@lix.polytechnique.fr · Dipartimento di Informatica, Università di Pisa, Pisa, Italy, e-mail: frangio@di.unipi.it

$$p_{\mathcal{A}} : \Pi \times C_{\mathcal{A}} \longrightarrow \mathbb{R} \tag{1}$$

mapping a pair (π, c) (instance, parameter configuration) to the outcome of running $\mathcal{A}$, configured by c, to solve an instance π. The encoding of $p_{\mathcal{A}}$ is a single continuous or discrete value. The performance $p_{\mathcal{A}}$ related to running $\mathcal{A}$ could be a cost measure (e.g., CPU time, number of iterations performed, etc.) or a quality measure (e.g., the accuracy achieved by a Machine Learning (ML) predictor at a certain iteration of the training process, the integrality gap reported by an optimization solver within a certain time limit, etc.). Depending on the case at hand, one aims at appropriately minimizing or maximizing it.

With the specifications given above, the ACP is formally defined as follows:

Definition 1 (ACP) Given a tuple $(\mathcal{A}, \bar{\pi}, p_{\mathcal{A}})$, $\bar{\pi} \in \Pi$, find the algorithmic configuration $c^*_{\bar{\pi}} \in C_{\mathcal{A}}$ providing the optimal performance $p_{\mathcal{A}}$ of $\mathcal{A}$ on $\bar{\pi}$.

A variant of the ACP is the Algorithm Selection Problem (ASP), where one seeks to pick, from a given set of configured algorithms, the best one for solving a specific instance. However, one can see the choice of which algorithm to pick as the one single parameter of a meta-algorithm for solving Π, and therefore the ASP is a special case of the ACP. The ACP is generally very hard both in theory and in practice, especially when, as it often happens, algorithms have a large number of configurable parameters. Yet, it has a large number of very relevant applications, such as the configuration of constraint programming or mathematical optimization solvers, the hyperparameter tuning of ML pipelines, the administration of ad-hoc medical treatments, and many others; the interested reader is referred, e.g.,to [14, 26] and the references therein for a more detailed treatment of the subject.

In this article, we supply a comprehensive framework for describing any approach to the ACP: we identify the core building components for designing an ACP solution strategy, and discuss their possible use patterns and implementation.

In Tab. 1, we recall the abbreviations used throughout the article:

extended name	shorthand
Algorithm Configuration Problem	ACP
Machine Learning	ML
Knowledge-encoding Process	K-EP
Per-Problem	PP
Per-Instance	PI

Table 1: Recurring abbreviations

2 Definitions

Research efforts on the ACP have focused on developing methodologies to solve it efficiently in an automated fashion. Although this has been implemented in several different ways, all approaches share the same goal, i.e., the construction of a *recommender*:

Definition 2 (recommender) The recommender is a function

$$\Psi_{\mathcal{M}} : \Pi \rightarrow C_{\mathcal{A}} \tag{2}$$

which, given an instance $\pi \in \Pi$, is capable of selecting a configuration $c^*_\pi \in C_{\mathcal{A}}$ for solving π more efficiently than with other configurations of $\mathcal{A}$.

Thus, c^*_π is, hopefully, a good approximation of the optimal configuration for π, with respect to the performance function $p_{\mathcal{A}}$. In other words, a recommender is a heuristic for the ACP. The fundamental constraint is that the recommender should be able to produce its output in a "short" time, so as not to offset the advantages due to choosing a better configuration.

Our notation underlines the fact that the structure of $\Psi_{\mathcal{M}}$ is usually determined by a model $\mathcal{M}$ encoding some knowledge about $p_{\mathcal{A}}$. In fact, an important phase of all ACP methodologies is devoted to the building of $\mathcal{M}$. It is in general difficult (if at all possible) to construct accurate enough analytical models of the performances of complex algorithms. Thus, most practical $\mathcal{M}$ are "data-driven", in the sense that they are constructed from experiments. For the recommender to be able to choose the right c^*_π for a given π, it should be able to assess $p_{\mathcal{A}}$ anywhere on the set $\Pi \times C_{\mathcal{A}}$. However, an exhaustive evaluation of $p_{\mathcal{A}}$ over $\Pi \times C_{\mathcal{A}}$ is almost always impossible in practice. In fact, Π is an infinite set, and $C_{\mathcal{A}}$ usually grows exponentially in the number of parameters, which can be large. Furthermore, since $p_{\mathcal{A}}$ itself is typically a black-box function (i.e., it has no analytic form), the only way to evaluate it is to directly run $\mathcal{A}$, which can be extremely costly. Therefore, a significant component of ACP approaches is how the set $\Pi \times C_{\mathcal{A}}$ is explored.

