On the Effective Configuration of Planning Domain Models

Mauro Vallati
University of Huddersfield
m.vallati@hud.ac.uk

Frank Hutter
University of Freiburg
fh@informatik.uni-freiburg.de

Lukáš Chrpa and **Thomas L. McCluskey**
University of Huddersfield
{l.chrpa,t.l.mccluskey}@hud.ac.uk

Abstract

The development of domain-independent planners within the AI Planning community is leading to "off the shelf" technology that can be used in a wide range of applications. Moreover, it allows a modular approach – in which planners and domain knowledge are modules of larger software applications – that facilitates substitutions or improvements of individual modules without changing the rest of the system. This approach also supports the use of reformulation and configuration techniques, which transform how a model is represented in order to improve the efficiency of plan generation.

In this paper, we investigate how the performance of planners is affected by *domain model configuration*. We introduce a fully automated method for this configuration task, and show in an extensive experimental analysis with six planners and seven domains that this process (which can, in principle, be combined with other forms of reformulation and configuration) can have a remarkable impact on performance across planners. Furthermore, studying the obtained domain model configurations can provide useful information to effectively engineer planning domain models.

1 Introduction

The development of domain-independent planners within the AI Planning community is leading to the use of this "off the shelf" technology to a wide range of applications. This is despite the complexity issues inherent in plan generation, which are exacerbated by the separation of planner logic from domain knowledge. One important class of application is where the planner is an embedded component within a larger framework, with the domain model and planning problem being generated by hand, or potentially automatically. This application class has the advantage that planners which accept the same domain model language and deliver the same type of plans can be interchanged in a modular way, without any changes to the rest of the system.

This modular approach also supports the use of reformulation and configuration techniques which can automatically re-formulate, re-represent or tune the domain model and/or problem description in order to increase the efficiency of a planner and increase the scope of problems solved. The idea is to make these techniques to some degree independent of domain and planner (that is, applicable to a range of domains and planning engine technologies), and use them to form a wrapper around a planner, improving its overall performance for the domain to which it is applied. Types of reformulation include macro-learning [Botea *et al.*, 2005; Newton *et al.*, 2007], action schema splitting [Areces *et al.*, 2014] and entanglements [Chrpa and McCluskey, 2012]: here the domain model is transformed to a more efficient form that is fed into the planner. The transformation is then reversed after a solution has been found, such that the solution is rephrased in the original formulation language. Another general approach for improving planner efficiency is to apply automated algorithm configuration tools [Hutter *et al.*, 2009] to tune the free parameters of a planner to a particular type of instances. In planning, this approach has mainly been applied to optimise LPG [Vallati *et al.*, 2013] and Fast Downward [Fawcett *et al.*, 2011], yielding average speedup factors in satisficing planning of up to 23 and 118, respectively.

In this context, we explore the *automated configuration of domain models* instead of algorithm parameters. We investigate this area by considering domain models written in PDDL, using as the parameters of configuration the potential orderings of predicate declarations, the ordering of operators, and the ordering of predicates within pre- and post conditions. Through comprehensive experiments using six state-of-the-art planners on seven planning domains and approximately 3,800 planning problems, we demonstrate that configuration is an effective approach for improving planners' performance. Next to making a significant contribution to planning speed-up, this work (i) introduces a general framework that can be exploited for improving planners' performance by tailoring the domain model configuration; (ii) gives insights into how to view results from competitions such as the International Planning Competition (IPC), by showing that even small changes in the models can change the final ranking of participants; and (iii) provides useful information to help engineer more efficient planning domain models.

Howe and Dahlman (2002) investigated how experiment design decisions affect the performance of planning systems, noticing that changing the order of elements in either domain

models or problem specifications can affect performance. Our investigation expands their pioneering work in a number of ways: (i) while their analysis dates back 13 years and was focused on STRIPS problems only, we consider a larger set of PDDL features, and current planners and instances; (ii) while their analysis was limited to a fairly small number of permutations (10 per problem) we consider the full combinatorial space of permutations for each planner; and (iii) we evaluate the impact of domain model configuration using various performance metrics, namely coverage, IPC score and penalized average runtime. Maybe more fundamentally, (iv) we introduce an automatic approach for configuring domain models in order to improve planners' performance; and (v) we describe which changes have a stronger impact on planners' performance, thus providing help for engineering future domain models.

2 Domain Model Configuration

In this section, we first describe some background on automated planning and algorithm configuration, then describe the choices we have in designing domain models, and finally show how to use configuration to make these choices automatically.

2.1 Automated Planning

Automated planning, and specifically classical planning, deals with finding a (partially or totally ordered) sequence of actions transforming a static, deterministic and fully observable environment from some initial state to a desired goal state [Ghallab *et al.*, 2004].

In the classical planning, environment is represented by first order logic *predicates*. *States* are defined as sets of ground predicates.

