A learning-based mathematical programming formulation for the automatic configuration of optimization solvers ⋆

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

[1] LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France,
{dambrosio,giommazz,liberti}@lix.polytechnique.fr
[2] Dip. di Informatica, Università di Pisa, Pisa, Italy,
frangio@di.unipi.it

Abstract. We propose a methodology, based on machine learning and optimization, for selecting a solver configuration for a given instance. First, we employ a set of solved instances and configurations in order to learn a performance function of the solver. Secondly, we formulate a mixed-integer nonlinear program where the objective/constraints explicitly encode the learnt information, and which we solve, upon the arrival of an unknown instance, to find the best solver configuration for that instance, based on the performance function. The main novelty of our approach lies in the fact that the configuration set search problem is formulated as a mathematical program, which allows us to a) enforce hard dependence and compatibility constraints on the configurations, and b) solve it efficiently with off-the-shelf optimization tools.

Keywords: automatic algorithm configuration, mathematical programming, machine learning, optimization solver configuration, hydro unit committment

1 Introduction

We address the problem of finding instance-wise optimal configurations for general Mathematical Programming (MP) solvers. We are particularly motivated by state-of-the-art general-purpose solvers, which combine a large set of diverse algorithmic components (relaxations, heuristics, cutting planes, branching, ...) and therefore have a long list of user-configurable parameters; tweaking them can have a significant impact on the quality of the obtained solution and/or on the efficiency of the solution process (cf., e.g., [20]). Good solvers have effective default parameter configurations, carefully selected to provide good performances

⋆ This paper has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement n. 764759 "MINOA".

in most cases. Furthermore, tuning tools may be available (e.g., [24, Ch. 10]) which run the solver, with different configurations, on one or more instances within a given time limit, and record the best parameter values encountered. Despite all this, the produced parameter configurations may still be highly suboptimal with specific instances. Hence, a manual search for the best parameter values may be required. This is a highly nontrivial and time-consuming task, due to the large amount of available parameters (see, e.g., [25]), which requires a profound knowledge of the application at hand and an extensive experience in solver usage. It is therefore of significant interest to develop general approaches capable of performing it efficiently and effectively in an automatic way.

This setting is an instance of the Algorithm Configuration Problem (ACP) [15, 33]. Our approach for addressing the ACP on MP solvers is based on a two-fold process:

(i) in the *Performance Map Learning Phase* (PMLP), supervised Machine Learning (ML) techniques [31] are used to learn a *performance function* which maps some features of the instance being solved and the parameter configuration into some measure of solver efficiency and effectiveness;

(ii) the formal model underlying the ML methodology used in the PMLP is translated into MP terms; the resulting formulation, together with constraints encoding the compatibility of the configuration parameter values, yields the *Configuration Set Search Problem* (CSSP), a Mixed-Integer Nonlinear Program (MINLP) which, for a given instance, finds the configuration providing optimal performance w.r.t. the performance function.

The main novelty of our approach lies in the fact that we explicitly model and optimize the CSSP using the mathematical description of the PMLP technique. This is in contrast to most of the existing ACP approaches, which instead employ heuristics such as local searches [2, 22], genetic algorithms [3], evolutionary strategies [9] and other methods [30]. Basically, most approaches consider the performance function as a black box, even when it is estimated by means of some ML technique, and therefore cannot reasonably hope to find a global minima when the number of parameter grows. Rather, we can exploit the mathematical structure of the CSSP solving it with sophisticated, off-the-shelf MP solvers. Moreover, formulating the CSSP by MP allows the seamless integration of the compatibility constraints on the configuration parameters, which is something that other ACP methods may struggle with. The idea of using a ML predictor to define the unknown components (constraints, objective) of a MP has been already explored in *data-driven optimization.* In general, it is possible to represent the ML model of a mapping/relation as a MP (or, equivalently, a Constraint Programming model) and optimize upon this [29]. However, while this is in principle possible, the set of successful applications in practice is limited. Indeed, using this approach in the ACP context is, to the best of our knowledge, new; it also comes with some specific twists. We tested this idea with the following components: we configured nine parameters of the IBM ILOG CPLEX solver [24], which we employed to solve instances of the Hydro Unit Commitment (HUC) problem [8], we chose Support Vector Regression (SVR) [34] as the PMLP learning methodology, and we use the off-the-shelf MINLP solver Bonmin [6] to solve the CSSP.

