Final Version appears in IEEE Trans. on Robotics, Vol. 20, No. 4, 2004

Improving Efficiency in Mobile Robot Task Planning through World Abstraction

Cipriano Galindo, Juan-Antonio Fernandez-Madrigal, and Javier Gonzalez

Abstract— Task planning in mobile robotics should be performed efficiently due to real time requirements of robot-environment interaction. Its computational efficiency depends both on the number of *operators* (actions the robot can perform without planning) and the size of the *world states* (descriptions of the world before and after the application of operators). Thus, in real robotic applications where both components can be large, planning may turn inefficient and even unsolvable.

In the AI literature on planning, little attention has been put into efficient management of large-scale world descriptions. In real large-scale situations, conventional AI planners (in spite of the most modern improvements) may consume intractable amounts of storage and computing time due to the huge amount of information.

This paper proposes a new approach to task planning called Hierarchical Task Planning through World Abstraction (HPWA) that, by arranging hierarchically the world representation, becomes a good complement of STRIPS-style planners, improving significantly their computational efficiency. Broadly speaking, our approach works by firstly solving the task planning problem in a highly abstracted model of the environment of the robot, and then refines the solution under more detailed models where irrelevant world elements can be ignored due to the results previously obtained at abstracted levels.

Among the different implementations that can be made with our general strategy, we describe two that use a graph-based hierarchical world representation named AH-graph. We show experiments as well as results of a mobile robot operating in a large-scale environment that demonstrate an important improvement in efficiency of our algorithms with respect to conventional (both hierarchical and non-hierarchical) planning, and also their nice integration with existing planners.

Index Terms—Intelligent robotics, hierarchical task planning, world abstraction, world modeling, mobile robotics.

This work was supported by the Spanish Government under research contract DPI2002-01319.

The authors are with the System Engineering and Automation Department, University of Málaga, Campus Teatinos – Complejo Tecnológico, 29071 Málaga, Spain. E-mail: {cipriano,jafma,jgonzalez}@ctima.uma.es

I. INTRODUCTION

Intelligent mobile robotics relies on planning to carry out deliberative actions based on its knowledge about the environment. Robotic task planning has its origins in the more general problem of planning, which historically constitutes an important issue in Artificial Intelligence, being widely studied since the 1960's. Besides robotic task planning, AI planning has been particularized to a variety of problems such as path and motion planning ([[32],[37]), assembly sequence planning ([5],[11]), scheduling ([46],[53]), production planning [47], etc. In general, the purpose of planning is to synthesize an abstract trajectory in some search space (also named *state space*, since it consists of possible states of the world), predicting outcomes, choosing and organizing actions of different types for reaching goals or for optimizing some utility functions.

Maybe surprisingly, there are not many AI classical planners integrated into robotic architectures, mainly due to difficulties in integrating symbolic treatment of information (planning) into non-symbolic sources of data acquired from the real-world. Rather, most of planning in robotics uses specific algorithms intended to guide the execution of particular robotic tasks (and not others) ([1],[42]), as it is the case of route planning for navigation. Among the few generic (in the AI sense) planners implemented for robotic architectures, some remarkable approaches are STRIPS [19], that was the first planner used for deliberation in a robot (the Shakey robot [8]), and PRODIGY [49], the planner of the Xavier mobile robot.

None of the works where task planners are employed for mobile robots have addressed the problem of computational efficiency, perhaps because they do not deal with a complex large-scale space with hundreds of elements. However, this situation is easily encountered by a real mobile robot that moves within many different places, or when the robot, for example equipped with a manipulator on-board, may interact with a lot of world objects.

In general, the high computational cost of planning arises from the combinatorial search in the state space. This cost depends both on the complexity of the description of the states, and on the number of *operators*[1] that can be applied to each state to obtain another. In the case of robotic task planning, the former corresponds to the symbolic model of the world managed by the robot, while the latter refers to actions that the robot can carry out without planning (for example, move forward, turn, grasp an object, etc.). The lack of efficiency in planning may even lead to the intractability of the problem because one of these two reasons (or both).

There has been great effort in AI literature devoted to improve the efficiency of planning, but the focus has been mostly on managing the possibly large number of operators involved in a complex problem. We believe that not all the attention that it deserves has been paid to the world model, since in pure AI it uses to be of very manageable size. However, in real robotic applications like those that arise in large-scale environments (i.e., a mobile robot delivering objects in an office building), the world model can become large enough to make task planning intractable or very inefficient.

Our approach to improve efficiency of planning, called *Hierarchical task Planning through World Abstraction* (HPWA), is particularly oriented towards this kind of large-scale robotic scenarios. The approach consists of breaking down the combinatorial expansion of the search space through reduction on the size of the description of the world (the model), leaving unchanged the operators. Our method can also yield benefits in other non-robotics areas where planning is carried out on worlds with a large number of elements, such as in Intelligent Transportation Systems.

For our purpose, we use *abstraction*. Abstraction is understood here as eliminating unnecessary details from the information managed in some operation [21]. Abstraction has already been used as a mechanism for improving efficiency of planning. It has been demonstrated that planning with abstraction reduces the search space and it is usually more efficient than non-hierarchical planning ([22],[29]). In literature,

[1] Operators in planning terminology are the possible primitive actions that can be set for execution in a given state of the world in order to reach another state. The planner needs a formal definition of each operator, which in STRIPS-style domains includes its *preconditions* (world situations where it is applicable) and its *effects* (changes in the world description after its execution).

planners that use abstraction in some way to reduce computational cost are commonly called *hierarchical planners*. A hierarchical planner first solves a problem in a simpler abstract *domain*[2] and then refines the abstract solution, inserting details that were ignored in the more abstract domain. This is repeated until the solution is reached. We have found that previous hierarchical planners exploit any of the following three kinds of abstractions [52]: precondition-elimination abstraction, effect abstraction, and task abstraction. Some remarkable implementations are: ABSTRIPS [44], Pablo [9], Prodigy [29] (that uses the hierarchy automatically produced by Alpine ([30],[38])), HTN planners [12] (that use task abstraction), etc. In this paper, we explore a new kind of hierarchical planning consisting of abstracting only the description of the world, which has produced promising results, and also can provide other planning methods with an easy way of including abstraction capabilities.

Our HPWA methods are basically procedures that use an existing planner (the so-called *embedded* planner) at different levels of world abstraction, without changing the internal structure or the behaviour of such embedded planner. Although our experiments have only been run with Graphplan [7] and Metric-FF [25] (they have become very popular planners during the last years), we don't find any limitation in using any other STRIPS-style one. It should be noticed that the embedded planner may already include some other techniques for improving computational efficiency of planning; hence, the overall improvement of the HPWA framework can be high.

Two implementations of HPWA are presented in this paper. Both of them use a hierarchical representation of the world called AH-graph [15]. An AH-graph is a graph-based hierarchical structure that can model symbolically the robot environment at different levels of detail.

The paper is organized as follows: section 2 introduces briefly the hierarchical, symbolic model of the environment called AH-graph and its automatic construction by an intelligent mobile robot. Section 3

[2] Generally, the term *domain* refers to both the description of the world and the definition of operators (which includes their preconditions and effects).

presents two HPWA implementations. Section 4 shows some results and compares the new technique to conventional (both hierarchical and non-hierarchical) planning. Finally, section 5 outlines some conclusions of this work and future lines of research.

II. MODELING THE ROBOT ENVIRONMENT WITH ABSTRACTION USING AH-GRAPHS

Our approach takes advantage of abstraction in world modeling by using the AH-graph model, a symbolic representation successfully used in robotics applications ([12], [17]) which is capable of representing a hierarchy of abstraction upon ground information. For completeness, in section A we give a brief description of this model; a complete formalization can be found elsewhere [15]. Section B deals with the automatic construction of these hierarchies.

A. The AH-Graph Model

An AH-graph is a relational, graph representation of the environment which includes hierarchical information, that is, the possibility of abstracting groups of elements to "super-elements". This kind of abstraction produces different layers isolated from one another, called *hierarchical levels*, that represent the same environment at different amounts of detail. Hierarchical levels are denoted L^i, where i is the position of the level in the hierarchy. The lowest hierarchical level of the AH-graph, L^0, is called the *ground level*, which represents the world with the maximum available amount of detail. The highest hierarchical level, say L^{r-1}, is called the *universal level*, and it typically represents the robot environment with a single node. Each hierarchical level in an AH-graph is a flat multigraph[3]. For the sake of simplicity, this work will use equivalently the term "graph".

[3] A multigraph is a conventional graph in which there can be more than one arc connecting two nodes [49].

The nodes of each hierarchical level represent elements (or super-elements) of the world while the arcs represent relations between them with the possibility of holding weights representing the strength of those relations. For example, in mobile robotics, nodes can represent distinctive places [36], while the arcs can represent the navigability relation between them, with the geometric distance as the arc weight. Fig. 1 shows another example of an AH-graph with a single type of relation representing “rigidly joined”.

