Factored Planning

Eyal Amir and **Barbara Engelhardt**
Computer Science Division, University of California at Berkeley
Berkeley, CA 94720-1776, USA
{eyal,bee}@cs.berkeley.edu

Abstract

We present a general-purpose method for dynamically factoring a planning domain, whose structure is then exploited by our generic planning method to find sound and complete plans. The planning algorithm's time complexity scales linearly with the size of the domain, and at worst exponentially with the size of the largest subdomain and interaction between subdomains. The factorization procedure divides a planning domain into subdomains that are organized in a tree structure such that interaction between neighboring subdomains in the tree is minimized. The combined planning algorithm is sound and complete, and we demonstrate it on a representative planning domain. The algorithm appears to scale to very large problems regardless of the black box planner used.

1 Introduction

Many planning algorithms use independencies or loose interactions between components in the planning domain to find plans more efficiently. For example, hierarchical planners divide a goal into subgoals (high-level operators) using a decomposition of the domain into loosely interacting parts (e.g., [Knoblock, 1990; Erol *et al.*, 1994]). Planning is done at each level separately, and later the subplans are pieced together to build a valid plan. Other planners that use domain decomposition (e.g., [Lansky and Getoor, 1995]) are not limited to domains with explicit hierarchical structure, but require backtracking across subdomains, and often replan for subgoals that already have valid plans. In general, domain-decomposition planning does not scale well because backtracking between subdomains can dominate the complexity of the search, and the domain decomposition is often ad hoc.

In this paper we present an approach to planning that scales to very large domains by taking advantage of domain structure. This approach is composed of two procedures: factoring and planning. The factoring procedure partitions a planning domain into loosely interacting subdomains that are organized in a tree structure. Our factoring procedure uses decomposition algorithms from graph theory, and our contribution in this matter is in translating the planning-problem decomposition task into a graph decomposition task. After factoring, our planning procedure finds plans for multiple subgoals in each of the subdomains separately, using a generic black box planner. It searches over possible plans using complex action descriptors from each of the subdomains to form a plan for the overall goal. The planning procedure uses dynamic programming principles, and backtracking occurs only within a subdomain as part of the black box planner.

We prove that our planning procedure runs in time linear in the number of subdomains and takes time that is at most exponential in the size of the largest subdomain and the number of dependencies between subdomains. The type of factoring that we select is justified by this complexity result. We also prove that the combined algorithm is sound and complete, and that it can be applied to solve any planning problem using any generic black box planner for planning within subdomains. The complexity is upper bounded by the complexity of the black box planner on the unpartitioned domain.

We implemented and tested our planning algorithm on a simple domain to guide further development. We created two implementations, one with the IPP planning system [Koehler and Hoffmann, 2000] and one with the FF planner [Hoffmann and Nebel, 2001]. We compared the results of our algorithm with those of IPP and FF, and have shown that for a single domain our results scale much better than these planners alone. The example validates our analytical results and shows that our planner's performance scales linearly with the size of the domain for this problem, motivating further development.

2 Factored Planning

In this section we present an algorithm for planning with a partitioned planning domain, where the subdomains are arranged in a rooted tree. Each subdomain corresponds to a subset of the fluents, or single states, in the domain and a subset of the actions which use only those fluents. Neighboring subdomains may share fluents and actions, and the ideal partition has the smallest number of shared fluents. We analyze this algorithm computationally and present a procedure for finding a tree partition with close-to-optimal properties.

2.1 Partitioned Planning Problems

We restrict ourselves to partitioned planning problems that are described using a simple propositional action language. A *partitioned domain description* is a labeled graph that describes a set of subdomains and the connections between the

subdomains. Formally, a partitioned domain description is a labeled graph $G(\mathbb{D}, E, l)$ with

- $\mathbb{D} = \{D_i\}_{i \leq m}$ vertices: for all i, $D_i = \langle P_i, \mathcal{A}_i \rangle$ is a planning domain description with
  - P_i a set of propositional *fluents* (features of our domain that may change value over time),
  - $\mathcal{A}_i$ a set of action (operator) definitions over P_i.
- Edges E over $\mathbb{D}$, and
- A labeling l of the edges in E. For every $D_u, D_v \in \mathbb{D}$, $l(D_u, D_v)$ is a subset of fluents from $\bigcup_{i \leq m} P_i$.

The label $l(D_u, D_v)$ includes at least the fluents shared between D_u, D_v (i.e., $P_u \cap P_v$), but it may include other (e.g. global) fluents. We use the convention that *action* A_i refers to the action definition (also called *action schema*) and that a_i denotes the instantiation of the action in a plan. We say that action a_i is in subdomain D_u iff it is defined in $\mathcal{A}_u$.

We specify the preconditions and effects of actions in $\mathcal{A}_i$ using the simple language of situation calculus (with propositional fluents), and allow actions to have conditional effects. We use the simple compilation technique of [Reiter, 2001] to provide monotonic semantics for this language (and avoid the frame problem). All of our results apply to the PDDL language [Fox and Long, 2002] as well. PDDL is the representation language used in our example (Section 3) to model the domains, the subdomains, and the plans.

