HW[ ]: A Parametric System for Planning by Abstraction

Authors: **G. Armano, G. Cherchi, and E. Vargiu**

Abstract

In this article, a system devised and implemented to exploit abstraction techniques for solving complex problems is presented. The system is able to automatically generate a hierarchical planner starting from a generic planner *P*, provided that compliance with the STRIPS subset of PDDL 1.2 standard is ensured. The resulting hierarchical planner delegates to *P* the actual search at any required level of abstraction – including ground level. Some experiments have been conducted between three selected planners and their hierarchical counterparts. Results show that the proposed approach may speed up the search, especially for problems of medium / high comp lexity.

Introduction

There is experimental evidence that humans repeatedly use abstraction to solve everyday problems [21], thus justifying research in the field of automated planning by abstraction. Nevertheless, when applied to simple problems, the resort to abstraction may actually represent a waste of computational resources –due to the need to go back and forth across abstract spaces during the search. Conversely, abstraction can be a useful technique for simplifying the search when tackling complex problems. In fact, under certain assumptions, abstraction can even reduce the size of the search space from exponential to linear in the size of the solution [13].

Typically, abstraction techniques require the original search space to be mapped into abstract spaces in which irrelevant details are disregarded at different levels of granularity. Let us briefly recall some relevant techniques proposed in the literature: (i) action-based, (ii) state-based, (iii) Hierarchical Task Networks, and (iv) case-based.

The first combines a group of actions to form macro-operators [15]. The second exploits representations of the world given at a lower level of detail; its most significant forms rely on (a) relaxed models, obtained by dropping operators' applicability conditions [20], and on (b) reduced models [14], obtained by completely removing certain conditions from the problem space. Both models, while preserving the provability of plans that hold at the ground level, perform a "weakening" of the original problem space, thus suffering from the drawback of introducing "false" (i.e., not refinable) solutions at the abstract levels [11]. In the third (e.g., [9]), problem and operators are organized into a set of tasks. A high-level task can be reduced to a set of ordered lower-level tasks, and a task can be reduced in several ways. Reductions allow specifying how to obtain a detailed plan from an abstract one. In the fourth [5], abstract planning cases are automatically learned from given concrete cases, as done in the PARIS system, although the user must provide explicit refinement rules between adjacent levels in the hierarchy.

In the past, abstraction has been widely used to reduce search in a variety of planning systems (e.g., GPS [19], ABSTRIPS [20], ABTWEAK [22], PABLO [8], and PRODIGY [7]). However such planning systems have been mostly focusing on a specific algorithm, instead of taking into account the possibility of generalizing the approach. In one sense, we followed the latter direction when devising and implementing a system able to embed a generic planner as a black-box. The rationale of this choice lies in the possibility of directly assessing the impact of abstraction techniques –by comparing the performances of existing planners with their hierarchical counterparts.

The remainder of this article is organized as follows. First, the overall system architecture is illustrated. Then, the planning algorithm is presented and experiments are described. The results are then discussed and finally the conclusions are drawn.

System Architecture

The proposed system, devised and implemented to perform planning by abstraction, is able to embed any domain-independent planner –provided that compliance with the STRIPS subset of PDDL 1.2 standard [18] is ensured. The system has been called *HW[]*, standing for (parametric) Hierarchical Wrapper. It is worth noting that square brackets are part of the name, indicating the ability to embed an external planner. Thus, the notation *HW[P]* denotes an instance of *HW[]* that exploits the planning capabilities of *P*. The embedded planner is exploited at any level of the hierarchy, each level being characterized by its own definitions. [1]

A suitable decoupling between levels is guaranteed by using domain-specific rules that describe how to translate the representation language from one level to its superior and vice-versa. To express this type of translation rules, an extension to standard PDDL has been adopted, explicitly defined to support abstraction hierarchies [1].

Figure 1 sketches the architecture of the system, focusing on its main components, i.e., an engine and the embedded planner. The former controls the communication between adjacent levels, whereas the latter performs planning at any given level of abstraction.

[1] A straightforward extension would consist of allowing the system to embed different planners, one for each level of abstraction.