The paper is structured as follows: in Sec. 2 we will review existing work on algorithm configuration; in Sec. 3 we will detail our approach and provide the explicit formulation of the CSSP with SVR; in Sec. 4 we will discuss some computational results.

2 Background

2.1 The algorithm configuration problem

The ACP [33] is defined as follows: given a set of instances of a problem class, a target algorithm to solve them, its set of parameters, and a measure of the performance of the target algorithm on a pair (instance, algorithmic configuration), find the parameter configuration providing optimal algorithmic performance according to the given measure, on a specific instance or instance set.

Most algorithmic solvers have a very high number of configurable parameters of various types (boolean, categorical, integer, continuous), which usually makes the ACP very hard to solve in practice. Notably, this issue significantly affects MP solvers: they are highly complex pieces of software, embedding several computational components that tackle the different phases of the solution process; the many available algorithmic choices are exposed to the user as a long list of configurable parameters (for example, more than 150 in CPLEX [25]).

Approaches to the ACP can be compared based on how they fit into the following two categories: Per-Set (PS) or Per-Instance (PI); offline or online.

In PS approaches, the optimal configuration is defined as the one with the best overall performance over a set of instances belonging to the same problem class. Therefore, PS approaches first find the optimal configuration for a problem class and then use it for any instance pertaining to that class. The exploration of the configuration set is generally conducted by means of heuristics, such as various local search procedures [2, 22, 4], genetic algorithms [3] or other evolutionary algorithms [32], racing methods [30]. In this context, an exception is, e.g., the approached described in [21], which predicts the performance of the target algorithm by random forest regression and then uses it to guide the sampling in an iterative local search.

PS approaches, however, struggle when the target algorithm performance varies considerably among instances belonging to the same problem class. In these cases, PI methodologies, which assume that the optimal algorithmic configuration depends on the instance at hand, are likely to produce better configurations. PI approaches typically focus on learning a good surrogate map of the performance function: this approximation is used to direct the search in the configuration set. Some of them perform regression. In [23], for example, linear basis function regression is used to approximate the target algorithm runtime, which is defined as a map of both features and configurations; then, for a new instance with known features, the learnt map is evaluated at all configuration points in an exhaustive search, to find the estimated best one. However, other approaches may be used: in [9], for example, a map from instance features to optimal configuration is learnt by a neural network and then employed by an evolutionary algorithm in order to explore the configuration set upon the arrival of a new instance; in [7] the ACP is restricted to a single binary parameter of CPLEX, and a classifier is then trained to predict it. In [27], instead, CPLEX is run on a given instance for a certain amount of computational resources, then a learning-to-rank [31] ML model is trained, on-the-fly, to learn the ordering of branch and bound variables, and it is then used to predict the best branching variable, at each node, for the the rest of the execution. Another approach, presented in [26], relies instead on a PS methodology to find a good PI configuration: first, it performs clustering on a set of instances, then, it uses the algorithm described in [3] to find one good algorithmic configuration for each cluster. A different attempt at using clustering is detailed in [37], where instances are automatically clustered in the leaves of a trained decision tree, which also learns the best configuration for each leaf; at test time, a new instance is assigned to a leaf based on its features, and it receives the corresponding configuration.

Solving the ACP requires sampling the performance function. This involves expensive and, generally, repeated executions of the target algorithm. PI methodologies usually build a large training set all at once. On the other hand, PS methodologies focus on sampling only a small set of points at each iteration of the search and execute the target algorithm on them; the procedure learns which candidates must be discarded and where to focus the sampling at the next iteration. However, they potentially perform many more iterations than PI approaches.

The purpose of an ACP approach is to provide a good algorithmic configuration upon the arrival of an unseen instance. We call a methodology offline if the learning happens before that moment, which is the case for all the approaches cited above. Otherwise, we call an ACP methodology online; these approaches normally use reinforcement learning [35] techniques (see, e.g., [16, 13]) or other heuristics [5].

In our approach, we define the performance of the target algorithm as a function of both features and controls, in order to account for the fact that the best configuration of a solver may vary among instances belonging to the same class of problems; this makes our approach PI. Moreover, we perform the PMLP only once and then apply the resulting CSSP to all new instances. What makes our approach stand out from other methodologies is that the learning phase is treated as white-box: the prediction problem of the PMLP is formulated as a MP. This allows the explicit embedding of the a mathematical encoding of the estimated performance into the CSSP, as its objective/constraints, as opposed to treating the learned predictor as a black-box, and therefore using as an oracle in brute-force searches or similar heuristics, that typically do not scale well. Lastly, we chose to implement an offline approach because, since the exploration phase in our specific application must be solved quickly for each instance, and since the learning part can be very computationally heavy, the PMLP needs to be performed before solving the CSSP.