Given the difficulty of assessing algorithmic performance on $\Pi \times C_{\mathcal{A}}$ exhaustively, the construction of $\Psi_{\mathcal{M}}$ in Eq. (2) always involves the selection of sets $\Pi' \subset \Pi$ and $C'_{\mathcal{A}} \subseteq C_{\mathcal{A}}$. Of these, Π' is meant to be "representative" of Π, usually in the sense that it preserves the characteristics and information of Π. Sometimes, this can instead be taken as the fact that Π' contains the most "difficult" instances for $\mathcal{A}$, in that all others are solved efficiently and do not require a dedicated algorithmic configuration. Since there is no automatic way of choosing Π', most ACP approaches take it as given and rely on existing libraries, hand-picked by problem experts. Therefore, we assume that Π' is always available, or can be easily generated, for deploying an ACP methodology. Further, we assume that Π' is specified before the construction of $\mathcal{M}$, and never updated. Moreover, the algorithmic performance of different configurations is typically unknown before launching an ACP approach, otherwise, there would be no need to even construct a recommender $\Psi_{\mathcal{M}}$. This means that $C'_{\mathcal{A}}$ is often selected during the construction of $\Psi_{\mathcal{M}}$, instead of being

picked *a priori*, as in the case of Π'; we remark that the (partly constructed) model $\mathcal{M}$ can also be useful in this context.

In all approaches in the literature, solving the ACP fits into the same two-stage framework (see Fig. 1), encompassing the ordered execution of:

a) a *Knowledge-encoding Process* (for brevity, K-EP). The K-EP builds $\mathcal{M}$ and the accompanying $\Psi_{\mathcal{M}}$. A critical step in the computation of $\mathcal{M}$ is the sampling of the performance function, i.e., the evaluation of $p_{\mathcal{A}}$ over pairs $(\pi, c) \in \Pi' \times C_{\mathcal{A}}$. Since $C_{\mathcal{A}}$ may be quite large, in most cases computing $p_{\mathcal{A}}$ on all the configurations is too expensive. Thus, in all ACP approaches, the selection of an appropriate subset of $C_{\mathcal{A}}$ in the K-EP is a crucial task, which may require the use of $\mathcal{M}$ or additional models. Instead, when $C_{\mathcal{A}}$ is small, all the $c \in C_{\mathcal{A}}$ can be considered;
b) a *Recommendation Phase*. The recommendation phase deploys $\Psi_{\mathcal{M}}$, in order to produce a suitable configuration for a given instance. Thus, $\Psi_{\mathcal{M}}$ supplies a solution of the ACP for that instance.

Since, as we noted above, most $\mathcal{M}$ are data-driven, the K-EP can demand considerable computational resources. The recommendation phase can also be computationally expensive, in that exploiting $\mathcal{M}$ to produce the output configuration may involve, e.g., the solution of a nontrivial optimization problem in itself.

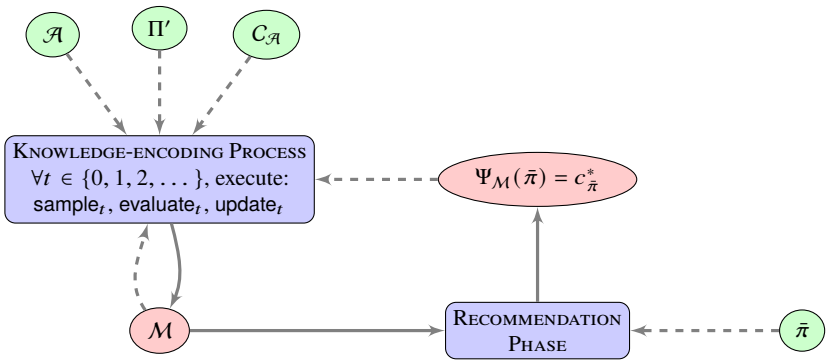

Fig. 1: Algorithmic schema of an ACP approach

The K-EP is an iterative procedure. It cycles through three phases at each iteration $t \in \{0, \dots, T\}$, until an allotted computational budget (quantified, e.g., in terms of allowed target algorithm runs, CPU/GPU time and power, memory usage, number of K-EP iterations) is used up, or $\mathcal{M}$ has attained a desired accuracy:

