Experimenting The Performance Of Abstraction Mechanisms Through A Parametric . . .

CONFERENCE PAPER · JANUARY 2003
Source: DBLP

CITATIONS
6

READS
21

3 AUTHORS, INCLUDING:

Giuliano Armano
Università degli studi di Cagliari

157 PUBLICATIONS  591 CITATIONS

Available from: Giuliano Armano
Retrieved on: 15 March 2016
EXPERIMENTING THE PERFORMANCE OF ABSTRACTION MECHANISMS THROUGH A PARAMETRIC HIERARCHICAL PLANNER

Giuliano Armano, Giancarlo Cherchi, and Eloisa Vargiu

DIEE Department of Electrical and Electronic Engineering, University of Cagliari
Piazza d’Armi, I-09123 Cagliari
{armano, cherchi, vargiu}@diee.unica.it

ABSTRACT
In this paper, the parametric hierarchical planner \textit{HW}\(1\) is presented, devised to perform experiments on abstraction mechanisms. The system has been designed to embed any PDDL-compliant planner –to be considered as a “parameter” with respect to the overall planning system. By suitably switching between abstraction levels, \textit{HW}\(1\) exploits the embedded planner to search for solutions at any required level of abstraction. Some preliminary experiments are reported, performed on five domains taken from the AIPS 2002, 2000 and 1998 planning competitions. A step further in validating experimental results has been made by embedding two different planners. Comparative results, obtained by assessing the performances measured with and without abstraction, put into evidence that the hierarchical approach can significantly speed up the search.

KEY WORDS

1 INTRODUCTION
Building an ordered set of abstractions for controlling the search has proven to be an effective approach for dealing with the complexity of planning tasks. This technique requires the original search space to be mapped into new abstract spaces, in which irrelevant details are disregarded at different levels of granularity. Two main abstraction mechanisms have been studied in the literature: action- and state-based. The former combines a group of actions to form macro-operators [1], whereas the latter exploits representations of the world given at a lower level of detail. The most significant forms of state-based abstraction rely on (i) relaxed models, obtained by dropping operators’ applicability conditions [2], and on (ii) reduced models, obtained by completely removing certain conditions from the problem space [3]. 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 [4].

For a given abstraction hierarchy, both the downward and upward solution properties may hold (DSP and USP [5], respectively). The former property ensures that, for every abstract solution, at least one corresponding ground solution exists. Conversely, the latter property ensures that, for any ground solution, at least one corresponding abstract solution exists.

Analysis and experimental results have shown that hierarchical planning is most effective when the hierarchy satisfies the downward refinement property (DRP) [6], whereby every abstract solution can be refined to a ground-level solution without backtracking across abstraction levels. However the DRP is a strong requirement, difficult to meet in practice; that is why, a weakened form of DRP, i.e., near-DRP, has been investigated, which requires the ratio between “false” and “true” abstract solutions be reasonably low.

In this paper a parametric hierarchical planner is presented able to embed any domain-independent (non-hierarchical) planner provided that a compliance with the PDDL 1.2 standard [7] is ensured. Assuming that the near-DRP holds for the set of problems used as a benchmark, the embedded planner is exploited at any level of the hierarchy, each level being characterized by the definition of a corresponding domain. A suitable decoupling between levels is guaranteed by using domain-specific rules that describe how predicates must be translated from a level to its superior and vice-versa. Translations are currently hand-coded and are given in a PDDL-like format explicitly defined to support abstractions. In principle, abstractions might occur on types, operators, and predicates, although in the current implementation of the system only abstractions with the expressive power of reduced models are allowed.

The remainder of this paper is organized as follows: First, the overall system architecture of the system is illustrated. Then, all customizations required to perform “parametric” hierarchical planning are described, with particular emphasis on the domain-specific translations required to exploit the embedded, non-hierarchical, planner at any level of abstraction—including the ground one. Subsequently, experimental results are discussed. Finally conclusions are drawn and future work is briefly outlined.

2 SYSTEM ARCHITECTURE
Being aimed at investigating the impact of abstraction mechanisms on the search complexity, we implemented a parametric system able to embed an external planner – which is in charge of solving planning problems at any level of abstraction.
Since the system operates as a “wrapper” for an external planner able to process domain and problem definitions in a PDDL syntax, it has been called HW\[P\] –to stress its nature of parametric Hierarchical Wrapper. It is worth noting that brackets are part of the name, pointing to the ability of embedding an external planner. Being $P$ any such planner, the notation $HW\{P\}$ should be used to denote an instantiation of $HW\[P\]$ able to exploit the planning capabilities of $P$.

