Synthesizing Plans for Multiple Domains

Abdelbaki Bouguerra and Lars Karlsson*

Applied Autonomous Sensor Systems Center,
Örebro University, Sweden
{abdelbaki.bouguerra, lars.karlsson}@tech.oru.se

Abstract. Intelligent agents acting in real world environments need to synthesize their course of action based on multiple sources of knowledge. They also need to generate plans that smoothly integrate actions from different domains. In this paper we present a generic approach to synthesize plans for solving planning problems involving multiple domains. The proposed approach performs search hierarchically by starting planning in one domain and considering subgoals related to the other domains as abstract tasks to be planned for later when their respective domains are considered. To plan in each domain, a domain-dependent planner can be used, making it possible to integrate different planners, possibly with different specializations. We outline the algorithm, and the assumptions underlying its functionality. We also demonstrate through a detailed example, how the proposed framework compares to planning in one global domain.

1 Introduction

A considerable amount of work in AI planning focuses on the use of abstraction to reduce the search space, where planning takes place at successive levels of more details. Hierarchical planning is such a paradigm that relies on abstraction and goal decoupling to produce effective plans [14],[17],[12],[5]. The planning problem is specified as a set of abstract tasks to achieve with ordering constraints over them. The planning process, repeatedly, refines the abstract tasks into more detailed tasks until the plan is composed only of executable tasks. Abstractions have mainly been supplied by the user as part of the knowledge bases used by the planner. However different approaches have been proposed to learn abstractions from the description of the planning problems [8],[4].

Decomposition of the planning problem into subproblems is also an approach aiming at reducing search complexity [16],[1]. The partitioning of the initial planning problem focuses on producing subproblems with minimum interaction in order to be able to find an efficient solution. It is worth noting that problem partitioning is generally combined with abstraction techniques to control the interaction between the different sub-components of the planning problem [10],[11].

* This work has been supported by the Swedish KK foundation and the Swedish research council.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 30–43, 2005.

In this paper we propose a framework to synthesize plans to solve planning problems involving multiple domains through the use of abstraction and goal ordering. In fact, goal and subgoal ordering approaches have been demonstrated to be effective ways in solving planning problems [13]. As demonstrated in [9], a total order relation between increasing sets of goals allows incremental planning by focusing search on the goals that appear earlier, leading to improved planning performances.

To solve problems involving multiple domains, one can envisage to solve in each domain the portion of the problem related to it and then glue all the results in a global plan, but doing so might result in degraded execution of the overall plan, because of the localized reasoning. Our work on plan execution on board mobile robots has motivated us to find a general approach that can utilize reasoning over different domains using different planners for problem solving so the resulting plan would execute smoothly and efficiently. Therefore the proposed approach guarantees to find a plan in an incremental way that interleaves actions from the different domains when only it is needed.

The general idea of our approach is to act on a set of domains ordered according to which domain gets its goals achieved earlier. The planning problem is solved incrementally starting with the leftmost domain, and going all the way to the last one. If a domain D, ordered before another domain D', needs to accomplish a subgoal involving the actions of D', then D places a request in the plan for D' to accomplish the desired subgoal. At a later stage, when planning to solve in D', the algorithm can use the actions of D' to solve the request. The planner used to plan in a particular domain can be a domain-(in)dependent planner, meaning that it is possible to integrate different efficient specialized planners to solve the global planning problem.

In the next section we detail the assumptions used to find a plan in multiple domains as well as operator transformation to reflect the interaction between the involved domains. Next we give a global overview of the approach and how planning in one domain introduces ordered abstract tasks to be solved in the subsequent domains. Section 4 outlines the hierarchical algorithms used to solve the multiple-domain planning problem. Before concluding we demonstrate the performance of the approach on two domains.

2 Domains Interaction

In this section we discuss the assumptions underlying the interactions between the planning domains, as well as the extensions to be made to the syntax of planning operators in order to be able to use the proposed framework. As stated before, the approach supposes that the agent has access to a planning system employing a domain-independent planner or a set of domain-dependent planners.