A *planning operator* $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$ is specified such that $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ (*op_name* is a unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator), $\mathit{pre}(o)$ is a set of predicates representing operator's precondition, $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of predicates representing operator's negative and positive effects. *Actions* are ground instances of planning operators. An action $a = (\mathit{pre}(a), \mathit{eff}^-(a), \mathit{eff}^+(a))$ is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

A *planning domain model* is specified via sets of predicates and planning operators. A *planning problem* is specified via a planning domain model, initial state and set of goal atoms. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

2.2 Algorithm Configuration

Most algorithms have several free parameters that can be adjusted to optimise performance (e.g., solution cost, or runtime to solve a set of instances). Formally, this *algorithm configuration* problem can be stated as follows: given a parametrised algorithm with possible configurations $\mathcal{C}$, a benchmark set Π, and a performance metric $m(c, \pi)$ that measures the performance of configuration $c \in \mathcal{C}$ on instance $\pi \in \Pi$, find a configuration $c \in \mathcal{C}$ that minimises m over Π , i.e., that minimises

$$f(c) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} m(c, \pi). \quad (1)$$

The AI community has recently developed dedicated algorithm configuration systems to tackle this problem [Hutter *et al.*, 2009; Ansótegui *et al.*, 2009; Yuan *et al.*, 2010; Hutter *et al.*, 2011]. Here we employ the sequential model-based algorithm configuration method SMAC [Hutter *et al.*, 2011], which represents the state of the art of configuration tools and, in contrast to ParamILS [Hutter *et al.*, 2009], can handle continuous parameters. SMAC uses predictive models of algorithm performance [Hutter *et al.*, 2014b] to guide its search for good configurations. More precisely, it uses previously observed $\langle$configuration, performance$\rangle$ pairs $\langle c, f(c)\rangle$ and supervised machine learning (in particular, random forests [Breiman, 2001]) to learn a function $\hat{f} : \mathcal{C} \to \mathbb{R}$ that predicts the performance of arbitrary parameter configurations (including those not yet evaluated). The performance data to fit these models is collected sequentially. In a nutshell, after an initialisation phase, SMAC iterates the following three steps: (1) use the performance measurements observed so far to fit a random forest model $\hat{f}$; (2) use $\hat{f}$ to select a promising configuration $c \in \mathcal{C}$ to evaluate next, trading off exploration in new parts of the configuration space and exploitation in parts of the space known to perform well; and (3) run c on one or more benchmark instances and compare its performance to the best configuration observed so far.

2.3 Degrees of Freedom in Domain Models

A key product of knowledge engineering for automated planning is the domain model. Currently, this knowledge engineering is somewhat of an ad-hoc process, where the skills of engineers may significantly influence the quality of the domain model, and therefore the performance of planners. Although usually underestimated, there are significant degrees of freedom in domain model descriptions.

Here, we are focusing on one particular aspect of knowledge engineering: given the components of a domain model (domain predicates and operators with pre- and post-conditions), in which order should these be listed in the domain model? Specifically, we considered as configurable the order in which domain predicates are declared and operators are listed in the PDDL domain model and, within every operator, the order in which pre and post condition predicates are shown.

Example 1. The simplest domain we consider here, Parking, contains 5 domain predicates and 4 operators, with 3, 4, 3, and 4 pre-conditions and 4, 5, 5, and 4 post-conditions, respectively. We need to specify 10 different orders: the order of the 5 domain predicates; the order of the 4 operators; 4 pre-condition orders (one per operator); and 4 post-condition orders (one per operator).

2.4 Configuration of Domain Models

In this work, we use SMAC for configuring domain models $\mathcal{M}$ in order to improve the runtime performance of a given planner $\mathcal{P}$. In this setting, it is possible to evaluate the "quality" of a domain model $\mathcal{M}$ by the performance of $\mathcal{P}$ when run on problem instances encoded using domain model $\mathcal{M}$. Therefore, on a high level our approach is to express the degrees of freedom in the domain model as parameters, and use algorithm configuration to set these parameters to optimise planner performance.

It is not directly obvious how to best encode the degrees of freedom in domain models as parameters. Configuration over orders is not natively supported by any general configuration procedure, but needs to be encoded using numerical or categorical parameters. The simplest encoding for an ordering of m elements would be to use a single categorical parameter with $m!$ values. However, next to being infeasible for large m, this encoding treats each ordering as independent and does not allow configuration procedures to discover and exploit patterns such as "element i should come early in the ordering". Two better encodings include:

1. $\binom{m}{2}$ **binary parameters**: each parameter expresses which of two elements should appear first in the ordering, with the actual order determined, e.g., by ordering elements according to the number of elements that should appear after them.
2. m **continuous parameters**: each element i is associated a continuous "precedence" value p_i from some interval (e.g., $[0, 1]$), and elements are ordered by their precedence.