2.2 Planning Algorithm

Here we present our planning algorithm, *PartPlan* (Figure 2), which finds plans in a partitioned domain. We use the following notation. For P a set of propositional fluents, $\mathcal{L}(P)$ is the language of P, i.e., the set of propositional sentences that can be built with P.

High-Level Overview

PartPlan processes a partitioned domain tree iteratively from the leaves to the root (the subdomain that holds the goal of *PartPlan*). It selects a leaf subdomain, tries to build plans for each of a set of possible preconditions and goals (Step 2b), records the successful attempts, adds them as new (macro-like) actions to this leaf's single parent subdomain (Step 2c), and removes that leaf subdomain from the graph.

The result of planning in each subdomain is not a complete plan, but rather a compilation of capabilities, which contain both subdomain actions and capabilities of descendant subdomains. Generally, every subdomain's *capabilities* are those plans that it can find (using its own actions and capabilities of its neighbors) that affect the fluents that it shares with its parent subdomain in the tree. Capabilities of a subdomain are shared by sending a message to the parent domain containing a complex action describing the capability.

When the tree is reduced to the single subdomain (after the iterative processing and removal of leaves) that contains the problem goal, we perform planning in that node to achieve the goal condition, based on the capabilities from the (eliminated) child subdomains (Step 3). Then, the plan found by this subdomain is expanded into a plan that achieves the overall problem goal, using the actions of this subdomain along with the reported capabilities of the children in the tree. Every capability that appears in this plan is expanded into a subplan by the subdomain that reported it. Those subplans can themselves contain capabilities that are then expanded, and the process continues until the plan contains only the original domain actions.

The Simple Algorithm is Incomplete

The view presented above leaves the exact nature of planning in each subdomain unspecified. It suggests, roughly, that planning in each subdomain takes the form of finding a plan that starts from some initial state of local fluents and reaches a specified end state of local fluents. Unfortunately, limiting ourselves to this kind of message from children subdomains to parents is incomplete, as demonstrated in the following example (see Figure 1 for an illustration).

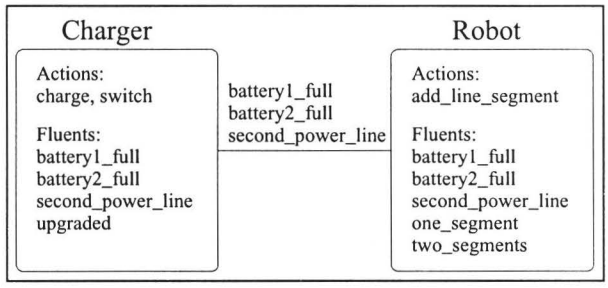

Figure 1: Subdomains for upgrading a battery charger.

Consider a scenario in which we have a charger that can charge batteries and a robot that can connect power cables together and that uses two batteries. Initially, every *charge* action of the charger charges exactly one battery, so two *charge* actions are required to charge both of the robot's batteries. However, if the charger is connected to an additional power source, then a single *charge* action charges two batteries at once. Our task is to upgrade the charger so that it can charge two batteries at once. For this, the robot needs to build a power line that is long enough to reach the charger, and the charger needs to switch to using both sources. Thus, building this line requires multiple charges interleaved with adding lines. A plan that solves this problem needs to charge the robot's batteries, add a line segment, charge again, add another line segment, charge again, add the final line segment, and switch the charger to the new mode.

We want our factored-planning approach to apply to any given planning problem and be complete (i.e., find a plan if there is one) regardless of the plan partition. We will show that there is a partitioning of this problem so that simple messages of the form "if X holds, then I can make Y hold" are not sufficient in a directed tree to find a complete plan. Messages of the form "if X_0 holds, then I can cause Y_0; if you subsequently make X_1 hold, then I can cause Y_1 to hold" are required for domains with interleaved interactions as in this example. These complex messages can be extended for larger numbers of interleaved preconditions and effects, and the algorithm we present allows that type of message.

One possible partitioning of the example above is presented in Figure 1. On the left-hand side we include the

actions *charge* and *switch*, and the features *battery1_full*, *battery2_full*, *second_power_line*, and *upgraded*. The action *charge* has the following effects and preconditions:

- *charge* causes *battery1_full*.
- If *battery1_full* or *upgraded*, then *charge* causes *battery2_full*.

Action *switch* has the following effects and preconditions:

- If *second_power_line*, then *switch* causes *upgraded*.

On the right-hand subdomain we include the action *add_line_segment*, and the features *battery1_full*, *battery2_full*, *second_power_line*, *one_segment*, and *two_segments*. The action *add_line_segment* has the following effects and preconditions:

- If *battery1_full* $\wedge$ *battery2_full*, then *add_line_segment* causes *one_segment* $\wedge$ $\neg$*battery1_full* $\wedge$ $\neg$*battery2_full*.
- If *one_segment*, then *add_line_segment* causes *two_segments*.
- If *two_segments*, then *add_line_segment* causes *second_power_line*.

Now, consider all the possible *simple* messages, i.e., messages of the form "if X, then I can cause Y to hold". We require each of those X and Y to be a combination of fluent values for fluents that are shared between the two subdomains. In our case, those shared fluents are *battery1_full*, *battery2_full*, and *second_power_line*.