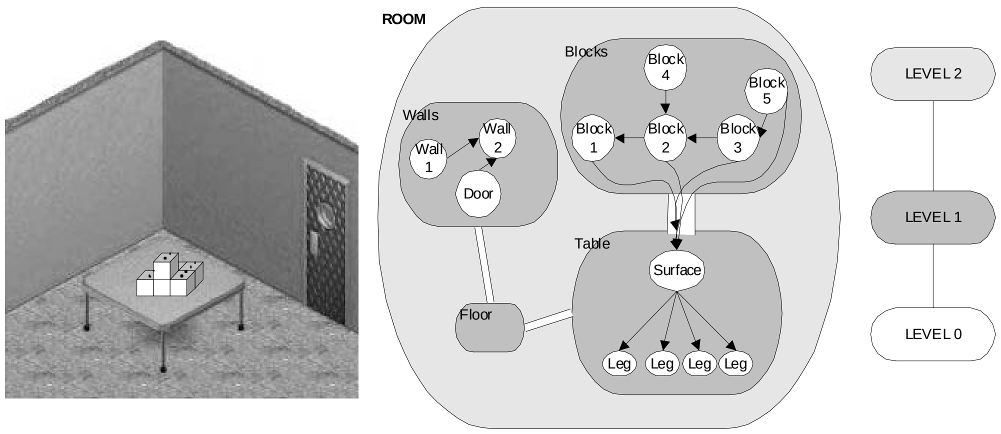

Fig 1. An example of a simple AH-graph model representing a possible abstraction of the spatial elements within a room. From left to right, a room, the hierarchical levels that model the room (represented by different gray shades), and the resulting hierarchy, are shown.

A group of nodes of a hierarchical level can be abstracted to a single node at the next higher hierarchical level, which becomes their *supernode* (the original nodes are called *subnodes* of that supernode). Analogously, a group of arcs of a hierarchical level can be represented by a single arc (their *superarc*) at the next higher level (see fig. 2).

Besides the structural information captured by the AH-graph through nodes, arcs, and hierarchical levels, both nodes and arcs can also hold non-structural information in the form of *annotations*. This information may include, but is not limited to, geometrical data gathered from the environment (i.e.: maps of obstacles), costs incurred by the robot when executing an action (i.e.: an arc that represents “navigability” from one location to another can store the expected cost energy of that navigation), etc. Non-structural information can be useful for planning and other algorithms. In particular, it is extensively used when the AH-graph model is employed for mobile robot navigation ([14],[18]).

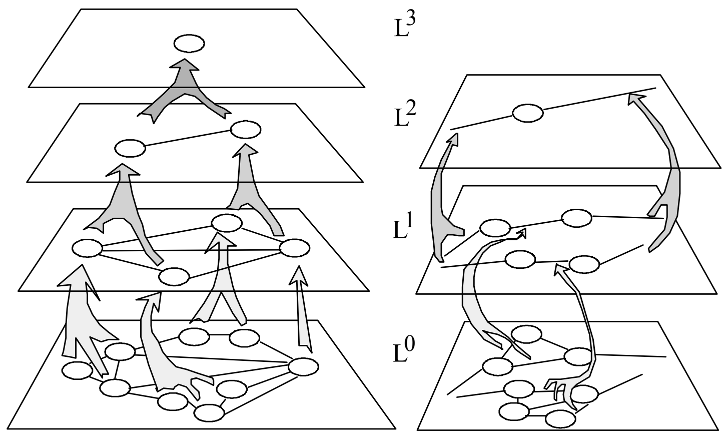

Fig 2. Two examples of AH-graphs of four and three hierarchical levels, respectively. The AH-graph on the left side illustrates the abstraction of nodes from the lowest to the highest hierarchical levels. The AH-graph on the right illustrates the abstraction of arcs.

B. Automatic Construction of Good Hierarchies of Abstraction for HPWA

Concerning the reduction of the computational cost of task planning, the power of the HPWA methods presented in this paper greatly depends on the hierarchy of abstraction (AH-graph) that is used. It must be pointed out that any abstraction-based computational approach (not only task planning ([15]), and not only through AH-graphs ([26],[33])) can encounter "degenerated" situations where it becomes even more time consuming than non-hierarchical methods. These situations are due either to the overhead involved in repeating operations at each hierarchical level, or to backtracking if no partial solution is met at some point, or to both.

Some interesting approaches for automatically constructing good hierarchies of abstraction for task planning have been proposed ([29],[27],[20]). In these works a polynomial time method for such construction is presented, although they do not guarantee to obtain the optimal hierarchy with respect to computational cost of task planning.

In general, the problem of finding the optimal hierarchy of abstraction is a combinatorial optimization not solvable in polynomial time when the following circumstances occur:

a) The space of all possible hierarchies of abstraction is exponentially large with respect to the elements involved in the definition of these abstractions (in our case, these elements are the nodes and arcs in the ground hierarchical level of the AH-graph; in other cases, such as the STRIPS-style domain, they are the symbols involved in the description of operators, the world, etc.).

b) Suitable (low-cost) approaches to calculate the goodness of a given hierarchy of abstraction are not available, that is, the goodness of a given hierarchy of abstraction can only be calculated directly when using it to solve the problem.

In the particular domain of mobile robotics, finding a good hierarchy of abstraction imposes additional limitations. First, consistency must be maintained across the hierarchical levels of the model. This problem is closely related to that of *aggregation-disaggregation* addressed in the community of Multi-Resolution Modeling ([43], [10]). In our AH-graph model, such consistency is guaranteed by the automatic construction of abstract levels from the ground level (as explained below in this section).

Secondly, the symbolic model of the world (the ground level) is not static, that is, it must be kept consistent with the real world, which is subjected to changes. In our approach, at each instant the task planner is executed, that is, under the task planning perspective, the ground level is taken as a symbolic "snapshot" of the world that must be reflect changes over time. Thus, for mobile robot applications, a procedure that finds a good hierarchy for task planning should be iterated along the robot life for adapting to new symbolic data and structures. Under that constraint, an algorithm that performs a complete optimization in the space of possible hierarchies each time the model of the world varies (each snapshot), even if it is an approximation based on heuristics, can lead to a high computational cost.

Therefore, we believe that in our case it is desirable an optimization algorithm that starts from an initial hierarchy of abstraction (probably not the optimal, even a bad hierarchy) and improves it in a continuous and gradual manner as the robot acquires experience from its operations, also adapting to world changes.

That approach fits well to mobile robotics, since it distributes the complexity of constructing the best hierarchy of abstraction along the active life of the robot, it assures that the model of the world is monotonically improved over time, and in addition, it allows the robot to adjust the hierarchy of abstraction for its particular necessities: the area of the world where it operates, the tasks at hand, etc. (It has been shown that a hierarchy that is good for solving a given problem may not be so good for solving a different one[4] ([29][27])).

We have previously presented ([15]) a software called C.L.AU.D.I.A. (*Concept Learning, AUtonomous Device, which Improves Abstraction*) that follows the previously commented guidelines and uses a multi-hierarchical extension of AH-graphs as the model for abstraction. It has been applied to solving hierarchical path search problems for mobile robot route planning in large-scale spaces, which has yielded important improvements in the performance of the robot. Modifying it to solve any other operation, such as task planning problems, consists just of varying the goodness evaluation function.

C.L.AU.D.I.A. is a software that samples the space of candidate hierarchies (AH-graphs in this case) that are coherent with a given ground information acquired from the environment. The ground information can be, for example, a flat topological map of space[5]. The system starts by constructing randomly an initial hierarchy of abstraction. Then, it carries out a hill-climbing algorithm for optimization that comprises the following steps: first it finds a set of new AH-graphs that are variants of the original one –its “neighbors” in the neighborhood of the space of AH-graphs-, second, it assigns appropriate goodness values to them (i.e.: the computational cost of task planning with each of these AH-graphs), and finally it selects the one with

[4] This has also been stated in the hierarchical path search domain ([16]), not only in the task planning domain. In fact, an extension of AH-graphs that includes more than one hierarchy of abstraction has been demonstrated to perform better than single-hierarchical approaches ([16] and [15]). For the sake of simplicity, this paper only deals with a single-hierarchy that adapts to all the task planning problems of the robot.

[5] Several approaches for constructing automatically topological maps for mobile robots can be found in ([3][40]).

the best goodness value and repeats the cycle. This optimization process is iterated while the robot is operating in its environment.

This outlined scheme contains two relevant procedures that need some further explanation:

a) *Automatic abstraction of hierarchical levels for producing AH-graphs.* A hierarchical level is abstracted to another higher level by clustering its nodes: each cluster will correspond to a supernode at the higher level, while superarcs can be generated in such a way that the same connectivity relations existing in the original graph are kept. There exist many graph clustering algorithms in literature ([23],[41]). Usually, they intend to optimize some characteristic of the resulting clustering, for example, minimizing the cost of the arcs that connect different clusters. In our approach, a STAR-like method [39] has been employed, due to its high efficiency (it is *O(a)*, where *a* is the number of arcs of the graph) as well as the wide range of diverse clusterings it can generate. Its input parameters include a number of "seed" nodes and a maximum size for the clusters, both lying into a finite range of discrete values.

b) *Constructing the neighborhood of an AH-graph.* Given an AH-graph, a number of neighbors can be automatically generated by a simple procedure. Firstly, a hierarchical level of the AH-graph is selected, called the *pivot level*. Then, an alternate clustering of that level is proposed by varying the input parameters employed in the clustering algorithm[6]. From that new hierarchical level, others can be constructed simply by repeatedly calling the clustering algorithm, obtaining a complete AH-graph. The portion of this new AH-graph that is kept unchanged from the original AH-graph can be

[6] Since these input parameters lie into a finite range of discrete values, they are enumerable. Therefore the number of possible clusterings that are generated can be easily controlled. Some graph isomorphism mechanism can be used to detect the case that different input parameters lead to equal clusterings, as discussed in depth in ([15]).

regulated by shifting upwards or downwards the pivot level. Figure 3 illustrates this procedure.