To select which of these two strategies to use in our analysis, we ran preliminary experiments to evaluate them; in these experiments, we configured one domain model (Depots) for each of two planners (LPG and Yahsp3 [Vidal, 2014]). In both of these experiments, the two strategies lead to domain model configurations of similar quality, with a slight advantage for the second strategy; we therefore selected that strategy for our study. Nevertheless, we would like to stress the preliminary nature of our comparison and that one could encode configuration across orders in many other possible ways; we believe that a comprehensive study of the possible strategies may well lead to substantial performance improvements (and would thereby boost domain configuration further). We leave such a study to future work to fully focus on domain configuration here.

We now describe the encoding we chose in more detail. To express domain configuration as an algorithm configuration problem we introduce a numerical parameter p_i (with domain $[0, 1]$) for each configurable element (i.e., each domain predicate, operator, pre-condition and post-condition of an operator). Thus, the configuration space for domains with m configurable elements is $\mathcal{C} = [0, 1]^m$. A configuration $c = \langle p_1, ..., p_m \rangle \in \mathcal{C}$ instantiates each of the parameters, and the corresponding domain model configuration for it is obtained by grouping and sorting the parameters as follows. For domains with k operators, PDDL elements are grouped into $2 + 2k$ groups: predicate declarations, operators and, for each operator, preconditions and post-condition predicates. Within each group, items are sorted in order of increasing parameter value; ties are broken alphabetically. Following this order, elements are listed in the resulting PDDL model. The simplest domain we consider here is Parking (see Example 1), with 41 parameters. The most complex domain we considered is Rovers, with 109 parameters.

Example 1 (continued). In the Parking example (5 domain predicates and 4 operators, with 3, 4, 3, and 4 pre-conditions and 4, 5, 5, and 4 post-conditions, respectively), our formulation of the domain configuration problem has 41 parameters with domain $[0, 1]$: 5 domain predicate precedence parameters, 4 operator precedence parameters, (3+4+3+4) pre-condition precedence parameters, and (4+5+5+4) post-condition precedence parameters. For the sake of simplicity, in the remainder of this example, we focus on configuring only the degrees of freedom of the first operator. It has 3 pre-conditions ($pre1$, $pre2$, $pre3$) and 4 post-conditions ($post1$, $post2$, $post3$, $post4$). This gives rise to 7 parameters: $p1$, $p2$, $p3$ (which control the order of pre-conditions) and $p4$, $p5$, $p6$, $p7$ (which control the order of post-conditions). SMAC therefore searches in the parameter space $[0, 1]^7$, and each parameter setting induces a domain model. If SMAC evaluates, e.g., [$p1$=0.32, $p2$=0.001, $p3$=0.7, $p4$=0.98, $p5$=0.11, $p6$=0.34, $p7$=0.35], this induces a domain model in which preconditions are ordered ($pre2$, $pre1$, $pre3$), and post-conditions are ordered ($post2$, $post3$, $post4$, $post1$).

A major difference between standard algorithm configuration on the one hand and domain configuration on the other is that the former only applies to planners that expose a large number of explicit tuning parameters. For this reason, within planning, standard algorithm configuration has only been applied to LPG [Vallati *et al.*, 2013]) and Fast Downward [Fawcett *et al.*, 2011]). In contrast, in domain configuration we configure over the *inputs* of a planner, which makes this technique planner-independent.

3 Experimental Analysis

Our experimental analysis aims to evaluate the impact that domain model configuration, as described in the previous section, has on state-of-the-art planning systems.

3.1 Settings

We selected 6 planners, based on their performance in the Agile track of the international planning competition (IPC) 2014 and/or the use of different planning approaches: Jasper [Xie *et al.*, 2014], Lpg [Gerevini *et al.*, 2003], Madagascar (in the following abbreviated as Mp) [Rintanen, 2014], Mercury [Katz and Hoffmann, 2014], Probe [Lipovetzky *et al.*, 2014], and Yahsp3 [Vidal, 2014].

No portfolio-based planners were included since so far no portfolios exist yet that exploit different domain models. Nevertheless, by opening up the design space of domain models and improving the best basic planners for particular domains, we substantially increase the potential of future portfolios.

We focused our study on domains that have been used in IPCs and for which a randomised problem generator is available. The models we chose had at least four operators and,

possibly, a large number of predicates: Blocksworld (4 ops version), Depots, Matching-Bw, Parking, Rovers, Tetris, and ZenoTravel.

For each domain we study, we created approximately 550 random instances with the domain's random instance generator. We split these instances into a training set (roughly 500 instances) and a test set (roughly 50 instances) in order to obtain an unbiased estimate of generalisation performance to previously unseen instances from the same distribution. Throughout the paper, we only report results on these test sets.

