Algorithm Configuration in the Unified Planning Framework

Dimitri Weiß[1], Andrea Micheli[2], and Kevin Tierney[1]

[1] Decision and Operation Technologies, Bielefeld University, Universitaetsstrasse 25, 33615 Bielefeld, Germany
{dimitri.weiss,kevin.tierney}@uni-bielefeld.de
[2] Digital Industry Center, Fondazione Bruno Kessler, Via Sommarive 18, 38123 Trento, Italy
amicheli@fbk.eu

Abstract. The Unified Planning Framework (UPF) provides convenient access to automated planning technology. It allows for problem formulation independent of a planning engine and the utilization of planners available on the system. However, choosing a suitable parameter configuration of the planning engine for a given problem constitutes a significant challenge. Manually finding a high-quality configuration requires domain knowledge and a considerable time investment, contradicting the intended ease-of-use of the UPF. This issue is addressed by Algorithm Configuration (AC) techniques, which aim to automatically find high-quality configurations. Algorithm runtime as well as quality of solutions found by the parameterized algorithm have been shown to be improved by AC methods in wide-ranging problem settings, which includes planning. We integrate three state-of-the-art AC methods into the UPF and perform AC runs with planning engines which are integrated in the UPF. To this end, we perform AC in runtime, solution quality and anytime planning scenarios on problem instance sets from several International Planning Competitions (IPC). We demonstrate that AC methods provide performance improvements for the IPC.

Keywords: Automatic algorithm configuration · unified-planning framework · planning.

1 Introduction

The Unified Planning Framework (UPF) [15] opens planning to a wider group of users by simplifying the use of planning technology in several ways. It offers the possibility to formulate planning problems in a simple, planner-independent way, by using Python data structures and provides a standardization of the possible interactions with planning engines. Furthermore, it allows reading in planning problems from files and converting them to its native format. Alongside the possibility to transform the problem in several ways, e.g. grounding, it is possible to let the UPF choose which planning engine to use with a given planning problem.

If the planning engine chosen by the UPF is not up to task for the problem at hand, e.g., due to the planner not solving problems fast enough, the next step is to adjust it by setting its parameters. Finding a suitable parameter configuration, however, requires domain knowledge and can require significant time cost and effort. Since the intention of the UPF is to make it easier for non-experts to use planning technology, this poses a contradiction to the goals of the UPF. However, we can overcome this limitation by means of algorithm configuration (AC), thus automating the task. AC methods including CALIBRA [1], ParamILS [12], GGA/OAT [3, 16], Irace [14], SMAC [11], ReACT/ReACTR [7, 6] and CPPL [4] have been shown to effectively improve the performance of algorithms in operations research. Additionally, AC has already been applied successfully in planning. The performance of the planner Fast-Downward [10] has been shown to be significantly improved on IPC domains from 2011 through applying AC [19]. FD-Autotune [5] successfully integrates ParamILS with Fast-Downward and places its performance in the upper half of the IPC 2011 contenders. Performance optimization through AC can target different performance metrics, e.g., algorithm runtime or solution quality. The primary contribution of this paper is the integration of three of the state-of-the-art AC methods in the UPF to allow for automated AC in several planning scenarios as well as a demonstration of their application.

This paper is organized as follows. Section 2 defines AC and outlines related work in planning. In Section 3 we describe which AC methods are integrated, which planning engines are targeted and how it is achieved. Section 4 describes an experimental setup and its results, which demonstrate the impact of AC in the UPF. Finally, we conclude and discuss possible future work with the AC integration in Section 5.

2 AC in Planning

AC offers the possibility to improve performance of parameterized algorithms. Planners are parameterized in many cases and can be adjusted to improve their performance by configuring their parameter settings. Note that AC approaches and approaches adjacent to AC were investigated in planning literature and partially implemented in planning applications. However, this branch of research was not pursued further since. In this section, we first define the underlying AC problem and then outline related work in the planning community.