Domains are defined in the usual way as consisting of a set of operators and a set of fluents, where the operators of one domain can use literals from other domains in their preconditions and effects. We use an incremental approach to plan in the different domains i.e. when trying to solve a planning problem in a

particular domain, the achieved goals, of solved problems of the other domains, have to be maintained. This incremental approach makes it possible to reduce search complexity, because goals do not have to be established, then destroyed, then established again [9]. This restriction leads us to the following consideration:

- **Consideration 1.** If an operator in a domain D_1 uses in its preconditions a literal l from another domain D_2, then finding a plan to achieve goals in D_1 would violate achieved goals in D_2, because the literal l might be sub-goaled in D_1 by the corresponding plan. Therefore the framework has to make sure that all the goals of D_1 are planned for first before planning for the goals of D_2. Note that this is coherent with the ordered monotonic refinement property [8].

Example 1. Suppose that the planning system has two domains: navigation, and blocks. The navigation domain is used to move a mobile robot equipped with an arm between rooms and corridors. The blocks domain is used to rearrange blocks in towers in different rooms. To manipulate blocks in a room r_1, the robot has to be in room r_1 too. Therefore the blocks domain operators (pick-up, put-down, stack, and unstack) use a literal in their preconditions to impose such restriction. Now, suppose that the agent wants to achieve the two goals $g_1 = (\text{on } b\ a)$, and $g_2 = (\text{robot-at} = r_2)$ (where both blocks a and b are in room r_3). If the agent achieves g_2 first then solving g_1 will violate g_2; because the robot has to move to room r_3 to be able to stack b on a. Consequently, g_1 has to be achieved before g_2.

We also want to keep the effects involving fluents from a domain D under the control of D. The aim of this restriction is to localize planning within the domains. This leads us to the second consideration:

- **Consideration 2.** If an operator o in domain D_1 achieves as a side effect a literal l from another domain D_2, then an abstract version of o is created in D_2 where only the preconditions related to D_2 are maintained, and the effects related to D_1 are posted as a task to be achieved in D_1.

Example 2. Considering the previous example. In the navigation domain, moving the robot from one room to another while holding a block will also move the block as a side effect. Therefore an abstract version of this operator is created in the blocks domain to reflect this change to the state of the block.

2.1 Domains Representation

Let $\mathscr{D} = \{D_1, D_2, \cdots, D_m\}$ be the set of the domains used by the planning system. A domain D_i is defined as a set of fluents F_i and a set of operators O_i. An operator $o \in O_i$ is a couple $\langle Pre(o), Eff(o) \rangle$, where *Pre(o)* are the preconditions of the operator, and *Eff(o)* are the effects (positive and negative) of the operator. Following the consideration 1, the preconditions *Pre(o)* component has the following syntax:

$$\textit{Pre(o)} :: ((local : \phi_i) \\ (foreign : (D_j : \psi_j)^+)^*)$$

where ϕ_i is a formula written only over the (local) fluents of the domain D_i, and ψ_j is a formula written only over the fluents of domain D_j $(i \neq j)$. This syntax reflects that the preconditions of the operator o might have a local precondition expressed as a formula over the fluents F_i of the current domain, and a list of foreign preconditions defined over the fluents of the other domains.

If o is an abstract version of another operator in another domain D_j $(j \neq i)$ (consideration 2), then the effects of o have the following form:

$$\textit{Eff(o)} :: ((local : eff_i) \\ (foreign : (D_j : eff_j)^+)^*)$$

where eff_i are effects over the fluents of the local domain D_i and eff_j are effects over the fluents of domain D_j $(j \neq i)$.

Example 3. The following is an operator from the blocks domain to pick up a block on the table at a location specified by the variable ?*loc*.

(pick-up ?b ?loc):	
param:	?b - BLOCK, ?loc - LOCATION
Pre:	((local: (and (clear ?b)(on-table ?b)(arm-free)(object-at ?b = ?loc))) (foreign: (navigation: (robot-at = ?loc)))
Eff:	((local: (and (holding ?b)(clear ?b = f)(arm-free = f) (on-table ?b = f))))

The $foreign$ part of the precondition specifies that the fluent (robot-at = ?*loc*) from the navigation domain has to be satisfied before executing the operator i.e. the robot must be at the same location as the block.