Configuration of domain models was done using SMAC version 2.08 (publicly available at `http://www.aclib.net/SMAC`). The performance metric we optimised trades off coverage (# instances for which we find a plan) and runtime for successful instances; specifically, we minimise Penalized Average Runtime (PAR), counting runs that crash or do not find a plan as ten times the cutoff time (PAR10).

We performed experiments on AMD Opteron™ machines with 2.4 GHz, 8 GB of RAM and Linux operating system. Each of our configuration runs was limited to a single core, and was given an overall runtime and memory limits of 5 days and 8GB, respectively. As in the Agile track of the IPC 2014, the cutoff time for each instance, both for training and testing purposes, was 300 seconds. We defined IPC score as in the Agile track: let $T_p(\mathcal{C})$ denote the time planner $\mathcal{C}$ requires to solve problem p and let T_p^* denote the minimum time required by the compared planners to solve the problem; then, $Score(\mathcal{C}, p)$ is 0 if $\mathcal{C}$ does not solve P and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise. The IPC score of a planner $\mathcal{C}$ on a set of problems P is then given by $\sum_{p \in P} Score(\mathcal{C}, p)$.

3.2 Results

Table 1 compares the results of planners running on the original domain model – the one that has been used in IPCs – and the specifically configured domain model, obtained by running SMAC. These results indicate that the configuration of the domain model has a substantial impact on the planners' performance, with several substantial and statistically significant improvements (as judged by a Wilcoxon signed rank test [Wilcoxon, 1945]).

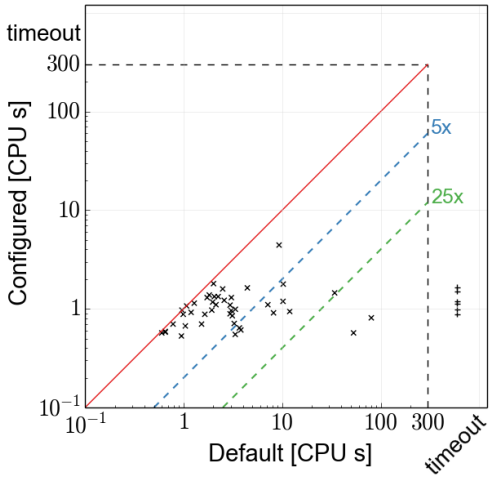
Figure 1 visualises the improvements for two cases. Figure 1(a) shows that Yahsp had mediocre performance on Depots using the original domain configuration (e.g., timing out on 12% of the instances after 300 seconds), but that after domain configuration it could solve every single instance in less than 10 seconds. Figure 1(b) shows that Probe benefited substantially from domain configuration on the Tetris domain, with both its coverage tripling and its runtime decreasing on solved instances.

Finally, of note is that in three domains domain configuration changed what is the best planner (according to PAR10):

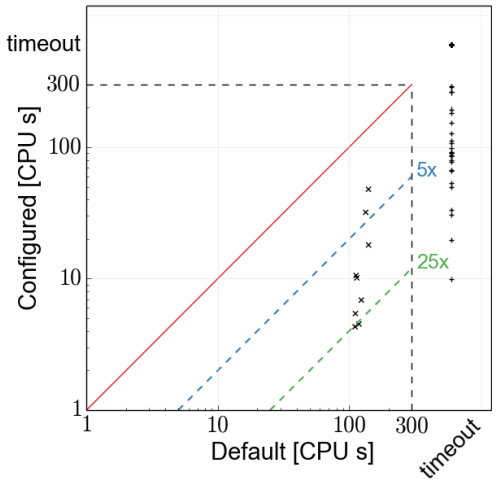
- In *Blocksworld*, LPG performed best on the original domain model, but its penalized average runtime was "only" halved by automatic domain configuration, whereas Yahsp was sped up by a factor of 30, ending up beating LPG.

