Warmstarting of Model-based Algorithm Configuration

Marius Lindauer and **Frank Hutter**
University of Freiburg
{lindauer,fh}@cs.uni-freiburg.de

Abstract

The performance of many hard combinatorial problem solvers depends strongly on their parameter settings, and since manual parameter tuning is both tedious and suboptimal the AI community has recently developed several algorithm configuration (AC) methods to automatically address this problem. While all existing AC methods start the configuration process of an algorithm A from scratch for each new type of benchmark instances, here we propose to exploit information about A's performance on previous benchmarks in order to warmstart its configuration on new types of benchmarks. We introduce two complementary ways in which we can exploit this information to warmstart AC methods based on a predictive model. Experiments for optimizing a flexible modern SAT solver on twelve different instance sets show that our methods often yield substantial speedups over existing AC methods (up to 165-fold) and can also find substantially better configurations given the same compute budget.

Introduction

Many algorithms in the field of artificial intelligence rely crucially on good parameter settings to yield strong performance; prominent examples include solvers for many hard combinatorial problems (e.g., the propositional satisfiability problem SAT (Hutter et al. 2017) or AI planning (Fawcett et al. 2011)) as well as a wide range of machine learning algorithms (in particular deep neural networks (Snoek, Larochelle, and Adams 2012) and automated machine learning frameworks (Feurer et al. 2015)). To overcome the tedious and error-prone task of manual parameter tuning for a given algorithm A, algorithm configuration (AC) procedures automatically determine a parameter configuration of A with low cost (e.g., runtime) on a given benchmark set. General algorithm configuration procedures fall into two categories: model-free approaches, such as *ParamILS* (Hutter et al. 2009), *irace* (López-Ibáñez et al. 2016) or *GGA* (Ansótegui, Sellmann, and Tierney 2009), and model-based approaches, such as *SMAC* (Hutter, Hoos, and Leyton-Brown 2011) or *GGA++* (Ansótegui et al. 2015).

Even though model-based approaches learn to predict the cost of different configurations on the benchmark instances at hand, so far all AC procedures start their configuration process from scratch when presented with a new set of benchmark instances. Compared with the way humans exploit information from past benchmark sets, this is obviously suboptimal. Inspired by the human ability to learn across different tasks, we propose to use performance measurements for an algorithm on previous benchmark sets in order to warmstart its configuration on a new benchmark set. As we will show in the experiments, our new warmstarting methods can substantially speed up AC procedures, by up to a factor of 165. In our experiments, this amounts to spending less than 20 minutes to obtain comparable performance as could previously be obtained within two days.

Preliminaries

Algorithm configuration (AC). Formally, given a target algorithm with configuration space $\mathbf{\Theta}$, a probability distribution $\mathcal{D}$ across problem instances, as well as a cost metric c to be minimized, the algorithm configuration (AC) problem is to determine a parameter configuration $\theta^* \in \mathbf{\Theta}$ with low expected cost on instances drawn from $\mathcal{D}$:

$$\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \mathbb{E}_{\pi \sim \mathcal{D}}[c(\theta, \pi)]. \quad (1)$$

In practice, $\pi \sim \mathcal{D}$ is typically approximated by a finite set of instances Π drawn from $\mathcal{D}$. An example AC problem is to set a SAT solver's parameters to minimize its average runtime on a given benchmark set of formal verification instances. Througout the paper, we refer to algorithms for solving the AC problem as *AC procedures*. They execute the target algorithm with different parameter configurations $\theta \in \mathbf{\Theta}$ on different instances $\pi \in \Pi$ and measure the resulting costs $c(\theta, \pi)$.

Empirical performance models (EPMs). A core ingredient in model-based approaches for AC is a probabilistic regression model $\hat{c} : \mathbf{\Theta} \times \Pi \to \mathbb{R}$ that is trained based on the cost values $\langle \mathbf{x} = [\theta, \pi], y = c(\theta, \pi) \rangle$ observed thus far and can be used to predict the cost of new parameter configurations $\theta \in \mathbf{\Theta}$ on new problem instances $\pi \in \Pi$. Since this regression model predicts empirical algorithm performance (i.e., its cost), it is known as an empirical performance model (EPM; Leyton-Brown, Nudelman, and Shoham; Hutter et al. 2009; 2014b). Random forests have been established as

Algorithm 1: Model-based Algorithm Configuration

Input : Configuration Space $\mathbf{\Theta}$, Instances Π, Configuration Budget B

1 $\theta_{\text{inc}}, \mathcal{H} \leftarrow$ initial_design($\mathbf{\Theta}$, Π);
2 **while** B *not exhausted* **do**
3 $\quad \hat{c} \leftarrow$ fit EPM based on $\mathcal{H}$;
4 $\quad \Theta_{\text{chall}} \leftarrow$ select challengers based on $\hat{c}$ and $\mathcal{H}$;
5 $\quad \theta_{\text{inc}}, \mathcal{H} \leftarrow$ race($\Theta_{\text{chall}} \cup \{\theta_{\text{inc}}\}, \Pi, \mathcal{H}$);
6 **return** θ_{inc}

the best-performing type of EPM and are thus used in all current model-based AC approaches.