Figure 2 illustrates the architecture of the system, focusing on its main components, i.e., an engine and an external planner. In principle, any domain-independent and PDDL-compliant planner could be embedded within the $HW\{P\}$ system, provided that a suitable description is given to the engine that controls the activation of $P$ on how the communication between abstraction levels occurs. In fact, the engine must operate bi-directional translations, aimed at permitting the communication between adjacent levels.

![Figure 2. The overall architecture of the $HW\{P\}$ system.](image)

We are assuming that each layer of the hierarchy contains its own set of operators, types and predicates devoted to cope with the given problem at different levels of granularity. Notwithstanding this general assumption, the current release of $HW\{P\}$ is able to deal only with abstractions on predicates, in accordance with the constraints that characterize reduced models. In other words, an on/off forwarding rule can currently be enforced on a predicate that belongs to a level, depending on the will of making it available or not to its superior. For the sake of simplicity, we assume that only one abstract level exists, giving rise to a two-levels hierarchical description, characterized only by ground and abstract levels. Furthermore, let us assume that the selected external planner is able to generate only linear plans.

### 2.1 Extending PDDL for Dealing with Abstraction

Inputs to the system are the given problem and a structured description of the domain. The problem is described in accordance with the standard PDDL 1.2 syntax, and makes use of the define problem statement. The description of the domain must be specified –for each level– through suitable define domain statements. Figure 1 illustrates a sample of definitions taken from the elevator domain, used in the AIPS 2000 planning competition [8]. To add a level of abstraction to this domain, separate information has been provided, consisting of (a) a specification of the abstract level, and (b) a specification of how bi-directional communication occurs between adjacent levels. As described in Figure 3 and Figure 4, we decided to maintain the former in a standard PDDL format, whereas for the latter a novel PDDL-like format has been defined and adopted.

Concentrating on Figure 4, let us point out that the general form for denoting an upward translation rule consists of specifying, with a Lisp-like clause, a predicate that belongs to the upper level and the corresponding translation –on the left and right part of the clause, respectively. Of course, this specification can be used to perform both upward and downward translations, depending on the operation performed by the engine to get the embedded planner $P$ working properly at the selected level of abstraction. In particular, when the abstract-level goal must be set, an upward translation occurs aimed at mapping the ground problem space into the abstract one.

Conversely, when an abstract operator must be refined, a downward translation occurs. As an example of translation rule, let us make a remark about the clause:

\[
\text{(define (domain elevator-ground)}
\text{(:requirements :typing)}
\text{(:types passenger floor)}
\text{(:predicates}
\text{ (origin ?p - passenger ?f - floor)}
\text{ ...etc...}
\text{ (lift-at ?f - floor))}
\text{ (:action board}
\text{ :parameters (?f1 ?f2 - floor)}
\text{ :precondition (and (lift-at ?f1))}
\text{ :effect (and (boarded ?p))})
\text{ ...etc...}
\text{ (:action down}
\text{ :parameters (?p - passenger ?f1 - floor)}
\text{ :precondition (and (lift-at ?f1))}
\text{ :effect (and (lift-at ?f2)}}
\text{ (not (lift-at ?f1)))})
\]

**Figure 1.** An extract of the ground-level definitions for the elevator domain.
backtracking is enforced. The whole process ends when a solution is found or the overall search fails. Note that the refinement of an abstract operator is performed by activating $P$, at the ground level, on the goal obtained by translating downward its effects. It is also worth noting that, to avoid incidental deletion of subgoals already attained during previous refinements, they are actually added to the list of subgoals that results from translating downward the effects of the current abstract operator to be refined.

3 EXPERIMENTAL RESULTS

The current prototype of the system has been implemented in Common Lisp Object System (CLOS), for the sake of rapid prototyping.

We made experiments using two external planners: GRAPHPLAN [9] and BLACKBOX [10]. In the following, $GP$ and $BB$ denote the GRAPHPLAN and BLACKBOX algorithms, as used in a stand-alone configuration, whereas $HW[GP]$ and $HW[BB]$ denote their hierarchical counterparts. Let us point out that the language adopted for implementing the wrapper scarcely affects the relevance of experimental results. In fact, only the relative performance between the hierarchical and the stand-alone versions of each planner should be directly compared. In particular, results obtained with $BB$ and $HW[BB]$ must be considered separately from the ones obtained with $GP$ and $HW[GP]$. A direct comparison between $HW[BB]$ and $HW[GP]$ is not relevant in this context, also because the selected implementations of $BB$ and $GP$ have been coded in C++ and Lisp, respectively.

