Experimenting Abstraction Mechanisms through an Agent-Based Hierarchical Planner

G. Armano, G. Cherchi, and E. Vargiu
DIEE Department of Electrical and Electronic EngineeringUniversity of Cagliari
Piazza d'Armi I-09123 Cagliari
{armano, cherchi, vargiu}@diee.unica.it

***Abstract*— In this paper, an agent-based architecture devised to perform experiments on hierarchical planning is described. The planning activity results from the interaction of a community of agents, some of them being explicitly devoted to embed one or more existing planners. The proposed architecture allows to exploit the characteristics of any external planner, under the hypothesis that a suitable wrapper –in form of planning agent– is provided. An implementation of the architecture, able to embed one planner of the graphplan family, has been used to directly assess whether or not abstraction mechanisms can help to reduce the time complexity of the search on specific domains. Some preliminary experiments are reported, focusing on problems taken from the AIPS 2002, 2000 and 1998 planning competitions. Comparative results, obtained by assessing the performances of the selected planner (used first in a stand-alone configuration and then embedded into the proposed multi-agent architecture), put into evidence that abstraction may significantly speed up the search.**

***Index Terms*—Abstraction, Hierarchical Planning, Multi-agent Interactions.**

I. 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 [7], 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 [10], and on (ii) reduced models, obtained by completely removing certain conditions from the problem space [6]. 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" solutions at the abstract levels [4].

For a given abstraction hierarchy, both the downward and upward solution properties may hold (DSP and USP [11], 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) [2], whereby every abstract solution can be refined to a concrete-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 being reasonably low.

In this paper, an agent-based hierarchical planner is presented, able to embed any domain-independent (non hierarchical) planner provided that a compliance with the PDDL 1.2 standard [9] 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 is illustrated. Then, all customizations required to perform hierarchical planning with multi-agent interactions 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.

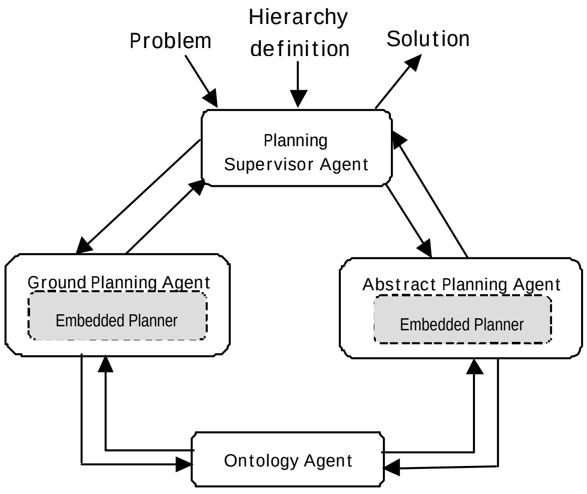

Fig. 1. The overall agent-based architecture.

II. System Architecture

Being aimed at investigating the impact of abstraction mechanisms on the search complexity, we implemented an agent-based system where hierarchical planning is performed through the interaction between a planning supervisor agent (PSA) and a community of planning agents, at least one for each level of abstraction, including the ground level. Each planning agent operates as a "wrapper" for an external planner able to process domain and problem definitions in a PDDL syntax. The supervisor's main responsibility is to coordinate the work of planning agents by feeding them with sub-problems issued at the proper abstraction level. In general, each planning agent can embed a different planner, thus giving rise to an agent-based planning system built upon a heterogeneous set of external planners. An ontology agent (OA) is responsible for providing any required domain description, no matter which level of abstraction is requested.

Headed at assessing the improvement obtained by enforcing abstraction on a particular planning system, we decided to embed the same planner in all planning agents. In this way, the significance of experimental results is ensured by the fact that comparisons can be made using the same planner, with and without abstraction mechanisms. Furthermore, the following constraints hold: (a) there are two levels of granularity; denoted as ground and abstract, and (b) planning agents can generate only linear plans. Figure 1 illustrates how such constraints affect the generality of the architecture sketched above: only two planning agents are actually required, called ground planning agent (GPA) and abstract planning agent (APA), respectively.

We are assuming that GPA and APA contain their own set of operators, types and predicates devoted to cope with the given problem at the proper level of granularity. Notwithstanding this general assumption, in this work we gave planning agents the capability of dealing 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 belong to a level, depending on the will of making it available or not to its superior.