For the purposes of this regression model, the instances π are characterized by instance features. These features reach from simple ones (such as the number of clauses and variables of a SAT formula) to more complex ones (such as statistics gathered by briefly running a probing algorithm). Nowadays, informative instance features are available for most hard combinatorial problems (e.g., SAT (Nudelman et al. 2004), mixed integer programming (Hutter et al. 2014b), AI planning (Fawcett et al. 2014), and answer set programming (Hoos, Lindauer, and Schaub 2014)).

Model-based algorithm configuration. The core idea of sequential model-based algorithm configuration is to iteratively fit an EPM based on the cost data observed so far and use it to guide the search for well-performing parameter configurations. Algorithm 1 outlines the model-based algorithm configuration framework, similarly as introduced by Hutter, Hoos, and Leyton-Brown (2011) for the AC procedure *SMAC*, but also encompassing the *GGA++* approach by Ansótegui et al. (2015). We now discuss this algorithm framework in detail since our warmstarting extensions will adapt its various elements.

First, in Line 1 a model-based AC procedure runs the algorithm to be optimized with configurations in a so-called *initial design*, keeping track of their costs and of the best configuration θ_{inc} seen so far (the so-called *incumbent*). It also keeps track of a *runhistory* $\mathcal{H}$, which contains tuples $\langle \theta, \pi, c(\theta, \pi) \rangle$ of the cost $c(\theta, \pi)$ obtained when evaluating configuration θ on instance π. To obtain good anytime performance, by default *SMAC* only executes a single run of a user-defined default configuration θ_{def} on a randomly-chosen instance as its initial design and uses θ_{def} as its initial incumbent θ_{inc}. *GGA++* samples a set of configurations as initial generation and races them against each other on a subset of the instances.

In Lines 2-5, the AC procedure performs the model-based search. While a user-specified configuration budget B (e.g., number of algorithm runs or wall-clock time) is not exhausted, it fits a random-forest-based EPM on the existing cost data in $\mathcal{H}$ (Line 3), aggregates the EPM's predictions over the instances Π in order to obtain marginal cost predictions $\hat{c}(\theta)$ for each configuration $\theta \in \mathbf{\Theta}$ and then uses these predictions in order to select a set of promising configurations Θ_{chall} to challenge the incumbent θ_{inc} (Line 4) (*SMAC*) or to generate well-performing offsprings (*GGA++*). For this step, a so-called *acquisition function* trades off exploitation of promising areas of the configuration space versus exploration of areas for which the model is still uncertain; common choices are expected improvement (Jones, Schonlau, and Welch 1998), upper confidence bounds (Srinivas et al. 2010) or entropy search (Hennig and Schuler 2012).

To determine a new incumbent configuration θ_{inc}, in Line 5 the AC procedure races these challengers and the current incumbent by evaluating them on individual instances $\pi \in \Pi$ and adding the observed data to $\mathcal{H}$. Since these evaluations can be computationally costly the race only evaluates as many instances as needed per configuration and terminates slow runs early (Hutter et al. 2009).

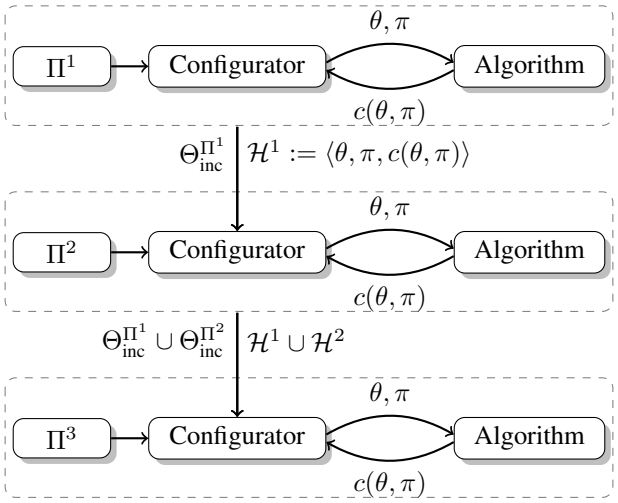


Figure 1: Control flow of warmstarting information

Warmstarting Approaches for AC

In this section, we discuss how the efficiency of model-based AC procedures (as described in the previous section) can be improved by warmstarting the search from data generated in previous AC runs. We assume that the algorithm to be optimized and its configuration space $\mathbf{\Theta}$ is the same in all runs, but the set of instances Π can change between the runs. To warmstart a new AC run, we consider the following data from previous AC runs on previous instance sets Π^i:

- Sets of optimized configurations $\Theta_{\text{inc}}^{\Pi^i}$ found in previous AC runs on Π^i—potentially, multiple runs were performed on the same instance set to return the result with best training performance such that $\Theta_{\text{inc}}^{\Pi^i}$ contains the final incumbents from each of these runs;
- We denote the union of previous instances as $\Pi' := \bigcup_{i \in \mathcal{I}} \Pi^i$ for set superscripts $i \in \mathcal{I}$.
- Runhistory data $\mathcal{H}' := \bigcup_{i \in \mathcal{I}} \mathcal{H}^i$ of all AC runs on previous instance sets Π^i. [1]

[1] If the set of instances Π and the runhistory $\mathcal{H}$ are not indexed, we always refer to the ones of the current AC run.

To design warmstarting approaches, we consider the following desired properties:

1. When the performance data gathered on previous instance sets is informative about performance on the current instance set, it should speed up our method.
2. When said performance data is misleading, our method should stop using it and should not be much slower than without it.
3. The runtime overhead generated by using the prior data should be fairly small.