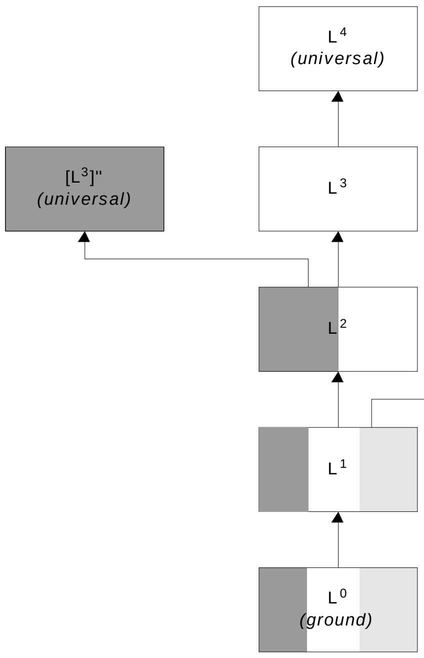

Fig. 3. Example of neighborhood of two elements for a given AH-graph (white-filled). The first neighbor (lightgray-filled) has been constructed using pivot level L^1 of the original AH-graph. The second one (darkgray-filled) has used pivot level L^2.

III. HIERARCHICAL PLANNING USING WORLD ABSTRACTION

Our HPWA framework is inspired from human beings. It seems that we humans organize knowledge of the environment hierarchically, since this representation is used for planning tasks more efficiently ([35],[34],[51]). Let us give an example to best elucidate this fact: if we want to go shopping the first plan we may make is “Leave the house, take the car and drive to the supermarket”. Although this sequence of actions does not give us details about how to solve the “go shopping” task, it is a first approximation to it. Each action of this first approximation can be refined further in order to achieve a plan which physically solves the task. Thus, the “Leave the house” action may be refined into “dress yourself, take the car keys, take some money and cross the door”. In fact, these actions may be refined over and over until achieving primitive actions, which can no longer be refined (the operators). Examples like this are common in the AI literature in order to justify the inspiration of other hierarchical task planning approaches in the human experience, although an important point must be highlighted: as we can notice, these abstractions do not only involve abstractions of operators, but also abstraction of *world elements*, i.e. house → rooms → door → lock.

This last idea is what we call *world abstraction*, and may be applied to task planning to perform in the same way human beings do. For that purpose, different strategies can be constructed upon that basic idea. In the following sections, we develop two of them, but first, in section A, some preliminary terminology is introduced. In sections B and C, two implementations of HPWA are presented. Although in this paper we assume that the robot environment is modelled by an AH-graph, other hierarchical representations could be possible under the same philosophy. Finally, section D analyses the influence of some planning anomalies in our hierarchical method.

A. Hierarchical Planning Definitions

In this section we introduce some concepts required for the subsequent formalization of HPWA.

Definition 1 (Logical Predicate and State). A *logical predicate* or *atomic sentence* of a first-order language *L,* with k parameters is defined as:

$$\text{<predicate symbol> } param_0...param_{k-1}$$

where $param_i$ is a language constant. We assume that each $param_i$ represents an element of the world, that is, a node of the AH-graph. There exists, therefore, a *one-to-one* correspondence between nodes of the AH-graph and the parameters of the logical predicates. It should be pointed out that all parameters of a logical predicate must represent nodes of a *same* hierarchical level.

A *state* is a finite and consistent set of predicates that represent the world. Within a state, all the parameters of the predicates must correspond to nodes of the *same* AH-graph hierarchical level.

Definition 2 (Problem Space and Operator). A *problem space* is composed of *L,* a first-order language, and *O,* a set of operators.

An *operator* $o \in O$ is a quadruple *{Param,Pre,Del,Add}*, where *Param* is a list of variables that parameterize the operator and can be substituted by world elements, *Pre* is the set of predicates representing the preconditions of the operator, and *Del,* and *Add* are the operator effects on state *s* on which *o* is applied (*Del* is the set of predicates that will be eliminated from *s,* and *Add* is the set of predicates that will be added to *s*).

Definition 3 (Abstraction Function for Predicates). The *abstraction function for predicates* s_p^i is formally defined at any hierarchical level L^i, except at the universal level, L^{r-1}, as:

s_p^i: PREDS$^i \rightarrow$ PREDS^{i+1} , $i< r-1$

where *PREDS*^{i} is the set of all the possible logical predicates whose parameters correspond to nodes at hierarchical level L^i. The superscript *i* in s_p^i stands for level L^i.

$s_p^i(p)$ works by abstracting nodes (that is, parameters) of the predicate *p*, in order to obtain another predicate *p'* defined at level L^{i+1}. Thus, the abstraction function for predicates s_p^i reduces the amount of detail from *p* to *p'*.

The hierarchy may be *incomplete* [15], that is, there may be nodes that are not abstracted into any supernode. If *p* contains a node that can not be abstracted to a supernode, then *p* can not be abstracted to the next hierarchical level. For simplicity, we will consider in the rest of the paper that the AH-graph is complete (the proposed algorithms work with incomplete hierarchies too, although that might lead to smaller improvements in computational efficiency due to the limited power of abstraction in that case).

The inverse of the s_p^i function is called the *refining function for predicates* and it is defined $\forall L^i : i>0,$ as:

$$[s_p^{i-1}]^{-1} : \text{PREDS}^i \rightarrow \text{power}(\text{PREDS}^{i-1})$$

$$[s_p^{i-1}]^{-1}(p_j) = \{p_h \in \text{PREDS}^{i-1} : s_p^{i-1}(p_h)=p_j\}$$

where *power(PREDS*$^{i-1}$*)* is the power set of *PREDS*$^{i-1}$. This function increases the amount of detail of predicates.

Notice that predicates of level L^0 can not be refined, since neither s_p^{-1} nor its inverse exist.

Definition 4 (Abstraction Chains of Predicates). A predicate can be abstracted more than once obtaining a *chain of abstraction*. This is accomplished by using the abstraction function for predicates recursively (assuming that it is defined at all steps). The *length* of a chain of abstraction is the number of times the predicate has been abstracted. An example of a chain of abstraction of length 4 for predicate *p* ($p \in PREDS^i$) is:

$$s_p^{i+3}(s_p^{i+2}(s_p^{i+1}(s_p^{i}(p))))$$

Definition 5 (Abstraction Function for States). The *abstraction function for states* s_s^i is defined at any hierarchical level L^i, except at the universal level, L^{r-1}, as:

$$s_s^i : S^i \rightarrow S^{i+1}, \; i < r-1$$

where S^i is the set of all states that can be defined at level L^i.

The abstraction function for states $s_s^i(s)$ works by applying the abstraction function for predicates s_p^i to each predicate of *s* in order to obtain another state *s'* defined at level L^{i+1}.

The inverse of this function is called the *refining function for states* and is defined $\forall L^i : i>0$, as:

$$[s_s^{i-1}]^{-1} : S^i \rightarrow power(S^{i-1})$$

$$[s_s^{i-1}]^{-1}(s_j) = \{s_h \in S^{i-1} : s_s^{i-1}(s_h) = s_j\}$$

where $power(S^{i-1})$ is the power set of S^{i-1}. Notice that states of level L^0 can not be refined since neither s_s^{-1} nor its inverse exist.

The s_s^i function, as well as the abstraction function for predicates, can be applied recursively to obtain a chain of abstraction. The length of this abstraction chain is limited by the possibility of abstraction of all predicates that make up the state. An abstraction chain for states may not reach to the highest hierarchical level (universal) if the hierarchy is not complete.

B. First HPWA Method: Planning Without Plan Guidance

This section presents a first implementation of the HPWA approach that, for short, we will call *HPWA-1*. Without loss of generality, we have chosen to enhance with HPWA two popular and efficient graphplan-like planners: Graphplan [7] and Metric-FF[7] [25]. Graphplan-like planners are well-known general-purpose planners for STRIPS-style domains that have been improved over time in a variety of directions ([2],[6], and [31]). The main reasons for choosing these planners include the availability of their C implementations as well as their efficiency [7].

An important feature of HPWA-1 is that it sets the best bound on the optimality of the plans that can be obtained with a refinement of the world, using the embedded planner included in the method (as happens in hierarchical path search by refinement [15]).

The general idea of HPWA-1 is to solve the plan at a high level of world abstraction by running the embedded planner at this level, and then use the resulting abstract plan at the next lower level ignoring the irrelevant world elements that do not take part in the abstract plan. This process is repeated recursively. Fig. 5 illustrates the steps taken by the algorithm.

More precisely, HPWA-1 works as follows (see fig. 4): first, it selects the highest hierarchical level where the plan can be solved. To do this, the abstraction chains for both the initial and goal states are generated as long as possible[8]. The highest common hierarchical level reached by both abstraction chains will be the initial level for the planning process. Let L^w be that initial hierarchical level for planning (w is the minimum length of both abstraction chains). The embedded planner (Graphplan or Metric-FF in our case) is run to provide an abstract plan at this level, say Π. For that purpose, the input provided to the embedded planner

[7] The main reason to choose Metric-FF is its capability to produce plans while minimizing a cost function. Such feature is useful when developing real robotic applications.

[8] Remember that this is limited by the possibility of abstracting the parameters of the predicates that constitute the states.

consists of the abstracted initial ($s_s^w[s_0]$) and goal ($s_s^w[s_g]$) states and the set of operators O, defined in the problem space.