3 The PMLP and the CSSP

Let $\mathcal{A}$ be the target algorithm, and:

- $\mathcal{C}_{\mathcal{A}}$ be the set of feasible configurations of $\mathcal{A}$. We assume that each configuration $c \in \mathcal{C}_{\mathcal{A}}$ can be encoded into a vector of binary and/or discrete values representing categorical and numerical parameters, and $\mathcal{C}_{\mathcal{A}}$ can be described by means of linear constraints;
- Π be the problem to be solved, consisting of an infinite set of instances, and $\Pi' \subset \Pi$ be the (finite) set of instances used for the PMLP phase;
- F_{Π} be the set of feature vectors used to describe instances, encoded by vectors of continuous or discrete/categorical values (in the latter case they are labelled by reals);
- $p_{\mathcal{A}} : F_{\Pi} \times \mathcal{C}_{\mathcal{A}} \longrightarrow \mathbb{R}$ be the performance function which maps a pair (f, c) (instance feature vector, configuration) to the outcome of an execution of $\mathcal{A}$ (say in terms of the integrality gap reported by the solver after a time limit, but other measures are possible).

With the above definitions, the PMLP and the CSSP are detailed as follows.

3.1 Performance Map Learning Phase

In the PMLP phase we use a supervised ML predictor, e.g., SVR, to learn the coefficient vector $\bar{\theta}$ providing the parameters of a prediction model $\bar{p}_{\mathcal{A}}(\cdot, \cdot, \theta) : F_{\Pi} \times \mathcal{C}_{\mathcal{A}} \to \mathbb{R}$ of the performance function $p_{\mathcal{A}}(\cdot, \cdot)$. The training set for the PMLP is

$$\mathcal{S} = \big\{(f_i, c_i, p_{\mathcal{A}}(f_i, c_i)) \mid i \in \{1 \dots s\}\big\} \;\subseteq\; F_{\Pi'} \times \mathcal{C}_{\mathcal{A}} \times \mathbb{R}, \tag{1}$$

where $s = |\mathcal{S}|$ and the training set labels $p_{\mathcal{A}}(f_i, c_i)$ are computed on the training vectors (f_i, c_i).

3.2 Configuration Space Search Problem

For a given instance f and parameter vector θ, $\mathsf{CSSP}(f, \theta)$ is the problem of finding the configuration with best estimated performance $\bar{p}_{\mathcal{A}}(f, c, \theta)$:

$$\mathsf{CSSP}(f, \theta) \equiv \min_{c \in \mathcal{C}_{\mathcal{A}}} \bar{p}_{\mathcal{A}}(f, c, \theta)\,. \tag{2}$$

The actual implementation of $\mathsf{CSSP}(f, \theta)$ depends on the MP formulation selected to encode $\bar{p}_{\mathcal{A}}$, which may require auxiliary variables and constraints to define the properties of the ML predictor. If $\bar{p}_{\mathcal{A}}$ yields an accurate estimate of $p_{\mathcal{A}}$, we expect the optimum $\bar{c}$ of $\mathsf{CSSP}(f, \theta)$ to be a good approximation of the true optimal configuration c^* for solving f. However, we remark that a) $\mathsf{CSSP}(f, \theta)$ can be hard to solve, and b) it needs to be solved quickly (otherwise one might as well solve the instance f directly). Achieving a balance between PMLP accuracy and CSSP cost is one of the challenges of this research.

4 Experimental results

We tested our approach on 108 instances of the HUC problem and on 9 parameters of CPLEX, version 12.7. The PMLP and CSSP experiments were conducted on an Intel Xeon CPU E5-2620 v4 @ 2.10GHz architecture, while CPLEX was run on an Intel Xeon Gold 5118 CPU @ 2.30GHz. In the following, we detail the algorithmic set-up that we employed.

