

Factored Planning: How, When, and When Not

Ronen I. Brafman*

Department of Computer Science
Stanford University, CA, USA
brafman@cs.stanford.edu

Carmel Domshlak

Faculty of Industrial Engineering and Management
Technion, Israel
dcarmel@ie.technion.ac.il

Abstract

Automated domain factoring, and planning methods that utilize them, have long been of interest to planning researchers. Recent work in this area yielded new theoretical insight and algorithms, but left many questions open: How to decompose a domain into factors? How to work with these factors? And whether and when decomposition-based methods are useful? This paper provides theoretical analysis that answers many of these questions: it proposes a novel approach to factored planning; proves its theoretical superiority over previous methods; provides insight into how to factor domains; and uses its novel complexity results to analyze when factored planning is likely to perform well, and when not. It also establishes the key role played by the domain's causal graph in the complexity analysis of planning algorithms.

Introduction

Factored planning is a collective name for planning algorithms that exploit independence within a planning problem to *decompose* the domain, and then work on each sub-domain (= factor) separately while trying to piece the constructed sub-plans into a valid global plan. Hierarchical planners (Knoblock 1994; Lansky & Getoor 1995) are probably the best-known examples of such algorithms. They vertically factor the domain into a set of increasingly more detailed abstraction levels. They plan in each level separately while reusing the solution of more abstract levels. The problem, however, is that hierarchical decomposition works well only in domains where one component's value has little direct *and* indirect influence on that of others. When such structure is missing, abstraction-generation techniques such as (Sacerdoti 1974; Knoblock 1994) yield no or only minor decomposition, and backtracking between sub-domains in the latter case can dominate the complexity of solving the non-decomposed problem.

The spectacular improvement in standard planning algorithm over the past decade, together with the above limitations of vertical factoring, pushed factored planning to the backstage of domain-independent planning research.¹

*On leave from Ben-Gurion University.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

¹Here we refer to methods that automatically induce a hierarchy from domain description, unlike HTN planning (Erol, Hendler, &

Recently, however, this situation has somewhat changed. First, a number of papers on the formal complexity of planning as a function of certain factored decomposition appeared (Brafman & Domshlak 2003; Domshlak & Dinitz 2001). Second, recent developments in heuristic-search planning have shown that factored problem decompositions and abstractions can provide extremely effective heuristic guidance (Helmert 2004). Finally, recent work by Amir & Engelhardt 2003 (henceforth referred to as AE) has produced a systematic, general-purpose approach to factored planning, with a clear worst-case complexity analysis.

This recent evolution of work on domain-independent factored planning leaves open two major questions. The first question is *how*, i.e., what is the best way to decompose a problem? Previous factoring methods used various graphical structures to drive the factorization process. The structure of such a graph is a significant parameter in the success of each method. Hence, finding a graphical structure leading to a provably better (or even optimal) factorization is clearly of interest. The second, closely related question is *when*: When should factored planning be expected to work better than standard planning. Addressing this question requires better understanding of the complexity of factored and non-factored planning and the parameters affecting them.

In this paper we address these two questions of *how* and *when* through the lens of worst-case complexity analysis. We identify the domain's *causal graph* as an essential structure in the analysis of factored planning, showing that it captures all the sufficient and necessary information about variable interactions. In particular, we show that our approach based on causal graphs is strictly more efficient (by up to an exponential factor) than the AE approach. We show that the tree-width of causal graphs plays a key role in the complexity of both our approach to factored planning, as well as existing methods for *non-factored* step-optimal planning. This finding allows us to relate factored and non-factored methods and understand when each is likely to work best.

Background

We start with a few basic definitions of the planning problem as defined in the SAS⁺ formalism (Bäckström & Nebel Nao 1994) where the hierarchy provides additional domain knowledge that often significantly improves performance.

1995) followed by the definition of the causal graph. The SAS⁺ formalism models domains using multi-valued state variables. It distinguishes between *pre*-conditions and *pre*-vail conditions of an action. The former are required values of variables that are affected by the action. The latter are required values of variables that are not affected by the action. The post-conditions of an action describe the new values its precondition variables. For example, having a visa is a pre-vail condition for applying the action Enter-USA, while having a valid ticket is a precondition of the action Fly-To-USA, as its value changes from true to false following the action's execution. An action is applicable if and only if both its pre- and prevail conditions are satisfied.