2.1 Automated Planning

Automated planning is the problem of synthesizing a course of actions to achieve a desired objective (called a "goal"), given a formal description of a system to be controlled [9]. Over the years different flavors of planning have been proposed, depending on the assumptions on the system specification and the kind of plan being synthesized. For the sake of this paper, we focus on classical planning problems, where the system is described using a finite set of Boolean variables (called predicates) and the actions are deterministic.

2.2 AC Problem

The AC problem underlying the methods integrated in the UPF is in the offline setting. To define the problem, we follow the formal notation introduced in [18]. Given a parameterized algorithm $\mathcal{A}$ and a set of problem instances $i \in \mathcal{I}$, the objective is to find a single high-quality configuration. $\mathcal{A}$ has configurable parameters $p_1, \ldots, p_k$, with parameters p_i having a domain Θ_i, such that $\Theta \subseteq \Theta_1 \times ... \times \Theta_k$ defines the search space of feasible configurations. $\mathcal{I}$ represents the space of problem instances over which the probability distribution $\mathcal{P}$ is defined. Furthermore, the problem instances i can optionally be described by features $f_{i,1}, \ldots f_{i,d}$ in vectors $\boldsymbol{f}_i \in \mathbb{R}^d$. A cost function from the cost function space $\mathcal{C}$, denoted as $c : \mathcal{I} \times \Theta \to \mathbb{R}$, represents the performance metric. The goal in offline AC is to find $\boldsymbol{\theta}^* \in \Theta$ which minimizes the cost function

$$\boldsymbol{\theta}^* \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \int_{\mathcal{I}} c(i, \boldsymbol{\theta}) d\mathcal{P}(i).$$

Since the distribution $\mathcal{P}$ over $\mathcal{I}$ is unknown, an alternative problem is solved in practical applications. Using a set of training instances $\mathcal{I}_{\text{train}} \subseteq \mathcal{I}$ the aggregation function $m : \mathcal{C} \times 2^{\mathcal{I}} \times \Theta \to \mathbb{R}$ is minimized to approximate $\boldsymbol{\theta}^*$. Typically, m is calculated as the arithmetic mean or a comparable function by evaluating $\hat{\boldsymbol{\theta}}$ with $\mathcal{I}_{\text{train}}$. The objective in practical applications becomes

$$\hat{\boldsymbol{\theta}} \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} m(c, \mathcal{I}_{\text{train}}, \boldsymbol{\theta}).$$

The cost function c, approximated by m, in offline AC can represent different performance metrics of the algorithm $\mathcal{A}$. In this work, we target the runtime, plan quality and the plan quality of the first plan found during anytime planning.

2.3 Configuration of Planners

Configuration of planners and the resulting performance improvement has been investigated in the planning community. The planner LPG [8], which is included in the UPF, is configured to improve performance on specific planning domains in [22]. In this work, the AC method used is ParamILS, and it significantly improves the performance of the planner on IPC 2011 domains. Another planner included in the UPF, Fast-Downward [10], is shown to improve performance on IPC 2011 domains using the AC method ParamILS in [19]. Fast-Downward and ParamILS are unified into FD-Autotune as described in [5]. This planner system participates numerous times in the IPC. In later work, Fast-Downward is configured by SMAC in [20]. Both, LPG and Fast-Downward are included in the Algorithm Configuration Library (AClib) [13]. AClib is a benchmark library commonly used for testing and development purposes in AC. We extend the idea of configuring planners by applying several state-of-the-art AC methods to a wider range of planning engines and integrating it in the UPF.

3 AC in the UPF

We enable the use of AC in the UPF by a generic AC interface. Based on this interface, four AC methods are integrated and target six of the planning engines available in the UPF. Our approach also allows integrating further AC methods and planning engines in the UPF[3]. In the following, we describe the AC interface, the included AC methods and the planning engines that can be configured.

