A General Approach for Configuring PDDL Problem Models

Mauro Vallati
School of Computing and Engineering
University of Huddersfield, UK

Ivan Serina
Dept. of Information Engineering
University of Brescia, Italy

Abstract

The development of a large number of domain-independent planners is leading to the use of planning engines in 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. However, this separation supports the use of reformulation and configuration techniques, which transform the model representation in order to improve the planner's performance.

In this paper, we investigate how the performance of domain-independent planners can be improved by problem model configuration. We introduce a fully automated method for this configuration task, that considers problem-specific aspects extracted by exploiting a problem- and domain-independent representation of the instance. Our extensive experimental analysis shows that this reformulation technique can have a significant impact on planners' performance.

Introduction

Domain-independent planning engines can now be exploited as embedded components within a larger framework. Since they accept the domain and problem description in a standardized interface language (often PDDL) and return plans using the same syntax, they can be interchanged without modifying the rest of the system.

This modular approach also supports the use of reformulation and configuration techniques which can automatically re-formulate or re-represent the domain model and/or problem description in order to increase the efficiency of a planner and increase the number of solved problems. 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 model is transformed to a more efficient form that is fed into the planner.

Among the other reformulation approaches, ordering in planning models has also shown to have considerable impact on the planning process (Howe and Dahlman 2002). Recently, Vallati et al. (2015) developed an approach that automatically configures domain models by re-ordering their elements. The underlying idea is that the way in which operators are encoded in PDDL carries some knowledge about the expected use of the corresponding actions. Such knowledge can therefore be exploited for providing a more suitable ordering of operators, for improving the performance of the planner that will be used for solving the given problem.

In this context, we propose an approach for performing the automatic configuration of PDDL *problem models*, in terms of ordering of propositional facts in the initial and goal states. Notably, this reformulation task is very different from domain model configuration (Vallati et al. 2015), as it requires to reason with both domain- and problem-related knowledge in order to be efficient and effective. Intuitively, it is not worth to design a problem-specific configuration technique, as performance improvement will be outweighed by the configuration overhead. Instead, a domain-specific configuration must be identified, and applied on problem models on the basis of the objects and predicates involved. In order to deal with the mentioned configuration challenges, we rely on the Planning Encoding Graph representation (Serina 2010) of problems, that provides a domain- and problem-independent, yet extremely informative, ground for performing the configuration. Through comprehensive experiments using five planners on seven planning domains, we demonstrate that problem model configuration has a remarkable impact on performance, and we show how it relates with the existing domain model configuration. Next to making a significant contribution to planning speed-up, this work provides some guidelines to encode problem models in order to maximise expected planners' performance.

Problem Model Configuration

In this section, we first describe the degrees of freedom in problem models. After that, we briefly introduce the Planning Encoding Graph, and we show how it has been exploited for the automated configuration of problem models.

Degrees of Freedom in Problem Models

The problem model describes, according to a corresponding domain model, an actual instance to solve. The main components of a problem models are the descriptions of the initial and goal states, that come under the form of lists of propositional facts. Currently, those lists are not ordered following a principled approach. Orders may derive from the way in which generators have been engineered, alphabetical order, or following the intuitions of knowledge engineers.

Here we focus on the following question: *given the facts of the initial and goal states of a problem model, in which order should they be listed so as to maximise the performance of a given domain-independent planner?*

As it is apparent, differently from the configuration of a domain model, the configuration of a problem model should be problem-specific, but should work for all the problems sharing the same domain model. In other words, the configuration process should be able to identify a domain-wise general configuration, that is then exploited to configure a problem model according to the specific problem facts and structure. This is because it would be extremely costly, and pointless, to design a configuration approach that needs to be tailored for every single problem model, as the time spent in learning the configuration would significantly overcome any performance boost.

A structure that is able to support the depicted configuration process, i.e. to combine together domain-specific aspects and problem-specific features is the Planning Encoding Graph (PEG) (Serina 2010). We used the PEG for extracting information to be used in the configuration process.

The Planning Encoding Graph

The underlying idea of the PEG is to provide a description of the "topology" of a planning problem without making assumptions on the domain structure or regarding the importance of specific problem features for the encoding.