Definition 1 A SAS⁺ problem instance is given by a quadruple $\Pi = \langle V, A, I, G \rangle$, where:

- $V = \{v_1, \dots, v_n\}$ is a set of state variables with finite domains $\text{dom}(v_i)$. The domain $\text{dom}(v_i)$ of the variable v_i induces an extended domain $\text{dom}^+(v_i) = \text{dom}(v_i) \cup \{u\}$, where u denotes the value: unspecified.
- I is a fully specified initial state, that is, $I \in \times \text{dom}(v_i)$. By $I[i]$ we denote the value of v_i in I .
- G specifies a set of alternative goal states. Adopting the standard practice in the planning research, we assume that such a set is specified by a partial assignment on V , that is, $G \in \times \text{dom}^+(v_i)$. By $G[i]$ we denote the value provided by G to v_i (with, possibly, $G[i] = u$.)
- $A = \{a_1, \dots, a_m\}$ is a finite set of actions. Each action a_i is a tuple $\langle \text{pre}(a_i), \text{post}(a_i), \text{prv}(a_i) \rangle$, where $\text{pre}(a_i), \text{post}(a_i), \text{prv}(a_i) \subseteq \times \text{dom}^+(v_i)$ denote the pre-, post-, and prevail conditions of a_i , respectively. In what follows, by $\text{pre}(a)[i]$, $\text{post}(a)[i]$, and $\text{prv}(a)[i]$ we denote the corresponding values of v_i .

The factorization of planning problems we propose here is based on the well-known *causal graph* structure (Bacchus & Yang 1994; Knoblock 1994; Brafman & Domshlak 2003; Domshlak & Dinitz 2001; Helmert 2004; Williams & Nayak 1997).

Definition 2 Given a planning problem $\Pi = \langle V, A, I, G \rangle$, the causal graph CG_Π of Π is a mixed (directed/undirected) graph over the nodes V . A directed edge $(\overrightarrow{v_i, v_j})$ appears in CG_Π if (and only if) some action in A that changes the value of v_j has a prevail condition involving some value of v_i . An undirected edge (v_i, v_j) appears in CG_Π if (and only if) some action in A changes the values of v_i and v_j simultaneously.

Informally, the immediate predecessors of v in CG_Π are all those variables that directly affect our ability to change the value of v . It is worth noting that nothing in Definition 2 prevents us from having for some pair of variables $v_i, v_j \in V$ in CG_Π both $(\overrightarrow{v_i, v_j})$, and $(\overrightarrow{v_j, v_i})$, and (v_i, v_j) . In any case, it is evident that constructing the causal graph CG_Π of any given SAS⁺ planning problem Π is straightforward.

Example 1 Suppose we have two packages, A and B , a rocket, and two locations, E and M . Packages can be either in a location or in the rocket, and the rocket requires

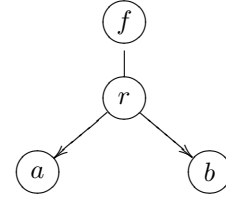


Figure 1: Causal graph for Example 1.

fuel to fly. The actions correspond to loading and unloading the packages, flying the rocket, and fueling the rocket. Flying the rocket consumes the fuel, but it can be fueled in any location. We model this problem in the SAS⁺ formalism as follows. The variables are r, a, b, f ; r denotes the position of the rocket and its domain is at-E, at-M. a and b denote positions of the packages A and B and their domains are: at-E, at-M, at-rocket. f denotes whether or not the rocket has fuel with values full and empty. For $x \neq y \in \{e, m\}$, the actions are fly-x-y, load-x-y, unload-x-y, and fuel. The fly-x-y actions have two precondition – $f=\text{full}$ and $r=\text{at-x}$. Their post-conditions are $f=\text{empty}$ and $r=\text{at-y}$. The load-x-y actions have one precondition – $x=\text{at-y}$ – and one prevail condition – $r=\text{at-y}$. The single post-condition is $x=\text{at-rocket}$. Finally, the action fuel has one pre-condition and one post-condition: empty and full respectively. The causal graph for this problem is shown in Figure 1.