In the following subsections, we describe different warmstarting approaches that satisfy these properties.

Warmstarting Initial Design (INIT)

The first approach we consider for warmstarting our model-based AC procedure is to adapt its initial design (Line 1 of Algorithm 1) to start from configurations that performed well in the past. Specifically, we include the incumbent configurations $\Theta_{\text{inc}}^{\Pi^i}$ from all previous AC runs as well as the user-specified default θ_{def}.

Evaluating all previous incumbents $\Theta_{\text{inc}}^{\Pi^i}$ in the initial design can be inefficient (contradicting Property 3), particularly if they are very similar. This can happen when the previous instance sets are quite similar, or when multiple runs were performed on a single instance set.

To obtain a complementary set of configurations that covers all previously optimized instances well but is not redundant, we propose to use a two step approach. First, we determine the best configuration for each previous Π^i.

$$\Theta_{\text{inc}} := \bigcup_{i \in \mathcal{I}} \operatorname*{arg\,min}_{\theta \in \Theta_{\text{inc}}^{\Pi^i}} \sum_{\pi \in \Pi^i} c(\theta, \pi) \tag{2}$$

Secondly, we use an iterative, greedy forward search to select a complementary set of configurations across all previous instance sets—inspired by the per-instance selection procedure *Hydra* (Xu, Hoos, and Leyton-Brown 2010). Specifically, for the second step we define the *mincost* $\tilde{c}(\Theta_j)$ of a set of configurations Θ_j on the union of all previous instances Π' as

$$\tilde{c}(\Theta_j) := \frac{1}{|\Pi'|} \sum_{\pi \in \Pi'} \min_{\theta \in \Theta_j} c(\theta, \pi), \tag{3}$$

start with $\Theta_1 := \{\theta_{\text{def}}\}$, and at each iteration, add the configuration $\theta' \in \Theta_{\text{inc}}$ to Θ_j that minimizes $\tilde{c}(\Theta_j \cup \{\theta'\})$. Because $\tilde{c}(\cdot)$ is a supermodular set function this greedy algorithm is guaranteed to select a set of configurations whose mincost is within a factor of $(1 - 1/e) \approx 0.63$ of optimal among sets of the same size (Krause and Golovin 2012).

Since we do not necessarily know the empirical cost of all $\theta' \in \Theta_{\text{inc}}$ on all $\pi \in \Pi'$, we use an EPM $\hat{c} : \boldsymbol{\Theta} \times \Pi \to \mathbb{R}$ as a plug-in estimator to predict these costs. We train this EPM on all previous runhistory data $\mathcal{H}'$. In order to enable this, the benchmark sets for all previous AC runs have to be characterized with the same set of instance features.

In *SMAC*, we use this set of complementary configurations in the initial design using the same racing function as in comparing challengers to the incumbent (Line 5) to obtain the initial incumbent; to avoid rejecting challengers too quickly, a challenger is compared on at least 3 instances before it can be rejected. In *GGA++*, these configurations can be included in the first generation of configurations.

Data-Driven Model-Warmstarting (DMW)

Since model-based AC procedures are guided by their EPM, we considered to warmstart this EPM by including all cost data $\mathcal{H}'$ gathered in previous AC runs as part of its training data. In the beginning, the predictions of this EPM would mostly rely on $\mathcal{H}'$, and as more data is acquired on the current benchmark this would increasingly affect the model.

However, this approach has two disadvantages:

1. When a lot of warmstarting data is available it requires many evaluations on the current instance set to affect model predictions. If the previous data is misleading, this would violate our desired Property 2.
2. Fitting the EPM on $\mathcal{H} \cup \mathcal{H}'$ will be expensive even in early iterations, because $\mathcal{H}'$ will typically contain many observations. Even by using *SMAC*'s mechanism to invest at least the same amount of time in Lines 3 and 4 as in Line 5, in preliminary experiments this slowed down *SMAC* substantially (violating Property 3).

For these two reasons, we do not use this approach for warmstarting but propose an alternative. Specifically, to avoid the computational overhead of refitting a very large EPM in each iteration, and to allow our model to discard misleading previous data, we propose to fit individual EPMs $\hat{c}_i$ for each $\mathcal{H}^i$ once and to combine their predictions with those of an EPM $\hat{c}$ fitted on the newly gathered cost data $\mathcal{H}$. This relates to stacking in ensemble learning (Wolpert 1992); however in our case, each constituent EPM is trained on a different dataset. Hence, in principle we could even use different instance features for each instance set.

To aggregate predictions of the individual EPMs, we propose to use a linear combination:

$$\hat{c}_{\text{DMW}}(\theta, \pi) := w_0 + w_{\hat{c}} \cdot \hat{c}(\theta, \pi) + \sum_{i \in \mathcal{I}} w_i \cdot \hat{c}_i(\theta, \pi) \tag{4}$$

where w are weights fitted with stochastic gradient descent (SGD) to minimize the combined model's root mean squared error (RMSE). To avoid overfitting of the weights, we randomly split the current $\mathcal{H}$ into a training and validation set $(2:1)$, use the training set to fit $\hat{c}$, and then compute predictions of $\hat{c}$ and each $\hat{c}_i$ on the validation set, which are used to fit the weights $\mathbf{w}$. Finally, we re-fit the EPM $\hat{c}$ on all data in $\mathcal{H}$ to obtain a maximally informed model.