	PAR10		Solved		IPC score	
Planner	C	O	C	O	C	O
BlocksWorld						
Jasper	**307.3**	470.1	90.9	85.4	49.0	45.0
Lpg	**5.9**	10.3	100.0	100.0	53.4	45.1
Mp	2896.0	2949.6	3.6	1.8	2.0	0.9
Mercury	90.6	90.3	98.2	98.2	53.5	53.8
Probe	**29.3**	30.4	100.0	100.0	54.7	53.9
Yahsp3	2.9	57.5	100.0	98.2	47.4	47.8
Depots						
Jasper	2295.0	2241.3	26.4	24.5	12.9	11.6
LPG	**9.0**	78.3	100.0	98.0	47.0	40.9
Mp	3.7	3.7	100.0	100.0	48.0	47.9
Mercury	2883.4	2883.4	4.1	4.1	2.0	2.0
Probe	14.1	14.2	100.0	100.0	47.8	47.4
Yahsp3	**1.1**	373.4	100.0	87.8	49.0	31.7
Matching-Bw						
Jasper	524.1	532.1	84.0	84.0	38.7	37.9
LPG	**685.2**	1278.0	78.0	58.0	35.8	22.9
Mp	3000.0	3000.0	0.0	0.0	–	–
Mercury	1695.3	1694.5	44.0	44.0	21.7	21.8
Probe	1464.0	1338.6	52.0	56.0	23.3	25.8
Yahsp3	**18.5**	185.7	100.0	94.0	34.9	44.2
Parking						
Jasper	1446.2	1448.9	53.6	53.6	29.8	29.2
LPG	2947.0	2846.1	1.8	5.4	1.0	2.7
Mp	855.3	858.1	73.2	73.2	39.8	38.9
Mercury	1446.2	1444.8	53.6	53.6	29.7	29.9
Probe	1268.0	1418.1	58.9	53.6	29.6	28.1
Yahsp3	2059.8	1955.1	32.1	35.7	16.8	19.6
Rovers						
Jasper	**2498.6**	2544.1	18.3	16.7	11.0	9.9
Lpg	**23.7**	26.1	100.0	100.0	59.1	56.9
Mp	3000.0	3000.0	0.0	0.0	–	–
Mercury	**1173.4**	1536.9	66.7	53.3	40.0	31.7
Probe	3000.0	3000.0	0.0	0.0	–	–
Yahsp3	7.5	7.5	100	100.0	59.9	59.8
Tetris						
Jasper	1937.3	1944.1	37.0	37.0	19.5	18.9
LPG	**1907.8**	2723.1	37.0	9.3	19.8	3.6
Mp	1860.9	1863.3	38.9	38.9	21.0	20.3
Mercury	1985.1	1983.4	35.2	35.2	18.6	18.9
Probe	**1490.6**	2520.5	51.9	16.7	28.0	4.5
Yahsp3	2615.8	2614.6	13.0	13.0	6.4	7.0
ZenoTravel						
Jasper	**174.9**	251.4	100.0	98.0	50.0	46.8
LPG	**36.4**	43.1	100.0	100.0	49.1	46.0
Mp	9.5	9.4	100.0	100.0	49.2	49.1
Mercury	84.6	**82.4**	100.0	100.0	49.4	49.9
Probe	138.9	143.1	100.0	100.0	49.1	48.3
Yahsp3	2.5	2.8	100.0	100.0	47.9	46.1

Table 1: Test set results in terms of PAR10, percentage of solved problems and IPC score, of planners running on original domain models (O) and SMAC-configured models (C). Boldface indicates statistically significant improvements of PAR10 performance. IPC score was calculated separately for each planner, considering only the performance of its two domain model configurations. The decrease in IPC score of Yahsp3 on Matching-Bw by configuration is not a typo: even though the optimisation objective (PAR10) substantially improved, it did so at the cost of solving easy instances slower.



(a) Yahsp, Depots



(b) Probe, Tetris

Figure 1: Scatter plots comparing runtime of planners using configured (y-axis) and original (x-axis) domain models; timeouts at 300s are plotted separately, dashed lines indicate 5-fold and 25-fold speedups, respectively.

- In *Depots*, Mp performed best on the original domain model, but since automatic domain configuration helped Yahsp reduce its penalized average runtime by a factor of more than 300 (and did not improve Mp), Yahsp clearly performed best on this domain after configuration.
- In *Tetris*, a similar situation occurred: Mp performed best on the original domain model, but since automatic domain configuration helped Probe solve three times more problems (and did not improve Mp), Probe clearly performed best on this domain after configuration.

For providing a better overview of the impact of different domain model configurations on the considered planners, we ran all planners on all domain model configurations identified by SMAC. Table 2 shows the results of this comparison. Generally speaking, performance of planners varied across these domain models, but not substantially. This indicates

Planners	PAR10					
	Js	LPG	Mp	Me	Pb	Y3
Jasper	**1293.2**	1376.9	1318.2	1338.7	1360.7	1354.9
LPG	867.9	**701.4**	893.0	918.9	894.6	757.2
Mp	1833.3	1818.3	1802.6	1796.6	**1781.1**	1796.2
Mercury	1327.1	1341.8	1349.0	**1312.7**	1320.0	1319.9
Probe	672.4	593.3	707.6	588.3	**578.9**	619.8
Yahsp3	311.4	319.6	391.0	352.5	326.2	**309.8**
Planners	% Solved Problems					
	Js	LPG	Mp	Me	Pb	Y3
Jasper	**59.1**	56.1	58.3	57.5	56.7	57.0
LPG	71.7	**77.3**	70.9	70.1	70.9	75.4
Mp	39.3	39.8	40.4	40.6	**41.2**	40.6
Mercury	57.8	57.2	57.0	**58.3**	58.0	58.0
Probe	78.9	81.6	77.8	81.8	**82.1**	80.7
Yahsp3	90.1	89.8	87.4	88.8	89.6	**90.1**
Planners	IPC score					
	Js	LPG	Mp	Me	Pb	Y3
Jasper	**90.6**	89.0	87.8	88.6	86.8	86.5
LPG	170.3	**192.6**	167.2	164.8	167.5	183.9
Mp	100.1	100.4	**104.4**	102.7	103.3	102.2
Mercury	90.3	89.5	89.1	**91.1**	90.7	90.6
Probe	147.4	156.8	142.5	154.2	**158.1**	152.2
Yahsp3	285.6	285.9	272.0	259.6	285.0	**292.3**