The resulting abstract plan will be used to discard irrelevant information at the next lower level L^{w-1} by simply ignoring the subnodes of those world elements that do not appear in Π. Let μ be the set of nodes of level L^w corresponding to the parameters[9] of all operators that form the abstract plan Π. Then, the subnodes of nodes that do not appear in μ will not longer be considered in the planning process. To refine the plan to hierarchical level L^{w-1}, our method considers only $s_s^{w-1}[s_g]$ and $s_s^{w-1}[s_0]$ as the desired goal and initial states, as well as the subnodes of the elements of μ.

This process is repeated until the lowest level of the model is reached (see fig. 4). In general, the lowest level to be reached will be the one where the initial and goal states are given. If at some step of refinement, no plan can be obtained, backtracking occurs and a new abstract plan must be retrieved (notice that backtracking is reduced by the automatic construction of good AH-graphs mentioned in section II.B, since it involves computational cost increments that C.L.AU.D.I.A. tends to avoid).

The algorithm for HPWA-1 is completed with a procedure to translate the AH-graph world information (nodes and arcs) into logical predicates.

In order to illustrate the HPWA-1 method, let us consider a relatively simple environment where a mobile robot with a manipulator on board has to perform different tasks involving a given object of the world named "*box*". Figure 6a shows a hierarchical model of the environment with three zones (a laboratory, a room, and a corridor), the mobile robot itself, and the box to be handled. In this model, three hierarchical levels have been considered: the ground level, L^0, with the maximum amount of detail (nodes represent distinctive locations for navigation and manipulation), the first level, L^1, for grouping nodes of L^0 in three

[9] As stated earlier in definition 1, the parameters of the operators correspond to world elements (nodes of the AH-graph).

different areas that allow the robot to reset odometric error, and finally, level L^2, which exhibits the minimum amount of detail.

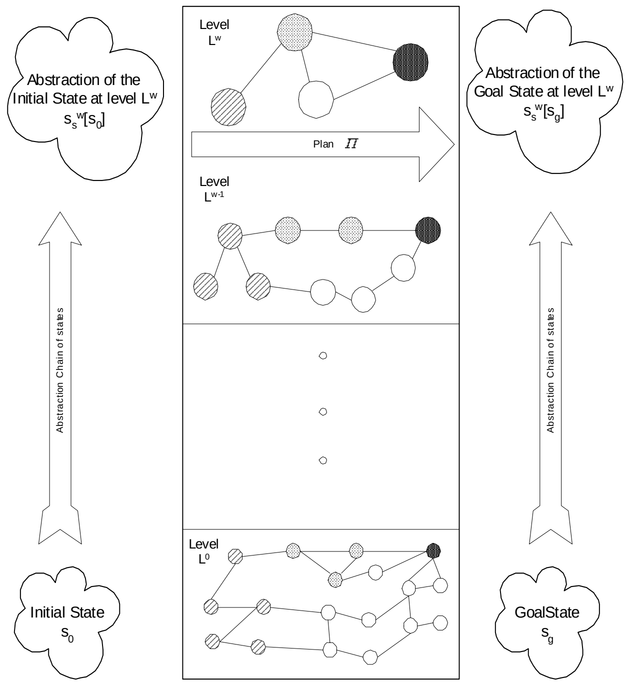
Abstraction of the Initial State at level L^w $s_s^w[s_0]$
Level L^w
Abstraction of the Goal State at level L^w $s_s^w[s_g]$
Plan Π
Level L^{w-1}
Abstraction Chain of states
Abstraction Chain of states
Level L^0
Initial State s_0
GoalState s_g

Fig. 4. Procedure to ignore irrelevant elements. In the figure, only the filled nodes (belonging to μ; as explained in the text) take part in the abstract plan Π at level L^w, so at lower levels, the subnodes of no-filled nodes are ignored. This process is repeated until reaching the ground level, where all no-filled nodes are discarded to obtain the plan that solves the proposed task.

The logical predicates that set up the states of the world are easily obtained from the arcs of each hierarchical level (see fig. 6) as follows:

If there is an arc between two locations *l1* and *l2*, a new logical predicate *(nav l1,l2)* is added to the description of the state.

If there is an arc between an object *o* and a location *l*, the predicate *(at o l)* is also added.

Finally, if there is an arc between the node that represents the mobile robot and a location *l*, the predicate *(at-robot l)* is added.

```
PROCEDURE HPWA-1 (State $s_g$, AH-graph ah):Plan
n=NodesFromLevel($L^w$)
pred= TransformWorldToPredicates(ah,$L^w$ ,n)
FOR i = $L^w$ DOWNTO GroundLevel(ah)
post= StateAbstraction(ah,$s_g$,i)
plan=EmbeddedPlanner(pred,post,operations)
param=PlanParameters(plan)
pred=NULL
IF (i != GroundLevel(ah)
FOR p=1 TO Length(param)
n= Subnodes(ah,param[p])
pred=pred + TransformWorldToPredicates(ah, i-1,n)
END
END
END
RETURN (plan)
END
```

Fig. 5. Pseudocode of the HPWA-1 implementation. Based on an abstract plan, this algorithm ignores irrelevant world elements at the lower hierarchical level. *NodesFromLevel(x)* procedure returns all nodes of hierarchical level *x*. *TransformWorldToPredicates(x,y,z)* produces those logical predicates in which a certain group of nodes *z* (from a hierarchical level *y* of an AH-graph *x*) are involved. *GroundLevel(x)* returns the ground level of the AH-graph *x*. *StateAbstraction(x,y,z)* returns the *z-th* abstraction for a given state *y* of an AH-graph *x*. *PlanParameters(x)* returns the list of parameters that appear within a plan *x*. Finally, the *Subnodes(x,y)* procedure returns the subnodes of a node *y* in the AH-graph *x*.

According to HPWA-1, the "take box" task is planned as follows:

i) Planning at the universal level L^2

At the ground level the predicate that represents the goal state is (*holding Box*) (operator definitions are as shown in figure 7). This state is abstracted up to the universal level L^2 as (*holding Box''*).

At the universal level, the predicates that model the initial state of the world are:

(at Box'' Floor)
(at Robot'' Floor)

The embedded planner is run using s_s^2 and it finds the following solution (see the operator definition shown in figure 7):

(PICKUP Box'' Floor)

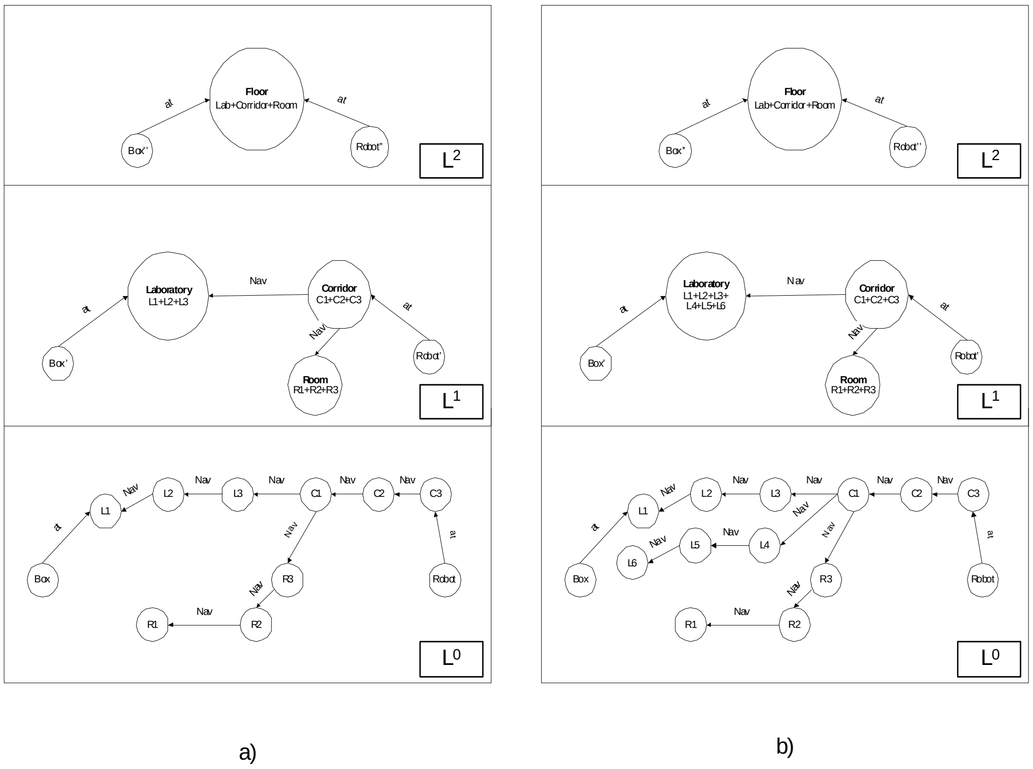

Fig. 6. Example of two different hierarchical models of a robot environment. For clarity, from the hierarchical level L^0 upwards, each node label includes its subnodes. For example, in a), the *Laboratory* node is the supernode of *L1, L2,* and *L3*.

```
(operator
 GO
  (params (<x> LOCATION) (<y> LOCATION) )
  (preconds      (at-robot <x>) (nav <x> <y>)  )
  (effects        (del at-robot <x>)  (at-robot <y>)))

(operator
  PICKUP
  (params (<x> OBJECT) <y> LOCATION ))
  (preconds     (at-robot <y>)(at-object <x> <y>))
  (effects        (holding <x>)(del at-object <x> <y>)))
```

Fig. 7. Definition of the operators used in this example. Observe that they involve several preconditions and effects.

ii) Planning at level L^1

Now, the abstract plan is moved down (refined) to level L^1, where only the subnodes of nodes of L^2 that take part in the plan are considered. In this particular case no node is ignored. At level L^1, the predicates that model the state of the world are:

(at Box' Laboratory)
(at Robot' Corridor)
(nav Corridor Laboratory)
(nav Corridor Room)

while the goal state is *(holding Box')*

A solution for this task provided by the embedded planner is:

(GO Corridor Laboratory)
(PICKUP Box' Laboratory)

Observe that the world element "Room" does not appear in the plan (it is irrelevant for the task[10]) and therefore, at the next lower level (ground level), its subnodes will be discarded.

iii) Planning at the ground level L^0

Finally, at the ground level, the predicates that model the initial state of the world are:

(at Box L1)
(at Robot C3)
(nav C3 C2)
(nav C2 C1)
(nav C1 L3)
(nav L3 L2)
(nav L2 L1)

Observe that no node within the *Room* is included in this world state, hence reducing the computational cost of planning. (This is only an illustrative example: more extensive results on the reduction in computational cost are presented in section IV).