The PEG of a planning problem Π is the union of the directed labelled graphs encoding the initial and goal facts. The *Initial Fact Encoding Graph* of a propositional initial fact $p = (\rho \quad o_1 \dots o_n)$ – where ρ is the predicate symbol of p and $o_1, \dots, o_n$ are the corresponding problem objects – is a directed graph where the first node is a predicate node I_ρ, which is connected with o_1 by an arc with label $\{I_\rho^{0,1}\}$. The object node o_i is labelled with the type of the object o_i and it is connected with all the remaining nodes o_j by an arc with label $\{I_\rho^{i,j}\}$. Differently from other data structures, the PEG is compact, extremely quick to generate and analyse since the time required for building it is directly proportional to the number of initial and goal facts of the problem considered, and yet it gives useful information about the structure of the problem as confirmed by its usefulness in case-based planning (Bonisoli et al. 2015; Vallati et al. 2016).

For example, considering the initial fact (`on C A`) of our running example of Figure 1, it determines an Initial Fact Encoding Graph that connects the I_{on} predicate node to the `C` node (the label of this arc is $\{I_{on}^{0,1}\}$), which is connected with the node `A` by an arc with label $\{I_{on}^{1,2}\}$. The object nodes `A` and `C` are labelled by the corresponding object type (`block`). Similarly, it is possible to define the *Goal Fact Encoding Graph* of a propositional goal p using the token G_ρ instead of token I_ρ.

Using this representation and the instantiable actions associated to a specific planning problem, we can extract a large amount of information that allows to define a set of features that characterize it. In particular, we can derive for each propositional fact p of our planning problem: the number of actions of which it is (*i*) precondition, (*ii*) additive effect, (*iii*) delete effect, the sum of the (*iv*) input and (*v*) output degree sequences of the object nodes of p considering only arcs derived by initial facts, the sum of the (*vi*) input and (*vii*) output degree sequences of the object nodes of p considering only arcs derived by goal facts, the (*viii*) degree sequence of the predicate node related to p derived by initial facts, the (*ix*) degree sequence of the predicate node related to p derived by goal facts, the total (*x*) number of arcs that connect two object nodes of p, the total (*xi*) number of arcs that connect a predicate node related to p to an object node of p, the (*xii*) total number of arcs of the PEG associated to p and finally (*xiii*) if p is/not an inertial fact[1].

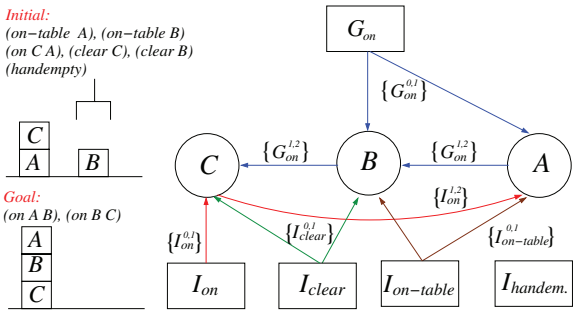

Figure 1: Planning Encoding Graph for a planning problem in the BlocksWorld domain.

Configuration of Problem Models

In this work we use the state-of-the-art SMAC (Hutter, Hoos, and Leyton-Brown 2011) configuration approach for identifying a domain-wise configuration of problem models that improves the runtime performance of a given domain-independent planner. Runtime performance are measured by considering the Penalised Average Runtime (PAR10). It is the average runtime where unsolved instances count as 10*cutoff time. PAR10 is a metric usually exploited in machine learning and algorithm configuration techniques, as it allows to consider coverage and runtime at the same time.

SMAC uses predictive models of performance (Hutter et al. 2014) to guide its search for good configurations. More precisely, it uses previously observed ⟨configuration, performance⟩ pairs $\langle c, f(c)\rangle$ and supervised machine learning (random forests (Breiman 2001)) to learn a function $\hat{f} : \mathcal{C} \rightarrow \mathbb{R}$ that predicts the performance of arbitrary parameter configurations.

There are different ways for encoding the degrees of freedom in problem models as parameters, mainly because orders are not natively supported by general configuration techniques. Here we generate 12 continuous parameters, which corresponds to the aforementioned features extracted from the PEG and from the instantiable actions of the given problem. These 12 parameters can be used for ordering either the initial state or the goal state, which leads to 24 pa-

[1]Inertial facts correspond to facts that are neither in the delete nor in the additive effects of any action of the planning problem; obviously their true value can influence heavily the planning process since they can be in the preconditions of the applicable actions.

rameters in total. Two additional categorical selectors are included: one which allows to decide if the order used for listing initial state facts must be the same used for ordering the goal state or not, and the other allows to decide if inertial facts have to be listed on top (bottom) of the list or not.