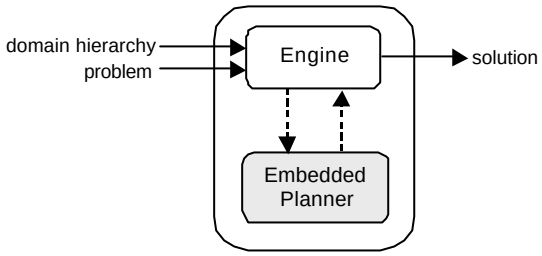

Figure 1 The architecture of the system.

We are assuming that each layer of the hierarchy contains its own set of operators, types and predicates for coping with the given problem at different levels of granularity. To facilitate the setting of abstract spaces, as an alternative to the hand-coded approach used in [2], a novel semi-automatic technique for generating abstraction hierarchies starting from ground-level domain descriptions has been adopted [3].

Although the system supports a multiple-level hierarchy, for the sake of simplicity, in the following we assume that only one abstract level exists, giving rise to a two-level (i.e., *ground* and *abstract*) hierarchical description.

The Planning Algorithm

Being *P* the embedded planner, *HW[P]* takes a problem and the corresponding domain hierarchy. The problem is described at the ground-level following the standard "*define problem*" PDDL notation. The hierarchy, expressed according to the extended "*define hierarchy*" notation proposed in [1], contains a structured description of the domains at any level of abstraction, including a set of rules to be used when mapping ground into abstract states and vice-versa. In fact, to perform planning at different levels of abstraction, the engine of *HW[]* must operate bi-directional translations (upwards and downwards) to permit communication between adjacent levels.

To find a solution of a given problem, first the engine of *HW[P]* translates the *init* and *goal* sections from the ground to the abstract level. The embedded planner *P* is then invoked to search for an abstract solution. Subsequently, each abstract operator is refined by repeatedly invoking *P*. The refinement of an abstract operator is performed by activating *P*, at the ground level, on the goal obtained by translating downward its effects. Note that the initial state of each refinement depends on previous refinements; thus, dictating that refinements must be performed according to the order specified by the abstract plan. To avoid accidental deletion of subgoals already attained during previous refinements, they are added to the list of subgoals that results from translating downward the effects of the current abstract operator to be refined.

When the attempt to refine the current abstract solution fails, *P* is invoked to find the next abstract solution, [2] unless the number of abstract solutions found so far exceeds a given threshold (*m*). If no abstract solution could be successfully refined, to ensure the completeness of the algorithm, an overall search is performed at the ground level. The whole process ends when a ground solution is found or the overall search fails.

Experimental Results

The current prototype of the system has been implemented in C++. Experiments have been performed with three planners: GRAPHPLAN [6], BLACKBOX [12], and LPG [10]. In the following, *GP*, *BB*, and *LPG* shall be used to denote the GRAPHPLAN, BLACKBOX, and LPG planners, as used in a stand-alone configuration, whereas *HW[GP]*, *HW[BB]*, and *HW[LPG]* shall be used to denote their hierarchical counterparts.

Let us point out that only the relative performance between the hierarchical and stand-alone versions of each planner has been directly compared. For instance, results obtained with *BB* and *HW[BB]* have been considered separately from those obtained with *GP* and *HW[GP]*. In fact, a direct comparison between *HW[BB]* and *HW[GP]* is not relevant in this context.

To illustrate the ability of abstraction to improve the search, in this section we present some tests on five domains taken from AIPS planning competitions (2002, 2000 and 1998) [17] [4] [16]: *elevator*, *logistics*, *blocks-world*, *zeno-travel* and *gripper*. Experiments were conducted on a machine powered by an Intel Celeron CPU working at 1200 Mhz with 256Mb of RAM. A time bound of 1000 CPU seconds has also been adopted, and the threshold (*m*) used to limit the search for abstract solutions has been set to 1 for each planner.

Table 1. Performance comparisons between stand-alone planners together with their hierarchical counterparts (i.e., *GP* vs. *HW[GP]*, *BB* vs. *HW[BB]*, and *LPG* vs. *HW[LPG]*).