3.1 Integration in the UPF

The generic interface manages information flows between planning engines and AC methods, as depicted in Figure 1. The AC methods are initialized with an AC scenario, which includes planning engine name and minimization metric, parameter space definitions, problem instance sets, maximum allowed runtime for an evaluation, penalty costs for not solving a problem instance, AC runtime, number of available CPUs and tournament configuration (if applicable). We implement pre-defined parameter spaces for the planners included in the experiments, but a custom definition can be loaded from files in the PCS format [13], a standard format for describing algorithm parameters, or be formulated in code by the user.

During the AC process, AC methods suggest parameter settings which need to be evaluated by running the configured planner on problem instances. The AC method also chooses which problem instances from the sets the configuration needs to be evaluated on. The interface converts the configurations to a format suitable for the planning engine and calls the planning engine with the problem instance. If the planning engine produces a plan within the given time limit, its output is converted to a format suitable for the AC method in use and passed to it. The AC interface terminates planning engines should the time limit be reached while the planning engine still runs.

3.2 Integrated AC Methods

We integrate three state-of-the-art methods into the UPF. The first method, Irace [14], employs the racing principle to effectively utilize resources. In run-

[3] Our approach is available via pip at: https://github.com/DimitriWeiss/up-ac, an installation guide and documentation at: https://up-ac.readthedocs.io/en/latest/

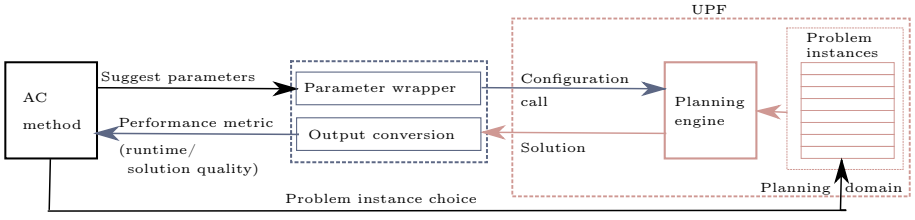

Fig. 1. Information flow between AC method, interface and planning engine.

time optimization, configuration runs can be capped before the time limit is reached. In this way, configurations exhibiting high runtime are not evaluated for a longer time period than necessary. Furthermore, Irace includes forbidden and conditional parameter value constraints into its parameter space search, which prevents the evaluation of non-functional configurations. This is also the case for the next AC method, SMAC [11], which is a Bayesian Optimization approach to AC. SMAC includes forbidden and conditional parameter value constraints and is additionally capable to include problem instance features in the AC process to assist it. We integrate the Optano Algorithm Tuner (OAT) [16] into the UPF. This AC method provides multiple AC approaches combined. OAT is based on the GGA [3] configurator, which uses a genetic algorithm variant that exploits dependencies between parameters. Finally, we integrate the ensemble-based algorithm configurator Selector, described in [23] which is currently under revision. The ensemble includes functionalities and models derived from GGA, SMAC and the realtime algorithm configurator CPPL. The Python iplementation of this method is available under the name "selector-ac" on PyPI.

3.3 Targeted Planners

The choice of planning engines to include is based on whether they are integrated in the UPF and whether they can be configured. The version of the UPF used in this work includes eight planning engines, two of which (Aries and FMAP) are not integrated with the AC interface. The planning engine Aries does not have parameters, which excludes it from AC. The multi-agent planner FMAP is not integrated because not enough multi-agent problem instances could be found for testing during the implementation.