Assume that we chose the root node in our partition to be the left-hand side subdomain. The only non-trivial message that the right-hand side can send is

- If *battery1_full* $\wedge$ *battery2_full*, then I can cause $\neg$*battery1_full* $\wedge$ $\neg$*battery2_full*.

We would like to find a plan in the root subdomain (the left-hand side subdomain) that uses these messages and that results in the charger being upgraded. Note that this is not possible with only simple messages.

One way to resolve this problem is to allow more complex messages. Particularly, in our example, we can allow the following message to be sent:

If *battery1_full* $\wedge$ *battery2_full*,
then I can cause $\neg$*battery1_full* $\wedge$ $\neg$*battery2_full* and
if subsequently *battery1_full* $\wedge$ *battery2_full*,
then I can cause (1)
$\neg$*battery1_full* $\wedge$ $\neg$*battery2_full* and
if subsequently *battery1_full* $\wedge$ *battery2_full*,
then I can cause *second_power_line*.

If our planner on the left-hand subdomain can handle such complex actions, then it can use such a message to find a plan that succeeds in upgrading the charger. Indeed allowing this kind of messages makes our planning algorithm complete. This is consistent with results achieved by [Amir and McIlraith, 2000; McIlraith and Amir, 2001] for inference in first-order logic, and is also consistent with [Amir, 2002] that presented results for projection in factored domains.

An alternative to using such long messages is to include internal (i.e., not shared) fluents in shorter messages. However, this has the effect of breaking the encapsulation and structure that we strive to employ, reducing our problem to that of traditional planning on the entire domain and forcing backtracking across different subdomains.

Detailing the Planning Algorithm

Procedure *PartPlan* is presented in Figure 2 and its subroutines are presented in Figures 3 and 4. *PartPlan* is given a partitioned planning domain $G(\mathbb{D}, E, l)$, $\mathbb{D} = \{D_i\}_{i \leq m}$, a set of initial conditions $I = \{I_i\}_{i \leq m}$, indexed by subdomain such that I_i is the initial state for the fluents in P_i, a goal state condition Q in $\mathcal{L}(P_t)$ for root node D_t, and search parameters k (interactions) and d (depth). It returns a plan for achieving the specified goal conditions in the given domain, if such a plan exists under the search constraints. Notice that we show goals that can be represented in a single subdomain. We illustrate how to get around this restriction in Section 2.5.

PROCEDURE *PartPlan*($G(\mathbb{D}, E, l)$, I, Q, k, d)

$\mathbb{D} = \{D_i\}_{i \leq m}$ a partitioned planning domain with G a tree, $I = \{I_i\}_{i \leq m}$ a set of initial conditions, indexed by subdomain, Q a goal in $\mathcal{L}(P_t)$ ($t \leq m$), and k, d parameters.

1. Set $W \leftarrow \mathbb{D}$, $G_W \leftarrow G$. Orient the tree G_W such that it is rooted in t (i.e., every $v \in W$, $v \neq t$, is a descendant of t).
2. Do until $W = \{t\}$ (i.e., until W includes only the root):
   (a) Take $D_v \in W$ a leaf in G_W, and let D_u be its parent.
   (b) For every $\{\mathsf{pre}_i\}_{i<k}$, $\{\mathsf{eff}_i\}_{i \leq k}$ truth assignments to $l(D_u, D_v)$:
   $\langle \pi, j \rangle \leftarrow$ *SinglePlan*($D_v, I_v, \{\mathsf{pre}_i\}_{i<k}, \{\mathsf{eff}_i\}_{i \leq k}, d$)
   If $\pi \neq$ *nil*,
   call *StorePlan*($\pi, T, v, \{\mathsf{pre}_i\}_{i \leq j}, \{\mathsf{eff}_i\}_{i \leq j}, \mathsf{eff}_k$).
   (c) Call *SendMessage*($v, T, \mathcal{A}_u$).
   (d) Remove D_v from G_W, and set $W \leftarrow W \setminus \{D_v\}$.
3. Let $\langle \pi, j \rangle \leftarrow$ *SinglePlan*($D_t, I_t, \{Q\}, d$).
   If $\pi =$ *nil*, return "FAILURE for d, k".
4. Do until $W = \mathbb{D}$ (reconstructing G_W):
   (a) Select $D_u \in W$ and $D_v \in \mathbb{D} \setminus W$ neighbors in G.
   (b) $\pi \leftarrow$ *ExpandPlan*(π, D_u, D_v);
   (c) $W \leftarrow W \cup \{D_v\}$.
5. Return π.

Figure 2: Planning with Subdomains

We discuss two of the subroutines of *PartPlan* in detail. The first, *SinglePlan*, is presented in Figure 3. It determines whether there is a plan with at most d actions in subdomain D_v that starts with the state described by the initial state I_v and arrives at a state that satisfies eff_k. We require the plan to have $k-1$ preconditions interleaved within k effects, corresponding to the number of interleaved conditions in the message (e.g., in (1), the message sent in our example above, $k = 4$). To find such an interleaving message the planner uses *fluent-setting actions*, which set the fluents in $l(D_u, D_v)$ to a truth assignment in one of $\{\mathsf{pre}_i\}_{i<k}$. They correspond to (yet undetermined) action sequences that can be performed in ancestor (or *cousin*) subdomains. If the parent subdomain wishes to use the result of this planning, it will need to find a sequence of actions that replaces these *fluent-setting actions*.