1. sample$_t$: picks a set

$$\mathscr{S}_t \subset \Pi' \times C_{\mathcal{A}} \quad s.t. \quad \mathscr{S}_t \cap \bigcup_{h<t} \mathscr{S}_h = \emptyset, \tag{3}$$

i.e., $\mathscr{S}_t$ only contains previously unsampled couples. This selection is nontrivial, as it requires addressing the trade-off between uniformly exploring $\Pi' \times C_{\mathcal{A}}$, so that no promising area is left unexplored (diversification), and concentrating on the areas containing the most promising candidates, so as to find better solutions (intensification);

2. $\mathsf{evaluate}_t$: executes (possibly, in parallel) the target algorithm $\mathcal{A}$ on all the points picked by sample_t, to compute $p_{\mathcal{A}}$ at those points and build the set

$$\mathcal{S}_t = \left\{ \left(\pi,\ c,\ p_{\mathcal{A}}(\pi, c) \right) \mid (\pi, c) \in \mathscr{S}_t \right\}; \tag{4}$$

3. update_t: updates the models employed in the K-EP, i.e., the model $\mathcal{M}$ of $\Psi_{\mathcal{M}}$ and, potentially, other models used for sampling purposes. For instance, it may entail training or re-training an ML model. This phase exploits the set $\bigcup_{h \leq t} \mathscr{S}_h$, the performance values computed at the points of that set and, sometimes, some other information collected in the $\mathsf{evaluate}_t$ phase.

The data generated by the recommendation phase may be employed in an adaptive "meta-sampling" loop whereby: a) when a new instance $\bar{\pi} \in \Pi \setminus \Pi'$ is given (either by the user of the recommender, or by a dedicated process aimed at improving its quality), one computes $\Psi_{\mathcal{M}}(\bar{\pi})$; b) the recommended configuration and/or $\bar{\pi}$ are fed into the K-EP, which can be performed again to improve $\mathcal{M}$. One example of this approach is presented in [22].

The implementation of the K-EP and the recommendation phase depends on the choice of the following components:

- a model $\mathcal{M}$ and the associated recommender $\Psi_{\mathcal{M}}$;
- whether $\Psi_{\mathcal{M}}$ actually depends on a specific instance or always provides the same answer for a set of instances: we call *Per-Instance* (PI) ACP approaches of the former type, and *Per-Problem* (PP) approaches of the second type;
- whether one commits to building $\mathcal{M}$ before actually solving an unknown instance by $\mathcal{A}$ (*offline* ACP methodologies) or $\mathcal{M}$ is constructed during an algorithm run (*online* methodologies).

3 Formulations

3.1 The construction of $\mathcal{M}$

In the literature, the model $\mathcal{M}$ built in the K-EP is one of the following:

(a) a function

$$\zeta_{\mathcal{A}} : \Pi \longrightarrow C_{\mathcal{A}}, \tag{5}$$

mapping an instance encoding to the configuration recommended for that instance. The function in Eq. (5) is usually constructed by ML techniques [8, 7, 10, 21];

(b) a function

$$\bar{p}_{\mathcal{A}} : \Pi \times C_{\mathcal{A}} \longrightarrow \mathbb{R}, \tag{6}$$

computing an approximation of the performance function $p_{\mathcal{A}}$ defined by Eq. (1), also generally built by ML techniques [6, 9, 16, 20]. Sometimes [11], $\bar{p}_{\mathcal{A}}$ is aggregated (e.g., averaged) over the instances in Π', which yields an estimate

$$\chi_{\mathcal{A}} : C_{\mathcal{A}} \longrightarrow \mathbb{R} \tag{7}$$

of the performance of single configurations over Π. The functions $\bar{p}_{\mathcal{A}}$ or $\chi_{\mathcal{A}}$ are then used as a proxy to recommend a configuration for the new instance, typically, by solving an optimization problem having them in the objective;

(c) a partition

$$\mathscr{P}_{\mathcal{A}} = \{\Pi'_i \subseteq \Pi'\}_{i \in C} \tag{8}$$

of Π' into C disjoint subsets (or "clusters") Π'_i, whereby each Π'_i is specified by choosing the corresponding recommended configuration c^*_i and, for instance, a representative instance π_i. When a new instance $\bar{\pi} \in \Pi$ has to be solved, the cluster to which it belongs is determined (e.g., by finding the closest representative π_i, under some appropriate distance metric) and the corresponding c^*_i is retrieved.