The goal state is *(holding Box)* and the plan at this level results:

(GO C3 C2)
(GO C2 C1)
(GO C1 L3)
(GO L3 L2)
(GO L2 L1)
(PICKUP Box L1)

[10] Notice that although the *Robot* element does not appear in plans, it is an implicit world element that cannot be removed.

These actions are primitive actions that the mobile robot can execute, so this is the final plan for the initial task at hand.

C. Second HPWA Method: Planning With Plan Guidance

In this section we go a step further in the HPWA framework by providing a more sophisticated implementation called *HPWA-2*. The main difference to HPWA-1 is that HPWA-2 uses the abstract plans not only to ignore irrelevant elements at lower hierarchical levels of the world representation, but also to guide refinement into those levels. This can lead, in normal situations, to a higher reduction in computational cost; however, due to the poor look-ahead of this refining process, HPWA-2 is more sensitive to *backtracking* than HPWA-1[11]. Figure 8 sketches the HPWA-2 algorithm.

HPWA-2 starts from an abstracted plan, taking each action individually in order to refine it at the next lower level. This action refinement is in fact a refinement of its post-conditions, which consists of substituting each parameter involved in that post-condition (corresponding to a node of the AH-graph) by one of its subnodes. In our implementation, the subnode is selected randomly, although different heuristics could be used, like selecting a *border* subnode (one that is connected directly by arcs to a subnode of a different supernode), or the subnode with the highest order (that is, one which is connected the most to other ones). Other techniques based on *constraint propagation* are also applicable to guide the subnode selection process[4].

[11] Notice that, as in HPWA-1, our automatic construction method for obtaining AH-graphs will tend to avoid backtracking since it leads to increase the computational cost.

The advantage of refining the abstract plan in this way is that the amount of information necessary to refine a unique action is generally much smaller than the information required to plan with all relevant subelements. However, backtracking in HPWA-2 is more likely to occur.

We illustrate the operation of this method with the world of figure 6b, where the ground level is a slight variation of example 6a. Since the plan at L^2 coincides with the one of section III.B, we start from the abstract plan at level L^1:

(GO Corridor Laboratory)
(PICKUP Box' Laboratory)

The action *(GO Corridor Laboratory)* has the post-condition *(at-robot Laboratory)*. To refine this action its post-condition must be refined, so a subnode of *Laboratory* must be chosen. The procedure to choose a subnode can be implemented heuristically, randomly, or in another way (as commented earlier), but all of these possibilities may produce wrong elections that make the plan to fail and then, the algorithm must backtrack to select another subnode. For example, if subnode *L6* is chosen, the refined post-condition is *(at-robot L6)*. Obviously, with this election, a plan for *(holding Box)* will not be found. In this case, HPWA-2 must choose another subnode until it finds a feasible plan. If another node is selected, for this action, i.e. L2 (at-robot L2), the action (GO Corridor Laboratory) can be performed as:

(GO C3 C2)
(GO C2 C1)
(GO C1 L3)
(GO L3 L2)

In this world state, the next action can be planned successfully: the post-condition of the pickup action, (holding Box') is refined using its unique node to (holding Box''), which produces the next plan:

(GO L2 L1)
(PICKUP Box L1)

```
PROCEDURE HPWA-2 (State $s_g$, AH-graph $ah$):Plan
n=NodesFromLevel($L^w$)
pred= TransformWorldToPredicates($ah$,$L^w$,$n$)
post=StateAbstraction($ah$,$s_g$,$w$)
plan=EmbeddedPlanner(pred,post,operations)
newplan=NULL

FOR $i = L^w - 1$  DOWNTO GroundLevel($ah$)
   action=FirstActionPlan(plan)

   WHILE (action!=NULL)
        r_action=ActionRefinement(i,action)
        IF (r_action==NULL)
            action=PreviousActionPlan(plan)
        ELSE
         pred=ActionPredicates(i,r_action)
            post= ActionPostcondition(i, r_action)
            planaux=EmbeddedPlanner(pred,post,operations)
            IF (planaux!=NULL)
                newplan=Concatenate(newplan, planaux)
                action=NextActionPlan(plan)
            ELSE
              newplan=EliminateLast(newplan)
               action=PreviousActionPlan(plan)
            END
        END
  END
  plan=newplan
END
RETURN (plan)
```

Fig. 8. Pseudocode of HPWA-2. Functions and procedures used in this pseudocode are: TransformWorldToPredicates(x,y,z) produces the logical predicates related to a group of nodes z, from a hierarchical level y of an AH-graph x. GroundLevel(x) returns the ground level of the AH-graph x. StateAbstraction(x,y,z) returns the z-th abstraction for a given state y of an AH-graph x. FirstAction(x), NextAction(x) and PreviousAction(x) return the first, next and previous action, respectively, from the plan x. ActionRefinement(x,y) chooses a possible refinement for the action y at the hierarchical level x; if no refinement is available, return NULL. ActionPredicates(x,y) returns the pre-conditions set of the action y at the hierarchical level x. EliminateLast(x) deletes the last added plan from the plan x. Finally, PostCondition(x,y) function returns the post-conditions of the action y at the hierarchical level x.

D. Planning Anomalies

In any task planner there exist some anomalies that must be studied. In this section, two of them are analyzed in the context of the HPWA methods presented.

The Sussman's Anomaly consists of losing part of the achievements (goals) previously stated by the planner when it intends to achieve new ones [47]. This problem is not directly related to abstraction, although it could appear in any abstraction-based planner. The essential condition under which a given planner is sensitive to this anomaly is that the goals of a plan are achieved separately and considered independently [47]. Since the HPWA methods use an embedded planner that solves a given plan in only one, atomic operation, no separation of goals is carried out. Thus, as long as the embedded planner does not suffer from the Sussman's Anomaly, HPWA does not either.

A slightly different anomaly that can be present in any abstraction planner appears when an abstract plan loses its truth at refinement, that is, the refinement of an abstract plan makes false some achieved abstract goal. A way to prevent this, as proposed in [30], is to impose a strong requirement on the hierarchy of abstraction. The method presented in this paper for constructing AH-graphs does not take into account this anomaly, but the only effect of it in HPWA is not to find a refinement for the abstract plan, since no "false" or incorrect plan can be produced. In these cases, backtracking will take care of redoing the abstract plan (finding a different one) and trying again.

In addition, C.L.AU.D.I.A. will tend to avoid hierarchies that produce this type of problem, since they will be discarded during optimization due to their high computational costs for task planning.

IV. IMPLEMENTATION AND RESULTS

This section describes some software details and numerical results of experiments with the above planning methods, when applied to larger and more realistic worlds than that of figure 6.

A. Implementation Details of HPWA

Since we intend to use HPWA for real applications, HPWA-1 and HPWA-2 have been implemented on a distributed software architecture built upon a system for integrating robotic software modules. These modules communicate through a real-time implementation of the CORBA specification for distributed applications [24] called ACE+TAO [45] which provides portability and extendibility to our mobile robot software [18].

The software modules implemented for our experiments consists of the following:

- *AH-graph Manager*. It creates, modifies, and maintains the AH-graph model that represents the world.
- *AH-graph / Predicates Translator*. This module takes information from the AH-graph manager module and generates all predicates that represent the world state corresponding to a given hierarchical level.
- *Embedded Planner*. It is a wrapper for the task planner being used (Graphplan or Metric-FF), that communicates it to other modules of the architecture.
- *Hierarchical Task Planning Module*. This module implements the world abstraction planning methods presented in section III (HPWA-1 and HPWA-2).

B. Experiments and results

Planning complex tasks in a real office environment, as the one depicted in figure 9, is a challenge for any intelligent robot. This type of scenario may be composed of a large number of rooms and corridors where a mobile robot (perhaps equipped with a robotic arm) can perform a certain number of operations. Notice that we are dealing daily with even much more complex environments, with much more rooms, floors, and buildings, etc.

Within this environment, a person can ask the robot to pick up her/his mail (i.e., an envelope) from her/his office. The requested mail can be stacked, so the robot may need to unstack other envelopes on top of it before taking it. Once the mail is delivered, the robot must go to a specific place to recharge energy (the *supply room*). For the sake of simplicity we assume that all doors are opened. The definitions of the operators used in the tests presented in this section are more complex than the ones shown in figure 6, since they include *unstack object* and *recharge energy* operators.