Every fluent-setting action, b_i, can be used at most once, and all of $b_1, ..., b_{i-1}$ must have been used before this action. Also, prior to using this fluent-setting action, eff_{i-1} must hold. This condition corresponds to the state that is required by the other subdomain's complex action as a precondition for the subsequent complex action.

SUBROUTINE *SinglePlan*(D_v, I_v, $\{\mathsf{pre}_i\}_{i<k}$, $\{\mathsf{eff}_i\}_{i\leq k}$, d)

$D_v = \langle P_v, E_v \rangle$ a domain description, I_v an initial condition formula for D_v, $\{\mathsf{pre}_i\}_{i\leq k-1}$ (preconditions) and $\{\mathsf{eff}_i\}_{i\leq k}$ (effects) are state description formulae in some set of fluents $P \subseteq P_v$, d the depth of planning explored.

1. Let $P' \leftarrow P_v$, $E' \leftarrow E_v$ and $I' \leftarrow I_v$.
2. Add the fluents $\{did(b_i)\}_{0<i<k}$ to P'.
3. For every $0 < i < k$ add a new action definition to E' (for $i = 1$ omit $did(b_{i-1})$):

$$Poss(b_i, s) \Leftrightarrow Holds(did(b_{i-1}) \wedge \neg did(b_i) \wedge \mathsf{eff}_{i-1}, s)$$
$$Poss(b_i, s) \Rightarrow Holds(\mathsf{pre}_i \wedge did(b_i), res(b_i, s))$$

4. Add the following initial condition to I':

$$Holds(\neg did(b_1) \wedge ... \wedge \neg did(b_{k-1}), S0)$$

5. Search for a plan in P', E', I' that achieves eff_k with at most d actions. If found plan π, then take $j = max\{i \mid b_i \in \pi\} \cup \{0\}$ and return $\langle \pi, j \rangle$. Otherwise, return nil.

SUBROUTINE *StorePlan*(π, T, v $\{\mathsf{pre}_i\}_{i<j}$, $\{\mathsf{eff}_i\}_{i\leq j}$, eff_k)

P a plan, T a table, v a subdomain index, $\{\mathsf{pre}_i\}_{i<k}$, $\{\mathsf{eff}_i\}_{i\leq k}$ and eff_k are state description formulae.

1. Set $T(v, \{\mathsf{pre}_i\}_{i\leq j}, \{\mathsf{eff}_i\}_{i\leq j}, \mathsf{eff}_k) \leftarrow \pi$.
2. For all $j' < j$,
   (a) Let $\pi_{j'}$ be the initial part of π up to $b_{j'}$ (not including).
   (b) Set $T(v, \{\mathsf{pre}_i\}_{i<j'}, \{\mathsf{eff}_i\}_{i<j'}, \mathsf{eff}_{j'}) \leftarrow \pi_{j'}$.

Figure 3: Planning in a single subdomain, and storing a plan.

SUBROUTINE *SendMessage*(v ,T, D_u)

v a node from which messages are sent, T a table containing plans, D_u a domain for part u.

1. For every table entry $T(v, \langle \mathsf{pre}_i \rangle_{i\leq j}, \langle \mathsf{eff}_i \rangle_{i\leq j}, \mathsf{eff}_k)$ ($j \leq k$), create a complex action symbol $c_{\langle \mathsf{pre}_i \rangle_{i\leq j}, \langle \mathsf{eff}_i \rangle_{i\leq j}, \mathsf{eff}_k}$.
2. For every table entry $T(v, \langle \mathsf{pre}_i \rangle_{i\leq j}, \langle \mathsf{eff}_i \rangle_{i\leq j}, \mathsf{eff}_k)$ ($j \leq k$), add the following new complex action definition to $\mathcal{A}_u$:

$$Poss(c, s) \Leftrightarrow Holds(\mathsf{pre}_j \wedge (last(v) = c'), s)$$
$$Poss(c, s) \Rightarrow Holds(\mathsf{eff}_k \wedge (last(v) = c), res(c, s))$$

$c = c_{\langle \mathsf{pre}_i \rangle_{i\leq j}, \langle \mathsf{eff}_i \rangle_{i\leq j}, \mathsf{eff}_k}$, $c' = c_{\langle \mathsf{pre}_i \rangle_{i\leq j-1}, \langle \mathsf{eff}_i \rangle_{i\leq j-1}, \mathsf{eff}_j}$ (for $j = 1$ let $c' = nil$).

3. Add the following new initial state declaration to I_u:

$$Holds(last(v) = nil), S0)$$

SUBROUTINE *ExpandPlan*(π, D_u ,D_v)

π is a plan, D_u, D_v planning domains.