Example 4. The operator (move $?l_1$ $?l_2$) from the navigation domain is used to move a robot from location $?l_1$ to another location $?l_2$. If the robot is holding a block ?*b*, then the block changes location too (i.e. (object-at ?*b* =$?l_1$)). So an abstract operator has to be created in the blocks domain to reflect this value change to the fluent object-at. When planning in the blocks domain, the abstract operator is used the same way as the other operators to cause state change, except that it does not appear in the plan as its real effects are achieved by the operator (move $?l_1$ $?l_2$) of the navigation domain (it is for this reason it is qualified as abstract).

(move-block ?b ?l1 ?l2): ABSTRACT	
param:	?b BLOCK, ?l1 ?l2 - LOCATION
Pre:	((local: (and (holding ?b)(object-at ?b = ?l1)))
Eff:	((local: (object-at ?b = ?l2)) (foreign: (navigation: (robot-at = ?l2))))

2.2 Ordering Domains

Consideration 1 imposes on the multi-domain plan synthesizer to find a total order on the set of domains $\mathscr{D} = \{D_1, D_2, \cdots, D_m\}$. A domain D_i is ordered before another domain D_j (noted: $D_i \prec D_j$) if at least one of the operators of D_i uses in its preconditions a fluent from D_j:

$$\exists f_j \in F_j, \exists o \in O_i : f_j \text{ appears in } Pre(o) \Rightarrow D_i \prec D_j.$$

The total order can be directly given by the user, or it can be extracted automatically as the result of a topological sort applied on a graph whose nodes represent the domains and arcs represent the constraint $\prec$.

3 Multiple-Domain Planning Overview

To be able to synthesize a plan to solve a planning problem involving the achievement of goals related to more than one domain, we need first to find a total order on the involved domains, and create abstract operators to fulfill the second requirement as described in the previous section. Once this is done, we can solve the planning problem in an incremental way: starting in the left-most domain we solve its planning problem, then the resulting plan is passed to the next domain (according to their order) where the abstract tasks related to the new domain are solved in the order they appear in the plan. This process continues until reaching the right-most domain where the plan would be completely refined.

3.1 Planning in One Domain

A planning problem in a domain D_i is specified by the initial state expressed as a conjunct of fluents from F_i, a goal state expressed as a first order logic formula, and the set of operators O_i. To solve a planning problem to achieve goals in domain D_i, the planner associated with D_i selects an instantiated operator $o \in O_i$ to insert in the plan, if its local preconditions are satisfied in the current state s defined over F_i. Since the foreign preconditions of the operator o have also to be satisfied in their respective domains, the planner prepends to o requests to achieve the foreign preconditions in their respective domains. If an operator has more than one foreign component, the planner has to make sure to order the requests according to the order of their domains. i.e. if the foreign components of the preconditions *Pre(o)* of o are $(foreign : (D_1 : \psi_1)(D_2 : \psi_2))$ such that $D_1 \prec D_2$, then as a result of selecting o, the planner inserts the following in the plan:

$$\texttt{(achieve } (D_1 : \psi_1)\texttt{)};\texttt{(achieve } (D_2 : \psi_2)\texttt{)} \ ; o$$

where `(achieve` $(D_i : \psi_i)$`)` formulates a request to achieve the conditions ψ_i in domain D_i.

If o is an abstract operator with foreign effects i.e. $(foreign : (D_j : eff_j)) \in$ *Eff(o)*, then the planner inserts those foreign effects as requests to be achieved in their respective domains i.e. `(achieve` $(D_j : eff_j)$`)`. Note that in this case, only the foreign effects of o are inserted in the plan as abstract tasks, but not o itself.

Each request, posted on a foreign component defines an abstract task to be achieved in its domain. This means that a plan might encompass unsolved abstract tasks inserted by the planners of the antecedent domains. The unsolved tasks are further refined when planning in the subsequent domains.

Example 5. The instantiated blocks domain operator (pick-up b_1 r_1) is applicable in a state s if the local formula (and (clear b_1)(on-table b_1)(arm-free)(block-at b_1= r_1)) holds in s. If it is selected by the planner then, the planner prepends it with the abstract task "`(achieve` (Navigation: (robot-at = r_1)`)`" which is solved later when planning in the navigation domain.