In the beginning of a new AC run, with few data in $\mathcal{H}$, $\hat{c}$ will not be very accurate, causing its weight $w_{\hat{c}}$ to be low, such that the previous models $\hat{c}_i$ will substantially influence the cost predictions. As more data is gathered in $\mathcal{H}$, the predictive accuracy of $\hat{c}$ will improve and the predictions of the previous models $\hat{c}_i$ will become less important.

Besides weighting based on the accuracy of the individual models, the weights have the second purpose of scaling the individual model's predictions appropriately: these

scales reflect the different hardnesses of the instance sets they were trained on and by setting the weights to minimize RMSE of the combined model on the current instances Π, they will automatically normalize for scale.

The performance predictions of DMW can be used in any model-based AC procedure, such as *SMAC* and *GGA++*.

Combining INIT and DMW (IDMW)

Importantly, the two methods we propose are complementary. A warmstarted initial design (INIT) can be easily combined with data-driven model-warmstarting (DMW) because both approaches affect different parts of model-based algorithm configuration: where to start from and how to integrate the full performance data from the current and the previous benchmarks to decide where to sample next. In fact, the two warmstarting methods can even synergize to yield more than the sum of their pieces: by evaluating strong configurations from previous AC runs in the initial design through INIT, the weights of the stacked model in DMW can be fitted on these important observations early on, improving the accuracy of its predictions even in early iterations.

Experiments

We evaluated how our three warmstarting approaches improve the state-of-the-art AC procedure *SMAC*.[2] In particular, we were interested in the following research questions:

Q1 Can warmstarted *SMAC* find better performing configurations within the same configuration budget?

Q2 Can warmstarted *SMAC* find well-performing configurations faster than default *SMAC*?

Q3 What is the effect of using warmstarting data $\mathcal{H}^i$ from related and unrelated benchmarks?

Experimental Setup To answer these questions, we ran *SMAC* (0.5.0) and our warmstarting variants[3] on twelve well-studied AC tasks from the configurable SAT solver challenge (Hutter et al. 2017), which are publicly available in the algorithm configuration library (Hutter et al. 2014a). Since our warmstarting approaches have to generalize across different instance sets and not across algorithms, we considered AC tasks of the highly flexible and robust SAT solver *SparrowToRiss* across 12 instance sets. *SparrowToRiss* is a combination of two well-performing solvers: *Riss* (Manthey 2014) is a tree-based solver that performs well on industrial and hand-crafted instances; *Sparrow* (Balint et al. 2011) is a local-search solver that performs well on random, satisfiable instances. *SparrowToRiss* first runs *Sparrow* for a parametrized amount of time and then runs *Riss* if *Sparrow* could not find a satisfying assignment. Thus, *SparrowToRiss* can be applied to a large variety of different SAT instances. *Riss*, *Sparrow* and *SparrowToRiss* also won several medals in the international SAT competition. Furthermore, configuring *SparrowToRiss* is a challenging task because it has a very large configuration space with 222 parameters and 176 conditional dependencies.

To study warmstarting on different categories of instances, the AC tasks consider SAT instances from applications with a lot of internal structure, hand-crafted instances with some internal structure, and randomly-generated SAT instances with little structure. We ran *SparrowToRiss* on

- application instances from bounded-model checking (*BMC*), hardware verification (*IBM*) and fuzz testing based on circuits (*CF*);
- hand-crafted instances from graph-isomorphism (*GI*), low autocorrelation binary sequence (*LABS*) and n-rooks instances (*N-Rooks*);
- randomly generated instances, specifically, 3-SAT instances at the phase transition from the ToughSAT instance generator (*3cnf*), a mix of satisfiable and unsatisfiable 3-SAT instances at the phase transition (*K3*), and unsatisfiable 5-SAT instances from a generator used in the SAT Challenge 2012 and SAT Competition 2013 (*UNSAT-k5*); and on
- randomly generated satisfiable instances, specifically, instances with 3 literals per clause and 1000 clauses (*3SAT1k*), instances with 5 literals per clause and 500 clauses (*5SAT500*) and instances with 7 literals per clause and 90 clauses (*7SAT90*).

Further details on these instances are given in the description of the configurable SAT solver challenge (Hutter et al. 2017). The instances were split into a training set for configuration and a test set to validate the performance of the configured *SparrowToRiss* on unseen instances.

For each configuration run on a benchmark set in one of the categories, our warmstarting methods had access to observations on the other two benchmark sets in the category. For example, warmstarted *SMAC* optimizing *SparrowToRiss* on *IBM* had access to the observations and final incumbents of *SparrowToRiss* on *CF* and *BMC*.

As a cost metric, we chose the commonly-used penalized average runtime metric (PAR10, i.e., counting each timeout as 10 times the runtime cutoff) with a cutoff of 300 CPU seconds. To avoid a constant inflation of the PAR10 values, we removed all test instances post hoc that were never solved by any configuration in our experiments (11 *CF* instances, 69 *IBM* instances, 17 *BMC* instances, 21 *GI* instances, 72 *LABS* instances and 73 *3cnf* instances).