We integrate the remaining six planning engines with the interface. The planner LPG [8] is based on local search and planning graphs. The parameter space of this planning engine includes numerous parameters only applying to specific planning domains, hence, we use a reduced parameter space definition both in the experiments in Section 4 and in the implementation. Fast-Downward [10] is a classical planning engine. It is based on heuristic search and has a unique and complex parameter space. The parameters are not set by assigning values to

IPC	**2014**	**2018**	**2023**
Domain	Barman	agricola	folding
	CaveDiving	caldera	labyrinth
	Childsnack	caldera-split	quantum-layout
	CityCar	data-network	recharging-robots
	Floortile	flashfill	ricochet-robots
	GED	nurikabe	rubiks-cube
	Hiking	organic-synthesis	slitherlink
	Maintenance	organic-synthesis-split	
	Openstacks	settlers	
	Parking	snake	
	Tetris	spider	
	Thoughtful	termes	
	Transport		
	Visitall		

Table 1. Planning domains from the IPC 2014, 2018, and 2023 used in the experiments.

individual parameters but are rather nested functions passed as a single string. Due to this fact, we also use a fraction of the possible parameter space with Fast-Downward. The planning engine SymK is a classical optimal and top-k planner based on symbolic search, which extends Fast-Downward. Since it has the same parameter space, we handle it as described for Fast-Downward. The remaining three planning engines all have a parameter space of two parameters. ENHSP [17] is an expressive numeric heuristic planner. It is capable of processing a range of domains wider than any other planning engine integrated. The planning engine Tamer [21] is based on heuristic search and a satisfiability modulo theory solver. Pyperplan [2] is based on search heuristics. While it is intended to be used as a teaching or prototyping tool and does not offer state-of-the-art performance, we nonetheless integrate it with the AC interface.

4 Experimental Results

We conduct several experiments to demonstrate the possible performance gain of the planning engines in the UPF through AC. The experiments include runtime, plan quality and anytime planning optimization scenarios. The runtime or the quality metric of a problem domain, either for the final or an intermediate plan, are minimized for each planning engine. In the anytime optimization scenario, the plan quality is also optimized. However, it is the quality the first intermediate plan that is returned by the planning engine within the time limit that is optimized. For this, we use a custom function to compute the plan quality metric that is minimized as per the domain file, or the plan length if no minimization metric is specified. We use this function to compute plan quality for Pyperplan and Tamer, since both planning engines do not output plan quality metrics. All planning engines are configured in all scenarios, except for Pyperplan and Tamer in the anytime scenario, since no anytime planning is implemented in the unified planning framework for these planners.

4.1 Problem Domains

For the experiments, we use problem instances derived from the IPC to obtain a challenging and diverse set of instances. The problem instances stem from the classical satisfiability tracks of the years 2014, 2018 and 2023. We note that the planners ENHSP and Tamer are numeric and temporal planners, respectively, thus they are strictly more expressive than the other planners, and we expect that they will underperform the other planners on these sets.

Every domain listed in Table 1 entails 20 problem instances, which amounts to 660 problem instances in total. We choose these instances because the planning problem underlying the instances can be used in runtime, plan quality and anytime planning optimization scenarios alike. Besides this, problem instances from the IPC are designed to be challenging, which fits the use case for AC.

Planner	Fast-Downward	SymK	LPG	ENHSP	tamer	pyperplan
Train instances	260	280	240	380	100	80
Test instances	140	120	120	180	60	40
$\sum$ Instances	400	400	360	560	160	120

Table 2. Numbers of instances available after filtering per planner.

4.2 Filtering Problem Instances

The planning algorithms contending in the IPC aim to solve as many of the problem instances of the track that they participate as quickly or to as good a solution quality as possible. However, it is not given that the planning engines are designed to perform best on exactly the selection of domains we use in the experiments. The UPF automatically evaluates whether the characteristics of a domain match the capabilities of a planning engine or not. Additionally, the UPF performs a sanity check of a problem definition when loading problem instances and domains. Both procedures exclude problem instances from use with a planner, if necessary.

We determine the problem instance sets to be used in the experiments as follows. First, the domains in Table 1 are filtered through the sanity check of the UPF. This excludes five domains, which results in 560 problem instances left. Next, the domains are filtered by the planning engine's capabilities, which reduces the set for all planning engines, except ENHSP, even further. For the AC process, we split the problem instances into training and test sets by domains. We roughly use a third of the domains for the test sets. The resulting numbers can be seen in Table 2.