3.2 Planning in Multiple Domains

Figure 1 gives an overview of how to synthesize a plan involving three domains $\mathscr{D} = \{D_1, D_2, D_3\}$ such that $D_1 \prec D_2 \prec D_3$. The goals involving one domain are considered as an abstract task to achieve. Therefore, a task is created on the final goals of each domain, giving us three initial abstract tasks: $T_{init}(D_1), T_{init}(D_2), T_{init}(D_3)$ where each $T_{init}(D_j)_{j=1,2,3}$ is formulated as `(achieve` $(D_j$: goals of $D_j)$`)`. The initial version of the global abstract plan is created by ordering the three initial abstract tasks according to the order defined over their respective domains i.e. $T_{init}(D_1) \prec T_{init}(D_2) \prec T_{init}(D_3)$.

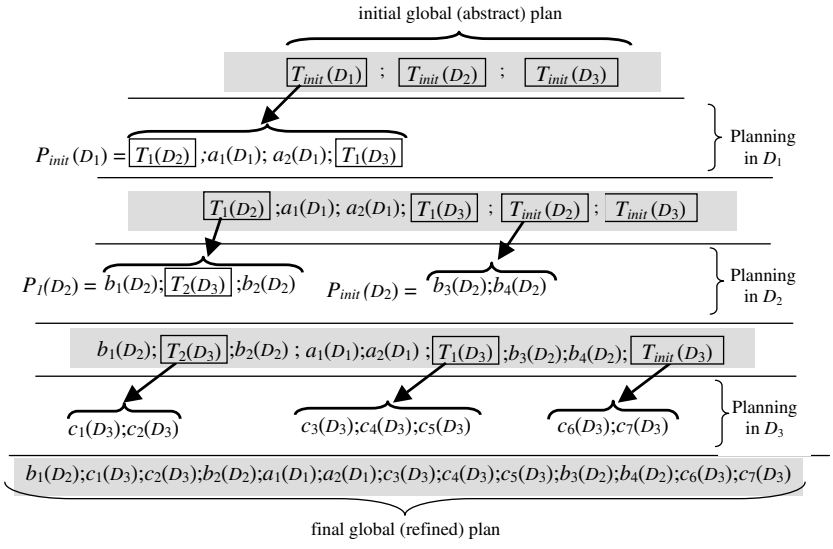


Fig. 1. Global view of planning in three domains

The plan synthesizing process proceeds top-down. It starts by planning in D_1 to solve $T_{init}(D_1)$ which yields a plan $P_{init}(D_1) = T_1(D_2); a_1(D1); a_2(D1); T_1(D_3)$ that contains two abstract tasks $T_1(D_2)$(resp. $T_1(D_3)$) to be achieved in D_2 (resp. D_3) and two instantiated operators of domain D_1: $a_1(D1)$ and $a_2(D1)$. $T_{init}(D_1)$ is then replaced by $P_{init}(D_1)$ to produce a global plan which is subsequently refined in D_2. When planning in D_2, the planner plans to solve the abstract tasks related to D_2 in the same order they appear i.e. it plans to solve $T_1(D_2)$ before planning to solve $T_{init}(D_2)$. The next step replaces the abstract tasks just planned for by their corresponding plans ($T_1(D_2)$ is replaced by $P_1(D_2)$, and $T_{init}(D_2)$ is replaced by $P_{init}(D_2)$). The resulting plan is next refined in the last domain D_3 where abstract tasks related to D_3 are solved and replaced by plans that contain only instantiated operators from D_3. At this stage the resulting plan is composed only of instantiated operators and it solves all the goals related to the three domains. Please note that there might be backtracking to the previous domain or just within a domain itself if a task can not be solved.

4 The Planning Algorithm

The multi-domain planning algorithm shown in Fig. 2. is a forward chaining algorithm. It takes as input a set of initial states S_0, a set of goals to achieve G, and a list of all the domains *Domains* given in the order defined in section 2. The elements of S_0 are the initial states of the domains involved in planning. G comprises goal sets, each of which is related to one domain in *Domains*.