Sequence-Based CSP Planning

The central questions for any factored planning approach are how to decompose the problem and how to piece together solutions from different sub-domains. Our initial answer to the first question, which later we generalize, is very simple: factor = variable. We can now focus on the questions of how to combine a set of given local plans for different factors, and then, how to generate these local plans. The causal graph plays a key role in the algorithm we propose. Its tree-width plays an equally important role in the complexity analysis of the algorithm.

Locally-Optimal Factored Planning

Let $A_i \subseteq A$ denote the set of all actions affecting $v_i \in V$. Suppose that, for every v_i , we are given a set of pre-scheduled action sequences $SPlan(v_i)$ where each $\rho_i \in SPlan(v_i)$ is a finite sequence of pairs (a, t) with $a \in A_i$, and $t \in \mathbb{Z}^+$ is the time point at which a is to be performed. We now ask ourselves how we might construct plans using these n sets of action sequences $SPlan(v_i)$. A key observation is that this particular problem can be solved by compiling it into a binary CSP (denoted SeqCSP) over n variables X_1, \dots, X_n where:

1. The domain of X_i is exactly $SPlan(v_i)$, and
2. The constraints of SeqCSP bijectively correspond to the edges of the causal graph CG_Π .

Informally, the constraint corresponding to a directed edge $(\overrightarrow{v_i, v_j}) \in \text{CG}_\Pi$ ensures that the action sequence selected for v_i provides all the prevail conditions required by the actions of the sequence selected for v_j , and that the timing of

procedure LID

```

 $d := 1$ 
loop
  for  $i := 1 \dots n$  do
     $Dom(X_i) :=$  all sub-plans for  $v_i$  of length up to  $d$ ,
    over all schedules across  $nd$  time points.
  Construct  $SeqCSP_{\Pi}(d)$  over  $X_1, \dots, X_n$ .
  if ( solve-csp( $SeqCSP_{\Pi}(d)$ ) ) then
    Reconstruct a plan  $\rho$  from the solution of  $SeqCSP_{\Pi}(d)$ .
    return  $\rho$ 
  else
     $d := d + 1$ 
  endloop

```

Figure 2: Factored planning via local iterative deepening.

this provision is correct. The constraint corresponding to an undirected edge $(v_i, v_j) \in CG_{\Pi}$ ensures that the restrictions of the sequences selected for v_i and v_j to $A_i \cap A_j$ are identical. This causal-graph based problem reformulation allows us to formalize the worst-case complexity of solving our “sequences combination” problem in a structure-informed manner. Since each constraint of $SeqCSP$ can be verified in polynomial time, from a classical result on tractable CSPs (Dechter 2003) we have that the time complexity of the “sub-plans combination” problem is $O(n\sigma^{w+1})$, where w is the tree-width of the undirected graph underlying CG_{Π} , and $\sigma = \max_i \{|SPlan(v_i)|\}$.

While in itself our “sequences combination” problem is not of general interest, the distance between its CSP formulation, and that of general planning problems is not large. Specifically, given a planning problem $\Pi = \langle V, A, I, G \rangle$, one can solve it using the LID (short for *local iterative deepening*) procedure depicted in Figure 2. LID searches for a plan by performing local iterative deepening on the maximal number of changes that a plan might induce on a single state variable. Given such an upper bound $d \geq 1$, LID formulates a constraint satisfaction problem $SeqCSP_{\Pi}(d)$ where, for $1 \leq i \leq n$, $SPlan(v_i)$ contains all (consistent with I and G) sequences of length at most d of actions affecting v_i . Each such sequence is considered with respect to all possible time schedules of its actions. Since each state variable in iteration d is allowed to change its value up to d times, it is sufficient to consider a time horizon of nd . If, at some iteration d , $SeqCSP_{\Pi}(d)$ is solvable, a valid plan (containing no idle time points) can be easily extracted from the corresponding solution of $SeqCSP_{\Pi}(d)$.