Each continuous parameter has associated a real value in the interval $[-1.0, +1.0]$ which represents (in absolute value) the weight given to the corresponding feature when exploited as an ordering criterion. Thus, the configuration space is $\Sigma = [-1.0, +1.0]^{24} \times 3 \times 2$, where 3 are the possible values of the parameter on the importance of inertial facts, and 2 are the possible values of the categorical parameter describing whether the goal facts should be listed according to the order used for the initial state ones or not. A configuration σ instantiates each of the 26 parameters, and can be used on any problem model of the considered domain. Given a problem model and a configuration σ, the corresponding problem configuration is obtained as follows. For each propositional fact in the initial state, an ordering score $o(p)$ is defined as:

$$o(p) = \sum_{c \in C} (value(p, c) \times weight(c)) + inertial(p) \quad (1)$$

where c is a continuous ordering criterion in the set C of the 24 available continuous parameters, $value(p, c)$ is the numerical value of the corresponding aspect for the propositional fact p, and $weight(c)$ is the weight assigned to the corresponding continuous parameter by the configuration technique. $inertial(p)$ is an offset that, according to the value of the corresponding parameter, can prioritise (deprioritise) inertial facts. It should be noted that $value(p, c)$ takes into account all the components of p in the PEG, i.e. the predicate and object(s) nodes. If the 26th parameter is set to force the order of goal facts to be the same of the order of initial state facts, then propositional facts in the goal state are ordered using the same $weight(c)$ values used for the initial facts. Otherwise, the $weight(c)$ values provided via the 12 goal-specific parameters are considered instead.

Propositional facts are then listed according to their $o(p)$, following a descending order. Ties are broken following the order in the "original" problem model. Notably, the weight assigned to an ordering criterion can be negative: this allows the configurator to "push" propositional facts with an high value of the corresponding aspect later in the list.

For instance, let us consider the initial state of the problem presented in Figure 1. Suppose that we are interested in listing propositional facts according to the total number of arcs they are involved in the PEG (feature *xii* in the previous section). This can be done by leaving all the parameters to the default value 0.0, but the one controlling feature *xii* to 1.0. Propositional facts in the initial state would then be listed as follows:

Original: `(on-table A), (on-table B), (on C A), (clear C), (clear B), (handempty)`

Configured: `(on C A), (on-table B), (on-table A), (clear B), (clear C), (handempty)`

Considering only feature *xii*, the fact `(on C A)` has a $o(p)$ score of $(3 + 4 + 4) \times 1.0 = 11$: `on` is involved in 3 arcs (two from the goal state, and one from the initial state) and `C` and `A` are involved in 4 arcs each; while considering the fact `(on-table A)`, it has a score of $(2 + 4) \times 1.0 = 6$.

Experimental Analysis

We selected 5 planners, based on their performance in the Agile track of IPC 2014 and/or the use of different planning approaches: Lama (Richter and Westphal 2010), Lpg (Gerevini, Saetti, and Serina 2003), Madagascar (Mp) (Rintanen 2014), Probe (Lipovetzky et al. 2014), and Yahsp3 (Vidal 2014). No portfolio-based planners were included, but by improving the best basic planners the performance of portfolios based on them can also be expected to improve.

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 some variety in terms of the number of objects involved in the problems and of predicates: Blocksworld (4 ops version), Depots, Matching-Bw, Parking, Rovers, Tetris, and ZenoTravel. For our experiments we created approximately 550 random instances per domain, and split them randomly into roughly 500 training and 50 test instances.[2]

Configuration of domain models was done using SMAC 2.08, and OAKplan 2.1[3] has been used for extracting PEGs. In order to automatically re-order a PDDL problem model according to the specified configuration, we developed a wrapper in Python. On the benchmarks considered in our experimental analysis, the re-ordering takes less than 0.5 CPU-time second.

Experiments were performed on a Intel Xeon E5-2620 2.00GHz. Each configuration run 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.

Table 1 compares the performance, in terms of PAR10, coverage, and IPC score, achieved by planners run using the original (O) and the automatically-configured (C) problem model. The original configuration is the one provided by generators, and we may therefore reasonably assume that planners have been developed and tested (implicitly "tuned") on it when dealing with the existing benchmarks. In fat, on a preliminary set of experiments, we observed that randomly picked configurations usually lead planners to worse performance than those achieved on the original. Also, the original configuration is the initial configuration considered by SMAC during the training process. In other words, it is the baseline used by SMAC during the configuration for evaluating the expected performance improvement. Providing SMAC with different initial configurations may lead to different final configurations, according to the quality of the initial models.