The algorithm builds for every domain $D_i \in Domains$ a task which is simply expressed as "`(achieve` $(D_i : g_{D_i})$)", where $g_{D_i} \in G$ is the goal set related to domain D_i. An abstract version of the global plan $GlobalP$ is created by ordering the tasks of the different domains according to the order of their respective domains (the Init phase). After the initialization phase, the algorithm retrieves the first domain D from *Domains* (step 2), and extracts all the abstract tasks D_{Tasks} from $GlobalP$ related to D keeping them in the same order as they appear in $GlobalP$ (step 4). The procedure "Find-Plan" is called to compute a plan D_{plan} to solve the ordered list of abstract tasks D_{Tasks} in D (step 6). D_{plan} is actually a list of sub-plans each solving one task in D_{Tasks}. If all the tasks in D_{Tasks} are solvable in D (i.e $D_{plan} \neq fail$), then $GlobalP$ is refined by substituting every abstract task that appears in D_{Tasks} by the portion of the sub-plan that solves it (step 8) (this is a refinement step that iterates over the elements of $GlobalP$ to replace an abstract task related to D by a sub-plan from D_{plan} if it solves it).

The same process repeats with the rest of the domains until finishing all of them. When all the domains are planned in, the global plan $GlobalP$ is completely refined. $GlobalP$ is returned as a solution for the multiple-domain planning problem (step 1). In case, there is no plan to solve the tasks in D_{Tasks} (step 7), The multiple-domain planner returns *fail* (step 13).

Since we might have backtracking when planning in the subsequent domains (steps 9 and 10), "next(Find-Plan($s_{0d} \in S_0, D_{Tasks}, D$))" (step 6) is supposed

Input:
$S_0 = \{s_{0i}$: initial state of domain $D_i\}$
$G = \{g_{Di}$: goal set of domain $D_i\}$
$Domains$: the ordered list of domains to use
Output:
A plan that achieves G starting from S_0
Init:
$GlobalP$ = build-and-order-abstract-tasks(G)

Algorithm MD-Plan($S_0, GlobalP, Domains$)
1. **if** $Domains$ is empty then return $GlobalP$ **endif**
2. D = first of $Domains$
3. RD = rest of $Domains$
4. D_{Tasks} = extract-tasks($GlobalP, D$)
5. **do**
6. D_{plan} = next(Find-Plan($s_{0d} \in S_0, D_{Tasks}, D$))
7. **if** $D_{plan} \neq fail$ **then**
8. $GlobalP$ = substitute-plan($GlobalP, D_{plan}$)
9. RD_{plan} = MD-Plan($S_0, GlobalP, RD$)
10. **if** $RD_{plan} \neq fail$ return RD_{plan} **endif**
11. **endif**
12. **until** $D_{plan} = fail$
13. return $fail$

END

Fig. 2. The multi-domain planning algorithm

to give the next valid plan D_{plan} solving the abstract tasks D_{Tasks} in domain D i.e. a plan that has not been considered yet (the call to the function "next" can be considered as iterating over a set of valid plans that solve D_{Tasks} in D).

The algorithm used to solve an ordered list of tasks in one domain is outlined in Fig. 3. It gets as input the initial state related to the domain s_0, an ordered list of tasks to solve $Tasks$, and the relevant domain D. The algorithm retrieves the first task g from $Tasks$ (step 2) and calls a planner to solve it starting in the initial state s_0 (step 5). If the task is solvable, the planner returns a plan P_g that solves it along with the goal state s_g where the task g is satisfied. As mentioned before, the planner can be specialized to solve planning problems related to the current domain, or a generic planner (domain-independent). In the next recursive call (step 7), the algorithm tries to solve the rest of the tasks starting from s_g this time i.e. the initial state for the next task is s_g. The algorithm continues recursively doing so until solving all the tasks where it returns $success$ (step 1). If $success$ is returned then the plan that solves all the ordered tasks specified in $Tasks$ is the concatenation of the plans solving each task apart (step 9). Please note that as in the previous algorithm, the use of next (in step 5) returns the next valid pair (plan, goal-state). This means that next iterates over a set of

Input:
s_0 : initial state
$Tasks$: a list of ordered tasks to achieve
D : domain to use
Output:
A plan that achieves $Tasks$ starting from s_0

Algorithm Find-Plan($s_0, Tasks, D$)
1. **if** $Tasks$ is empty **then return** *success* **endif**
2. g = the first task in $Tasks$
3. R = the rest of $Tasks$
4. **do**
5. (P_g, s_g) = next(PLANNER(s_0, g, D))
6. **if** $P_g \neq fail$ **then**
7. P_R = Find-Plan(s_g, R, D)
8. **if** $P_R \neq fail$ **then**
9. **return** $P_g; P_R$
10. **endif**
11. **endif**
12. **until** $P_g = fail$
13. **return** *fail*

END

Fig. 3. The Find-Plan planning algorithm

plans that can be generated by the planner to solve one planning problem. A failure is returned (step 12) if the current task can not be solved, or if all the plans that solve the current task make the subsequent tasks unsolvable.