Theorem 1 *LID is a sound and complete planning algorithm. Moreover, if LID terminates with a plan ρ at iteration d , then, for any other plan ρ' for the considered problem instance, there exists a state variable that changes its value on ρ' at least d times.*

The CSP encoding used in LID may seem a bit crude, but it is simple to understand, and all the essential ideas and formal results of this work already fall out from it. In the technical report we describe an *equivalent*, yet technically more involved, encoding in the spirit of standard planning-as-CSP encodings. For the next result we introduce the following notation: Let $Plan(\Pi)$ be the (possibly infinite) set of all plans for Π . For each plan $\rho \in Plan(\Pi)$, and each

$1 \leq i \leq n$, let ρ_i denote the subset of all actions in ρ affecting variable v_i . Finally, let G_{CG} denote the undirected graph underlying causal graph CG_{Π} .

Theorem 2 *Given a planning problem Π , it can be solved using LID in time*

$$O(n(n\delta a)^{w\delta+\delta}) \quad (1)$$

where $a = \max_i \{|A_i|\}$, w is the tree-width of G_{CG} , and δ is the local depth of Π defined as:

$$\delta = \min_{\rho \in Plan(\Pi)} \max_{1 \leq i \leq n} \{|\rho_i|\} \quad (2)$$

Theorem 2 expresses the complexity of LID in terms of two parameters. The tree width of the domain’s causal graph measures the level of interaction between the domain variables. The parameter δ is problem-instance dependent and it expresses the minmax amount of work required on a single variable. In particular, we note that Theorem 2 establishes a new tractable class of planning problems, because for problems with both w and δ bounded by some constants, Eq. 1 trivially reduces to a polynomial.

Generalized Factoring

So far, we assumed that factor = variable, yet it is not clear that this factorization leads to the best possible worst-case performance. Here we take a closer look at this question by drawing on our previous analysis to understand possible effects of using a different factoring.

Two parameters affect the worst-case complexity of factored planning: tree-width and minmax number of changes per factor (local depth, for factor=variable.) Thus, we need to understand the effect of alternative factorizations on these parameters. Consider variables v_1, \dots, v_k that change their value c_1, \dots, c_k times in a locally optimal plan when single variable factors are used. If we combine these variables into a single factor, this new factor will change its value at most $\sum_{i=1}^k c_i$ times in any locally optimal (for the new factorization) plan. In general, it is not hard to verify that the minmax number of changes per factor under factorization with maximal factor size k could be as large as $k\delta$.

While this seems like a big loss, observe that it can be offset by a reduction in the tree-width of the constraint graph. Indeed, it is well known that for each CSP whose primal graph has tree-width w , there is a tree-decomposition with maximal node size $w + 1$. Such a tree-decomposition defines an equivalent CSP whose variables (our new factors) are cross-products of the original variables, and whose constraint graph forms a tree², that is, has tree-width of 1. We also already know that d_f , the minmax number of value changes per new factor, is upper bounded by $(w+1)\delta$. However, observe that it can also be much better. For any tree-decomposition, we know that d_f is bounded by the maximal sum of value changes of original variables in any new factor. If so, then unless all the variables clustered together have to

²Though constructing an optimal tree-decomposition (i.e., one with maximal node size $w + 1$) is NP-hard (Arnborg, Cornell, & Proskurowski 1987), there are numerous effective, fast approximation and heuristic algorithms for this problem.

change δ times each, d_f would be less than $(w + 1)\delta$, and possibly much less, down to δ !

Consequently, we can adapt LID to any tree-decomposition, and any form of factoring. Instead of iterating over the maximal number of value changes of a variable, we iterate over the maximal number of value changes of a factor, i.e., a node of the given tree-decomposition. We refer to this procedure as LID-GF, and its complexity is described by Theorem 3.