4.1 Building the dataset

1. *Features.* The HUC is the problem of finding the optimal scheduling of a pump-storage hydro power station, where the commitment and the power generation of the plant must be decided in a short term period, in which inflows and electricity prices are previously forecast. The goal is to maximize the revenue given by power selling (see, e.g., [1]). The time horizon is fixed to 24h and the underlying hydro system is also fixed, so that all the instances have the same size. Thus, only 55 elements that vary from day to day are features: the date, 24 hourly prices, 24 hourly inflows, initial and target water volumes, upper and lower bound admitted on the water volumes. We encode them in a vector f of 55 continuous/discrete components. All the instances have been randomly generated with an existing generator that accurately reproduces realistic settings.
2. *Configuration parameters.* Thanks to preliminary tests, we select a subset of 9 discrete CPLEX parameters (`fpheur`, `dive`, `probe`, `heuristicfreq`, `startalgorithm` and `subalgorithm` from `mip.strategy`; `crossover` from `barrier`; `mircuts` and `flowcovers`, from `mip.cuts`), for each of which we consider between 2 and 4 different values. We then combine them so as to obtain 2304 parameter configurations. A configuration is encoded by a vector $c \in \{0, 1\}^{23}$, where each categorical parameter is represented by its incidence vector.
3. *Performance measure.* We use the integrality gap to define $p(f, c)$. It has been shown that MIP solvers can be affected by performance variability issues (see, e.g., [28]), due to executing the solver on different computing platforms, permuting rows/columns of a model, adding valid and redundant constraints, performing apparently neutral changes to the solution process, etc. In order to tackle this issue, first we sample three different random seeds. For each instance feature vector f and each configuration c, we then carry out the following procedure: (i) we run CPLEX (using the Python API) 3 times on the instance, using the different random seeds, for 60 seconds; (ii) we record the middle out of the three obtained performance values, to be assigned to the pair (f, c). At this point, our dataset contains $108 \times 2304 = 248832$ records, each with dimension $55 + 23 + 1 = 79$. The performance measure $\varrho(f, c)$, thus, obtained from CPLEX output usually contains some extremely large floating point values (e.g., whenever the CPLEX gap has a value close to zero in the denominator), which unduly bias the learning process. We deal with this issue as follows: we compute the maximum $\bar{\varrho}$ over all values of (the

range of) ϱ lower than a given threshold (set to 1e+5 in our experiments), re-set all values of ϱ larger than the threshold to $\bar{\varrho} + 100$, then rescale ϱ so that it lies within the interval $[\,0\,,\,1\,]$.

4. *Feature engineering.* We process the date in order to extract the season, the week-day, the year-day, two flags called `isHoliday` and `isWeekend`, and we perform several sine/cosine encodings, that are customarily used to treat cyclical features. Moreover, we craft new features by computing statistics on the remaining 52 features. This task takes around 12 minutes to complete for the whole data set.
5. *Splitting the dataset.* We randomly divide the instances into 81 In-Sample (IS) and 27 Out-of-Sample (OS), and split the dataset rows accordingly (186624 IS and 62208 OS). We use the IS data to perform Feature Selection (FS) and to train the SVR predictor; then, we assess the performance of the PMLP-CSSP pipeline both on OS instances, to test its generalization capabilities to unseen input, and on IS instances, to evaluate its performance on the data that we learn from, as detailed below.
6. *Feature selection for the PMLP.* We use Pearson's Linear Correlation (LC), decision trees' Feature Importance (FI) and Mutual Information (MI) to get insights on which features contribute the most to yield accurate predictions. A detailed explanation of the employed FS techniques falls outside of the scope of this document, but the interested reader can refer to [18] (for MI and LC), [19, Ch. 10.13,15.3] (for FI) and [11, Ch. 2] (for MI), among many others. In the following, we use the shorthand "attributes" to refer to the whole list of columns of the learning dataset. In order to perform FS, we use a dedicated subset of the IS dataset, composed of 27994 records and only employed for this task; performing the selected FS techniques on this dataset takes around 8 minutes, and reduces f to 21 components. For the configuration vectors we consider two FS scenarios: `fewC`, where FS is applied more aggressively, and `manyC`, yielding c vectors with, respectively, 14 and 18 components. We then filter the PMLP dataset according to the selected attributes. For each instance: a) we delete all the dataset rows where the components eliminated from c vectors by FS are not set to their `default` value, and b) for each subset of remaining duplicates, we compute the average p keeping only one row. Lastly, we delete the dataset columns that we singled out by FS. At this point, the `fewC` PMLP dataset is composed of 26658 records and the `manyC` one has 106325, but in both scenarios we only use 8k points for the PMLP.