5 Detailed Example

This section demonstrates the performance of the proposed approach in the two domains navigation and blocks, shown in Fig. 4. A planning problem in the navigation domain consists in moving a mobile robot, equipped with an arm, from one location to another. The different locations are connected by doors that can be open or closed. The blocks domain is the standard AI planning bench mark domain used to form towers of blocks according to a set of constraints over the positions of the blocks. In our experiment, it is the mobile robot that is responsible of forming the towers of blocks. Furthermore, a block is constrained to be at a specific location. Consequently, in order to execute an action in the blocks domain, the mobile robot has to be at the same location as the relevant blocks (the blocks that undergo the action). As a result of consideration 1, the blocks domain is augmented with the abstract operator "move-block" derived from the navigation domain operator "move-in". In this scenario the blocks

Domain: Blocks

(pickup ?b ?l)
param: `?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (clear ?b)(on-table ?b) (arm-free) (object-at ?b = ?l))) (foreign: (navigation: (robot-at = ?l)) ))`
Eff: `((local:(and (holding ?b)(clear ?b = f) (on-table ?b = f)(arm-free = f))))`

(putdown ?b ?l)
param: `?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (holding ?b)(object-at ?b = ?l))) (foreign: (navigation: (robot-at = ?l) )))`
Eff: `((local:(and (holding ?b = f)(clear ?b = t) (on-table ?b = t)(arm-free = t))))`

(unstack ?a ?b ?l)
param: `?a ?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (clear ?a)(on ?a ?b)(object-at ?b = ?l) (object-at ?a = ?l) (arm-free))) (foreign: (navigation: (robot-at = ?l) )))`
Eff: `((local:(and (holding ?a)(clear ?b)(clear ?a = f) (on ?a ?b = f)(arm-free = f)))))`

(stack ?a ?b ?l)
param: `?a ?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (holding ?a)(clear ?b) (object-at ?b = ?l)(object-at ?a = ?l))) (foreign: (navigation: (robot-at = ?l))))`
Eff: `((local: (and (holding ?a = f)(clear ?b = f) (clear ?a) (on ?a ?b)(arm-free))))`

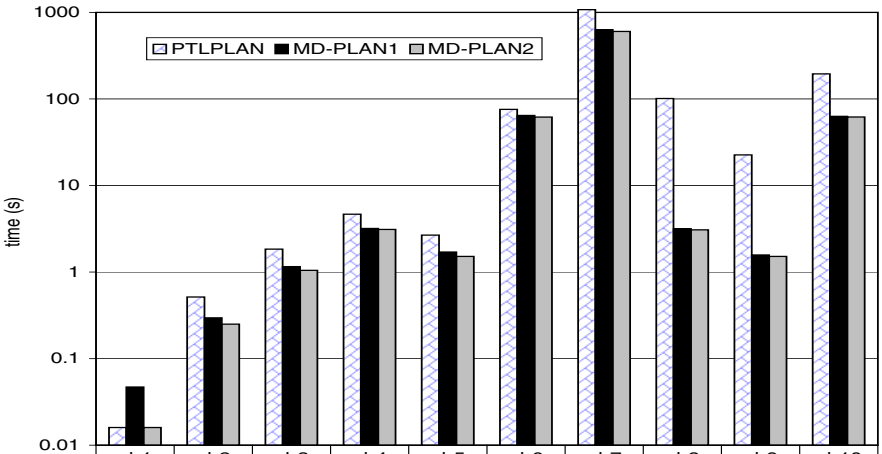
(move-block ?b ?l1 ?l2): ABSTRACT
param: `?b BLOCK, ?l1 ?l2 - LOCATION`
Pre: `((local: (and (holding ?b)(object-at ?b = ?l1)))`
Eff: `((local: (object-at ?b = ?l2)) (foreign: (navigation: (robot-at = ?l2))))`