Results in Table 1 indicate that the configuration of the problem model has a significant impact on the planners' performance. Many of the differences (highlighted in bold

[2]Benchmarks and wrappers can be found at http://helios.hud.ac.uk/scommv/storage/VallatiSerina.zip

[3]http://eracle.ing.unibs.it/OAKplan/

Planner	PAR10 C	PAR10 O	Solved C	Solved O	IPC score C	IPC score O
BlocksWorld						
Lama	7.3	7.6	100.0	100.0	53.0	51.3
Lpg	**6.5**	8.7	100.0	100.0	51.8	46.2
Mp	555.3	556.3	82.0	82.0	40.7	39.6
Probe	**12.2**	15.8	100.0	100.0	54.8	49.7
Yahsp3	**6.6**	7.9	100.0	100.0	50.0	42.1
Depots						
Lama	2241.2	2303.3	26.0	24.0	12.7	11.1
Lpg	**1.4**	13.7	100.0	100.0	48.4	35.6
Mp	**1.6**	62.8	100.0	98.0	50.0	40.9
Probe	**5.4**	7.9	100.0	100.0	50.0	43.8
Yahsp3	427.4	481.5	86.0	84.0	37.9	37.9
Matching-Bw						
Lama	362.6	422.3	88.0	86.0	41.9	41.4
Lpg	**1446.3**	1685.0	52.0	44.0	22.2	20.6
Mp	3000.0	3000.0	0.0	0.0	0.0	0.0
Probe	1397.9	1456.7	54.0	52.0	26.3	23.0
Yahsp3	**184.1**	307.9	94.0	90.0	43.4	33.9
Parking						
Lama	783.5	895.2	75.0	71.4	39.6	37.0
Lpg	2683.8	2845.4	10.7	5.4	5.8	2.3
Mp	**1241.7**	1670.1	58.9	44.6	32.9	20.2
Probe	1156.3	1263.9	62.5	58.9	32.1	29.0
Yahsp3	1840.2	1844.5	39.3	39.3	21.4	20.5
Rovers						
Lama	69.0	67.8	100.0	100.0	59.3	59.7
Lpg	19.8	19.4	100.0	100.0	57.0	57.6
Mp	3000.0	3000.0	0.0	0.0	0.0	0.0
Probe	3000.0	3000.0	0.0	0.0	0.0	0.0
Yahsp3	5.4	5.3	100.0	100.0	59.3	59.3
Tetris						
Lama	1249.3	1304.3	59.3	57.4	31.2	29.0
Lpg	2562.0	2670.1	14.8	11.1	7.9	4.9
Mp	2234.9	2234.9	25.9	25.9	14.0	13.7
Probe	810.0	866.7	74.1	72.2	38.7	35.6
Yahsp3	2670.8	2668.7	11.1	11.1	5.2	5.3
ZenoTravel						
Lama	**54.3**	59.2	100.0	100.0	47.7	46.4
Lpg	36.2	35.7	100.0	100.0	46.4	47.4
Mp	8.7	8.8	100.0	100.0	48.6	48.5
Probe	61.9	121.4	100.0	98.0	48.6	47.4
Yahsp3	1.8	1.8	100.0	100.0	47.3	47.1

Table 1: Results in terms of PAR10, percentage of solved problems and IPC score, of planners running on original problem models (O) and configured models (C). Bold indicates statistically different PAR10 performance. IPC score was calculated separately for each planner, considering only the performance of its two domain model configurations. Therefore, IPC scores cannot be compared across planners.

in the table) are statistically significant, according to the Wilcoxon signed rank test ($p = 0.05$). Even in cases where differences are not statistically significant, improvements in terms of coverage can usually be observed. Remarkably, it can be noted that in two domains (BlocksWorld and Depots) the configuration of the problem model changes the best performing planner (according to PAR10 score).

Our intuition about the observed substantial performance variations is that different ordering of the propositional facts in the initial state can influence the internal representation of search states. Propositional facts to be satisfied in order to achieve the goals, as well as landmarks, may then be considered in a different order, resulting in a different exploration of the search space. Intuitively, the way in which goal facts are ordered can affect the order in which goals are tackled.

Table 1 also seems to indicate that, for some of the considered planners, the configuration of problem models does not lead to conspicuous performance improvements in Rovers and ZenoTravel. According to our observations, this is due to the fact that in both domains, very few different predicates are considered in the PDDL model, and initial states are described as long sequences of instances of the same predicate(s). This is the case of the `visible`, and `can_traverse` predicates in Rovers, and `at` predicate in ZenoTravel. Under these circumstances, also all the objects of the problem are very similar to each other, thus few knowledge can be extracted by analysing the structure of the problem.