4.3 Performance Gains on the Test Sets

All AC runs are performed on compute nodes with AMD Milan 7763 processors with 128 cores running at 2.45 GHz. The allocated AC runtime is 24 hours. We choose this runtime due to the small parameter space of only two parameters some of the planners and to avoid overfitting on some of the training sets that are small compared to typical AC settings. We evaluate the performances of the planning engines in all scenarios with the default configurations and the configurations found by the AC methods three times each. Evaluations in the runtime scenario reaching the time limit are recorded as running for 300 seconds, while evaluations not finding results within the time limit in the quality and anytime planning scenarios are recorded as values of $10,000$. The test and training instance sets used in the three scenarios are the same. The results on the test sets are listed as averages in Tables 3, 4 and 5 for comparison.

Overall, the AC methods provided configurations partly reducing the runtimes of the planning engines, with few exceptions, as illustrated in Table 3. The potential time saving of using configured planners is depicted in Figure 2. It is noteworthy that Selector provided a configuration performing worse than the

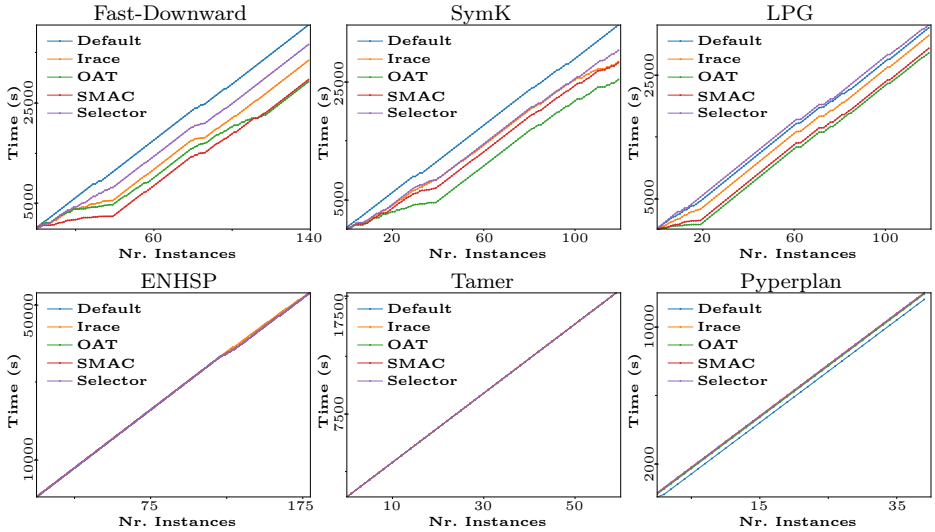

Fig. 2. Instance runtimes.

default configuration. Due to it being an ensemble method and being comprised of multiple models, we assume that the low AC runtime allocation leads to too few evaluation cycles for Selector to perform well with a decent sized parameter space. On the other hand, Selector provided a configuration for Tamer that at least solved one problem instance within the time limit, contrasted to all other configurations for Tamer in this scenario. However, it can not be excluded that the solution of this problem instance was not due to the heuristic nature of Tamer and more of a coincidence. Similarly, the performance of ENHSP could not be significantly improved. Note that all AC methods, but Irace, concluded with the default configuration. The runtime of Pyperplan, however, was degraded in this experiment. Due to Pyperplan having a parameter space of only two parameters and the experiment running on the smallest instance set size, we conclude that this is a case of overfitting to the training set.

	Performance				
Planner	Default	Irace	OAT	SMAC	Selector
Fast-Downward	288.77	238.55	208.50	211.37	261.09
SymK	288.73	236.90	211.55	235.58	252.95
LPG	181.89	174.89	159.11	163.30	184.42
ENHSP	293.34	294.41	293.42	293.34	293.41
Tamer	300.00	300.00	300.00	300.00	299.80
Pyperplan	290.28	300.00	298.07	299.87	300.00

Table 3. Average performances in runtime configuration scenario.