4.2 PMLP experimental setup

The PMLP methodology of choice in this paper is SVR. Its advantages are: (a) the PMLP for training an SVR can be formulated as a convex Quadratic Program (QP), which can be solved efficiently; (b) even complicated and possibly nonlinear performance functions can be learned by using the "kernel trick" [34]; (c) the solution of the PMLP for SVR provides a closed-form algebraic expression of the performance map $\bar{p}_{\mathcal{A}}$, which yields an easier formulation of $\mathsf{CSSP}(f, \theta)$. We use a gaussian kernel during SVR training, which is the default choice in absence of any other meaningful prior [12]. We assess the prediction error of the predictor by Nested Cross Validation (NCV) [10, 36]; furthermore, our training includes a phase for determining and saving the hyperparameters and the model coefficients of the SVR. These two task take approximately 1h in the `fewC` scenario and 1.5h in the `manyC` one.

A common issue in data-driven optimization is that using customary ML error metrics may not lead to good solutions of the optimization problem (see, for example, [14, 17]). We tackled this issue by comparing the classical Mean Absolute Error, $\mathsf{MAE}_S = \sum_{i \in S} |p_i - \bar{p}_i|$, where $p_i = p_i(f, c)$ and $\bar{p}_i = \bar{p}_i(f, c)$, to the custom metric $\mathsf{cMAE}_S(\delta) = \sum_{i \in S} loss_i$, where

$$loss_i = \begin{cases} (\bar{p}_i - p_i) \cdot (1 + \frac{1}{1+exp(p_i - \bar{p}_i)}) & \text{if } p_i \leq \delta,\ \bar{p}_i > p_i \\ (p_i - \bar{p}_i) \cdot (1 + \frac{1}{1+exp(\bar{p}_i - p_i)}) & \text{if } p_i \geq 1 - \delta,\ \bar{p}_i < p_i \\ (p_i - \bar{p}_i) & \text{if } \delta \leq p_i \leq 1 - \delta \\ 0 & \text{otherwise.} \end{cases}$$

We remark that we prefer the MAE to the mean squared error $\sum_{i \in S} (p_i - \bar{p}_i)^2$ because it is less sensitive to outliers. Moreover, we want $\bar{p}$ to accurately predict the points around the global minimum/maximum of p. In order to achieve this, at δ-minima (i.e., points s.t. $p_i \leq \delta$), the cMAE penalizes prediction overestimates while allowing any underestimates, and viceversa at δ-maxima; at points s.t. $\delta \leq p_i \leq 1 - \delta$, instead, the cMAE behaves exactly like the MAE.

4.3 CSSP experimental setup

The choice of a gaussian kernel in the SVR formulation makes the CSSP a MINLP with a nonconvex objective function $\bar{p}_{\mathcal{A}}$. More precisely, our CSSP is:

$$\min_{c \in \mathcal{C}_{\mathcal{A}}} \sum_{i=1}^{s} \alpha_i \exp\left(-\gamma \|(f_i, c_i) - (\bar{f}, c)\|_2^2 \right) \tag{3}$$

where, for all $i \leq s$, (f_i, c_i) belong to the training set, α_i are the dual solutions of the SVR, γ is the scaling parameter of the Gaussian kernel, and $\mathcal{C}_{\mathcal{A}}$ is defined by mixed-integer linear programming constraints encoding the dependences/compatibility of the configurations. We use the nonlinear solver Bonmin [6], manually configured and with a time limit of 60 seconds, to solve CSSP; then we retrieve, for each instance f, the Bonmin solution `CSSPsol`. Since we have enumerated all possible configurations, we can also compute the "true" global optimum $\texttt{CSSPglobMin} = \arg\min\{\bar{p}(f, c), c \in \mathcal{C}_{\mathcal{A}}\}$ for sake of comparison. In Table 1, we report the percentage of cases where $\texttt{CSSPsol} = \texttt{CSSPglobmin}$, ("%glob. min. found") and, for all the instances where this is not true, the average distance between $\bar{p}(\texttt{CSSPsol})$ and $\bar{p}(\texttt{CSSPglobmin})$, over all the instances of the considered set ("avg. dist. from glob. min").