We remark that there are two "extreme" cases of Eq. (8):

- one in which $C = |\Pi'|$, i.e., each Π'_i contains exactly one instance, [5, 30]. We refer to this case as "$\mathscr{P}_{\mathcal{A},|\Pi'|}$";
- one whereby $\Pi'_0 = \Pi'$ and $C = 1$, i.e., the partition is trivial and a single configuration c^*_0 will be recommended regardless of the input $\bar{\pi}$. This is customary in PP approaches [1, 2, 3, 4, 13, 11, 17, 18, 29, 28]. We refer to this case as "$\mathscr{P}_{\mathcal{A},1}$".

Moreover, we refer to intermediate case, whereby $1 < C < |\Pi'|$, as "$\mathscr{P}_{\mathcal{A},C}$".

It should be noted that this choice of $\mathcal{M}$ is typically strongly coupled with the sample$_t$ phase in the K-EP, in the sense that it is often the direct result of the algorithmic decisions there.

The strategy implemented to construct $\mathscr{P}_{\mathcal{A}}$ is that of solving the problem

$$c^*_i = \arg\min\{\, \mathsf{agg}_{\pi \in \Pi_i} p_{\mathcal{A}}(\pi, c) \mid c \in C_{\mathcal{A}} \,\}, \tag{9}$$

where agg is some aggregation function (say, the average). The issue here is to find the configuration providing the best aggregated performance with respect to the subset Π'_i at hand. Since $C_{\mathcal{A}}$ is often very large and $p_{\mathcal{A}}$ is "black-box", the problem in Eq. (9) is typically treated by heuristic search algorithms, such as local searches, evolutionary algorithms or other metaheuristics, providing a local solution. Further, such heuristic algorithms can themselves naturally identify clusters on the fly, even in the extreme cases; for instance, evolutionary algorithms produce a population of the fittest individuals, each of which can be used to define a cluster representative π_i.

We remark that the strategy of building $\mathscr{P}_{\mathcal{A}}$ by solving Eq. (9), usually adopted by offline ACP methodologies, quite naturally extends to the online setting. In the online ACP, further evaluations of $p_{\mathcal{A}}$ are often triggered by the arrival of a new π, that restarts the optimization process.

Some ACP methodologies combine more than one model; the possible combinations are shown in Tab. 2.

3.2 Computing $\Psi_{\mathcal{M}}$

Given an instance $\bar{\pi} \in \Pi$, the implementation of $\Psi_{\mathcal{M}}(\bar{\pi})$ depends on the model $\mathcal{M}$ built during the K-EP:

(a) If $\mathcal{M}$ is the one in Eq. (5), then $\Psi_{\mathcal{M}}(\bar{\pi}) := \zeta(\bar{\pi})$;
(b) if $\mathcal{M}$ is the one in Eq. (6), then

$$\Psi_{\mathcal{M}}(\bar{\pi}) := \arg\min\{\, \bar{p}_{\mathcal{A}}(\bar{\pi}, c) \mid c \in C_{\mathcal{A}} \,\}\,. \tag{10}$$

The problem in Eq. (10) may be a hard one, calling for heuristic solution algorithms akin to those possibly employed in the K-EP, i.e., treating the approximation $\bar{p}_{\mathcal{A}}$ as a "black box" (e.g., [6, 16, 32, 33, 11, 27]). However, exploiting the mathematical structure of $\bar{p}_{\mathcal{A}}$ is also possible, allowing the use of exact approaches [20];