	Performance				
Planner	Default	Irace	OAT	SMAC	Selector
Fast-Downward	9530.28	6494.64	6510.64	6366.52	6511.64
SymK	8084.60	5296.35	4956.90	5265.79	5183.90
LPG	5619.19	6666.66	6432.89	6486.30	6006.66
ENHSP	9668.21	10000.00	9668.21	9723.34	9668.21
Tamer	10000.00	9837.45	10000.00	10000.00	9891.54
Pyperplan	9504.18	10000.00	10000.00	9504.18	10000.00

Table 4. Average performances in quality configuration scenario.

	Performance				
Planner	Default	Irace	OAT	SMAC	Selector
Fast-Downward	9530.28	7387.12	6541.05	6562.22	6662.98
SymK	9319.27	9371.10	7181.87	9400.87	6418.85
LPG	8421.63	8437.77	8867.95	8443.15	8380.64
ENHSP	9723.52	9723.52	10000.00	9889.27	9723.52

Table 5. Average performances in anytime planning configuration scenario.

The results in the quality scenario mirror the results of the runtime scenario, as illustrated in Table 4. In this scenario, however, the performance of LPG could not be improved, likely due to the AC runtime being too short.

The results of the anytime scenario are shown in Table 5. While the performance of Fast-Downward could be improved by all four methods, finding a configuration of the other planners which yields initial plans with lower quality metrics is a challenge. The success of AC, however, is higher in this scenario. This can be explained by a higher amount of evaluations in this scenario, due to intermediate plans being found quicker than final plans. For ENHSP, Irace and Selector provided the default configuration, while the configurations from OAT and SMAC resulted in worse performance.

	Performance				
Planner	Default	Irace	OAT	SMAC	Selector
Fast-Downward	279.86	215.42	220.08	220.49	219.63
SymK	280.68	235.22	206.83	235.95	245.55
LPG	150.02	139.42	149.94	142.66	158.27
ENHSP	247.65	275.03	247.85	248.32	248.07
Tamer	246.02	242.79	231.17	232.65	231.47
Pyperplan	286.36	285.36	285.79	285.48	285.35

Table 6. Average performances in runtime configuration scenario on training sets.

4.4 Performance Gains on the Training Sets

Performance gains on training sets provide further clarification about the results. The results illustrated in Table 6 confirm overfitting of small parameter space planners, e.g. very clearly for Tamer. The performance on the training set is improved, while there was no improvement on the test set. The rest of the results is in accordance to the results of the test sets.

On the training set of the quality scenario, illustrated in Table 7, The effect of the separation of domains between training and test set for small instance sets and small parameter spaces is illustrated with Pyperplan. The performance of different configurations is barely distinguishable to the default configuration on the training set but differs significantly on the test set, which amounts to unusable feedback for the AC methods. This also affects the other planning engines with only two parameters in an attenuated form.

The results in Table 8 mirror the results on the test sets. This can be explained by the higher amount of evaluations in the anytime optimization scenario, since an intermediate plan is generated quicker than a final plan, as is the criterion for solving an instance in the quality as well as the runtime scenario.

	Performance				
Planner	Default	Irace	OAT	SMAC	Selector
Fast-Downward	9335.46	7435.61	6239.51	6730.42	6603.71
SymK	9889.02	7608.19	7044.91	7634.79	7570.87
LPG	5375.75	5957.98	5575.07	5088.04	5714.65
ENHSP	8080.96	9660.22	8107.26	8349.58	8107.27
Tamer	8108.93	7412.46	10000.00	8009.35	7412.46
Pyperplan	9376.98	9376.85	9376.85	9376.98	9376.98

Table 7. Average performances in quality configuration scenario on training sets.

