# Cost-Optimal Factored Planning: Promises and Pitfalls

**Eric Fabre**
INRIA Rennes - Bretagne Atlantique
Rennes, France
Eric.Fabre@inria.fr

**Loïg Jezequel**
ENS Cachan Bretagne
Rennes, France
Loig.Jezequel@irisa.fr

**Patrik Haslum** and **Sylvie Thiébaux**
Australian National University & NICTA
Canberra, Australia
Patrik.Haslum@anu.edu.au
Sylvie.Thiebaux@anu.edu.au

**Abstract**

Factored planning methods aim to exploit locality to efficiently solve large but "loosely coupled" planning problems by computing solutions locally and propagating limited information between components. However, all factored planning methods presented so far work with representations that require certain parameters to be bounded (e.g. number of coordination points between local plans considered); the satisfaction of those bounds by a given problem instance is difficult to establish *a priori*, and the influence of those parameters on the problem complexity is unclear. We present an instance of the factored planning framework using a representation of the (regular) sets of local plans by finite automata, which does not require any such bound. By substituting weighted automata, we can even do factored cost-optimal planning. We test an implementation of the method on the few standard planning benchmarks that we have found to be amenable to factoring. We show that this method runs in polynomial time under conditions similar to those considered in previous work, but not only under those conditions. Thus, what constitutes an essential measure of "factorability" remains obscure.

## Introduction

Identifying and exploiting problem structure is one of the key challenges faced by automated planning and more generally AI. For problems structured as a set of "loosely coupled" subproblems (or components), factored methods which compute local component solutions and propagate limited information between components are appealing and promise exponential reduction of problem-solving effort. Such methods have successfully been applied in constraint satisfaction, probabilistic reasoning, image and speech processing, digital communication, and in many distributed management algorithms for large modular systems.

A number of factored *planning* methods have been proposed, but with more limited success (Amir and Engelhardt 2003; Brafman and Domshlak 2006; 2008; Kelareva et al. 2007). Existing methods provide either no or weak optimality guarantees (Brafman and Domshlak 2008). A mere couple of implementations have been reported, which have been shown to excel only in very few, particular domains (Amir and Engelhardt 2003; Kelareva et al. 2007). Moreover, whilst all published methods are exponential in the tree-width of the instance's interaction graph, which captures the level of coupling between components, they additionally impose bounds on other parameters, such as the length of the local plans considered, and/or the maximal number of coordination points between pairs of local plans. It is unclear whether those additional restrictions are motivated by the representations adopted by these particular methods (e.g. constraints or SAT), or whether they intrinsically influence the complexity of the factored planning *problem*. They lead existing factored planners to search a bounded plan space that may not contain a solution, and to resort to iterative deepening to achieve even completeness.

We advance the state of the art in the theory and practice of factored planning, by presenting an instantiation of the factored planning framework which does not impose any bound on the set of plans considered. This is achieved by representing components as finite automata recognising the (regular) language of valid local plans. This enables us to manipulate *all* local plans for a component at once, and classical message passing narrows down local plan sets to ensure they are globally compatible. Adopting this more general representation, we make the following contributions:

**Cost optimal factored planning**: by substituting *weighted* automata (Mohri 2009), i.e. transducers mapping valid local plans to their cost, we devise the first cost-optimal factored planning algorithm for arbitrary non-negative action costs. We are not aware of any factored planner providing as strong optimality guarantees.

**Benchmark analysis**: previous factored planners, insofar as they have been implemented, have used a couple of simple, ad-hoc domains for evaluation. We test our (cost-optimal) factored planner on IPC benchmarks. Most of them do not decompose well, yet we are able to identify a few that have the required structure for factored planning to shine, whether at solving problems or proving them unsolvable.

**Complexity analysis**: Bounding tree-width of the interaction graph alone is not sufficient to achieve polynomial time factored planning. We prove that if in addition the length bound considered by Brafman and Domshlak (2008) holds for all plans, then the planner runs in polynomial time.[1] However, this is not the only condition under which it does.

Our investigations reveal that there are still significant gaps in the understanding of factored planning, and suggest a number of avenues for future research.