On each AC task, we ran 10 independent *SMAC* runs with a configuration budget of 2 days each. All runs were run on a compute cluster with nodes equipped with two Intel Xeon E5-2630v4 and 128GB memory running CentOS 7.

Baselines

As baselines, we ran (I) the user-specified default configuration θ_{def} to show the effect of algorithm configuration, (II) *SMAC* without warmstarting, and (III) a state-of-the-art warmstarting approach for hyperparameter optimizers proposed by Wistuba, Schilling, and Schmidt-Thieme (2016), which we abbreviate as "adapted acquisition function" (AAF). The goal of AAF is to bias the acquisition

[2] The source code of *GGA++* is not publicly available and thus, we could not run experiments on *GGA++*.

[3] Code and data is publicly available at: `http://www.ml4aad.org/smac/`.

	PAR10 scores						Speedup over default SMAC			
	θ_{def}	SMAC	AAF	INIT	DMW	IDMW	AAF	INIT	DMW	IDMW
CF	326.5	**125.8**	140.0	**126.6**	**122.0**	**116.4**	0.1	0.5	0.7	**2.7**
IBM	150.6	**50.6**	49.0	**47.8**	**47.5**	48.8	3.9	**16.2**	1.4	9
BMC	421.5	209.6	230.7	203.1	**155.9**	**137.4**	1.2	1	11	**29.3**
GI	314.1	**165.0**	**165.6**	**165.6**	**165.4**	**152.7**	**25.6**	0.6	7.1	19.4
LABS	330.1	**232.9**	291.2	271.2	285.9	286.7	0.8	0.8	0.8	0.8
N-Rooks	116.7	**8.6**	18.1	27.3	27.5	**12.7**	0.4	0.4	0.4	0.5
3cnf	890.5	890.5	**822.8**	**877.5**	890.3	**812.8**	**10.7**	1	1	8.4
K3	152.8	**30.1**	53.9	**42.9**	**39.9**	**29.7**	0.9	0.9	1.8	**1.8**
UNSAT-k5	151.9	**1.1**	1.2	**1.1**	1.3	1.2	1	1	1	1
3SAT1k	104.4	**76.6**	**75.2**	**75.2**	**82.6**	**69.8**	3.1	2.1	2.1	**3.8**
5SAT500	3000	20.5	14.6	**14.7**	**8.3**	**9.6**	**6**	0.7	0.7	0.8
7SAT90	52.3	38.0	31.7	**20.2**	**19.7**	**31.2**	53.5	2.3	0.5	**165.3**
							∅ 2.4	1.1	1.3	4.3

Table 1: **Left:** PAR10 score (sec) of θ_{def}, i.e., the default configuation of *SparrowToRiss*, and the final *SparrowToRiss* configurations returned by the different *SMAC* variants; median across 10 *SMAC* runs. The "SMAC" column shows the performance of default *SMAC* without warmstarting. Best PAR10 is underlined and we highlighted runs in bold face for which there is no statistical evidence according to a (one-sided) Mann-Whitney U test ($\alpha = 0.05$) that they performed worse than the best configurator. **Right:** Speedup of warmstarted *SMAC* compared to default *SMAC*. This is computed by comparing the time points of *SMAC* with and without warmstarting after which they do not perform significantly worse (according to a permutation test) than *SMAC* with the full budget. Speedups > 1 indicate that warmstarted *SMAC* reached the final performance of default *SMAC* faster, speedups < 1 indicate that default *SMAC* was faster. We marked the best speedup (> 1) in bold-face. The last row shows the geometric average across all speedups.

function (Line 4 in Algorithm 1) towards previously well-performing regions in the configuration space.[4] To generalize AAF to algorithm configuration, we use marginalized prediction across all instances $\hat{c}(\theta) := \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \hat{c}(\theta, \pi)$.

Q1: Same configuration Budget

The left part of Table 1 shows the median PAR10 test scores of the finally-returned configurations θ_{inc} across the 10 *SMAC* runs. Default *SMAC* nearly always improved the PAR10 scores of *SparrowToRiss* substantially compared to the *SparrowToRiss* default, yielding up to a 138-fold speedup (on *UNSAT-k5*). Warmstarted *SMAC* performed significantly better yet on 4 of the AC tasks (*BMC*, *3cnf*, *5SAT500* and *7SAT90*), with additional speedups up to 2.1-fold (on *5SAT500*). On two of the crafted instance sets (*LABS* and *N-Rooks*), the warmstarting approaches performed worse than default *SMAC*—details discussed later.

Overall, the best results were achieved by the combination of our approaches, IDMW. This yielded the best performance of all approaches in 6 of the 12 scenarios (with sometimes substantial improvements over default SMAC) and statistically insignificantly different results than the best approach in 3 of the scenarios. Notably, IDMW performed better on average than its individual components INIT and DMW and clearly outperformed AAF.

[4]We note that combining AAF and INIT is not effective because evaluating the incumbents of INIT would nullify the acquisition function bias of AAF.