In principle, any domain-independent and PDDL-compliant planner could be embedded within a planning agent, together with a suitable description –provided to the PSA– aimed at specifying how the domain translation between the GPA and the APA (and vice versa) must be performed. For the sake of simplicity, we assume that the selected external planner is able to generate only linear plans.

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

Fig. 2. An extract of the ground-level definitions for the elevator domain.

A. 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 2 illustrates a sample of definitions taken from the elevator domain, used in the AIPS 2000 planning competition [1].

To add a level of abstraction to this domain, separate information has been provided, consisting of: (i) a specification of the abstract domain, and (ii) a specification of how bi-directional translations occur 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

```
(define (domain elevator-abstract)
 (:requirements :typing)
 (:types passenger floor)
 (:predicates
  (origin ?person - passenger ?floor - floor)
  (destin ?person - passenger ?floor - floor)
  (boarded ?person - passenger)
  (served ?person - passenger))
 (:action load
  :parameters (?p - passenger ?f - floor)
  :precondition (and (origin ?p ?f))
  :effect (and (boarded ?p)))
 (:action unload
  :parameters (?p - passenger ?f - floor)
  :precondition (and (boarded ?p)
                     (destin ?p ?f))
  :effect (and (served ?p))))
```

Fig. 3. The abstract-level definitions for the elevator domain.

form for denoting an upward translation rule consists of specifying, with a Lisp 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. In particular, when the abstract-level goal must be set, an upward translation occurs aimed at mapping the ground problem space into an 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:

```
((origin ?p – passenger ?f - floor)
 (origin ?p - passenger ?f - floor))
```

which specifies that the predicate origin must be preserved while going upward and vice-versa. A further example illustrates how a predicate can be disregarded while going upward:

```
(nil (lift-at ?f - floor))
```

which specifies that the predicate lift-at is not translated into any abstract-level predicate. Note that this kind of clauses can be also omitted, being assumed by default. However, for the sake of clarity, a knowledge engineer may explicitly state which predicates should not be translated upward according to the given syntax.

Let us point out that the translations described above allow to specify Knoblock's reduced models [6], although the expressive power of such notation could be able to represent also more sophisticated translation strategies (for example, is-a and part-of abstractions performed over types and / or predicates are feasible).

B. *Planning as a Result of Multi-Agent Interactions*

The PSA takes as inputs a ground-level problem, a

```
(define (hierarchy elevator)
 (:domains elevator-ground elevator-abstract)
  (:translations
   (:level 0
    (:predicates
       ((origin ?p - passenger ?f - floor)
        (origin ?p - passenger ?f - floor))
       ((destin ?p - passenger ?f - floor)
        (destin ?p - passenger ?f - floor))
       ((boarded ?p - passenger)
        (boarded ?p - passenger))
       ((served ?p - passenger)
        (served ?p - passenger))
       (nil
        (above ?f1 - floor ?f2 - floor))
       (nil
        (lift-at ?f - floor)))))))
```

Fig. 4. Hierarchical translations for the *elevator* domain.

description of the domain (splitted into ground and abstract level), and a set of rules –to be used while translating ground into abstract predicates and vice-versa.

To search for a solution, first the PSA translates from the ground to the abstract space all the facts asserted (in form of predicates) in the init section together with the predicates expressed in the goal section, so that the APA can be fed with this information. Then, the PSA sends a planning request, through a FIPA-ACL request performative, issued to the APA, which invokes the embodied planner to search for all existing (abstract) solutions. In case the APA does not currently know the domain ontology, a request for loading it is issued to the OA, which takes care of feeding the APA with the requested ontology, otherwise a failure performative is sent back to the APA. Afterwards, a solution –if any– is selected and sent to the PSA through a suitable inform performative. The PSA translates each operator of the abstract plan into a sub-problem (understandable by the ground level planner), and sends a planning request to the GPA, which invokes the embedded planner to find possible refinements. If a refinement cannot be found, a failure performative is sent to the PSA, which enforces backtracking on the APA. The whole process ends when a solution is found or the PSA notifies an overall search failure. Note that the refinement of an abstract operator is performed by activating the planner embedded in the GPA on the goal obtained by translating downward its effects. It is worth pointing out 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.