(c) if $\mathcal{M}$ is specified by a set of clusters $\Pi'_i \subseteq \Pi'$ as in Eq. (8), then $\Psi_{\mathcal{M}}(\bar{\pi})$ is implemented by first solving

$$h(\bar{\pi}) = \operatorname{argmin}_i \mathsf{dist}(\Pi'_i, \bar{\pi})\,, \tag{11}$$

for some appropriate distance function $\mathsf{dist}(\cdot,\cdot)$, and then returning $c^*_{h(\bar{\pi})}$. If the clusters are specified via a representative instance, say, π_i for cluster Π_i, then

$$\mathsf{dist}(\Pi'_i, \bar{\pi}) = \|\bar{\pi} - \pi_i\|\,,$$

for some appropriate norm $\|\cdot\|$. When the number of clusters is low, as it usually happens, Eq. (11) can be quickly solved by direct enumeration. In the extreme case with only one cluster, instead, the problem is trivial.

3.3 Classifying algorithm configuration approaches

PP approaches search for the configuration with the best overall performance over a problem set; they are usually based on one of the models $\mathcal{M}$ described at point (c) of Sec. 3.1. The main risk of PP approaches is that they produce suboptimal ACP solutions when the performance of the target algorithm varies considerably

between instances of a problem. This is to be expected for large problem classes, e.g. MILP, where instances can represent very different problems that are hard for very different reasons (say, having very few feasible solutions, so that finding even one is challenging, or very many feasible solutions with very close objective, so that finding the optimal one is challenging).

When the risk of selecting suboptimal configurations, PI methodologies, which assume that the ACP-optimal algorithmic configuration depends on the instance at hand, are likely to achieve better results.

An ACP methodology is offline if it commits to building $\mathcal{M}$ before the recommendation phase, during which, therefore, $\mathcal{M}$ is fixed. Instead, an online methodology performs the entire K-EP during the execution of $\mathcal{A}$. In this case, recommendation phase and K-EP coincide, and the information retrieved by running $\mathcal{A}$ is dynamically exploited, on-the-fly, to build $\mathcal{M}$. Online methods can only be employed when the target algorithm is launched to solve a sequence of instances, or is tasked with making a series of decisions during its run.

An online procedure can be used as a component of a larger offline/online approach. One way to accomplish this would be to construct, online, $\mathcal{M}$ through experiments on Π', and, then, deploy it as a recommender for instances similar to those in Π'. A very straightforward implementation of this approach would be to, say, run an optimization solver on a sequence of instances Π', within a prescribed time limit, trying different parameter configurations on each of them. This would allow the selection and storage of a set of parameter values ensuring high solver performance (as in model $\mathscr{P}_{\mathcal{A}}$ in Eq. (8)); they could be reused to configure the solver, and to run it on different instances in Π. For example, the CPLEX Tuning Tool [19, Ch. 10], of the IBM ILOG CPLEX solver, implements this procedure.

Another way to reuse an online $\mathcal{M}$ could be to employ it in the update_0 phase of a subsequent K-EP, to initialize the construction of a new model. Yet another option may be to use the points (instance, configuration, related performance), sampled to construct $\mathcal{M}$ online (by the sample_t and the $\mathsf{evaluate}_t$ phases), in the sample_0 phase of a following K-EP.

Since the purpose of online approaches is to solve the ACP on-the-run, they are usually based on simple algorithms, which allow for rapid decision-making. However, these approaches are often impractical to scale to large configuration sets. For this reason, they are ordinarily used for solving the ASP, rather than the ACP.

In Tab. 2, we give an overview of the main ACP approaches in the literature. From the rightmost column of the table, we gather that all PP methodologies are based on the model $\mathscr{P}_{\mathcal{A}}$ described by Eq. (8), notably, on its $\mathscr{P}_{\mathcal{A},1}$ variant. Instead, the $\mathscr{P}_{\mathcal{A},C}$ and $\mathscr{P}_{\mathcal{A},|\Pi'|}$ variants are always used in the PI setting. Further, most PI approaches rely on the ML-derived models $\bar{p}_{\mathcal{A}}$ of Eq. (6) and $\zeta_{\mathcal{A}}$ of Eq. (5). In fact, PI methodologies are required to capture/encode the complicated, possibly nonlinear relationships between $p_{\mathcal{A}}$, $C_{\mathcal{A}}$ and Π, and several ML paradigms are capable of producing extremely accurate approximation. In some cases, ML-based models and variants of $\mathscr{P}_{\mathcal{A}}$ are combined, in order to implement PI procedures: see, e.g., the approaches on the second to last line of the table, which rely on both $\mathscr{P}_{\mathcal{A},1}$ and $\bar{p}_{\mathcal{A}}$ for online and offline algorithm configuration.