Theorem 3 *Given a planning problem Π , the time complexity of solving it using LID-GF on an optimal tree-decomposition of G_{CG} is $O(n(wa + a)^{d_f})$, where w is the treewidth of G_{CG} , $a = \max_i \{|A_i|\}$, and $d_f \leq (w + 1)\delta$.*

It is now apparent that by moving from the extreme of factor = variable to an optimal tree-decomposition, we cannot lose, and are most likely to improve our worst-case complexity. The complexity now changes from exponential in $w\delta$ to something that is at least as good as $w d_{avg}$, where d_{avg} is the maximal (over the factors) average number of variable changes within the factor. Thus, when constructing a tree-decomposition, one needs to consider both the cluster size and its variability, where the value to keep in mind is the (unknown) sum of value changes of variables in a cluster. The good news is that even if we know nothing about the domain, Theorems 2 and 3 imply that we cannot lose by moving to an optimal tree-decomposition.

Perhaps the better news is that we have here a concrete role for domain knowledge. Suppose we have some idea about which variables are likely to change a lot and which variables are likely to change just a little. In that case, we can impose some constraints on the tree-decomposition, ensuring that certain variables appear together in it. We can do this by constructing a *constrained* tree-decomposition, that is, a tree decomposition in which we *a priori* require certain problem variables to be together. This could lead us to tree decomposition with larger nodes, but with smaller sum-of-value-changes, leading to improved performance.

Comparison with AE

Having our generalized LID-GF approach, we now show that it provides better complexity guarantees than AE, a recently proposed approach to factored planning with the first clear complexity analysis (Amir & Engelhardt 2003).

Similarly to LID-GF, AE has a single factoring phase, followed by a sequence of planning phases invoked in an iterative deepening fashion over the upper-bound on the depth of the local plans. The factoring phase takes a certain graph induced by the given problem instance, and constructs a tree decomposition of this graph (named here AEG_{Π}) using one of the off-the-shelf algorithms for close-to-optimal tree decomposition. Given such a tree of planning factors (each factor corresponding to a subset of state variables), each planning phase processes this tree incrementally in a bottom-up fashion. In processing each sub-domain, AE looks for a local plan of a bounded depth over a certain set of complex

macro actions. The search for local plans is performed using a generic black-box planner.³

Though algorithmically different, both LID-GF and AE use local iterative deepening to search for plans, and provide *similar guarantees on the quality* of the resulting plan. That is, plans returned by both approaches are guaranteed to be locally optimal at the level of factors of tree decomposition in use. However, the worst-case complexity of these two approaches is not the same. First, while both approaches scale linearly in the number of state variables, the worst-case complexity of AE grows exponentially in $\Theta(w_{ae}^2 \delta)$ where w_{ae} is the tree-width of AEG_{Π} , while that of LID-GF grows exponentially in $\Theta(w\delta)$. Assuming for a moment that the tree-width of the causal graph and this of AEG_{Π} are comparable, this already shows that LID-GF is worst-case more efficient than AE. However, Theorem 4 shows that the actual difference is much larger, and that it can be exponential in $\Theta(n)$.

Theorem 4 *Given a planning problem Π , let w be the tree-width of G_{CG} , and w_{ae} be the tree-width of AEG_{Π} . For all planning problems Π , we have $w_d \leq w_{ae}$, and there are problems for which we have $w_d = O(1)$ and $w_{ae} = \Theta(n)$.*

Such a gap between the time complexity of LID-GF and AE stems from the structure of the dependencies between the state variables that these two approaches exploit. While problem decomposition in LID-GF is based on the causal graph, AEG_{Π} is an undirected graph over the nodes V , containing an edge (v_i, v_j) iff there is an action $a \in A$ that *somehow* involves both v_i and v_j , that is,

$$\begin{aligned} &(\text{pre}(a)[i] \neq u \vee \text{post}(a)[i] \neq u \vee \text{prv}(a)[i] \neq u) \wedge \\ &(\text{pre}(a)[j] \neq u \vee \text{post}(a)[j] \neq u \vee \text{prv}(a)[j] \neq u) \end{aligned}$$

Given that, it is easy to verify that G_{CG} is a *subgraph* of AEG_{Π} , and thus $w \leq w_{ae}$. (To prove Theorem 4 we provide an example of a linear difference between w and w_{ae} .)