1. While π has complex actions from D_v:
   (a) Let c be the last complex action from D_v in π.
   (b) Find $T(v, \langle \mathsf{pre}_i \rangle_{i\leq j}, \langle \mathsf{eff}_i \rangle_{i\leq j}, \mathsf{eff}_k)$ the matching table entry for c in T, for some $k > j > 0$. Let π_c be the plan in this entry.
   (c) Find the latest instances of $c_1, ..., c_j$ complex actions of D_v in π such that for every $l \leq j$ $c_l = c_{\langle \mathsf{pre}_i \rangle_{i<l}, \langle \mathsf{eff}_i \rangle_{i<l}, \mathsf{eff}_l}$.
   (d) For every $1 < i \leq j$, replace c_i in π with the sequence of actions in π_c that is between b_{i-1} and b_i. Replace c_1 in π with the sequence of actions in π_c that is before b_1. Replace c in π with the sequence of actions in π_c that is after b_j.
2. Return P.

Figure 4: Sending a message, and expanding a plan.

Subroutine *SendMessage* (Figure 4) takes a plan found by subdomain D_v (stored in table T by subroutine StorePlan) and adds a new *complex* action definition in the parent subdomain, D_u, whose initial state and goal are fluents found in D_v. This new action contains complex action definitions from child subdomains and actions local to D_u. Plans execute sequentially to ensure that the preconditions for complex actions are satisfied at the appropriate times.

2.3 Properties of *PartPlan*

In this section we prove the completeness and soundness of *PartPlan* and analyze its computational behavior. We pursue the intuition that our planning algorithm can find a plan more efficiently if the number of interactions between subdomains is small. The maximum shared fluents across the partition determines the time complexity of the algorithm.

Definition 2.1 (Width of a Plan) *Let $G(\{D_i\}_{i\leq m}, E, l)$ be a partitioned planning domain tree, and $\pi = \langle a_1, ..., a_l \rangle$ a sequence of actions. The* width *of π for G is the largest k (minus one) for which there is a subsequence of actions and an edge (D_u, D_v) in G such that a_i, a_{i+1} are in subdomains on different sides of (D_u, D_v) in G, for every $i < k$.*

The plan-width k for our partitioned domain is the least k that will allow PartPlan to find a plan. In our example in Figure 1 $k = 4$ because for $k < 4$ there is no message that we can send that will allow us to find a plan in the left-hand subdomain. Also, notice that every plan π for D has width at most $|\pi| - 1$.

A *tree decomposition* of planning domain D is a partitioned domain description tree $G(\mathbb{D}, E, l)$ satisfying the running intersection property: if a fluent f appears in P_i and also in P_j, then all the edge-labels on the path between D_i, D_j in G must include f.

Theorem 2.2 *Let $G(\mathbb{D}, E, l)$, be a tree decomposition of planning domain D with m subdomains. PartPlan(G, I, Q, k, d) returns a plan π of that achieves Q in D starting from I, if and only if one exists with width at most $2k$ and at most $d \cdot m$ actions.*

The role of d is to bound the length of the complex actions to at most d interactions; bounding this variable can be used in

an iterative deepening algorithm to find plans with the shortest possible complex actions. The parameter k can be used similarly, and its role is to restrict the length of messages that are sent from children subdomains to parent subdomains.

Corollary 2.3 *Iterative deepening of search parameters k, d in PartPlan is a sound and complete planning algorithm.*

The time bound for procedure *PartPlan* is exponential in the number of fluents in the links between subdomains and the plan-width of the partitioned-planning problem, but is linear in the number of subdomains.

Theorem 2.4 *PartPlan(D, I, Q, k, d) terminates in time $O(m \cdot 2^{2k+l} \cdot min((a+k)^d, k \cdot 2^w))$, for a, w being the largest number of action symbols and largest number of fluent symbols in any subdomain, respectively, and l being the largest number of symbols in $l(D_u, D_v)$ for all subdomains D_u, D_v.*

2.4 Automatic Factoring

Theorem 2.2 assumes that a tree decomposition $G(\mathbb{D}, E, l)$ is given. For a planning domain D we can build the decompositions by hand, adding a knowledge-engineering and planning perspectives to the domain description. We can also find a decomposition automatically using graph decomposition techniques. In this section we describe such an algorithm.

The notion of tree decomposition in planning domains is a special case of tree decomposition in graphs [Robertson and Seymour, 1986]. The width of a tree-decomposition is the width of the largest subset in the tree minus one. The *treewidth* of a graph $G(V, E)$ is the minimum width over all tree-decompositions of G minus one

Finding optimal tree decompositions is NP-hard. Nonetheless, there are several algorithms that find close-to-optimal tree decompositions (e.g., [Becker and Geiger, 1996; Amir, 2001]), and there are also some heuristics (e.g., [Kjaerulff, 1993]) that have been applied successfully in inference algorithms for Bayesian networks and theorem proving.

The reduction of the planning decomposition problem into a graph decomposition problem is as follows. We create a graph $G(V, E)$ with a vertex $v \in V$ for every propositional fluent in our domain D. E is the set of edges in this graph, and it includes an edge between $v, u \in V$ if and only if there is an action definition that includes both v, u (either as a precondition or as an effect). In the next step we find a tree decomposition (S, T), $S = \{X_i \mid i \leq m\}$, $T = (I, F)$ for this graph using one of the algorithms mentioned above. Finally, we create a partitioned domain description $G_D(\{D_i\}_{i \leq m}, E_D, l)$ as follows: The vertices (domain descriptions) $D_i = \langle P_i, \mathcal{A}_i \rangle$ are chosen such that P_i includes all the fluents in X_i, and $\mathcal{A}_i$ includes all the action definitions that can be expressed with the fluents in P_i. The edges in E_D are the tree edges, F, in T. The labeling l of a edge $(D_u, D_v) \in E_D$ is the set of fluents that are shared between the vertices D_u, D_v, i.e., $l(u, v) = P_u \cap P_v$.