	Performance				
Planner	Default	Irace	OAT	SMAC	Selector
Fast-Downward	9284.41	6922.65	6908.66	7022.21	7010.48
SymK	9359.49	9359.45	7527.87	9383.23	6842.50
LPG	8364.21	8337.44	9462.92	8336.96	8334.08
ENHSP	8107.27	8107.27	9340.31	8806.83	8107.26

Table 8. Average performances in anytime planning configuration scenario on training sets.

5 Conclusion and Future Work

The results of the three AC scenarios show that AC can improve performance of planning algorithms. The impact of AC, however, is dependent on an adequate amount of resources allocated to the AC process. Planning engines with small parameter spaces can lead to overfitting, while planning engines with a bigger parameter space need to be configured for a reasonable amount of time. The computational effort of applying AC clearly outweighs the savings in runtime on the instance sets used in the experiments. The value, however, lies in improved performance of an algorithm in a competition or in the reoccuring runtime savings of a configured algorithm applied to further problem instances in practice. The same is true for quality and anytime quality. The results with Fast-Downward and SymK demonstrate the utility of AC in adequate settings. Given that the parameter spaces of Fast-Downard and SymK were reduced due to their vastness, it cannot be ruled out that parameters with high impact on the performance of the planning engines were not considered in the experiments. It is possible that higher performance gains can be achieved through AC with these two planning engines. We note that the AC runtime certainly needs to be increased accordingly since a larger parameter space would need to be searched. The experiments with ENHSP included the biggest instance set and demonstrated that the default configuration already generalizes over the diverse set of domains. Taking into account that the problem instance sets are very challenging and different domains were evaluated in testing than were used in training, the results are adequate.

Future work can be done with the integration of further AC methods' functionalities. For example, OAT includes a gray-box functionality that is not yet integrated in the UPF. Applying gray-box AC allows for capturing and processing intermediate algorithm output to learn about configurations and prematurely terminate their runs. This allows for more evaluations in the same allocated AC runtime and proved to further improve the AC process [24].

Acknowledgments. This work was partially supported by the AIplan4eu Grant Agreement number: 101016442 - AIPlan4EU-H2020-ICT-2018-20 / H2020-ICT-020-2, in Track A Agreement Number 1961741. The authors gratefully acknowledge the computing time provided to them on the high-performance computer Noctua 1 at the NHR Center PC2. These are funded by the Federal Ministry of Education and Research and the state governments participating on the basis of the resolutions of the GWK for the national high-performance computing at universities (www.nhr-verein.de/unsere-partner). Andrea Micheli was supported by the STEP-RL project funded by the European Research Council under GA n. 101115870.

Disclosure of Interests. The authors have no competing interests.