Factoring and Plan Optimality

Classical planning offers a few notions of plan optimality, with the most standard being sequential optimality (henceforth, OP), which corresponds to a plan with a minimal number of actions. Step-optimal planning (SOP) is an alternative that stands for minimizing the number of time steps in which a plan can be executed under a valid parallelizing of its actions. Depending on the application, SOP can be either of interest on its own, or considered as a reasonable compromise when OP is beyond reach. We argue that, from this perspective, the notion of local optimality (LOP) targeted by factored planners is not any different. In some applications, LOP is of interest on its own, e.g., in the context of distributed systems. And viewed as an approximation to OP, LOP and SOP provide similar guarantees, as shown below.

Lemma 1 *Given a planning problem Π , let m_{op}, m_{sop}, m_{lop} denote the number of actions in an optimal, step-optimal, and locally optimal plan, respectively. We have that $m_{sop} \leq n \cdot m_{op}$ and $m_{lop} \leq n \cdot m_{op}$*

³For detailed description of AE, see (Amir & Engelhardt 2003).

(where n is the number of variables in Π), and both these bounds are tight.

Given the ‘‘approximation equivalence’’ between SOP and LOP established by Lemma 1, we turn to consider the time complexity guarantees of standard methods for OP, SOP, and LOP. To the best of our knowledge, such worst-case time guarantees for OP are either exponential in the length of the optimal plan (e.g., state-space forward search using BFS), or exponential in the problem size (e.g., planning-as-CSP with a linear encodings (Kautz & Selman 1996)). At this point, for SOP, all methods with established complexity guarantees are of the second type, that is, worst-case exponential in the problem size – we will have something to say about this later. Thus, moving from OP to SOP appears to buy us nothing in terms of *formal* bounds on the time complexity. The situation with LOP, however, is different. Theorem 2 shows that the direct dependence of LID’s complexity on *both* the problem size and plan length is polynomial. The exponential dependence of LID is on two other, deeper problem characteristics, namely the tightness of problem structure (w), and the amount of local effort required on each problem factor in order to solve the problem (δ).

Below we take a closer look at the relationship between the complexity guarantees for LOP, SOP, and OP. In the course of this comparative analysis, we provide and exploit some new results on the complexity of SOP. In particular, these results show that in certain situations SOP can actually provide better upper bounds on time complexity than OP. Moreover, these results emphasize the importance of the causal graph in the analysis of planning, as its tree-width plays an important role in the analysis of SOP, as well.

Complexity of SOP using DK

To make our discussion concrete, we consider a characteristic planning-as-CSP approach to SOP described in (Do & Kambhampati 2001) (named here DK). While describing the DK encoding, we ignore the use of graphplan in DK to obtain reachability information in form of temporal mutexes. We make this simplification to separate between the core of the methods and their various possible extensions. The DK encoding is parameterized by an upper bound, m , on the step-length of a plan. Given m , the DK encoding includes a single variable $v_i^{[k]}$ for every problem variable v and every time stamp $1 \leq k \leq m$. The domain of each variable $v_i^{[k]}$ is the set of actions that can change the value of v . For any $1 \leq k \leq m$, the value of all variables $v_i^{[k]}$ encode the state of the system at time k . The following (binary) constraints are imposed:

- *Initial state:* If $v_i^{[1]} = a$, then $\text{pre}(a) \cup \text{prv}(a) \subseteq I$.
- *Goals:* If $v_i^{[m]} = a$ and $G[i] \neq u$, then $\text{post}(a)[i] = G[i]$.
- *Precondition:* If $v_i^{[k]} = a$ and $v_i^{[k-1]} = a'$, then $\text{post}(a')[i] = \text{pre}(a)[i]$.
- *Prevail condition:* If $v_i^{[k]} = a$, $v_j^{[k-1]} = a'$, $v_j^{[k]} = a''$, and $\text{prv}(a)[j] \neq u$, then $\text{post}(a')[j] = \text{post}(a'')[j] = \text{prv}(a)[j]$.