#	GP	HW [GP]	BB	HW [BB]	LPG	HW [LPG]
elevator						
1-4	0.01	0.06	0.1	0.33	0.01	0.11
3-1	0.23	0.36	1.34	1.20	0.02	0.15
4-1	1.96	0.83	1.03	1.74	0.02	0.16
4-4	10.11	0.84	311.5	1.79	0.02	0.16
5-1	364.7	2.03	180.8	2.54	0.02	0.18
7-2	--	12.04	--	3.89	0.03	0.29
logistics						
4-2	0.68	1.22	0.27	0.46	17.93	--
5-2	0.08	0.16	0.15	0.46	0.02	--
7-0	--	10.93	4.49	2.17	2.12	--
8-1	--	16.26	2.90	3.02	1.55	--
10-0	--	43.43	8.27	3.76	2.17	--
15-0	--	203.4	10.91	6.33	0.15	--
blocks-world						
4-0	0.34	0.32	0.16	0.67	0.02	0.08
6-0	3.04	1.82	0.26	1.68	0.05	0.23
8-0	31.61	11.13	0.92	2.46	0.36	0.31
10-0	--	--	6.82	5.00	0.62	0.67
11-0	--	--	16.23	4.25	4.23	0.83

[2] Due to the limitations of most of the existing planners, the process of incrementally querying for another solution may be simulated by preliminarily querying for *m* abstract solutions, to be released incrementally on demand.

14-0	--	--	--	9.84	5.00	1.91
15-0	--	--	--	--	7.49	2.07
17-0	--	--	--	--	33.93	3.49
20-0	--	--	--	--	66,78	7.88
22-0	--	--	--	--	183.16	12.21
25-0	--	--	--	--	668.98	24.94
zeno-travel						
1	0.02	0.52	0.22	0.36	0.02	0.03
8	--	42.55	0.94	2.36	0.14	0.49
9	--	--	0.34	3.37	0.13	1.08
11	--	--	11.20	2.78	0.16	1.06
13	--	--	62.99	20.52	0.42	2.47
14	--	--	--	20.04	3.90	21.93
gripper						
2	4.72	0.56	0.42	0.63	0.02	0.07
3	7.91	1.73	5.22	1.20	0.02	0.12
4	18.32	2.63	268.7	1.55	0.02	0.14
5	57.21	4.38	421.1	1.54	0.03	0.15
6	--	7.97	586.4	2.26	0.03	0.17
9	--	24.29	--	3.63	0.05	0.36

All domains have been structured according to a ground and an abstract level, the latter having been generated following the approach described in [3].

For each domain, several tests have been performed, characterized by increasing complexity. Table 1 compares the CPU time of each planner over the set of problems taken from the AIPS planning competitions. Dashes show problem instances that could not be solved by the corresponding system within the adopted time-bound.

Experiments performed on *GP* and *HW[GP]* show that the latter outperforms the former on problems of a certain length. In particular, for the *elevator* domain (see Figure 2), CPU time increases very rapidly when trying to solve problems of increasing length with *GP*, whereas *HW[GP]* keeps solving problems with greater regularity (although the relation between number of steps and CPU time still remains exponential). In the *logistics* domain

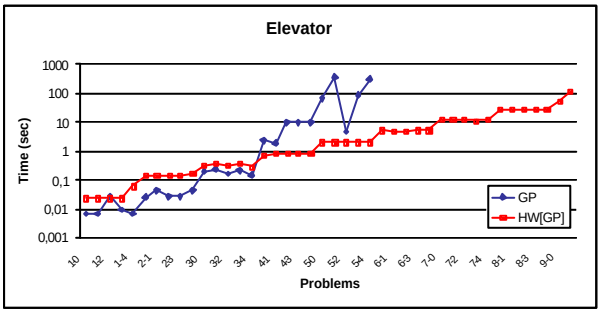

Figure 2 CPU time comparisons between *GP* and *HW[GP]* in the *elevator* domain.

GP is unable to solve some problems within the imposed time limits when a given length is exceeded; on the other hand, *HW[GP]* succeeds in solving problems of increasing length without encountering any such difficulties. Tests performed on the *blocks-world* domain reveal a similar trend for *GP* and *HW[GP]*, with the latter performing slightly better than the former. Unfortunately, neither *GP* nor *HW[GP]* are able to successfully tackle most of the *zeno-travel* domain problems within the adopted time bound. For the *gripper* domain, *HW[GP]* clearly outperforms its non-hierarchical counterpart (see Figure 3).