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] In fact, if we impose this bound on the problem artificially, and wrap it in an iterative deepening search, we obtain essentially the same algorithm.

## Factored Planning

We assume a classical planning problem, composed of a set of state variables (either propositions or finite domain variables) and a set of actions, each described by its preconditions and effects. The objective is to find a plan with minimum sum of (non-negative) action costs.

Factored planning methods decompose the problem into subproblems, called factors or components. Each component is a planning problem in itself, but they also interact. The graph where interacting components are connected with an edge is called the component interaction graph. Existing factored planning methods require this graph to be a tree: when it is not, it must be transformed into a tree by merging components. This is why the tree-width of the interaction graph, which measures the greatest number of components that have to be joined into one to obtain a tree, is often quoted as one of the basic measures of problem factorability.

To decompose the problem, we may partition the set of state variables and let components share actions (Brafman and Domshlak 2006), or we may partition actions and share variables across components (Amir and Engelhardt 2003; Brafman and Domshlak 2008). We choose the former.[2]

What matters, conceptually, is that each component has a set of *locally valid plans*. The factored planning problem is to select a valid plan for each component that is *compatible* with those selected for each of the components it interacts with (i.e., its neighbours in the interaction graph). The combination of those local (sequential) plans, synchronising only execution of shared actions, is then a valid (partially ordered) plan for the global problem.

### Languages and Plans

Since each component is an ordinary planning problem (essentially, an abstraction of the global problem onto the variables that make up the component), its set of locally valid plans forms a regular language over the alphabet of the components actions. In the shared action model, the condition of compatibility of local plans for two neighbouring components is that their local plans (which are words in their languages) become equal when projected onto the subalphabet of actions that they share. The combination of sets of local plans over overlapping action alphabets corresponds to forming the product of the languages. The product of all components' languages of locally valid plans equals the set of all globally valid plans.

Any concrete instantiation of the factored planning method has to manipulate *representations* of the regular sets of local plans. If we impose, for example, a constant bound on the length of any local plan, as previously done by Amir & Engelhardt (2003) and by Brafman & Domshlak (2006), these sets can be represented either exhaustively or by a propositional or constraint formula.

[2]There is no essential difference between the two options, as each can be transformed to the other with a linear size increase. To transform a shared action model to a shared variable model, duplicate shared actions and make all variabels they touch shared; in the opposite direction, duplicate shared variables and add actions to perform explicit synchronisation and alternating access.

Since the sets are regular, they can be represented by (deterministic or non-deterministic) finite automata. The operations that the factored problem solving method requires, viz. projection and product, can be performed – and performed efficiently – directly on automata. To guarantee plan optimality, we need a representation that also considers plan costs. For this, we turn to weighted languages and the corresponding weighted automata (Mohri 2009).

### Weighted Languages and Optimal Plans

A (regular) language is a set of strings. A *weighted language* is a mapping from strings to a domain of numeric values, which we take to be the non-negative reals. Weights are interpreted additively: the product operation on weighted languages sums the weights of the combined strings, while projection minimises over compatible strings.

Formally, a weighted language $L$ over alphabet $\Sigma$ is a function $L : \Sigma^* \to \mathbb{R}^+ \cup \{\infty\}$, with the convention that $L(u) = \infty$ when $u$ does not belong to the language. The projection $\Pi_{\Sigma'}(L)$ of $L$ on subalphabet $\Sigma' \subset \Sigma$ is such that

$$\Pi_{\Sigma'}(L)(u') = \min_{u \in \Sigma^*, u_{|\Sigma'} = u'} L(u), \tag{1}$$

where $u_{|\Sigma'}$ denotes ordinary, unweighted projection. The product $L_1 \times L_2$ of weighted languages $L_1$ and $L_2$ over alphabets $\Sigma_1$ and $\Sigma_2$, respectively, is the weighted language over $\Sigma_1 \cup \Sigma_2$ defined by

$$(L_1 \times L_2)(u) = L_1(u_{|\Sigma_1}) + L_2(u_{|\Sigma_2}). \tag{2}$$

This fits our interpretation of strings as plans and their weights as plan costs. If $L$ is the set of locally valid plans for a component, with associated costs, and $\Sigma'$ the set of actions it shares with its neighbours, the projection of $L$ onto $\Sigma'$ is the shared action sequences that this component can perform, i.e., an "outside view" of its local plan set, under the assumption that if the component has more than one way to perform a particular shared action sequence, it will choose the cheapest. Similarly, the cost of a plan in the product of two components languages, which is a valid plan for both, sums the costs of the two local plans. This implies that the cost of a shared action must be split between all components that have it; it does not matter how the division is done.