III. Experimental Results

The current prototype of the system has been implemented in Common Lisp Object System (CLOS), for the sake of rapid prototyping. The embedded planners used to perform experiments are GRAPHPLAN (GP) [3] and BLACKBOX [5]. In the following, GP and BB are used to denote the GRAPHPLAN and BLACKBOX algorithms, whereas HGP and HBB are used to denote their “agentified” hierarchical counterpart, respectively.

To assess how abstraction can improve the search, we performed some preliminary tests on five domains used in the AIPS planning competitions (2002 [12], 2000 [8], and 98 [9]): elevator, logistics, blocks-world, zeno-travel, and gripper. Experiments were conducted on a machine powered with an Intel Celeron CPU working at 1200 Mhz with 256Mb of RAM. A time bound of 1000 CPU seconds has also been adopted.

Table 1. Performance comparison of BB and GP, together with their hierarchical counterparts HBB, and HGP.

Problem	GP	HGP	BB	HBB
elevator-1-4	0.007	0.062	0.098	0.333
elevator-3-1	0.234	0.364	1.342	1.208
elevator-4-1	1.956	0.837	1.030	1.744
elevator-4-4	10.114	0.839	311.046	1.792
elevator-5-1	364.74	2.032	180.781	2.548
elevator-7-2	--	12.043	--	3.899
logistics-4-2	0.682	1.227	0.266	0.461
logistics-5-2	0.085	0.165	0.151	0.466
logistics-7-0	--	10.931	4.494	2.176
logistics-8-1	--	16.265	2.898	3.027
Logistics-10-0	--	43.435	8.272	3.763
Logistics-15-0	--	203.467	10.915	6.333
blocks-4-0	0.345	0.317	0.162	0.674
blocks-6-0	3.040	1.821	0.263	1.684
blocks-8-0	31.612	11.123	0.924	2.467
blocks-10-0	--	–	6.821	5.003
blocks-11-0	--	–	16.236	4.254
blocks-14-0	--	–	--	9.845
zeno-1	0.027	0.519	0.223	0.369
zeno-8	--	42.549	0.941	2.360
zeno-9	--	–	0.344	3.377
zeno-11	--	–	11.203	2.786
zeno-13	--	–	62.999	20.524
zeno-14	--	–	--	20.042
gripper-2	4.722	0.565	0.424	0.635
gripper-3	7.914	1.732	5.224	1.203
gripper-4	18.321	2.630	268.737	1.554
gripper-5	57.212	4.377	421.186	1.548
gripper-6	--	7.968	586.439	2.267
gripper-9	--	24.294	--	3.631

A. 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 stacking/unstacking a block onto/from another block, and for putting down or picking up a block.

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 2, Figure 3, and Figure 4). Since we are dealing with two levels of abstraction, only ground-to-abstract (i.e., upward) translation rules are actually required.

B. 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 5, whose y-axis is expressed in logarithmic scale), whereas HGP and HBB keep solving problems with greater regularity (although the relation between number of steps and CPU time remains exponential).

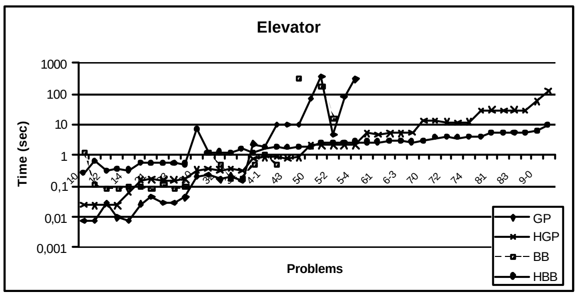

Fig. 5. CPU time comparisons in the *elevator* domain (x-axis reporting the problem ID).

Logistics. In the logistics domain, GP suffers from a phase transition problem (see Figure 6): 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, HGP keeps solving problems of increasing length without putting into evidence any such phenomenon. BB performs better than HBB for small problems, whereas HBB outperforms BB on more complex problems.

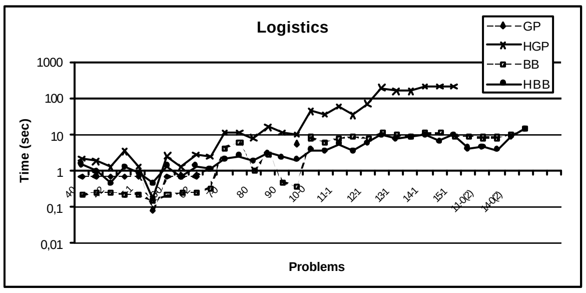

Fig. 6. CPU time comparisons in the *logistics* domain (x-axis reporting the problem ID).