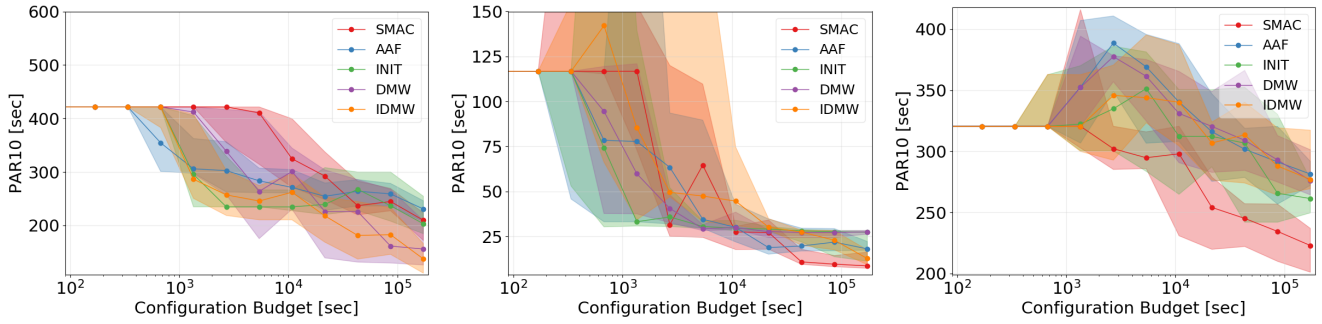
Q2: Speedup

The right part of Table 1 shows how much faster our warmstarted *SMAC* reached the PAR10 performance default that *SMAC* reached with the full configuration budget.[5] The warmstarting methods outperformed default SMAC in almost all cases (again except *LABS* and *N-Rooks*), with up to 165-fold speedups. The most consistent speedups were achieved by the combination of our warmstarting approaches, IDMW, with a geometric-average 4.3-fold speedup. We note that our baseline AAF also yielded good speedups (geometric average of 2.4), but its final performance was often quite poor (see left part of Table 1).

Figure 2 illustrates the anytime test performance of all *SMAC* variants.[6] In Figure 2a, AAF, INIT and IDMW improved the performance of *SparrowToRiss* very early (after roughly 700-1000 seconds), but only the DMW variants per-

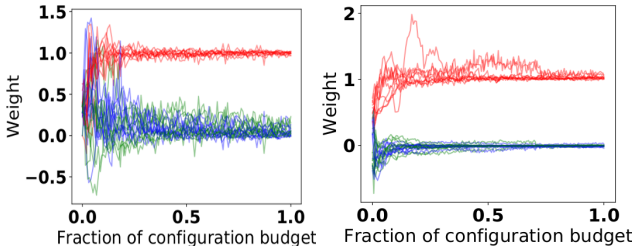
[5]A priori it is not clear how to define a speedup metric comparing algorithm configurators across several runs. To take noise into account across our 10 runs, we performed a permutation test (with $\alpha = 0.05$ with 10 000 permutations) to determine the first time point from which onwards there was no statistical evidence that default *SMAC* with a full budget would perform better. To take early convergence/stagnation of default *SMAC* into account, we compute the speedup of default *SMAC* to itself and divide the speedups by default *SMAC*'s speedup.

[6]Since Figure 2 shows test performance on unseen test instances, performance is not guaranteed to improve monotonically (a new best configuration on the training instances might not generalize well to the test instances).



(a) BMC (b) N-Rooks (c) LABS

Figure 2: Median PAR10 of *SparrowToRiss* over configuration time with 25% and 75% percentiles as uncertainties.



(a) red: IBM, blue: CF, green: BMC

(b) red: UNSAT-k5, blue: K3, green: 3cnf

Figure 3: Weights over time of all 10 runs *SMAC*+DMW. The red curve is the weight on EPM $\hat{c}$ on the current instances; the blue and green curves corresponds to weights on EPMs based on previously optimized instances.

formed well in the long run.

To study the effect of our worst results, Figure 2b and 2c show the anytime performance on *N-Rooks* and *LABS*, respectively. Figure 2b shows that warmstarted *SMAC* performed better in the beginning, but that default *SMAC* performed slightly better in the end. The better initial performance is not captured in our quantitative analysis in Table 1. In contrast, Figure 2c shows that for *LABS*, warmstarted *SMAC* was initially mislead and then started improving like default SMAC, but with a time lag; we note that we only observed this pattern on *LABS* and conclude that configurations found on *N-Rooks* and *GI* do not generalize to *LABS*.

Q3: Warmstarting Influence

To study how our warmstarting methods learn from previous data, in Figure 3 we show how the weights of the DMW approach changed over time. Figure 3a shows a representative plot: the weights were similar in the beginning (i.e., all EPMs contributed similarly to cost predictions) and over time, the weights of the previous models decreased, with the weight of the current EPM dominating. When optimizing on *IBM*, the EPM trained on observations from *CF* was the most important EPM in the beginning.

In contrast, Figure 3b shows a case in which the previous performance data acquired for benchmarks *K3* and *3cnf* do not help for cost predictions on *UNSAT-k5*. (This was to be expected, because *3cnf* comprises only satisfiable instances, *K3* a mix of satisfiable and unsatisfiable instances, and *UNSAT-k5* only unsatisfiable instances.) As the figure shows, our DMW approach briefly used the data from the mixed *K3* benchmark (blue curves), but quickly focused only on data from the current benchmark. These two examples illustrate that our DMW approach indeed successfully used data from related benchmarks and quickly ignored data from unrelated ones.