The `fewC` scenario achieves better results than the `manyC` one: the percentage of global optima found is higher and the average errors are lower. This is probably

		set type	%glob. min. found	avg. dist. from glob. min
`fewC`	cMAE	IS	76.54	1.7849E-02
		OS	74.07	1.8602E-02
	MAE	IS	69.14	3.0438E-02
		OS	85.19	1.0607E-02
`manyC`	cMAE	IS	61.73	3.7913E-02
		OS	55.56	4.7817E-02
	MAE	IS	61.73	3.9297E-02
		OS	70.37	3.2590E-02

Table 1. Quality of Bonmin's CSSP solutions

motivated by the fact that the `fewC` formulation has less variables and, therefore, may be easier to solve. Thus, devising more efficient techniques to solve the CSSP (say, reformulations, decomposition, ...) might be necessary if our approach is scaled to considerably more algorithmic parameters. However, in both cases the errors are fairly small, i.e., the local optima found by Bonmin are quite good ones.

4.4 Results

In order to assess the performance of the approach, we retrieve $p($`CSSPsol`$)$ and $p($`CPXdefault`$)$ (CPLEX default configuration, where all parameters are set to "auto") from the filtered dataset, for every IS and OS instance. Table 2 shows: the number of times that $p($`CSSPsol`$)$ is less than, equal to or greater than $p($`CPXdefault`$)$, by the first four decimal digits of p in scientific notation ("w-d-l", for wins-draws-losses); the percentage of "w" ("%wins"); the average difference between $p($`CPXdefault`$)$ and $p($`CSSPsol`$)$, over all the considered instances, in case of win ("avg wins"); the average difference between $p($`CSSPsol`$)$ and $p($`CPXdefault`$)$, over all the considered instances, in case of loss ("avg loss").

		set type	w-d-l	%wins	avg wins	avg loss
`fewC`	cMAE	IS	66-10-5	81.48	5.0193E-01	2.2005E-01
		OS	14-6-7	51.85	4.7891E-01	2.6643E-01
	MAE	IS	64-10-7	79.01	5.4334E-01	2.8343E-01
		OS	12-6-9	44.44	4.6950E-01	1.6058E-01
`manyC`	cMAE	IS	65-10-6	80.25	5.4402E-01	2.7065E-01
		OS	17-6-4	62.96	5.0012E-01	1.5118E-01
	MAE	IS	65-11-5	80.25	5.2016E-01	2.4697E-01
		OS	12-6-9	44.44	4.2594E-01	2.6081E-01

Table 2. Pipeline quality with `CSSPsol`

The percentage of wins on IS instances varies between 79% and 82%, while the non-worsenings (i.e., wins and draws, over the total) vary between 91% to 94%.

From this we gather that $\bar{p}$ provides an accurate approximation p's global minima at points belonging to the training set. Although the percentage of wins on OS instances lowers to 44%-63% and also shows higher variability, our approach still outperforms `CPXdefault` on more than half of the cases, and has 67% to 86% of non-worsenings. The `fewC` scenario presents worse "%wins" results than the `manyC` one. Moreover, we observe that cMAE-based PMLP models provide lower "avg wins" than MAE-based models on IS instances but higher on OS instances, and lower "avg losses" on both IS and OS instances. Furthermore, they have a better performance than MAE-based PMLP models, in terms of "avg wins", on OS instances in the `fewC` scenario; they always dominate, however, in the `fewC` one.

All in all, the results show that our approach is promising, in that it provides configurations that are, generally speaking, better than these provided by the heuristics inside CPLEX. The results also show that a number of important details have to be properly accounted for before the approach can deliver good performances. In particular, it is interesting that the custom cMAE error metric has higher generalization capabilities than the classical MAE, which it outperforms on OS instances. Since the choice of the PMLP error metric determines the quality of the CSSP solution, this can be expected. Yet, this indicates that applying ML techniques to the ACP requires taking into account the specific aspects of the problem.