Blocks-world. As shown in Figure 7, the tests performed on the blocks-world domain reveal a similar trend between GP and HGP, although the latter performs slightly better than the former. As for BB, it performs better than HBB for simple problems, whereas HBB outperforms BB on problems of medium complexity.

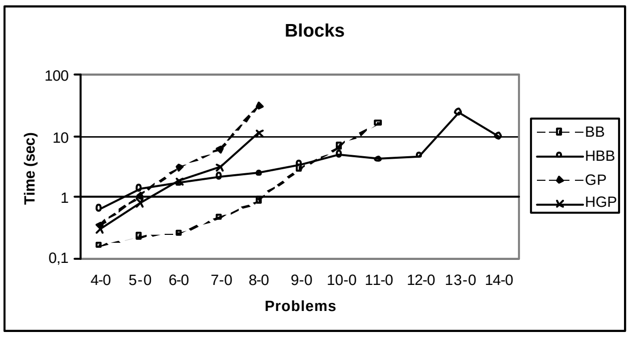

Fig. 7. CPU time comparisons in the *blocks-world* domain.

Zeno-travel. In this domain, an improvement of HBB 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 HGP are able to successfully tackle any problem of this domain.

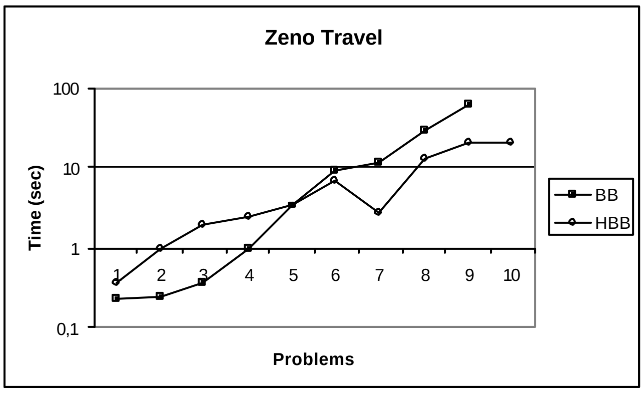

Fig. 8. CPU time comparisons in the *zeno-travel* domain.

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 HGP and HBB clearly overwhelm their non-hierarchical counterparts.

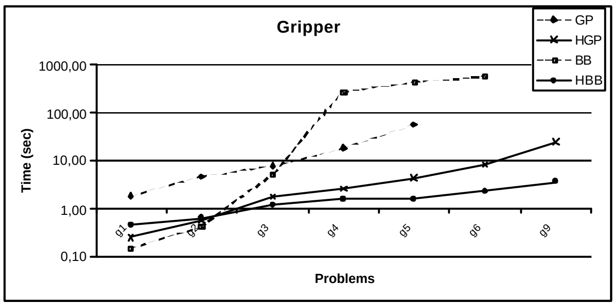

Fig. 9. CPU time comparisons in the *gripper* domain.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, an agent-based system has been described, devised to assess whether abstraction can significantly reduce the time complexity of planning problems. The proposed system is an agent-based hierarchical planner, able to embed a PDDL-compliant external planner, to be used for planning at any 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 performance 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 investigating a mechanism for automatically abstracting domains, given their ground-level description. This ability would be very helpful while trying to assess the performances of the system on a large set of domains.

REFERENCES

[1] Bacchus, F. Results of the AIPS 2000 Planning Competition. 2000. Url: http://www.cs.toronto.edu/aips2000.

[2] Bacchus, F., and Yang, Q. Downward Refinement and the Efficiency of Hierarchical Problem Solving. *Artificial Intelligence*. 1994. 71:43-100.

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

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

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

[6] Knoblock, C.A. Automatically Generating Abstractions for Planning. *Artificial Intelligence* 68(2), 1994.

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

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

[9] McDermott, D., and the AIPS-98 Planning Competition Committee. 1998. PDDL – The Planning Domain Definition Language.

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

[11] Tenenberg, J.D. *Abstraction in Planning.* Ph.D. thesis, Computer Science Department, University of Rochester, 1988.

[12] Results of the AIPS Planning Competition. (2002). 2002. Url: http://www.dur.ac.uk/d.p.long/competition.html.