Domain: Navigation

(move-in ?l ?2)
param: `?l1 ?l2 - LOCATION`
Pre: `((local: (and (robot-at = ?l1) (exists (?d - DOOR)(and (part-of ?d ?l1) (part-of ?d ?l2)(closed ?d = f))))))`
Eff: `((local: (robot-at = ?l2)))`

(open-door ?d)
param: `?d - DOOR`
Pre: `((local: (and (closed ?d)(exists (?l -LOCATION) (and (part-of ?d ?l)(robot-at = ?l))))))`
Eff: `((local:(closed ?d = F))`

Fig. 4. The blocks and navigation domains



	pb1	pb2	pb3	pb4	pb5	pb6	pb7	pb8	pb9	pb10
PTLPLAN	0.016	0.515	1.843	4.656	2.672	75.812	1075.3	101.2	22.563	194.8
MD-PLAN1	0.047	0.296	1.156	3.188	1.703	64.547	632.11	3.172	1.578	63.031
MD-PLAN2	0.016	0.25	1.047	3.109	1.515	61.765	602.06	3.078	1.515	61.844

problems

Fig. 5. Execution times for navigation and blocks domains

domain is ordered before the navigation domain as a result of considerations 1 & 2 stated in section 2.

To evaluate the generation of plans to solve planning problems involving both domains (blocks and navigation), we compared execution times taken by our approach against the execution times taken by the domain-independent planner PTLPLAN [6]. In order to use PTLPLAN, both domains were collapsed in one global domain. We used two versions of the multiple domain planning approach. In the first version called MD-PLAN1, the planner used to plan in the navigation domain as well as the blocks domain is PTLPLAN. In the second version, called MD-PLAN2, we used a specialized planner to plan in the navigation domain, and PTLPLAN to plan in the blocks domain. The navigation specialized planner is a graph-based search algorithm that returns all the different plans that can lead from one location to another one.

The tests were run on 10 problems, with different numbers of blocks and rooms. The different problems involved forming towers of blocks in different rooms which involved moving the blocks from their initial location to their goal location. Figure 5 shows a bar chart diagram (where the values of the y axis are logarithmic) as well as a table of the executions times in seconds taken by PTLPLAN, MD-PLAN1, and MD-PLAN2 to solve the ten problems.

The diagram shows that the two versions of the proposed approach outperform PTLPLAN applied to the two domains as one global domain. It is also worth noting that the second version MD-PLAN2 is slightly faster than the first version MD-PLAN1, which is clearly an advantage of using specialized domain-dependent planners. The performance of the proposed approach against planning in one global domain can be attributed to localized planning in the respective

domains and solving interactions through the use of abstract tasks and their orderings. We believe that many real world scenarios involve domains that have little interaction such as the blocks and navigation domains, therefore defining an ordered structure over them and solving problems local to each domain within the domain itself would greatly reduce the complexity of search.

6 Related Work

In this section we review some of the systems close to the proposed approach. Alpine [7, 8] is one of the first systems proposed to automate the generation of abstraction hierarchies for a specific planning problem by grouping literals appearing in the preconditions and effects of operators according to predefined constraints over operators conflicts. The partitions are then topologically ordered forming a directed graph of abstraction levels. Alpine solves the planning problem in the simpler abstract space, then refines the abstract solution at successive levels of detail by inserting operators to achieve the conditions that were ignored in the higher levels of abstraction. Alpine relies on the "Ordered Monotonicity Property" to refine abstract plans: the refinements of the abstract plans maintain the literals established in the higher levels of abstraction i.e. make sure not to violate what it has been achieved at higher levels. But as mentioned in [15] Alpine does not guarantee the construction of good hierarchies, because it ignores variable binding conflicts. Our approach differs from planning with alpine in different ways. First, Alpine plans with abstract spaces specific to planning problems, ours on the other hand uses abstraction on the domain level. Second, in Alpine each intermediate state in an abstract plan forms an intermediate goal (task to achieve) at the next level of detail; in our approach an abstract plan can have tasks to be achieved at all the next levels of detail.

Collage [10] partitions the overall planning problem into regions of actions and constraints over them. The localized partitioning relies on building a DAG of abstract partitions from constraints over actions and their scope i.e. their relevance to actions of the plan associated with their region and its subregions. The planning algorithm has to maintain the consistency between the different search regions involving a considerable amount of jumping between them to cope with their interactions.