	PI		PP
$\mathcal{M}$	offline	online	offline
$\zeta_{\mathcal{A}}$ (Eq. (5))	[8]*, [7, 10], **[21]**		
$\bar{p}_{\mathcal{A}}$ (Eq. (6))	[6, 9, 16], **[20]**		
$\mathscr{P}_{\mathcal{A},C}$ (Eq. (8))	[12, 22]*, [23]		
$\mathscr{P}_{\mathcal{A},1}$ (Eq. (8))		[5]	[1, 2, 3, 4, 13] [18, 29, 28]
$\mathscr{P}_{\mathcal{A},\lvert\Pi'\rvert} + \chi_{\mathcal{A}}$ (Eq. (8) + (7))		[30]	
$\mathscr{P}_{\mathcal{A},1} + \bar{p}_{\mathcal{A}}$ (Eq. (8) + (6))	[27, 32, 33]	[27]	[17]
$\mathscr{P}_{\mathcal{A},1} + \chi_{\mathcal{A}}$ (Eq. (8) + (7))			[11]

Table 2: A schematic summary the ACP literature; * indicates ASP approaches.

4 Conclusions

We introduced an algorithmic schema, common to all ACP methodologies, for constructing and deploying a recommender, i.e., a function capable of suggesting the optimal configuration of a given algorithm $\mathcal{A}$ for solving an instance of a given decision/optimization problem Π.

Despite all the research efforts in the field of algorithm configuration, the problem still remains extremely difficult to solve. In fact, it is usually impossible to know the algorithmic performance $p_{\mathcal{A}}$ over all of the many parameter configurations in $C_{\mathcal{A}}$, which can be up to the hundreds or thousands (especially in general-purpose optimization solvers, equipped with a diverse set of algorithmic components to solve famously NP-hard problems [24, 25]). Furthermore, while the ACP can be somewhat approached if we consider a single instance or a small subset of Π', it becomes intractable when we look at the whole set Π', which is normally of infinite size.

To overcome this complication, ACP methodologies are all based on multiple forms of approximation of: a) the algorithmic performance function, or of the parameter configuration allowing to achieve specific algorithmic performances, via some computable ML approximation; b) Π, via the manual selection of a subset of representative instances and corresponding representative configurations.

References

1. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using Fractional Experimental Design and Local Search. Operations Research **54**(1), 99–114 (2006)
2. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, pp. 142–157. Springer-Verlag, Berlin, Heidelberg (2009)