The robot world hierarchy used by our hierarchical planning methods is shown in figure 10. This world representation resembles the human cognitive map [34]. The ground level contains the maximum amount of detail: every distinctive place, environment objects and their relations. The first level groups places and objects to form the human concept of rooms and corridors. Upper hierarchical levels represent groups of rooms or areas, and finally, the universal level contains a single node that represents the whole environment. As commented earlier in section II, the world hierarchy can be automatically constructed and maintained by C.L.A.U.D.I.A.

Within this scenario, we compare the Graphplan and Metric-FF conventional planners to the implementation of HPWA that uses them as embedded planners. Also, an implementation of Graphplan that uses the hierarchies produced by the ABSTRIPS algorithm [44] has also been developed, in order to compare our planning results to other hierarchical approach.

The first experiment consists of planning a single fixed task while varying the complexity of the scenario. The planned robot task is to pick up a given envelope from the *mail room* (see fig. 9). This task requires the robot to navigate to this room, unstack other envelopes (if needed), and then pick up the requested envelope. The complexity of the scenario ranges from 6 to 48 rooms. Figure 11 shows a comparison of the computational costs of planning using HPWA-1 and HPWA-2 with Graphplan as embedded planner against the Metric-FF and GraphPlan conventional planners alone, and against the hierarchical version of Graphplan with ABSTRIPS. Observe that, without using abstraction, the computational cost grows exponentially with the number of rooms; however, HPWA-1 and HPWA-2 spend a constant time since the new rooms added to the environment, which are irrelevant for the task, are promptly discarded by the abstract planning process.

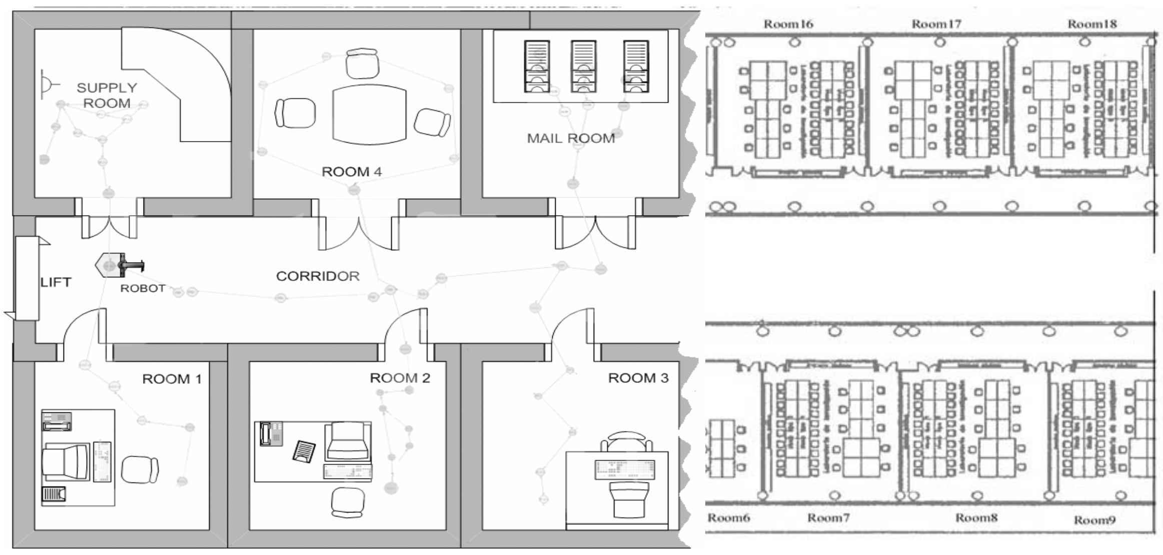

Fig 9. A large-scale office environment composed of many rooms. On the left, a detailed portion of the whole environment is zoomed. Part of the hierarchical ground level has been overprinted in order to clarify this partial view.

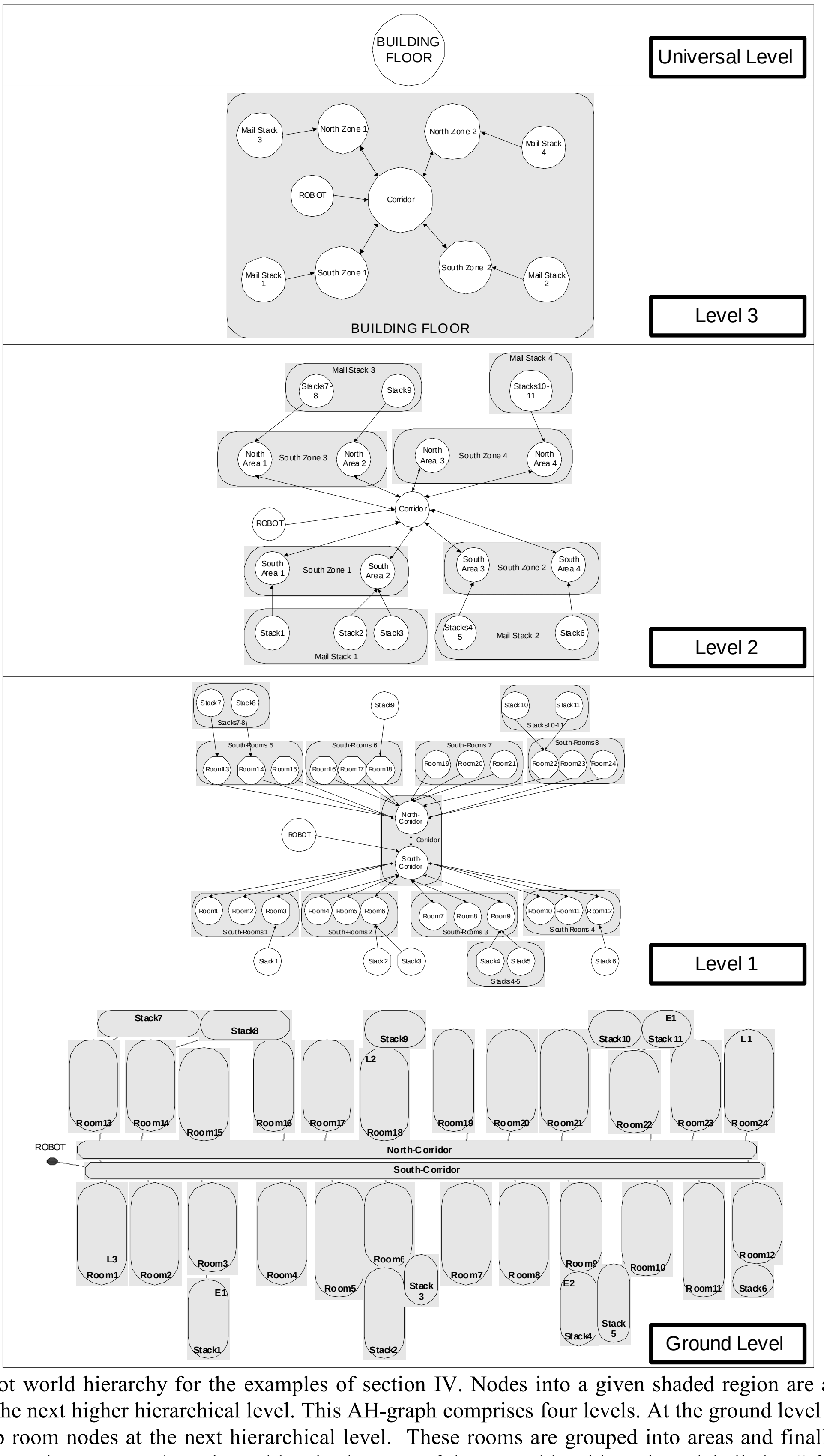

Fig 10. The robot world hierarchy for the examples of section IV. Nodes into a given shaded region are abstracted to the same supernode at the next higher hierarchical level. This AH-graph comprises four levels. At the ground level the elements are grouped to make up room nodes at the next hierarchical level. These rooms are grouped into areas and finally, a single node represents the whole environment at the universal level. Elements of the ground level have been labelled "E" for envelopes and "L" for destination locations.

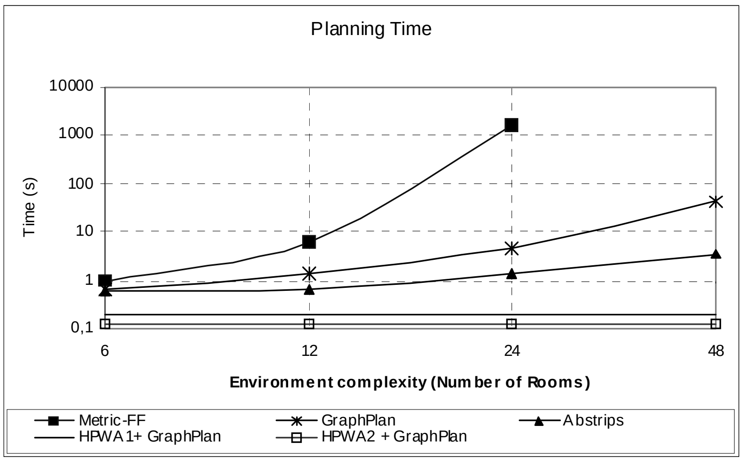

Fig. 11. Planning a single fixed task in an increasing complexity environment (ranging from 6 to 48 rooms). While the CPU planning time grows exponentially using the conventional planners considered here (hierarchical and non-hierarchical), both HPWA methods exhibit a constant CPU time, since irrelevant information for the task is discarded. This plot only shows the time spent by planners: neither pre- or post-processing, nor communication burdens have been added. No result is shown for Metric-FF for 48 rooms due to the large computational resources demanded.