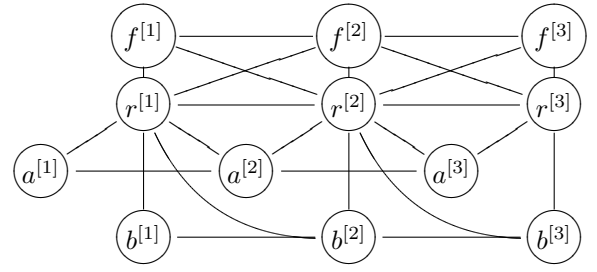


Figure 3: Primal graph $G_{DK}^{(d)}$ of $DK-CSP_{\Pi}(3)$ for example.

- *Simultaneity:* If $a \in A_i \cap A_j$, then $v_i^{[k]} = a$ iff $v_j^{[k]} = a$.

The DK encoding, when used in conjunction with iterative deepening on the plan-length bound m is guaranteed to yield a step-optimal plan (Do & Kambhampati 2001). Now, Figure 3 depicts the primal graph $G_{DK}^{(3)}$ of $DK-CSP_{\Pi}(3)$ for our running example. Observe that constraints between variables at adjacent time points in $DK-CSP_{\Pi}(m)$ (i.e., variables of the form $v_j^{[k]}$ and $v_i^{[k+1]}$) involve *only* neighboring variables within the causal graph. Indeed, $G_{DK}^{(m)}$ corresponds to connected layers of (undirected) causal graphs G_{CG} . Thus, we would expect the tree-width of $G_{DK}^{(m)}$ to be closely related to the tree-width w of G_{CG} , and this is indeed the case.

Lemma 2 *Let Π be a planning problem, and let w be the tree-width of G_{CG} . For any $m \geq 1$, the tree-width w_m of $G_{DK}^{(m)}$ is bounded by $w_m \leq \min\{wm, n\}$. This upper bound is tight for $m > n$, that is, we have $w_m = n$.*

Using Lemma 2, we can immediately provide a structure-aware complexity bound on the ‘‘global-length’’ iterative deepening approach to CSP-based planning.

Theorem 5 *Let Π be a planning problem, w be the tree-width of G_{CG} , and m be the minimal concurrent length of a plan for Π . Then, Π can be solved in time $O(\min\{nm \cdot a^{n+1}, nm \cdot a^{wm+1}\})$, where $a = \max_i \{|A_i|\}$.*

LID/LOP vs. DK/SOP

With these results, we can compare the relative strengths and weaknesses of DK and LID with respect to their time complexity guarantees. We believe that the overall insight holds for other SOP methods as well. We distinguish between a few cases based on the tightness of the causal graph (w), and the step-length of optimal plans (m).

(1) $w = \Theta(n)$. This is the case of very dense causal graphs, indicating strong interactions between variables. This case is a-priori unlikely to be favorable for a factored approach, and Theorems 2 and 5 concur.

(2) $w = O(1)$. In that case, LID’s complexity is exponential in δ , while DK is exponential in $\min\{m, n\}$. Since $\delta \leq m$, LID dominates whenever $\delta < n$. For example, when $m = O(n^2)$, we must have $\delta \geq n$. However, if $m = o(n \log n)$, and local plans are well balanced (recall our discussion of general factorization), we have $\delta < n$. And if $m = o(n)$, then $\delta < n$ for any factorization.

(3) $w = o(n)$. In such case (e.g., $w = \log(n)$), LID complexity is exponential in δw , whereas DK is exponential in $\min\{mw, n\}$. As in case (2), LID is a win when local plan length is not too large in comparison to n/w , e.g., if $w = \log n$ and $\delta = o(n/\log(n))$.

In short, considering scalability in terms of complexity guarantees, we see that LOP scales better than SOP when local plans are not too long (relatively to n), and the causal tree is not too dense, satisfying the relation $w\delta < \min\{wm, n\}$. Similarly, it can be shown that LOP scales better than OP if the domain preserves the relation $w\delta < \min\{m_{op}, n\}$. Intuitively, if the number of factors grows proportionally to the number of problem variables, and the topology of the causal graph and the required local efforts on the factors remain bounded, LOP will scale up. It is then natural to ask whether interesting problems have such features. While this ultimately requires empirical evaluation, we can already point out a few very encouraging indications.