Table 2: Performance of each planner (one row per planner) running on the domain configurations configured by SMAC (each column represents the domain model configured for one planner). Respectively, columns Js, LPG, Mp, Me, Pb and Y3 indicate the domain-model configurations identified by SMAC for the Jasper, LPG, Madagascar, Mercury, Probe and Yahsp3 planner. Performance is given as cumulative across all the test instances. IPC score is evaluated by considering all the planners and all the domain model configurations at the same time. Bold indicates best performance, also considering hidden decimals, with respect to each planner.

that there possibly exists a single configuration that can boost performance of all the considered planners. As could be expected, almost all the planners achieved slightly better performance when using the domain model specifically configured for them. The only exception is Mp, which usually performs slightly better, in terms of PAR10 and solved problems, when running on domain models configured for Probe. We hypothesise this is due to the fact that Mp timed out on more than half the training instances. This slows down the configuration process, likely causing SMAC (in its fixed budget of five CPU days) to only find configurations that are quite far from optimal for this solver.

Next, in order to study how much average impact a change of domain configuration can have across planners, for each domain we identified the "best fixed configuration", as the configuration that most improved the IPC score of a specific planner. (For instance, in Blocksworld the best fixed configuration is the one for LPG, which allows the planner to increase its IPC score by 8.3 points, see Table 1). In Table 3 we report the performance of the six planners exploiting, for each domain, the best fixed configuration. Interestingly, exploiting these configurations allows all the considered planners to achieve statistically significantly better performance than when using the original configuration. This indicates that the original domain models, i.e. the models which are currently used when benchmarking planning systems, are not in a "planner-friendly" shape. This supports the intuition

Planner	IPC score		PAR10		Solved	
	B	O	B	O	B	O
Jasper	**252.2**	195.9	975.3	1375.9	69.4	56.3
LPG	**261.2**	192.1	826.0	926.2	72.9	69.7
Mp	**232.5**	167.9	1109.4	1637.5	63.8	46.1
Mercury	**236.8**	183.9	1126.6	1381.5	63.5	55.8
Probe	**294.2**	205.5	602.3	1265.3	81.0	58.7
Yahsp3	**310.9**	260.0	310.7	390.6	90.1	87.4

Table 3: Results on all the testing instances, in terms of IPC score, PAR10 and percentage of solved problems, of planners running on original domain models (O) and the best fixed configured models (B). Bold indicates statistically different performance.

Pos	**Original**	**Configured**	**Best**
1	Yahsp (297)	Yahsp (303)	Yahsp (306)
2	Lpg (168)	Lpg (198)	Lpg (189)
3	Mp (121)	Probe (123)	Probe (173)
4	Probe (112)	Mp (117)	Jasper (123)
5	Mercury (91)	Mercury (95)	Mp (119)
6	Jasper (90)	Jasper (94)	Mercury (105)

Table 4: Relative position of considered planners (IPC score) on original domain models (Original), configured for each planner (Configured) and the configuration that provides the best improvements among all the considered (Best). Planners are listed in decreasing order of IPC score.

that there are knowledge engineering policies for encoding domain models that can lead to an average performance improvement of planning systems.

Finally, we selected the two domains in which the planners' performance mostly improved by using configured domain models: Tetris and Depots (see Table 1). On these domains, we also used SMAC for identifying a "bad" domain configuration, i.e., the configuration that *maximises* PAR10. Maximising PAR10 means identifying configurations that slow down the planning process, in order to increase the number of timeouts. We assessed how such configurations affect planners by comparing performance planners achieved when exploiting them, with those they obtained on the Original and Best models. We observed a noticeable –and statistically significant– deterioration in performance both in terms IPC score (-21.1 vs Original, -64.4 vs Best), solved problems (-2.0% vs Original, -6.6% vs Best) and average runtime on solved instances ($+4.4$ vs Original, $+16.3$ vs Best) among all the planners. Results are averages across both domains.