A factored planning problem consists of a network of components, each of which has its own weighted regular language, $L_i$ over an action set $\Sigma_i$. Their product, $L = L_1 \times \ldots \times L_n$ is the language of all globally valid plans. The projection of this language back to the alphabet of component $i$, $L'_i = \Pi_{\Sigma_i}(L)$, is the weighted language of plans that are both locally valid for component $i$ and compatible with some plan for every other component, where weights reflect the global plan cost.

**Theorem 1** *If $u^\star$ is an optimal plan in $L$, with cost $w^\star$, then $u^\star_{|\Sigma_i}$ is an optimal plan in $L'_i = \Pi_{\Sigma_i}(L)$, and $L'_i(u^\star_{|\Sigma_i}) = w^\star$. Conversely, for any optimal plan $u^\star_i$ in $L'_i$, there exists an optimal plan $u^\star$ in $L$ such that $u^\star_i = u^\star_{|\Sigma_i}$.*

In other words, knowing the "updated" local language $L'_i$ for each component allows us to construct a globally valid and optimal plan $u^\star$ by finding locally valid and optimal plans

$u_1^\star, \ldots, u_n^\star$ for each component. For illustration, contrast figure 1(a) and (d) which represent $L_1$ and $L_1'$: in 1(a), the best plan appears to be taking the shared action $\alpha$, with cost 1, while in 1(d) it can be seen that the global cost of this plan is 7, while the cost of the optimal plan, taking action $a$, is 3.

The global plan is partially ordered; synchronisation is needed only for shared actions, and then only between the components that share the action. In a problem that factors well, components are small, so extracting local optimal plans from a representation of each local language $L_i'$ is easy. The challenge of factored planning lies in computing those representations without computing any explicit representation of the global language $L$. This is what the message passing algorithm does.

## The Message Passing Algorithm

The message passing algorithm (MPA) is a generic distributed optimisation method. It has been used for, e.g., inference in belief networks (Pearl 1986), constraint optimisation (Dechter 2003) and other applications (Fabre 2003). It operates by sending messages along edges of the interaction graph. Messages are objects of the same type as components: in our setting, they are sets of (weighted) plans.

Message $M_{i.j}$ represents the knowledge of component $i$, incorporating messages it has received from components on its side of the tree, projected on the vocabulary $\Sigma_i \cap \Sigma_j$ shared with the receiving component $j$. After messages have stabilised, the product of the local language of component $i$ with its incomming messages, $L_i \times (\times_{k \in N(i)} M_{k,i})$, yields the updated language $L_i'$, from which a globally valid and optimal plan can be extracted. The algorithm requires that $\Pi_{\Sigma'}(L_1 \times L_2) = \Pi_{\Sigma'}(L_1) \times \Pi_{\Sigma'}(L_2)$ for any $\Sigma'$ that contains shared vocabulary of $L_1$ and $L_2$. This condition holds for weighted languages, and their representation by weighted automata (cf. Fabre and Jezequel 2009).

**Algorithm 1** The message passing algorithm (MPA) for factored planning. $N(i)$ denotes the neighbours of component $i$ in the interaction graph $G$. $\mathbb{I}$ is the neutral element of $\times$.

$M_{i,j} \leftarrow \mathbb{I}, \forall (i,j) \in G$
**until** stability of messages **do**
  select an edge $(i,j)$
  $M_{i,j} \leftarrow \Pi_{\Sigma_i \cap \Sigma_j} \left( L_i \times \left( \times_{k \in N(i) \setminus j} M_{k,i} \right) \right)$
**done**
extract solution from $L_i' = L_i \times \left( \times_{k \in N(i)} M_{k,i} \right), \forall i$

If the interaction graph is a tree the algorithm converges to a solution in finite time, no matter in what order messages are sent. Convergence with minimum number of messages is achieved by a two-pass scheme, where the first pass starts at the leaves, sending messages along every edge directed towards the (arbitrarily chosen) root, and the second runs back in the opposite direction, starting from the root. Thus, only one message in each direction along each edge is needed, resulting in polynomial runtime when the time to process each message is polynomial.