A second experiment is aimed to test the method in a medium-complexity environment (with 24 rooms) where six arbitrary tasks are planned. The first three tasks are "*take envelope E*" and the others "*carry envelope E to location L*" (see fig. 12). The elements involved in the tasks (*E* and *L*) have been selected arbitrarily from the robot world. All tasks involve navigation and manipulation operations, and some of the goals are reached in more than 40 actions. The whole robot world is composed of more than 250 distinctive places, 30 different objects that the robot can manipulate, and all the navigation and manipulation relations existing between these elements.

Task #1	Take Envelope E1
Task #2	Take Envelope E2
Task #3	Take Envelope E3
Task #4	Carry Envelope E1 to L1
Task #5	Carry Envelope E2 to L2
Task #6	Carry Envelope E3 to L3

Fig. 12. The six tasks planned for the second experiment. Three different objects (envelopes) and three locations of the environment have been chosen to test the hierarchical planning methods versus conventional planning. Please, see figure 10 to find these objects and locations in the ground level of the AH-graph.

Fig. 13 shows the results of this experiment using HPWA-1 and HPWA-2 with Graphplan as embedded planner, HPWA-1 with Metric-FF[12], a hierarchical version of Graphplan using the hierarchies produced by ABSTRIPS, and the conventional planners alone (GraphPlan and Metric-FF). Each chart shows the results of planning a task from the ones shown in fig. 12. It is clear the computational benefit of hierarchical planning through world abstraction against both non-hierarchical planning and ABSTRIPS. Also, notice that planning time is not shown for the last three tasks (Task 4, Task 5 and Task 6) for Graphplan planner and ABSTRIPS, because Graphplan was not able to find a plan due to the large computational resources demanded and ABSTRIPS fails in finding a correct plan that solves the tasks due to violation of previously achieved preconditions when refining a plan (the same problem is reported in [28]). In all these plots we only consider the time spent by the embedded planner, without taken into account the pre-processing time taken by the *AH-Graph manager* and *AH-Graph/Predicates Translator*, and the time spent in CORBA communications. Notice how HPWA-2 performs better than HPWA-1 in all tasks, except for the first two ones because of backtracking burden. Figure 14 compares the total time (which accounts also for pre-processing and communications) of HPWA-1 versus HPWA-2 when using Graphplan as embedded planner. In this case, HPWA-2 performs more inefficiently since backtracking does not only increase the number of plans to solve, but it also multiplies that increment by the CORBA communication and the predicate translation times. Also, observe how the total time of HPWA-1 is quite similar to its planning time (figure 13), since the additional cost of this method is negligible.

[12] HPWA-2 is not yet adapted for Metric-FF. Metric-FF planner requires a more sophisticated management of the world information that has not been considered for this work, remaining as one of the future goals in our research.

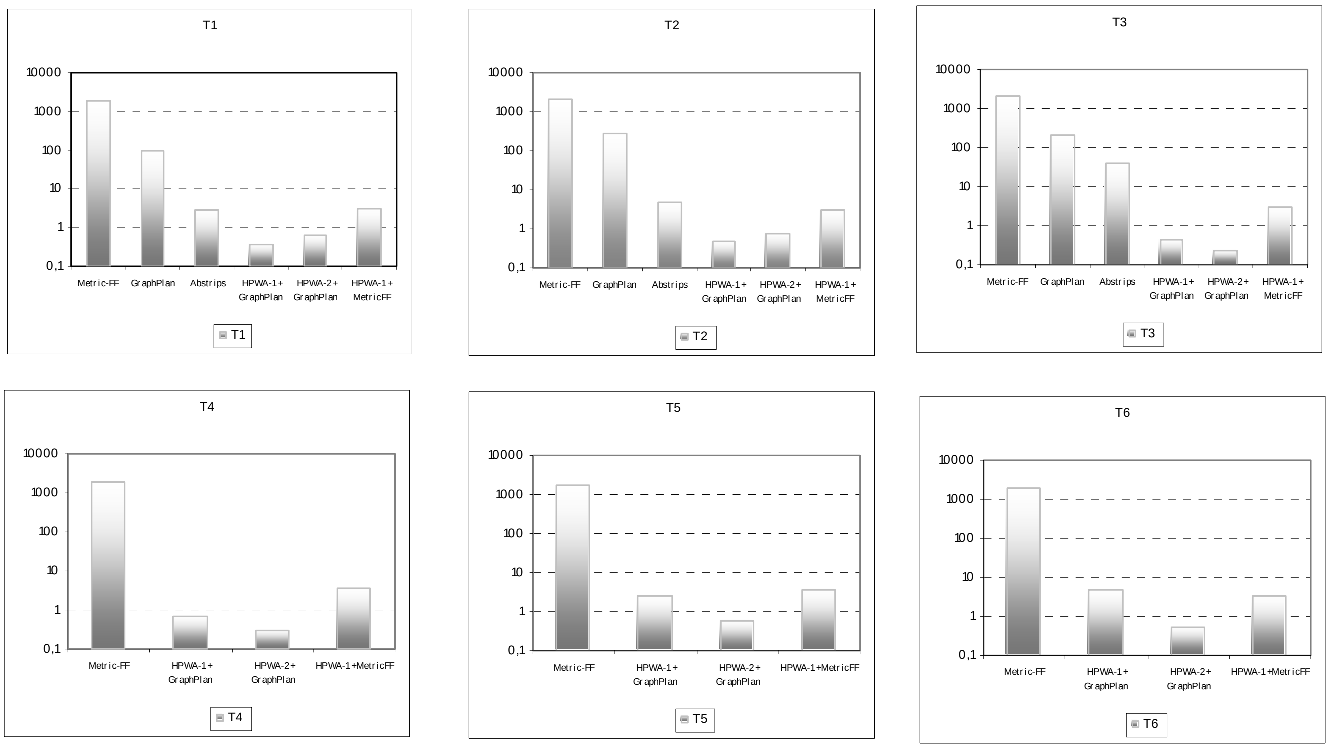

Fig. 13. Planning time for the proposed tasks (in seconds) in a Pentium IV at 1.7 MHz with 512 Mbytes of RAM. Both non-hierarchical and hierarchical planners exhibit worse CPU time than HPWA methods. Moreover, HPWA-2 exhibits the best planning time for all tasks except for the first two ones due to backtracking situations. Notice that the planning time is not shown for Graphplan and ABSTRIPS in the last three tasks, because they were not able to end up with a solution. In all these plots we consider only the time spent by the embedded planner, without taken into account the pre-processing time, CORBA communications, etc.

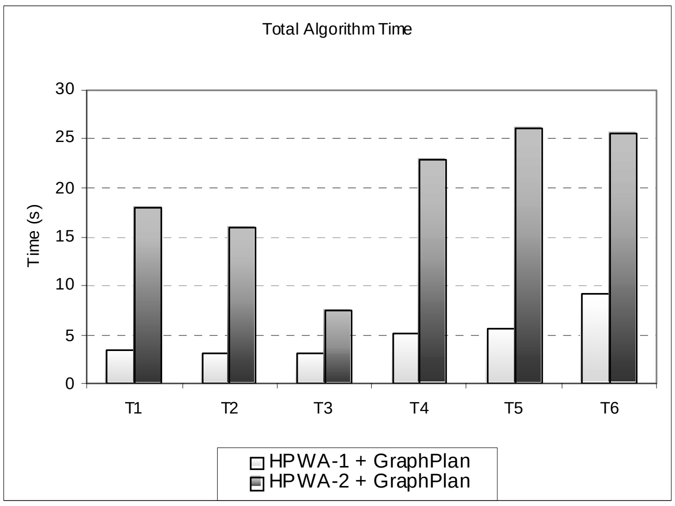

Fig. 14. Total time spent by our HPWA-1 and HPWA-2 implementations with Graphplan as embedded planner. Time of HPWA-2 is higher than HPWA-1 due to the extra pre-processing and CORBA communication costs involved in backtracking. In spite of this apparently bad result, HPWA-2 performs better than other approaches in fig. 13, even when they do not include the pre-processing and communication burden considered in this plot.

V. CONCLUSIONS AND FUTURE WORK

In AI literature, task planning has not dealt with large-scale spaces, which is a common situation in mobile robotics. In these cases, task planning, even with the most modern improvements, can exhibit a high computational cost, even be intractable (as shown in fig. 13). This paper has introduced a new scheme for task planning that takes advantage of a hierarchical arrangement of the robot model of the environment. This approach, called *Task Planning through World Abstraction,* performs planning at a high level of abstraction of the world and then, recursively, refines the resulting abstract plan at lower levels, ignoring irrelevant world elements that do not take part in the abstract plan. *HPWA* has been stated and formalized upon a graph-based hierarchical structure called *AH-graph,* although other hierarchical models could also take advantage of the general approach.

In particular, we have described two implementations of HPWA that embed other existing planners (Graphplan and Metric-FF) to solve planning at each level of abstraction of the world. Thus, we can benefit from any other kind of improvements on the embedded task planner, even abstraction (of operators, tasks, etc.). We have shown how our two implementations of HPWA have performed better than the embedded planners alone and other hierarchical approaches as ABSTRIPS. This good performance is tightly coupled with the use of adequate hierarchies of abstraction. Adequate hierarchies can be obtained by using the task-driven paradigm for automatic construction of abstractions outlined in section II and presented with more detail in [15].

Since world abstraction is intuitive and applicable to most real robotic environments, we believe this work may be of a great value in those task planning problems where the number of world elements makes conventional planning strategies impractical. In addition, other areas different from robotics can benefit from our framework provided that the world can be represented in a graph-like manner and the problem supports the progressive optimization carried out by C.L.A.U.D.I.A.