This algorithm complexity is taken up by the decomposition of the graph $G(V, E)$. Most heuristics are at most $O(n^2)$, and one approximation algorithm [Amir, 2001] takes time $O(n^3 \log^4 n \cdot k^5 \log k)$ (k is the treewidth of G, and is typically much smaller than n). This is an upper bound on the time taken by the planning domain decomposition algorithm.

2.5 Distribution of the Goal

So far we have assumed that the goal is represented in the set of fluents of a single subdomain. In many planning problems the goal consists of a conjunction of conditions on many different fluents. If we create a single subdomain that includes all those fluents, then the width of the tree decomposition will be high, and so will be the planning time. Instead, we can distribute the goal between subdomains, as we show here.

Assume that we are given a tree decomposition $G(\mathbb{D}, E, l)$ of domain D, with $\mathbb{D} = \{D_i\}_{i \leq m}$. Let a goal condition be $\bigwedge_{i \leq m} Q_i$, where Q_i is a part of the goal represented with the fluents in P_i. Arbitrarily, choose a single subdomain, D_t, as the root of the tree. To each subdomain D_i we add a new fluent symbol, q_i, and a new action, a_i^q. For every leaf subdomain, a_i^q's precondition is Q_i and its effect is q_i. For every internal node in the tree, with children $D_{j_1}, ..., D_{j_s}$, a_i^q's precondition is $q_{j_1} \wedge ... \wedge q_{j_s}$, and its effect is q_i.

When planning with this revised decomposition we use a single goal, q_t, in subdomain D_t. The revised planning problem in this tree decomposition is equivalent to the original one, and we have reduced a (worst case) exponentially large problem down to a linear one.

3 An Extended Example

Consider the following domain, where a robot can move around in a set of rooms connected by doors as a ring [Cimatti and Roveri, 2000]. Each room has a window which can be open, closed, or locked. The robot has four possible actions: move clockwise, move counterclockwise, close the window of the current room, and lock the window of the current room if the window is already closed. The goal state is that the windows of all the rooms are locked. The start state is the robot in one of the r rooms and some configuration of windows open, closed, and locked. In this simple example we assume that the state of windows and the location of the robot are known. In other words, the initial condition is one of $r \cdot 3^r$ possible initial configurations.

Define the following set of fluents:

- Agent location: $in_1, ..., in_r$
- Windows closed: $closed_1, ..., closed_r$
- Windows locked: $locked_1, ..., locked_r$
- Windows locked from room 1 up to room i: $all_1, ..., all_r$.

The all_i fluents represent conjunctions of the $locked_1, ..., locked_i$ fluents from parent and ancestor subdomains in the tree, as in the previous section. Define the following set of actions and corresponding effect axioms:

- Actions: $close$, $lock$, $move_{cl}$, $move_{ccl}$
- Effect axioms:

$$
\begin{aligned}
& in_i(s) \wedge \neg closed_i(s) \Rightarrow closed_i(res(close, s)) \\
& in_i(s) \wedge closed_i(s) \Rightarrow locked_i(res(locked, s)) \\
& in_i(s) \Rightarrow \\
& \quad in_{i+1}(res(move_{cl}, s)) \wedge \neg in_i(res(move_{cl}, s)) \\
& in_i(s) \Rightarrow \\
& \quad in_{i-1}(res(move_{ccl}, s)) \wedge \neg in_i(res(move_{ccl}, s)) \\
& locked_1(s) \Rightarrow all_1(s) \\
& all_{i-1}(s) \wedge locked_i(s) \Rightarrow all_i(s) \quad (1 < i \leq m)
\end{aligned}
$$

In the last set of axioms we use the convention that addition and subtraction are modulo the number of rooms in the ring. In other words, the next room after i is $(i \pm 1 + m) \bmod m$.

For the ring of rooms, an obvious partition turns out to be a good one. Specifically, we associate each subdomain with one room, letting the fluents and actions for subdomain D_i be

$$P_i = \{in_i, closed_i, locked_i, in_{i-1}, in_{i+1}, all_{i-1}, all_i\}$$
$$\mathcal{A}_i = \{close, lock, move_{cl}, move_{ccl}\}$$

Each subdomain contains all actions whose preconditions and effects mention only fluents in that subdomain.

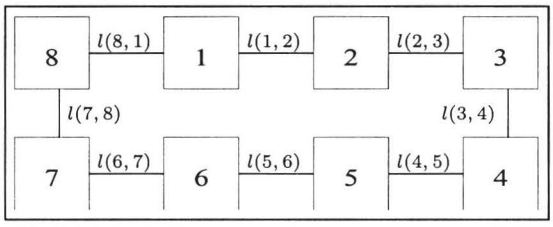

Figure 5: Subdomains for Ring of Rooms.