References

1. Adenso-Díaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research **54**, 99–114 (02 2006)
2. Alkhazraji, Y., Frorath, M., Grützner, M., Helmert, M., Liebetraut, T., Mattmüller, R., Ortlieb, M., Seipp, J., Springenberg, T., Stahl, P., Wülfing, J.: Pyperplan (Mar 2020), https://doi.org/10.5281/zenodo.3701399
3. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Principles and Practice of Constraint Programming. pp. 142–157 (09 2009). https://doi.org/10.1007/978-3-642-04244-7_14
4. El Mesaoudi-Paul, A., Weiß, D., Bengs, V., Hüllermeier, E., Tierney, K.: Pool-Based Realtime Algorithm Configuration: A Preselection Bandit Approach, Lecture Notes in Computer Science, vol. 12096, pp. 216–232. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-53552-0_22
5. Fawcett, C., Helmert, M., Hoos, H., Karpas, E., Röger, G., Seipp, J.: Fd-autotune: Domain-specific configuration using fast downward. ICAPS 2011 Workshop on Planning and Learning pp. 13–17 (2011)
6. Fitzgerald, T., Malitsky, Y., O'Sullivan, B.: Reactr: Realtime algorithm configuration through tournament rankings. In: International Joint Conferences on Artificial Intelligence Organization (IJCAI). pp. 304–310 (2015)
7. Fitzgerald, T., Malitsky, Y., O'Sullivan, B.J., Tierney, K.: ReACT: Real-Time Algorithm Configuration through Tournaments. In: Annual Symposium on Combinatorial Search (SoCS) (2014)
8. Gerevini, A., Serina, I.: Lpg: A planner based on local search for planning graphs with action costs. In: Conference: Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems. pp. 13–22 (01 2002)
9. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning - theory and practice. Elsevier (2004)
10. Helmert, M.: The fast downward planning system. J. Artif. Int. Res. **26**(1), 191–246 (jul 2006)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization (LION). p. 507–523 (2011)
12. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research (JAIR) pp. 267–306 (Sep 2009)
13. Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H., Leyton-Brown, K., Stützle, T.: Aclib: A benchmark library for algorithm configuration. In: International Conference on Learning and Intelligent Optimization (LION). pp. 36–40 (02 2014). https://doi.org/10.1007/978-3-319-09584-4_4
14. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives pp. 43–58 (01 2016). https://doi.org/https://doi.org/10.1016/j.orp.2016.09.002
15. Micheli, A., Bit-Monnot, A., Röger, G., Scala, E., Valentini, A., Framba, L., Rovetta, A., Trapasso, A., Bonassi, L., Gerevini, A.E., Iocchi, L., Ingrand, F., Köckemann, U., Patrizi, F., Saetti, A., Serina, I., Stock, S.: Unified Planning: Modeling, manipulating and solving AI planning problems in Python. SoftwareX **29**, 102012 (2025). https://doi.org/https://doi.org/10.1016/j.softx.2024.102012, https://www.sciencedirect.com/science/article/pii/S2352711024003820

16. OPTANO: OPTANO Algorithm Tuner documentation (2022), https://docs.optano.com/algorithm.tuner/current/
17. Scala, E., Haslum, P., Thiebaux, S., Ramirez, M.: Interval-based relaxation for general numeric planning. In: Proceedings of the Twenty-Second European Conference on Artificial Intelligence. p. 655–663. ECAI'16, IOS Press, NLD (2016). https://doi.org/10.3233/978-1-61499-672-9-655, https://doi.org/10.3233/978-1-61499-672-9-655
18. Schede, E., Brandt, J., Tornede, A., Wever, M., Bengs, V., Hüllermeier, E., Tierney, K.: A survey of methods for automated algorithm configuration. Journal of Artificial Intelligence Research **75**, 425–487 (10 2022). https://doi.org/10.1613/jair.1.13676
19. Seipp, J., Braun, M., Garimort, J., Helmert, M.: Learning portfolios of automatically tuned planners. Proceedings of the International Conference on Automated Planning and Scheduling **22**(1), 368–372 (May 2012). https://doi.org/10.1609/icaps.v22i1.13538, https://ojs.aaai.org/index.php/ICAPS/article/view/13538
20. Seipp, J., Sievers, S., Hutter, F.: Fast downward smac. In: IPC 2014 (2014), https://api.semanticscholar.org/CorpusID:8317657
21. Valentini, A., Micheli, A., Cimatti, A.: Temporal planning with intermediate conditions and effects. Proceedings of the AAAI Conference on Artificial Intelligence **34**, 9975–9982 (04 2020). https://doi.org/10.1609/aaai.v34i06.6553
22. Vallati, M., Fawcett, C., Gerevini, A., Hoos, H., Saetti, A.: Generating fast domain-specific planners by automatically configuring a generic parameterised planner. In: ICAPS 2011 (06 2011)
23. Weiss, D., Schede, E., Tierney, K.: Selector: Ensemble-based automated algorithm configuration (2025), under revision
24. Weiss, D., Tierney, K.: Realtime gray-box algorithm configuration using cost-sensitive classification. Annals of Mathematics and Artificial Intelligence pp. 1–22 (08 2023). https://doi.org/10.1007/s10472-023-09890-x