Our short term research aims to use the HPWA framework with other kinds of embedded planners, like decision-theoretic ones and others. Also, we are working on obtaining predicates not only from nodes and arcs, but also from non-structural information held by the AH-graph.

Our ultimate goal is the complete integration of the HPWA methods into a mobile robotic architecture, addressing the problem of interleaving planning and execution.

REFERENCES

[1] Alami R., Chatila R. (1998). *An Architecture for Autonomy*. International Journal of Robotics Research, Vol. 17, No. 4.

[2] Anderson C.R., Smith D. E., and Weld D.S. (1998). *Conditional Effects in Graphplan*. 4th Int. Conf. AI Planning Systems, Pittsburgh Pennsylvania, USA.

[3] Asada M., Fukui, Y., and Tsuji, S. (1990). *Representing Global World of a Mobile Robot with Relational Local Maps*. In IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, No.6.

[4] Baioletti M., Marcugini S., and Milani A. (2000). *DPPlan: an Algorithm for Fast Solutions Extraction from a Planning Graph*. Artificial Intelligence Planning Systems, Breckenridge, USA.

[5] Baker K. *Introduction to Sequencing and Scheduling*. (1974). Wiley, New York.

[6] Barish G. (1998). *Approaches to Integrating Abstractions In Graphplan-based Planning System*. Universtity of Southern California. CS541: Artificial Intelligence Planning. Final Project, December 11.

[7] Blum A.L. and Furst M.L. (1997). *Fast Planning Through Planning Graph Analysis*. Artificial Intelligence Vol. 90.

[8] Brooks, R.A. (1986). A robust layered control system for a mobile robot. IEEE J. Rob. Automation Vol. 2.

[9] Christensen J. (1990). *A Hierarchical Planner that Generates its Own Hierarchies*. 8th National Conf. On Artificial Intelligence, Boston, USA.

[10] Davis P.K., and Hillestad R.J. (1993). *Families of Models that Cross Levels of Resolution: Issues for Design, Calibration, and Management.* Winter Simulation Conference of the ACM. Los Angeles, USA.

[11] Delchambre A. and Gaspart P. (1992). KBAP: *An industrial prototype of knowledge-based assembly planner.* IEEE Intl. Conf. On Robotics and Automation, Nice (France).

[12] Erol K, Hendler J, and Nau D. (1994). *Semantics for Hierarchical Task network Planning*. Technical Report CS-TR-3239, Dept. of Computer Science, U. Maryland.

[13] Fernández J.A. and González J. (1998). *Hierarchical Path Search for Mobile Robot Path Planning.* IEEE International Conference on Robotics and Automations (ICRA), Leuven (Belgium).

[14] Fernández J.A. and González J. (1998). *Mobile Robot Path Planning and Navigation Using Hierarchical Graphs*. ICSC Symposium on Engineering of Intelligent Systems (EIS), Tenerife (Spain).

[15] Fernández J.A. and González J. (2001). *Multi-Hierarchical Representation of Large-Scale Space*. International Series on Microprocessor-based and Intelligent Systems Engineering. Vol. 24. Kluwer Academic Publishers, Netherlands .

[16] Fernández J.A. and González J. (2002). *Multihierarchical Graph Search*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, nº 1.

[17] Fernández J.A. and González J. (2002). *Task-driven, Multiple Abstraction for Modeling Mobile Robot Large-scale Space*. 15th IFAC World Congress on Automatic Control, Barcelona (Spain).

[18] Fernández J.A., González J., Galindo C., Reina A., and Muñoz A. (2002). *Assistive Navigation using a Hierarchical Model of the Environment.* 3rd International Congress on Engineering Intelligent Systems (EIS), Málaga (Spain).

[19] Fikes, R., and Nilson, N.J. (1971). *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence, Vol. 2.

[20] Fink E., Yang Q. (1993). *A Spectrum of Plan Justifications*. AAAI Spring Symposium, Standford, California.

[21] Giunchiglia E. and Walsh T. (1992). *A Theory of Abstraction*. In Artificial Intelligence, Vol 56 nº 2-3.

[22] Giunchiglia F., Walsh T. (1991). *Using Abstraction*. 8th Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour, Leeds, UK.

[23] Hendrickson B. and Leland R. (1993). *Multidimensional Spectral Load Balancing*. 6th SIAM Conference on Parallel Processing and Scientific Computation. Norfolk, Virginia, USA.

[24] Henning M., Vinoski S. (1999). *Advanced CORBA Programming with C++*. Addison Wesley Longman, Inc.

[25] Hoffmann Jörg and Nebel Bernhard. (2001). *The FF Planning System: Fast Plan Generation through Heuristic Search*. Journal of Artificial Intelligence Research, Vol. 14.

[26] Knoblock C.A. (1991). *Search Reduction in Hierarchical Problem Solving*. 9th National Conference on Artificial Intelligence, California, USA.

[27] Knoblock C.A. (1990). *Learning Abstraction Hierarchies for Problem Solving*. 8th National Conference on Artificial Intelligence, Boston, USA.

[28] Knoblock C.A. (1992). *An Analysis of ABSTRIPS*. Artificial Intelligence Planning Systems: Proceedings of the First International Conference, AIPS, Morgan Kaufmann.

[29] Knoblock C.A. (1993). *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning*. Kluwer Academic Publishers, Norwell, MA.

[30] Knoblock C.A. (1994). *Automatically Generating* Abstractions *for Planning*. Artificial Intelligence, Vol. 68, nº2.

[31] Koehler J., Nebel B., Hoffmann J., and Dimopoulos Y. (1997). *Extending Planning Graphs to an ADL Subset*. Institute for Computer Science Albert Ludwigs University. Technical Report No. 88.

[32] Koening S., Goodwing R., and Simmons R.. (1996). *Robot navigation with Markov models: A framework for path planning and learning with limited computational resources.*. In Dorst, L. van Lambarlge, M., and Voorbraak, R., eds., Reasoning with Uncertainty in Robotics, volume 1903 of Lecture Notes in Artificial Intelligence. Springer.

[33] Korf, Richard E. (1987). *Planning as search: A quantitative approach*. Artificial Intelligence, Vol. 33, nº1.

[34] Kuipers B.J. (1983). *The Cognitive Map: Could it Have Been Any Other Way?*. In Spatial Orientation: Theory, Research, and Application. Picks H.L. and Acredolo L.P. (eds.), New York, Plenum Press.

[35] Kuipers B.J. (2000). *The Spatial Semantic Hierarchy*. In Artificial Intelligence Vol. 119.

[36] Kuipers B.J. and Byun Y.T. (1987). *A Qualitative Approach to Robot Exploration and Map-Learning*. IEEE Workshop on Spatial Reasoning and Multi-Sensor Fusion. Los Altos, California.

[37] Latombe J.C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Boston.

[38] Minto S., Carbonell J.G., Knoblock C.A., Kuokka D.R., Etzioni O., and Gil Y. (1989). *Explanation-based learning: A problem solving perspective*. Artificial Intelligence, 40 (1-3).

[39] Mkadmi T. (1993). *Speeding Up State-Space Search by Automatic Abstraction*. Master's thesis, University of Ottawa.

[40] Park I-P, and Kender J.R. (1995). *Topological Direction-Giving and Visual Navigation in Large Environments*. In Artificial Intelligence, Vol. 78.

[41] Perona P. and Freeman W. (1998). *A Factorization Approach to Grouping*. Lecture Notes in Computer Science, Vol. 1406.

[42] Peterson G.L, and Cook, D.J. (2003). *Incorporating decision-theoretic planning in a robot architecture*. Robotics and Autonomous Systems 42.

[43] Reynolds P.F.J., Natrajan A., and Srinivasan S. (1997). *Consistency Maintenance in Multi-Resolution Simulations*. In ACM Transactions on Computer Modeling and Simulation, vol. 7, no. 3.

[44] Sacerdoti E.D. (1974). *Planning in a Hierarchy of Abstraction Spaces*. Artificial Intelligence, Vol. 5, nº2.

[45] Schmidt D., ACE+TAO Corba Homepage. http://www.cs.wustl.edu/~schmidt/TAO.html.

[46] Smith D.E., Frank J., and Jónsson A.K. (2000). *Bridging the Gap Between Planning and Scheduling*. Knowledge Engineering Review, 15 (1).

[47] Sussman G.J. (1975). *A computer model of skill acquisition*. American Elsevier Publishing Company.

[48] Tamaki H., Murac H., Kitamura S. (2002). *Decision-making strategies for decentralized production planning and scheduling*. IEEE International Symposium on Industrial Electronics, L'Aquila (Italy).

[49] Trudeau R. J. (1993). *Introduction to Graph Theory*. Dover Publications, New York.

[50] Veloso M, Carbonell J. (1995). *Integrating planning and learning: the prodigy architecture*. Journal of Experimental and Theoretical AI. Vol. 7, nº 1.

[51] Walker A., Hallam J., and Willshaw D. (1993). *Bee-havior in a Mobile Robot: The Construction of a Self-Organized Cognitive Map and its Use in Robot Navigation within a Complex, Natural Environment*. IEEE International Conference on Neural Networks, San Francisco, California.

[52] Yang Q. (1997). *Intelligent Planning: A Decomposition and Abstraction Based Approach*. Springer Verlag, Berlin, Germany.

[53] Zweben M. and Fox M. (1994). *Intelligent Scheduling*. Morgan Kaufmann.