3. Audet, C., Kien, D., Orban, D.: Algorithmic parameter optimization of the DFO method with the OPAL framework. Software Automatic Tuning: From Concepts to State-of-the-Art Results pp. 255–274 (2010)
4. Audet, C., Orban, D.: Finding optimal algorithmic parameters using Derivative-Free Optimization. SIAM Journal on Optimization **17**(3), 642–664 (2006)
5. Battiti, R., Brunato, M.: Reactive Search: machine learning for memory-based heuristics. Tech. rep., University of Trento (2005)
6. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Per instance algorithm configuration of CMA-ES with limited budget. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 681–688. ACM, New York, NY, USA (2017)
7. Berthold, T., Hendel, G.: Learning to scale mixed-integer programs. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 5, pp. 3661–3668. AAAI Press (2021)
8. Boas, M.G.V., Santos, H.G., de Campos Merschmann, L.H., Berghe, G.V.: Optimal decision trees for the algorithm selection problem: Integer programming based approaches. International Transactions in Operational Research **28** (2019)
9. Boas, M.G.V., Santos, H.G., de S. O. Martins, R., Merschmann, L.H.C.: Data mining approach for feature based parameter tuning for mixed-integer programming solvers. In: P. Koumoutsakos *et al.* (ed.) Proceedings of the International Conference on Computational Science, vol. 108, pp. 715–724. Elsevier (2017)
10. Bonami, P., Lodi, A., G. Zarpellon, G.: Learning a classification of mixed-integer quadratic programming problems. In: W.J. van Hoeve (ed.) Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Lecture Notes in Control and Information Sciences, vol. 10848, pp. 595–604. Springer, Cham (2018)
11. C. Ansótegui *et al.*: Model-based genetic algorithms for algorithm configuration. In: Proceedings of the 24th International Conference on Artificial Intelligence, pp. 733—739. AAAI Press (2015)
12. Collins, A., Tierney, L., Beel, J.: Per-instance algorithm selection for recommender systems via instance clustering. CoRR **abs/2012.15151** (2020)
13. Cáceres, L.P., Stützle, T.: Exploring variable neighborhood search for automatic algorithm configuration. Electronic Notes in Discrete Mathematics **58**, 167–174 (2017)
14. Eggensperger, K., Lindauer, M., Hutter, F.: Pitfalls and best practices in algorithm configuration. Journal of Artificial Intelligence Research **64**(1), 861—893 (2019)
15. Gupta, R., Roughgarden, T.: A PAC approach to application-specific algorithm selection. In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, pp. 123—-134. Association for Computing Machinery, New York, NY, USA (2016)
16. Hutter, F., Hamadi, Y.: Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Tech. Rep. MSR-TR-2005125, Microsoft Research (2005). Https://www.microsoft.com/en-us/research/publication/parameter-adjustment-based-on-performance-prediction-towards-an-instance-aware-problem-solver/
17. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential Model-based Optimization for general algorithm configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization, pp. 507–523. Springer-Verlag (2011)
18. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36**(1), 267–306 (2009)
19. IBM: IBM ILOG CPLEX Optimization Studio, CPLEX 12.7 User's Manual. IBM (2016)
20. Iommazzo, G., D'Ambrosio, C., Frangioni, A., Liberti, L.: A learning-based mathematical programming formulation for the automatic configuration of optimization solvers. In: G. Nicosia *et al.* (ed.) Proceedings of the 6th International Conference on Machine Learning, Optimization, and Data Science, Information Systems and Applications, incl. Internet/Web, and HCI, vol. 12565, pp. 700–712. Springer (2020)
21. Iommazzo, G., D'Ambrosio, C., Frangioni, A., Liberti, L.: Learning to configure mathematical programming solvers by mathematical programming. In: I.S. Kotsireas, P.M. Pardalos (eds.) Proceeedings of the 14th International Conference on Learning and Intelligent Optimization -

14th International Conference, Lecture Notes in Computer Science, vol. 12096, pp. 377–389. Springer (2020)
22. J. Pérez *et al.*: A statistical approach for algorithm selection. In: C.C. Ribeiro, S.L. Martins (eds.) Proceedings of the International Workshop on Experimental and Efficient Algorithms, pp. 417–431. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
23. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC: Instance Specific Algorithm Configuration. In: Proceedings of the 19th European Conference on Artificial Intelligence, pp. 751–756. IOS Press, Amsterdam, The Netherlands (2010)
24. Kannan, R., Monma, C.L.: On the Computational Complexity of Integer Programming Problems. In: R. Henn, B. Korte, W. Oettli (eds.) Optimization and Operations Research, pp. 161–172. Springer Berlin Heidelberg, Berlin, Heidelberg (1978)
25. Katta, K.G., Kabadi, S.N.: Some NP-complete problems in quadratic and nonlinear programming. Mathematical Programming **39**(2), 117–129 (1987)
26. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated Algorithm Selection: Survey and Perspectives. Evolutionary Computation **27**(1), 3–45 (2019)
27. M. Feurer *et al.*: Efficient and robust automated machine learning. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, pp. 2755—2763. MIT Press, Cambridge, MA, USA (2015)
28. M. López-Ibáñez *et al.*: The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
29. Nannen, V., Eiben, A.E.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, pp. 975–980. Morgan Kaufmann Publishers Inc. (2007)
30. Pavón, R., Díaz, F., Laza, R., Luzón, V.: Automatic parameter tuning with a Bayesian case-based reasoning system. a case of study. Expert Systems with Applications **36**(2), 3407–3420 (2009)
31. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
32. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically configuring algorithms for portfolio-based selection. In: Proceedings of the National Conference on Artificial Intelligence, vol. 1 (2010)
33. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: Proceedings of the RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (2012)