References

1. van Ackooij, W., D'Ambrosio, C., Liberti, L., Taktak, R., Thomopulos, D., Toubaline, S.: Shortest path problem variants for the hydro unit commitment problem. Electronic Notes in Discrete Mathematics **69**, 309 – 316 (2018), joint EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization (EURO/ALIO 2018)
2. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using Fractional Experimental Design and Local Search. Operations Research **54**(1), 99–114 (2006)
3. 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. CP'09, Springer-Verlag, Berlin, Heidelberg (2009)
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. Bonami, P., B., L.T., Conn, A.R., CornuéJols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., WäChter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization **5**(2), 186–204 (2008)
7. Bonami, P., Lodi, A., Zarpellon, G.: Learning a classification of mixed-integer quadratic programming problems. In: van Hoeve (eds), W. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2018. Lecture Notes in Control and Information Sciences, vol. vol. 10848, pp. 595–604. Springer, Cham (2018)

8. Borghetti, A., D'Ambrosio, C., Lodi, A., Martello, S.: An MILP approach for short-term hydro scheduling and unit commitment with head-dependent reservoir. IEEE Transactions on Power Systems **23**(3), 1115–1124 (2008)
9. Brendel, M., Schoenauer, M.: Instance-based parameter tuning for Evolutionary AI Planning. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation. pp. 591–598. GECCO '11, ACM (2011)
10. Cawley, G.C., Talbot, N.L.: On over-fitting in model selection and subsequent selection bias in performance evaluation. J. Mach. Learn. Res. **11**, 2079–2107 (2010)
11. Cover, T.M., Thomas, J.A.: Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing). Wiley-Interscience, USA (2006)
12. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods. Cambridge University Press (2000)
13. Degroote, H., Bischl, B., Kotthoff, L., De Causmaecker, P.: Reinforcement Learning for automatic online algorithm selection - an empirical study. In: Proceedings of the 16th ITAT Conference Information Technologies - Applications and Theory. pp. 93–101 (2016)
14. Demirović, E., Stuckey, P., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., Guns, T.: An investigation into prediction + optimisation for the knapsack problem. In: CPAIOR. pp. 241–257 (2019)
15. Eggensperger, K., Lindauer, M., Hutter, F.: Pitfalls and best practices in algorithm configuration. CoRR **abs/1705.06058** (2017)
16. Gagliolo, M., Schmidhuber, J.: Algorithm selection as a Bandit problem with unbounded losses. In: Blum, C., Battiti, R. (eds.) Learning and Intelligent Optimization: 4th International Conference, LION 4, 2010. Selected Papers, pp. 82–96. Springer Berlin Heidelberg (2010)
17. Grimes, D., Ifrim, G., O'Sullivan, B., Simonis, H.: Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. Sustainable Computing: Informatics and Systems (SUSCOM) **4**, 276–291 (2014)
18. Guyon, I.: An introduction to variable and feature selection. Journal of Machine Learning Research **3**, 1157–1182 (2003)
19. Hastie, T., Tibshirani, R., Friedman, J.H.: The elements of statistical learning: data mining, inference, and prediction, 2nd Edition. Springer series in statistics, Springer (2009)
20. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 186–202. CPAIOR'10, Springer-Verlag, Berlin, Heidelberg (2010)
21. 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. LION'05, Springer-Verlag (2011)
22. 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)
23. Hutter, F., Youssef, H.: Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Tech. rep., In: Technical Report: MSR-TR-2005125, Microsoft Research (2005)
24. IBM: IBM ILOG CPLEX Optimization Studio, CPLEX 12.7 User's Manual. IBM (2016)

25. IBM: IBM ILOG CPLEX Optimization Studio CPLEX Parameters Reference (2016)
26. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC: Instance Specific Algorithm Configuration. In: Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence. pp. 751–756. IOS Press, Amsterdam, The Netherlands (2010)
27. Khalil, E.B., Bodic, P.L., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 724–731. AAAI'16, AAAI Press (2016)
28. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. Tutorials in Operations Research, Vol. 10 pp. 1–12 (2013)
29. Lombardi, M., Milano, M., Bartolini, A.: Empirical decision model learning. Artificial Intelligence **244**, 343–367 (2017)
30. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
31. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. The MIT Press, 2nd edn. (2018)
32. 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. IJCAI'07, Morgan Kaufmann Publishers Inc. (2007)
33. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
34. Smola, A.J., Schölkopf, B.: A tutorial on support vector regression. Statistics and Computing **14**(3), 199–222 (2004)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, 2nd edn. (2018)
36. Varma, S., Simon, R.: Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics **7**(91) (2006)
37. Vilas Boas, M.G., Gambini Santos, H., de Campos Merschmann, L.H., Vanden Berghe, G.: Optimal decision trees for the algorithm selection problem: Integer programming based approaches. CoRR **abs/1907.02211** (2019)