In the light of the work done by Vallati et al. (2015), it is important to assess how the automated configurations of domain and problem models relate. We selected two planners due to their performance in the previous experiments, Lpg and Probe, and run them on all of our benchmarks with three different combinations of models: original domain and configured problem models, configured domain and original problem models, and on both configured models. As configured domain models, we used those generated by Vallati et al. (2015) in their work. Results of this comparison are presented in Table 2. Lpg seems to be able to exploit synergies in the configured models, and achieves best performance when both the models are configured. On the contrary, Probe delivers best performance when only the problem model is configured. Considered domains are usually similarly affected, we did not notice significant outliers. While observed behaviours may be due to the fact that the models (domain and problem) are separately configured, i.e. domain (problem) is configured while keeping the other model fixed, results seem to suggest that planners react differently to the configuration process; therefore, it is important to select the "right" configuration to maximise planner's performance.

In order to understand the influence of the considered configurable aspects of problem models, thus providing some guidelines for the effective formulation of models, we used the fAnova tool (Hutter, Hoos, and Leyton-Brown 2014) to assess parameters' importance in two domains, BlocksWorld and Depots. In these domains, general performance improvements, when using the configured problem models, can be observed among all the considered planners. First, we noticed that in order to maximise performance, initial and goal facts' ordering should be aligned. Second, propositional facts that are frequently in actions' preconditions should be listed earlier, followed by propositional facts that are often positive effects of grounded actions. This is possibly because their ordering can direct the search into promising areas of the search space. Finally, the order of propositional facts should follow their degree of connectivity in the PEG: propositional facts whose elements are

	Lpg		
	PAR10	Solved	IPC score
oD+cP	974.2	67.7	221.2
cD+oP	1008.5	66.6	212.9
cD+cP	**950.6**	**68.5**	**239.0**
	Probe		
oD+cP	**966.8**	**68.5**	**245.8**
cD+oP	973.8	68.2	238.1
cD+cP	976.1	68.2	237.2

Table 2: Results in terms of PAR10, percentage of solved problems and IPC score, of planners running on original domain model and configured problem models (oD+cP), configured domain model and original problem models (cD+oP), and configured models only (cD+cP). Bold indicates best performance.

highly connected are likely to be important, and should be listed earlier.

Conclusion

In this paper we proposed a general, i.e. domain- and problem-independent, approach for automatically configuring PDDL problem models by exploiting information extracted by the PEG. We considered as configurable the ordering of propositional facts as goals, and in the initial state.

The performed analysis, aimed at investigating how the configuration of problem models affects the performance of domain-independent planners: (i) demonstrates that configuration has a statistically significant impact on planners' performance; (ii) provides useful information to engineer more efficient models; and (iii) gives insights into possible synergies between domain and problem models configuration.

Future work includes the analysis of the impact of configuration on plans' quality, and the configuration of models encoded using different languages. We plan to test our techniques on planners that serialize goals (Lipovetzky and Geffner 2012) since in these planners the effect of goals' ordering could be significant. Finally, we would like to investigate the concurrent configuration of PDDL knowledge models and planning engines.

References

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

Bonisoli, A.; Gerevini, A. E.; Saetti, A.; and Serina, I. 2015. Effective plan retrieval in case-based planning for metric-temporal problems. *J. Exp. Theor. Artif. Intell.* 27(5):603–647.

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

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

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

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)* 20:239 – 290.

Howe, A. E., and Dahlman, E. 2002. A critical assessment of benchmark comparison in planning. *J. Artif. Intell. Res. (JAIR)* 17:1–33.

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

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

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

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 540–545.

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and inference based planners: Siw, bfs(f), and probe. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

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

Richter, S., and Westphal, M. 2010. The lama planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)* 39:127–177.

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

Serina, I. 2010. Kernel functions for case-based planning. *Artif. Intell.* 174(16-17):1369–1406.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the effective configuration of planning domain models. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

Vallati, M.; Serina, I.; Saetti, A.; and Gerevini, A. E. 2016. Identifying and exploiting features for effective plan retrieval in case-based planning. *Fundam. Inform.* 149(1-2):209–240.

Vidal, V. 2014. Yahsp3 and yahsp3-mt in the 8th international planning competition. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.