3.3 Discussion

In order to understand how the different models affect the relative performance of planners, we simulate the execution of three "planning competitions" between the six considered planners on the 50 testing instances of the selected domains. We used the same settings as for the Agile track of IPC-14. In the first competition, planners ran on the original domain models. In the second, all the planners ran on the best fixed domain model configurations (Best), selected as described in Section 3.2. Finally, in the third competition, every planner ran on the specifically configured domain models (Configured). Although the third competition is unusual, in the sense that planners are actually running on different models, it can provide some interesting insights. The results of the three competitions are shown in Table 4. Interestingly, the IPC score of all the planners increases when considering Configured models, and it usually improves further on the Best possible model. This is possibly due to the fact that good configurations are similar among considered planners. Moreover, for planners which did not solve many of the training problems, SMAC probably was not able to explore large areas of the configuration space; therefore the specifically configured domain models can be of lower quality than the best fixed model. Finally, we notice that in Table 4, the relative performance of planners, i.e. their ordering, changed in the three different settings of the competition.

Our intuition behind the substantial performance variations we have reported with different domain models throughout this paper is that a different ordering can influence tie breaking. For instance, in A* search, situations where several nodes have the same best f-value are common. Intuitively, all of these nodes are equally good and many implementations thus select the node that came first or last. Therefore, a different order of operators/predicates can drastically change the order in which nodes are considered, thus changing the behaviour of the search. Also, a smart ordering of preconditions can lead to performance improvements in cases where some preconditions are more likely to be unsatisfied than others; having them as first in the checking list can avoid a possibly large number of checks. A similar behaviour has been empirically observed by Valenzano *et al.* (2014), who noticed that changing the order in which operators are considered during search significantly affects the performance of greedy best-first-based planning systems. Moreover, we empirically observed that differently-configured domain models can lead to different SAS+ encodings.

In order to shed some light on the importance of elements' order, thus providing some useful information for knowledge engineers, we assessed the importance of parameters in two domains: Depots and Tetris. They have a very different structure, and domain configuration substantially improved Yahsp and Probe, respectively (compare Figure 1 above).

We used fANOVA, a general tool for assessing parameter importance [Hutter *et al.*, 2014a] after algorithm configuration, to find that different parameters are indeed important for these two domains. Although we believed that the order of operators may have the strongest impact on performance, the experimental analysis demonstrated that this is not always true. In the two domain models, the order of operators had different impact. In Depots, which has a strong *directionality* (some actions are very likely to be executed in initial steps and not in final steps, and vice-versa), ordering operators according to this directionality clearly improves planning performance. In particular, our analysis highlighted that two operators have a very significant impact on performance: `load` and `drop`. Figure 2 shows a clear pattern for Yahsp's performance as a function of two parameters in the domain model: in order to achieve a low penalized average runtime, operator

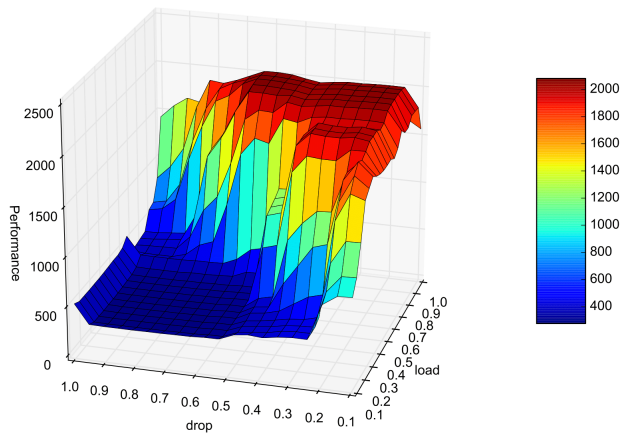


Figure 2: The average PAR10 performance of Yahsp in the Depots domain, as a function of the values of the parameters corresponding to the `load` and `drop` operators. The lower the PAR10 value, the better the performance. Best performance is achieved when the `load` operator is listed early in the model (low parameter value), and the `drop` operator is listed late (high parameter value).

`load` (which is mostly used in early steps for loading crates on trucks) should be listed early in the model description and operator `drop` should be listed late.

In the Tetris domain, where all the operators have potentially the same probability to be executed in any part of the plan, a good operators' order follows the frequency of action usage in the plan: most used first. Clearly, the frequency of action usage depends, to some extent, on the planner. Nevertheless, in domains such as Tetris, it is likely that the operators' ordering has a less significant impact than in domains with a strong directionality. In line with this explanation, the performance improvements for Tetris arose not by a single factor but as gains due to many of the configurable elements.

According to our observations in both Depots and Tetris domains, preconditions of operators which are most likely to be unsatisfied should be listed earlier. On the other hand, the impact of ordering of post-condition predicates is not very clear; our intuition is that the main effect for which an action is usually executed should be presented earlier. Finally, the configured order of predicate declarations tends to follow their frequency as operators' preconditions.