The graph of subdomains is not a tree (see Figure 5). One way to turn it into a tree decomposition is by removing one of the edges completing a cycle and adding (in the worst case) the label of that edge to the rest of the edges in this graph (this is an example of a more general algorithm that is outside the scope of this paper; see [Amir and McIlraith, 2000]). The result is presented in Figure 6. The graph G connects every two consecutive subdomains but D_1, D_m, for m being the number of rooms (and subdomains). The labeling on the edges is originally $l(i, i+1) = \{in_i, in_{i+1}, all_i\}$ for $i \leq m$. After transforming to graph into a tree we get the labeling

$$l'(i, i{+}1) = l(i, i{+}1) \cup l(m, 1) = \{in_i, in_{i+1}, all_i, in_m, in_1\}.$$

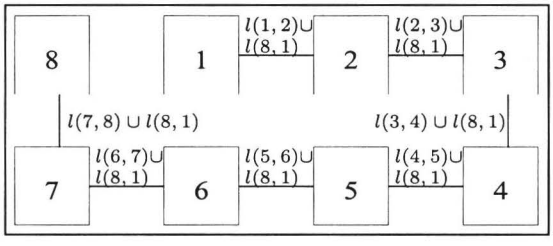

Figure 6: Tree decomposition for Ring of Rooms.

We show how *PartPlan* finds a plan for this domain. We set the goal to be all_m, which is expressed in the fluents of subdomain m. We set $t = m$ to be the root of the tree.

We take node $v = 1$, whose parent in the tree is $u = 2$, and find all the plans possible with $k = 2$, $d = 2$. One such plan is "if the robot is in room 1 and the window is open, then close, lock and move clockwise." We find a plan for D_1:

$$\mathsf{pre}_1 = in_1 \wedge \neg in_2 \wedge \neg all_1 \wedge \neg in_m$$
$$\mathsf{eff}_2 = \neg in_1 \wedge in_2 \wedge all_1 \wedge \neg in_m.$$

Both $\mathsf{pre}_1, \mathsf{eff}_2$ are complete state descriptions in $l'(1,2)$). There is a plan π_1 that *SinglePlan*(D_1, I_1, $\langle \mathsf{pre}_1 \rangle$, $\langle \mathsf{eff}_1 \rangle$, eff_2, d) will return. Thus, one message that is sent to subdomain D_u is "if in some state the robot is in room 1, then D_1 has a sequence of actions that results in all_1 being true and the robot moving to room 2." In more precise terms, the definition for this capability $c_{2,1}$ is added to subdomain D_u by subroutine *SendMessage*.

Now, we take node $v = 2$, whose parent in the tree is $u = 3$, and find all the plans possible with $k = 2$, $d = 2$. One such plan is, "if in some state the robot is in room 2, then move counter-clockwise, execute a subdomain-1-action, then close, lock and move clockwise." We find a plan for D_2:

$$\mathsf{pre}_1 = in_2 \wedge \neg in_3 \wedge \neg all_2 \wedge \neg in_m \wedge \neg in_1$$
$$\mathsf{eff}_2 = \neg in_2 \wedge in_3 \wedge all_2 \wedge \neg in_m \wedge \neg in_1,$$

These are all complete state descriptions in $l'(2,3)$. There is a plan $\pi(c_2)$ that *SinglePlan*(D_2, I_2, $\{\mathsf{pre}_1\}$, $\{\mathsf{eff}_1\}$, eff_2, d) will return. The message that is sent to subdomain D_u is roughly speaking "if the robot is in room 2, then D_2 has a sequence of actions that results in all_2 being true". In more precise terms, the definition for action $c_{2,1}$ is added to subdomain D_u by subroutine *SendMessage*.

This method continues for all the subdomains up to D_m that contains the goal. Subdomain m can conclude that there exists a plan for closing and locking all the windows, satisfying all_m. If the initial condition of our problem specifies that the robot is in room r, then starting in subdomain D_r we will already have a plan independent of the position of the robot. This will lead to a complete plan in subdomain m.

When the plan is found in subdomain m, the algorithm then goes on to expanding the complex actions into their capability plans. For example, when the complex action $c_{2,1}$ of subdomain 1 is encountered in the plan π, it is expanded into a segment $\langle close, lock, move_{cl} \rangle$ that is placed instead of $c_{2,1}$ in plan π. It results in a sequence of actions that is between the state satisfying condition eff_1 (i.e., when the robot appears in room 1) and the state satisfying eff_2.

Using this domain and associated tree decomposition above we implemented our algorithm to see that the run time scaled linearly as the number of rooms increased. The results are compared in figure (7), which shows four different planning algorithm on this domain as the size of the domain grows. For all runs, the optimal-sized plans were found. In very small domains, there is some constant time overhead involved in the *PartPlan* algorithm, but asymptotically the *PartPlan* algorithms begin to dominate because of the linear scaling with respect to the number of rooms. Although this example is not rigorous enough to show the empirical dominance of the *PartPlan* algorithm for every domain, it does motivate further development of the ideas in real-world domains.