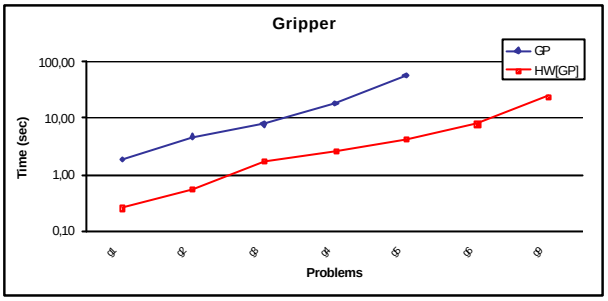

Figure 3 CPU time comparisons between *GP* and *HW[GP]*, in the *gripper* domain.

Experiments performed on *BB* and *HW[BB]* show that, in general, the former performs better than the latter on small problems, whereas the latter outperforms the former on more complex problems. In particular, for the *elevator* domain similar considerations can be made for *GP* and *HW[GP]*. In the *logistics* and *blocks-world*

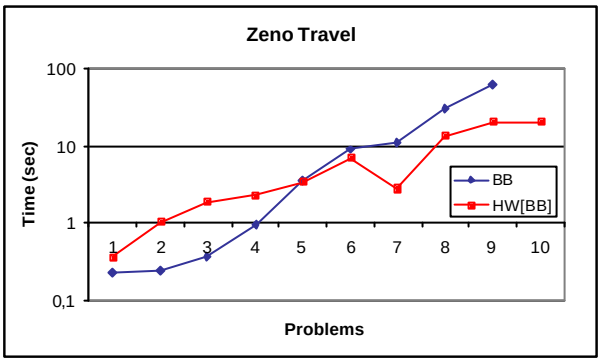

Figure 4 CPU time comparisons between *BB* and *HW[BB]* in the *zeno-travel* domain.

domains *BB* performs slightly better than *HW[BB]* on simple problems, whereas *HW[BB]* outperforms *BB* on problems of medium complexity. In the *zeno-travel* domain, a certain improvement of *HW[BB]* over *BB* can be observed (see Figure 4). In the *gripper* domain *HW[BB]* is clearly more efficient than *BB* (see Figure 5). Experiments performed on *LPG* and *HW[LPG]* show that the complexity of the selected problems is not adequate to stress *LPG*, which is able to solve any of them in a very short amount of time; thus doing away with the need to resort to *HW[LPG]*. This behavior can be observed in all the selected domains but the *blocks-world* domain. In this case, *HW[LPG]* clearly outperforms *LPG* on medium-size problems (see Figure 6).

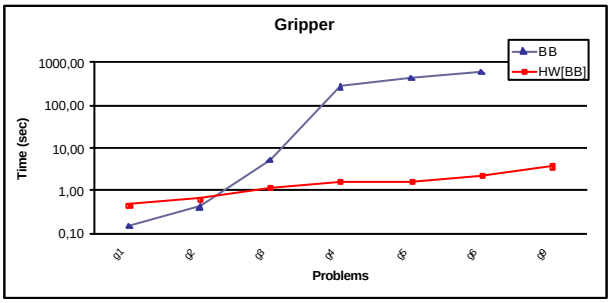

Figure 5 CPU time comparisons between *BB* and *HW[BB]* in the *gripper* domain.

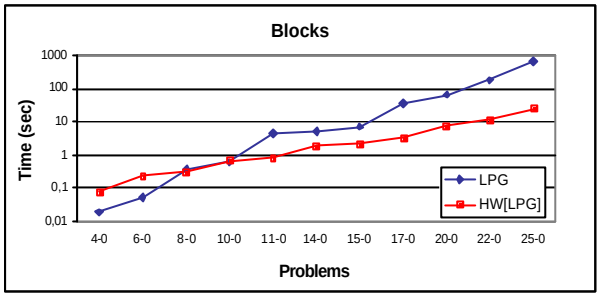

Figure 6 CPU time comparisons between *LPG* and *HW[LPG]* in the *blocks-world* domain.

Conclusions

In this article, the *HW[]* system has been described, devised to allow a generic PDDL-compliant planner to exploit abstraction techniques for solving complex problems.
Some tests have been performed on five domains taken from the AIPS 2002, 2000, and 1998 planning competitions. Three different planners have been embedded into the system, and comparisons have been made between each planner and its hierarchical counterpart. Experimental results highlight that abstraction is useful for classical planners, such as *GP* and *BB*. On the contrary, the usefulness of resorting to planning by abstraction for the latest-generation planner *LPG* emerges only in the *blocks-world* domain.