To assess how abstraction can improve the search, we performed some preliminary tests on five domains taken from AIPS planning competitions (2002, 2000 and 1998) [11][12][13]: 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.

3.1 BRIEF DESCRIPTION OF THE SELECTED DOMAINS

The elevator domain is characterized by a lift, devoted to carry passengers from a floor to another. A starting and a destination floor are specified for each passenger. The goal is to serve all passengers.

The logistics domain describes a set of cities, each containing several locations; some of which are airports. There are also trucks, used for in-city driving, and airplanes, used to fly between different cities. The goal is to move some packages to their (local or remote) destinations.

In the well-known blocks-world domain, stackable blocks need to be re-assembled on a table with unlimited space. A robot arm can be used for grabbing/ungrabbing a block onto/from another block, and for putting down or picking up a block.

2.2 THE PLANNING ALGORITHM

$HW[P]$ takes as inputs a ground-level problem, a description of the domain (split into ground and abstract level), and a set of rules –to be used while translating ground into abstract predicates and vice-versa. The significance of experimental results is ensured by the fact that comparisons can be made using the same planner, say $P$, with and without abstraction mechanisms.

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 all existing abstract solutions, ordered by length. Then, a solution is selected and each abstract operator is refined by repeatedly invoking $P$. When a refinement fails
3.2 TESTING THE SELECTED DOMAINS

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 chosen time-bound.

Elevator. Experiments performed on the elevator domain clearly show that—for GP and BB—the CPU time rises very rapidly while trying to solve problems of increasing length (see Figure 7, whose y-axis is expressed in logarithmic scale), whereas HW[GP] and HW[BB] keep solving problems with greater regularity (although the relation between number of steps and CPU time remains exponential).

Logistics. In the logistics domain, GP suffers from a phase transition problem (see Figure 5): it easily solves problems up to a certain length but it is unable to solve problems within the imposed time limits if a given threshold is exceeded. On the other hand, HW[GP] keeps solving problems of increasing length without putting into evidence any such phenomenon. BB performs better than HW[BB] for small problems, whereas HW[BB] outperforms BB on more complex problems.

Figure 5. CPU time comparisons in the logistics domain.

Figure 6. CPU time comparisons in the blocks-world domain.

The zeno-travel domain deals with people transportation using planes able to perform fast or slow movements; fast movements consume fuel faster than slow movements.

In the gripper domain there is a robot equipped with two grippers. The robot must be used to carry some balls from one room to another.

All domains have been structured according to a ground and an abstract level, following the approach sketched for the elevator domain (as reported in Figure 1, Figure 3, and Figure 4). Since we are dealing with two levels of abstraction, only ground-to-abstract (i.e., upward) translation rules\(^1\) are actually required.

---

\(^1\) Let us recall that the same set of rules allows performing also downward translations.
whereas $HW[BB]$ outperforms $BB$ on problems of medium complexity.

**Zeno-travel.** In this domain, an improvement of $HW[BB]$ over $BB$ can be observed, similar to the one shown for the blocks-world domain (see Figure 8). It is worth noting that, unfortunately, neither $GP$ nor $HW[GP]$ are able to successfully tackle any problem of this domain.

![Figure 8](image8.png)

**Figure 8.** CPU time comparisons in the zeno-travel domain.

**Gripper.** Figure 9 reports some information about the performances of $GP$, $BB$, and their hierarchical counterparts over the set of problems devised for assessing the gripper domain. Both $HW[GP]$ and $HW[BB]$ clearly overwhelm their non-hierarchical counterparts.

![Figure 9](image9.png)

**Figure 9.** CPU time comparisons in the gripper domain.

4 CONCLUSIONS AND FUTURE WORK

In this paper the $HW[]$ system has been described, devised to perform experiments on abstraction mechanisms. The proposed system is a parametric hierarchical planner, able to embed a PDDL-compliant external planner, to be used for planning at any required level of abstraction. Some preliminary tests have been made on five domains taken from the AIPS 2002, 2000, and 1998 planning competitions.

As expected, abstraction is not useful for improving the search performances on simple problems (characterized by solutions of limited length), due to the overhead introduced by the need of dealing with different levels of abstraction. On the other hand, especially for problems of increasing complexity, experimental results clearly show that abstraction-based techniques can significantly increase the performance of the search.

As for the future work, we are currently devising a mechanism for automatically abstracting domains, given their ground-level description. This ability would be very helpful while trying to assess the performance of $HW[]$ on a large set of domains. Furthermore, we are re-running the experiments described above by exploiting the capabilities of more recent planners (e.g., LPG [14]).

REFERENCES