Related Work

The most related work comes from the field of hyperparameter optimization (HPO) of machine learning algorithms. HPO, when cast as the optimization of (cross-)validation error, is a special case of AC. This special case does not require the concept of problem instances, does not require the modelling of runtimes of randomized algorithms, does not need to adaptively terminate slow algorithm runs and handle the resulting censored algorithm runtimes, and typically deals with fairly low-dimensional and all-continuous (hyper-)parameter configuration spaces. These works therefore do not directly transfer to the general AC problem.

Several warmstarting approaches exist for HPO. A prominent approach is to learn surrogate models across datasets (Swersky, Snoek, and Adams 2013; Bardenet et al. 2014; Yogatama and Mann 2014). All of these works are based on Gaussian process models whose computational complexity scales cubically in the number of data points, and therefore, all of them were limited to hundreds or at most thousands of data points. We generalize them to the AC setting (which, on top of the differences to HPO stated above, also needs to handle up to a million cost measurements for an algorithm) in our DMW approach.

Another approach for warmstarting HPO is by adapting the initial design. Feurer, Springenberg, and Hutter (2015) proposed to initialize HPO in the automatic machine learning framework *Auto-Sklearn* with well-performing configurations from previous datasets. They had optimized configurations from 57 different machine learning data sets available as warmstarting data and chose which of these to use for a new dataset based on its characteristics; specifically, they

used the optimized configurations from the k most similar datasets. This approach could be adapted to AC warmstarting in cases where we have many AC benchmarks. However, one disadvantage of the approach is that – unlike our INIT approach – it does not aim for complementarity in the selected configurations. Wistuba, Schilling, and Schmidt-Thieme (2015) proposed another approach for warmstarting the initial design which does not depend on instance features and is not limited to configurations returned in previous optimization experiments. They combined surrogate predictions from previous runs and used gradient descent to determine promising configurations. This approach is limited to continuous (hyper-)parameters and thus does not apply to the general AC setting.

One related variant of algorithm configuration is the problem of configuring on a stream of problem instances that changes over time. The *ReACT* approach (Fitzgerald et al. 2014) targets this problem setting, keeping track of configurations that worked well on previous instances. If the characteristics of the instances change over time, it also adapts the current configuration by combining observations on previous instances and on new instances. In contrast to our setting, *ReACT* does not return a single configuration for an instance set and requires parallel compute resources to run a parallel portfolio all the time.

Discussion & Conclusion

In this paper, we introduced several methods to warmstart model-based algorithm configuration (AC) using observations from previous AC experiments on different benchmark instance sets. As we showed in our experiments, warmstarting can speed up the configuration process up to 165-fold and can also improve the configurations finally returned.

While we focused on the state-of-the-art configurator *SMAC* in our experiments, our methods are also applicable to other model-based configurators, such as *GGA++*, and our warmstarted initial design approach is even applicable to model-free configurators, such as *ParamILS* and *irace*. We expect that our results would similarly generalize to these.

A practical limitation of our DMW approach (and thus also of IDMW) is that the memory consumption grows substantially with each additional EPM (at least when using random forests fitted on hundreds of thousands of observations). We also tried to study warmstarting *SMAC* for optimizing *SparrowToRiss* on all instance sets except the one at hand, but unfortunately, the memory consumption exceeded 12GB RAM. Therefore, one possible approach would be to reduce memory consumption and to use instance features to select a subset of EPMs constructed on similar instances.

Another direction for future work is to combine warmstarting with parameter importance analysis (Hutter, Hoos, and Leyton-Brown 2014; Biedenkapp et al. 2017), e.g., for determining important parameters on previous instance sets and focusing the search on these parameters for a new instance set. Finally, a promising future direction is to integrate warmstarting into iterative configuration procedures, such as *Hydra* (Xu, Hoos, and Leyton-Brown 2010), *ParHydra* (Lindauer et al. 2017), or *Cedalion* (Seipp et al. 2015), which construct portfolios of complementary configurations in an iterative fashion using multiple AC runs.

Acknowledgements

The authors acknowledge funding by the DFG (German Research Foundation) under Emmy Noether grant HU 1900/2-1 and support by the state of Baden-Württemberg through bwHPC and the DFG through grant no INST 39/963-1 FUGG.

References

[Ansótegui et al. 2015] Ansótegui, C.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2015. Model-based genetic algorithms for algorithm configuration. In Yang, Q., and Wooldridge, M., eds., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 733–739.

[Ansótegui, Sellmann, and Tierney 2009] Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I., ed., *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09)*, volume 5732 of *Lecture Notes in Computer Science*, 142–157. Springer-Verlag.

[Balint et al. 2011] Balint, A.; Frohlich, A.; Tompkins, D.; and Hoos, H. 2011. Sparrow2011. In *Proceedings of SAT Competition 2011*.

[Bardenet et al. 2014] Bardenet, R.; Brendel, M.; Kégl, B.; and Sebag, M. 2014. Collaborative hyperparameter tuning. In Dasgupta, S., and McAllester, D., eds., *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, 199–207. Omnipress.

[Biedenkapp et al. 2017] Biedenkapp, A.; Lindauer, M.; Eggensperger, K.; Fawcett, C.; Hoos, H.; and Hutter, F. 2017. Efficient parameter importance analysis via ablation with surrogates. In *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI'17)*, 773–779.