STRPLAN [11] is also a planning system that decomposes the original planning problem into sub-regions. The system uses a language that allows it to specify domain sub-regions and specify local planners to them. To find a global plan, a centralized control module coordinates the local planners by solving constraints over their sub-regions plans.

Perhaps the most related approach to ours is the one reported in [1] where the planning domain is partitioned into sub-domains and organized in a tree structure. The planning process works in two stages: first, abstracted actions coded as complex messages are computed at each sub-domain. Starting with the leaf sub-domains, the abstract action are added to the parent sub-domain until reaching the root sub-domain which contains the global planning problem

goal. The root sub-domain uses all its actions and the abstract actions from its descendant nodes to find an abstract plan that solves the original planning problem. The second stage consists in refining the plan found at the root node by replacing all the abstract actions in the root plan by a sub-plan from the actions of the sub-domains. The main difference with our approach is the construction of the abstract actions at every sub-domain to represent all plans the sub-domain can find to affect the fluents it shares with its parents. In our approach, only one plan is constructed at a domain level, this plan solves the goals of the corresponding domain and introduces new order constraints on the subgoals to be solved in the subsequent domains.

The proposed approach is also comparable to Hierarchical Task Networks "HTN" Planners such as SHOP [2], SHOP2 [12] and UMCP [3]. However HTN planning relies on an expert to hand-code the procedures that are used to refine abstract tasks which is error-prone and not easy for certain domains. Our approach relies on first principles planning to build abstract tasks automatically during planning.

7 Conclusion

We have presented in this paper an approach to synthesize plans involving multiple domains. The approach assumes that a total order of the domains exists to perform plan synthesizing by hierarchically refining abstract tasks defined as subgoals in their respective domains. The main advantages of using such approach, besides search complexity reduction, are its simplicity, and the possibility to use specialized planners when planning in the different domains. It can also be seen as an alternative to using one big global planning domain making it possible to write sub-domains by different experts. The use of abstraction to solve subgoals seems to be a natural way that humans use in their daily life when performing tasks implying different fields of knowledge. The assumption of a total order between domains limits the applicability of the proposed approach to domains with the kind of interactions discussed in section 2. In our future work, we will investigate how to extend the planning algorithms to domains that can not be totally ordered, i.e when there are cycles between the domains in terms of the relation $\prec$. We also envisage to integrate the presented framework with the plan executor implemented on board our mobile robots.

References

1. E. Amir and B. Engelhardt. Factored planning. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence(IJCAI)*, pages 929–935, Acapulco, Mexico, 2003. Morgan Kaufmann.
2. A. Lotem D. Nau, Y. Cao and H. Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 968–973, 1999.

3. K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.
4. E. Fink and Q. Yang. Automatically abstracting the effects of operators. In James Hendler, editor, *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems (AIPS92)*, pages 243–251, College Park, Maryland, USA, 1992. Morgan Kaufmann.
5. F. Giunchiglia, A. Villafiorita, and T. Walsh. Theories of abstraction. *Artificial Intelligence Communications*, 10(3-4):167–176, 1997.
6. L. Karlsson. Conditional progressive planning under uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 431–438, 2001.
7. C. A. Knoblock. Learning abstraction hierarchies for problem solving. In Thomas Dietterich and William Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence*, Menlo Park, California, 1990. AAAI Press.
8. C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243 – 302, 1994.
9. J. Koehler. Solving complex planning tasks through extraction of subproblems. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS98)*, pages 62–69, 1998.
10. A. L. Lansky and L. Getoor. Scope and abstraction: Two criteria for localized planning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1612–1619, 1995.
11. R. B. Llavori. Strplan: A distributed planner for object-centered application domains. *Applied Intelligence*, 10(2-3):259–275, 1999.
12. D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, December 2003.
13. J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP01)*, 2001.
14. E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.
15. D. E. Smith and M. A. Peot. A critical look at knoblock's hierarchy mechanism. In J. Hendler, editor, *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems(AIPS92)*, pages 307–308. Kaufmann, San Mateo, CA, 1992.
16. B. W. Wah and Y. Chen. Partitioning of temporal planning problems in mixed space using the theory of extended saddle points. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003)*, pages 266–273, 2003.
17. D. E. Wilkins. *Practical Planning: extending the classical AI paradigm.* Morgan Kaufmann, 1988.