4 Related Work

Some of the techniques that we apply have been used before in automated reasoning. Best known are methods for reasoning with probabilistic graphical models [Pearl, 1988]), logical theories [Dechter and Rish, 1994; Amir and McIlraith, 2000] and constraint satisfaction problems [Dechter and Pearl, 1989]. The common principle for those applications is the advantage taken of low *treewidth* available in many domains. With a tree decomposition of close-to-optimal width,

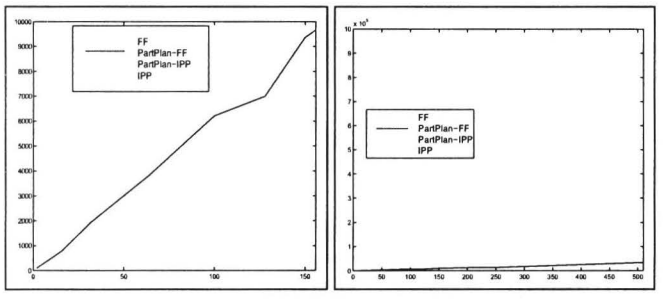

Figure 7: Comparison of run time for *IPP*, *FF*, *PartPlan-IPP*, and *PartPlan-FF*. The figure on the left is for 0-150 rooms, the figure on the right is for 0-500 rooms. The scale on the y-axis is in milliseconds.

these reasoning algorithms use two sweeps of the tree, an evidence collection and evidence distribution sweep, and the entire process is linear in the number of nodes.

Similar techniques to these have also been used in probabilistic planning to find *approximate* policies for factored MDPs (e.g., [Guestrin *et al.*, 2002]), and also in the context of multi-agent stochastic planning with MDPs. The main difference with this approach is that we find an *exact* solution, and we act in domains that do not have universal reachability (all states are reachable from all states), as is assumed in MDPs.

Section 1 discusses approaches for multi-agent planning [Lansky and Getoor, 1995; Ephrati and Rosenschein, 1994; Wolverton and desJardins, 1998].

5 Conclusions

Two contributions of our work are automated planning domain decomposition by minimizing subdomain interactions, and an algorithm to plan in this decomposed domain with no backtracking. The planning algorithm is sound and complete, and exploits a low treewidth partition for a particular planning domain, enabling planning to scale linearly with the domain size. Our automatic factoring algorithm for a planning domain attempts to optimize the run-time complexity of the planner. Theoretical results and empirical examples show that the planning algorithm scales well in domains with decomposable structure.

Directions for future work in this area include the development of algorithms for richer domains and description languages, such as nondeterministic planning actions, stochastic actions, and actions involving duration and numerical parameters. Our approach is built on the language of situation calculus (in which we developed our first version of the algorithms), leading us to believe that it will extend to these directions and more. Our work connects automated reasoning and planning, and we expect to extend this connection to allow first-order decomposition of planning and other related problems, such as diagnosis and reactive control.

Acknowledgments

The first author is supported under an ONR MURI Funds N00014-01-1-0890, and N00014-00-1-0637, and NSF grant ECS-9873474. The second author is supported under an NSF Graduate Research Fellowship.

References

[Amir and McIlraith, 2000] E. Amir and S. McIlraith. Paritition-based logical reasoning. In *Proc. KR '00*, pages 389–400. MK, 2000.

[Amir, 2001] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proc. UAI '01*, pages 7–15. MK, 2001.

[Amir, 2002] E. Amir. Projection in decomposed situation calculus. In *Proc. KR '02*, pages 315–326. MK, 2002.

[Becker and Geiger, 1996] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. UAI '96*, pages 81–89. MK, 1996.

[Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *JAIR*, 13:305–338, 2000.

[Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *AIJ*, 38:353–366, 1989.

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Proc. KR '94*, pages 134–145. Morgan Kaufmann, 1994.

[Ephrati and Rosenschein, 1994] E. Ephrati and J. S. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI '94*, pages 375–380. AAAI Press, 1994.

[Erol *et al.*, 1994] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proc. AAAI '94*, pages 1123–1128. AAAI Press, 1994.

[Fox and Long, 2002] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. http://www.dur.ac.uk/d.p.long/IPC/pddl.html, 2002.

[Guestrin *et al.*, 2002] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *Proc. NIPS'01*. MIT, 2002.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Kjaerulff, 1993] U. Kjaerulff. *Aspects of efficiency imporvement in bayesian networks*. PhD thesis, Aalborg University, 1993.

[Knoblock, 1990] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proc. AAAI '90*, 1990.

[Koehler and Hoffmann, 2000] J. Koehler and J. Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR*, 12:339–386, 2000.

[Lansky and Getoor, 1995] A. L. Lansky and L. C. Getoor. Scope and abstraction: Two criteria for localized planning. In *Proc. IJCAI '95*, pages 1612–1618, 1995.

[McIlraith and Amir, 2001] Sheila McIlraith and Eyal Amir. Theorem proving with structured theories. In *Proc. IJCAI '01*, pages 624–631. MK, 2001.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. MK, 1988.

[Reiter, 2001] R. Reiter. *Knowledge In Action*. MIT, 2001.

[Robertson and Seymour, 1986] N. Robertson and P. D. Seymour. Graph minors. II: algorithmic aspects of treewidth. *J. of Algorithms*, 7:309–322, 1986.

[Wolverton and desJardins, 1998] M. Wolverton and M. desJardins. Controlling communication in distributed planning using irrelevance reasoning. In *Proc. AAAI '98*, pages 868–874, 1998.