[Bonet and Koenig 2015] Bonet, B., and Koenig, S., eds. 2015. *Proceedings of the Twenty-nineth Conference on Artificial Intelligence (AAAI'15)*. AAAI Press.

[Fawcett et al. 2011] Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Roger, G.; and Seipp, J. 2011. Fd-autotune: Domain-specific configuration using fast-downward. In Helmert, M., and Edelkamp, S., eds., *Working notes of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-11), Workshop on Planning and Learning.*

[Fawcett et al. 2014] Fawcett, C.; Vallati, M.; Hutter, F.; Hoffmann, J.; Hoos, H.; and Leyton-Brown, K. 2014. Improved features for runtime prediction of domain-independent planners. In Chien, S.; Minh, D.; Fern, A.; and Ruml, W., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*. AAAI.

[Feurer et al. 2015] Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J. T.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NIPS'15).*

[Feurer, Springenberg, and Hutter 2015] Feurer, M.; Springenberg, T.; and Hutter, F. 2015. Initializing Bayesian hyperparameter optimization via meta-learning. In Bonet and Koenig (2015), 1128–1135.

[2014] Fitzgerald, T.; O'Sullivan, B.; Malitsky, Y.; and Tierney, K. 2014. React: Real-time algorithm configuration through tournaments. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SOCS'14)*. AAAI Press.

[2012] Hennig, P., and Schuler, C. 2012. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research* 98888(1):1809–1837.

[2014] Hoos, H.; Lindauer, M.; and Schaub, T. 2014. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming* 14:569–585.

[2009] Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

[2014a] Hutter, F.; López-Ibánez, M.; Fawcett, C.; Lindauer, M.; Hoos, H.; Leyton-Brown, K.; and Stützle, T. 2014a. Aclib: a benchmark library for algorithm configuration. In Pardalos, P., and Resende, M., eds., *Proceedings of the Eighth International Conference on Learning and Intelligent Optimization (LION'14)*, Lecture Notes in Computer Science, 36–40. Springer-Verlag.

[2014b] Hutter, F.; Xu, L.; Hoos, H.; and Leyton-Brown, K. 2014b. Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence* 206:79–111.

[2017] Hutter, F.; Lindauer, M.; Balint, A.; Bayless, S.; Hoos, H.; and Leyton-Brown, K. 2017. The configurable SAT solver challenge (CSSC). *Artificial Intelligence Journal (AIJ)* 243:1–25.

[2011] Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In Coello, C., ed., *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, 507–523. Springer-Verlag.

[2014] Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2014. An efficient approach for assessing hyperparameter importance. In Xing, E., and Jebara, T., eds., *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, 754–762. Omnipress.

[1998] Jones, D.; Schonlau, M.; and Welch, W. 1998. Efficient global optimization of expensive black box functions. *Journal of Global Optimization* 13:455–492.

[2012] Krause, A., and Golovin, D. 2012. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems* 3(19):8.

[2009] Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2009. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM* 56(4).

[2017] Lindauer, M.; Hoos, H.; Leyton-Brown, K.; and Schaub, T. 2017. Automatic construction of parallel portfolios via algorithm configuration. *Artificial Intelligence* 244:272–290.

[2016] López-Ibáñez, M.; Dubois-Lacoste, J.; Caceres, L. P.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3:43–58.

[2014] Manthey, N. 2014. Riss 4.27. In Belov, A.; Diepold, D.; Heule, M.; and Järvisalo, M., eds., *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, 65–67. University of Helsinki.

[2004] Nudelman, E.; Leyton-Brown, K.; Devkar, A.; Shoham, Y.; and Hoos, H. 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In Wallace, M., ed., *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, volume 3258 of *Lecture Notes in Computer Science*, 438–452. Springer-Verlag.

[2015] Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In Bonet and Koenig (2015), 3364–3370.

[2012] Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian optimization of machine learning algorithms. In Bartlett, P.; Pereira, F.; Burges, C.; Bottou, L.; and Weinberger, K., eds., *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*, 2960–2968.

[2010] Srinivas, N.; Krause, A.; Kakade, S.; and Seeger, M. 2010. Gaussian process optimization in the bandit setting: No regret and experimental design. In Fürnkranz, J., and Joachims, T., eds., *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, 1015–1022. Omnipress.

[2013] Swersky, K.; Snoek, J.; and Adams, R. 2013. Multi-task Bayesian optimization. In Burges, C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K., eds., *Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NIPS'13)*, 2004–2012.

[2015] Wistuba, M.; Schilling, N.; and Schmidt-Thieme, L. 2015. Learning hyperparameter optimization initializations. In *Proceedings of the International Conference on Data Science and Advanced Analytics (DSAA)*, 1–10. IEEE.

[2016] Wistuba, M.; Schilling, N.; and Schmidt-Thieme, L. 2016. Hyperparameter optimization machines. In *Proceedings of the International Conference on Data Science and Advanced Analytics (DSAA)*, 41–50. IEEE.

[1992] Wolpert, D. 1992. Stacked generalization. *Neural Networks* 5(2):241–259.

[2010] Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M., and Poole, D., eds., *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, 210–216. AAAI Press.

[2014] Yogatama, D., and Mann, G. 2014. Efficient transfer learning method for automatic hyperparameter tuning. In Kaski, S., and Corander, J., eds., *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33 of *JMLR Workshop and Conference Proceedings*, 1077–1085.