Since the results in Table 2 show that LPG tends to benefit most from domain configuration, we studied its performance in some more detail. In particular, we focused on the Tetris domain, where its percentage of solved instance was increased most. For gaining some insights, we qualitatively compared the Tetris domain model configured for LPG, with the models configured for other planners and with the aforementioned "bad" configuration. In terms of predicates, we notice that the `connected` predicate has a strong impact: it is the static predicate that describes connections between cells. Although static predicates are commonly used for describing aspects of the world that do not change, and are included also in some of the other domains considered in our empirical analysis, the `connected` used in Tetris is peculiar. Every operator incorporates at least two preconditions that are based on this predicate. LPG performance is improved when `connected` is the last predicate to be defined, but it is listed early as preconditions. In terms of operators, those which are rarely used by LPG should be the listed last.

4 Conclusion and Future Work

In this paper we proposed an approach for automatically configuring domain models. We considered PDDL domain models, configuring the order of predicate declarations, the order of operators, and predicates within pre and post-conditions. We investigated how configuring domain models affects the performance of domain-independent planners.

The analysis we performed in this work: (i) demonstrates that domain model configuration has a statistically significant impact on planners' performance; (ii) provides useful information to help engineer more efficient planning domain models; and (iii) gives insights into how to view competitions results; in particular, we showed that IPC results are influenced, in terms of the final ranking of participants, by the domain model configuration that is selected.

Domain model configuration opens up many possibilities for future work. Firstly, we plan to evaluate how it affects the quality of plans, particularly for models that include action costs. Moreover, we are interested in exploring the configuration of models encoded in languages different from PDDL, such as SAS+ and SAT. Such an analysis can also shed some light on the different impact that domain model configurations have on planning systems exploiting different techniques. Along similar lines, we also plan to study how domain configuration interacts with the configuration of planner parameters. We further plan to combine it with portfolio approaches to automatically configure a portfolio of domain models, for one or more planners. Finally, we would also like to explore whether problem models can be configured online, which could potentially lead to further performance improvements and to useful insights on the obscure task of describing and modelling problems.

Acknowledgments

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work. F. Hutter acknowledges support by the German Research Foundation (DFG) as part of Emmy Noether grant HU 1900/2-1. The research was partly funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1).

References

[Ansótegui *et al.*, 2009] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the Principles and Practice of Constraint Programming (CP)*, pages 142–157, 2009.

[Areces *et al.*, 2014] C. Areces, F. Bustos, M. Dominguez, and J. Hoffmann. Optimizing planning domains by automatic action schema splitting. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, (ICAPS)*, 2014.

[Botea *et al.*, 2005] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[Breiman, 2001] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[Chrpa and McCluskey, 2012] L. Chrpa and T. L. McCluskey. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 240–245, 2012.

[Fawcett *et al.*, 2011] C. Fawcett, M. Helmert, H.H. Hoos, E. Karpas, G. Röger, and J. Seipp. FD-Autotune: Domain-specific configuration using fast-downward. In *Workshop on Planning and Learning (PAL)*, 2011.

[Gerevini *et al.*, 2003] A. E. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research (JAIR)*, 20:239–290, 2003.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice*. Morgan Kaufmann Publishers, 2004.

[Howe and Dahlman, 2002] A. E. Howe and E. Dahlman. A critical assessment of benchmark comparison in planning. volume 17, pages 1–33, 2002.

[Hutter *et al.*, 2009] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 2009.

[Hutter *et al.*, 2011] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th Learning and Intelligent OptimizatioN Conference (LION)*, pages 507–523, 2011.

[Hutter *et al.*, 2014a] F. Hutter, H. H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of The 31st International Conference on Machine Learning*, pages 754–762, 2014.

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

[Katz and Hoffmann, 2014] M. Katz and J. Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.

[Lipovetzky *et al.*, 2014] N. Lipovetzky, M. Ramirez, C. Muise, and H. Geffner. Width and inference based planners: SIW, BFS(f), and PROBE. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.

[Newton *et al.*, 2007] M. A. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*, pages 256–263, 2007.

[Rintanen, 2014] J. Rintanen. Madagascar: Scalable planning with SAT. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.

[Valenzano *et al.*, 2014] Richard Valenzano, J Schaeffer, N Sturtevant, and Fan Xie. A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS-2014)*, pages 375–379, 2014.

[Vallati *et al.*, 2013] M. Vallati, C. Fawcett, A. E. Gerevini, H. H. Hoos, and A. Saetti. Automatic generation of efficient domain-optimized planners from generic parametrized planners. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS*, 2013.

[Vidal, 2014] V. Vidal. YAHSP3 and YAHSP3-MT in the 8th international planning competition. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.

[Wilcoxon, 1945] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[Xie *et al.*, 2014] F. Xie, M. Müller, and R. Holte. Jasper: the art of exploration in greedy best first search. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.

[Yuan *et al.*, 2010] Z. Yuan, T. Stützle, and M. Birattari. MADS/F-Race: Mesh adaptive direct search meets f-race. In *Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, pages 41–50, 2010.