First, upon examination of the standard benchmarks used in recent IPCs, we found⁴ that the step-optimal plan length in *all* these benchmarks is relatively low, and does not appear to grow faster than n . Second, if one considers the type of oversubscription planning problems recently discussed in the literature (Smith 2004; Benton, Do, & Kambhampati 2005), one sees that many such problems are characterized by the need to accomplish many, relatively independent and simple tasks (e.g., small experiments at different sites). Finally, (Williams & Nayak 1997) describe planning for mechanical systems with many parts possibly contributing to the plan, but only a small number of actions each. We believe that these observations strongly encourage theoretical and empirical analysis of factored planning.

Conclusion

The idea of divide and conquer through domain decomposition has always appealed to planning researchers. In this paper we provided a formal study of some of the fundamental questions factored planning brings up. This study resulted in a number of key results and insights. First, it provides a novel factored planning approach that is more efficient than the best previous method of (Amir & Engelhardt 2003). Second, it identifies the domain's causal graph as one of the key parameters in the complexity of factored *and* non-factored planning. Third, the complexity analysis provided enables us to compare between the complexity of standard and factored methods, and provides new classes of tractable planning problems. As we noted, these tractable classes appear to be of genuine practical interest, which has not often been the case for past results on tractable planning. Finally, our analysis helps to understand what makes one factorization better than another, and makes a concrete recommendation on how to factor a problem domain both in presence and in absence of additional domain knowledge.

Future work must examine how well our theoretical insights and new performance guarantees translate into prac-

⁴For additional closely related observations, see analysis of planning under "canonicity assumption" in (Vidal & Geffner 2006) where each action is assumed to be required at most once.

tical performance. Finally, we refer the reader to the full version of this paper which provides (i) Proofs of all theorems; (ii) An encoding with theoretical properties equivalent to *SeqCSP*, but with appearance similar to that of standard planning as CSP encodings, i.e., where variable values are single actions. Because of space constraints, we used here only the conceptually simpler *SeqCSP*; (iii) A more detailed comparison of factored vs. non-factored planning.

References

- Amir, E., and Engelhardt, B. 2003. Factored planning. In *IJCAI'03*, 929–935.
- Arnborg, S.; Cornell, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a k -tree. *SIAM J. Algebraic Discrete Methods* 8:277–284.
- Bacchus, F., and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *AIJ* 71(1):43–100.
- Bäckström, C., and Nebel, B. 1995. Complexity results for SAS⁺ planning. *Comp. Int.* 11(4):625–655.
- Benton, J.; Do, M. B.; and Kambhampati, S. 2005. Over-subscription planning with metric goals. In *IJCAI'05*, 1207–1213.
- Brafman, R. I., and Domshlak, C. 2003. Structure and complexity of planning with unary operators. *JAIR* 18:315–349.
- Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.
- Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *AIJ* 132(2):151–182.
- Domshlak, C., and Dinitz, Y. 2001. Multi-agent off-line coordination: Structure and complexity. In *ECP'01*.
- Erol, K.; Hendler, J.; and Nao, D. S. 1994. HTN planning: Complexity and expressivity. In *AAAI'94*, 1123–1128.
- Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS'04*, 161–170.
- Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI'96*, 1194–1201.
- Knoblock, C. 1994. Automatically generating abstractions for planning. *AIJ* 68(2):243–302.
- Lansky, A. L., and Getoor, L. C. 1995. Scope and abstraction: Two criteria for localized planning. In *IJCAI'95*, 1612–1618.
- Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *AIJ* 5:115–135.
- Smith, D. 2004. Choosing objectives in over-subscription planning. In *ICAPS'04*, 393–401.
- Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal poel planner based on constraint programming. *AIJ* 170(3):298–335.
- Williams, B., and Nayak, P. 1997. A reactive planner for a model-based executive. In *IJCAI'97*, 1178–1185.