Bibliography

[1] G. Armano, G. Cherchi, and E. Vargiu. An Extension to PDDL for Hierarchical Planning. *Workshop on PDDL (ICAPS'03)*, Trento, Italy, June 2003, to appear.

[2] G. Armano, G. Cherchi, and E. Vargiu. Experimenting the Performance of Abstraction Mechanisms through a Parametric Hierarchical Planner. *Proceedings of IASTED International Conference on Artificial Intelligence and Applications (AIA'2003)*, Innsbruck, Austria, February 2003.

[3] G. Armano, G. Cherchi, and Eloisa Vargiu. A Parametric Hierarchical Planner for Experimenting Abstraction Techniques. *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, August 2003, to appear.

[4] F. Bacchus. *Results of the AIPS 2000 planning competition*, 2000 (http://www.cs.toronto.edu/aips2000).

[5] R. Bergmann and W. Wilke. Building and Refining Abstract Planning Cases by Change of Representation Language, *Journal of Artificial Intelligence Research (JAIR)*, 1995, 3:53-118.

[6] A. Blum and M. Furst. Fast Planning through Planning Graph Analysis, *Artificial Intelligence*, *90*(1-2), 1997, 279-298.

[7] J. Carbonell, C.A. Knoblock, and S. Minton. PRODIGY: An integrated architecture for planning and learning. In D. Paul Benjamin (ed.) *Change of Representation and Inductive Bias*. Kluwer Academic Publisher, 1990, 125-146.

[8] J. Christensen. Automatic Abstraction in Planning. PhD thesis, Department of Computer Science, Standford University, 1991.

[9] K. Erol, J. Hendler, and Dana S. Nau. HTN Planning: Complexity and Expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, 1994, 1123-1128, AAAI Press / MIT Press, Seattle, WA.

[10] A. Gerevini and I. Serina, LPG: A Planner Based on Local Search for Planning Graphs. In *Proc. of the 6th Int. Conference on AI Planning and Scheduling (AIPS'02)*, AAAI Press, 2002.

[11] F. Giunchiglia and T. Walsh, A theory of Abstraction, *Technical Report 9001-14*, IRST, Trento (Italy), 1990.

[12] H. Kautz and B. Selman, BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving, *Working notes of the Workshop on Planning as Combinatorial Search, AIPS-98*, Pittsburg, PA, 1998, 58-60.

[13] C.A. Knoblock, Search Reduction in Hierarchical Problem Solving. *Proceedings of the 9th National Conference on Artificial Intelligence*, 1991, 686-691, Anaheim, CA.

[14] C.A. Knoblock, Automatically Generating Abstractions or Planning, *Artificial Intelligence*, *68*(2), 1994, 243-302.

[15] R.E. Korf, Planning as Search: A Quantitative Approach, *Artificial Intelligence*, *33*(1), 1987, 65–88.

[16] D. Long, The AIPS-98 Planning Competition, *AI Magazine*, *21*(2), 2000, 13-33.

[17] D. Long, *Results of the AIPS 2002 planning competition*, 2002 (http://www.dur.ac.uk/d.p.long/competition.html).

[18] D. McDermott, and the AIPS-98 Planning Competition Committee, *PDDL – The Planning Domain Definition Language*, 1988.

[19] A. Newell, and H.A. Simon. *Human Problem Solving*. Prentice Hall, Englewood Cliffs, NJ, 1972.

[20] E.D. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artificial Intelligence*, *5*, 1974, 115–135.

[21] N.A. Stillings, C.H. Chase, M.H. Feinstein, J.L. Garfield, E.L. Rissland, and S.E.Weisler. *Cognitive Science: An Introduction*. MIT Press, Cambridge, Massachusetts, 1987.

[22] Q. Yang, and J. Tenenberg. Abtweak: Abstracting a Nonlinear, Least Commitment Planner. *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990, 204-209, Boston, MA.

Author Information

Giuliano Armano, Giancarlo Cherchi, and Eloisa Vargiu
DIEE, Department of Electrical and Electronic Engineering, University of Cagliari, Piazza d'Armi, I-09123 Cagliari, Italy,
{armano, cherchi, vargiu}@diee.unica.it