In factored planning, it is not necessary to compute an explicit representation of $L_i'$, as long as a (minimum cost) plan can somehow be extracted from (representations of) $L_i$ and $\times_{k \in N(i)} M_{k,i}$. In fact, apart from final solution extraction, the MPA works only with projections of $L_i$ onto shared subalphabets. If $L_{\text{pub}(i)} = \Pi_{\Sigma_{\text{pub}(i)}}(L_i)$, where $\Sigma_{\text{pub}(i)}$ contains all actions component $i$ shares with any neighbour, then every step of the factored planning algorithm apart from the last may be carried out using $L_{\text{pub}(i)}$ in place of $L_i$.

Any implementation of the MPA must operate on a concrete representation of the sets of plans sent as messages. For this, we will use weighted automata.

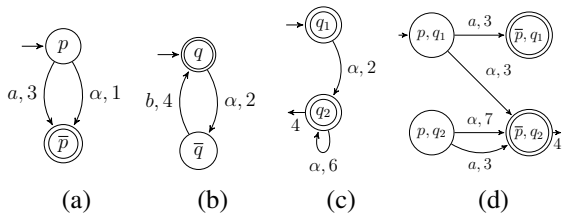



Figure 1: (a)–(b) Weighted automata, representing locally valid plans of components 1 and 2. (c) Projection of (b) on the set of shared actions, $\{\alpha\}$ (after minimisation). This is message $M_{2,1}$ (see description of the MPA). (d) Product of (c) and (a), representing the final plan set of component 1. Note the final state cost of 4 in (c)–(d).

## Weighted Automata

Weighted automata (Mohri 2009) are finite state transducers from strings to numbers, which we take to be non-negative reals. Formally, a weighted automaton is a tuple $A = (S, \Sigma, T, I, F, c^i, c^f)$ where $S$ is a finite set of states, among which $I, F \subseteq S$ are the initial and final states, $\Sigma$ is the finite alphabet (of actions), $T \subseteq S \times \Sigma \times \mathbb{R}^+ \times S$ is a finite set of weighted transitions, and the functions $c^i : I \mapsto \mathbb{R}^+$ and $c^f : F \mapsto \mathbb{R}^+$ assign weights to initial and final states.[3] An accepting path in $A$ is a sequence of transitions, $\pi = t_1, \ldots, t_k$, that forms a contiguous path from an initial state $s_0 \in I$ to a final state $s_k \in F$. The word produced by the path, $\sigma(\pi)$, is the corresponding sequence of transition labels. As for ordinary automata, we assume the existence of a distinguished "silent" transition label $\epsilon$; transitions labelled by $\epsilon$ are invisible in the word produced by the path. The weight of the path is $c(\pi) = c^i(s_0) + (\sum_{i=1\ldots k} c(t_i)) + c^f(s_k)$, where $c(t_i)$ is the weight of $t_i$. The weighted language of $A$ is defined by

$$\mathcal{L}(A)(u) = \min_{\text{paths } \pi \text{ in } A \text{ s.t. } \sigma(\pi) = u} c(\pi) \qquad (3)$$

where the minimum is $\infty$ if $A$ has no path accepting $u$. In other words, the sequence of actions $u$ is in the language if there is an accepting path that produces it, as usual, and its weight is the minimal weight over all paths that produce $u$.

As usual, $A$ is said to be deterministic if it has a single initial state ($|I| = 1$) and for any state $s$ and action $a \in \Sigma$, there is at most one transition $(s, a, c, s') \in T$.

[3] The initial and final state costs play a role in the algorithms for manipulating weighted automata.

## Operations on Weighted Automata

To use weighted automata as our representation of plan sets in the factored planning algorithm, we need to be able to form products and project on subsets of actions. For efficiency, it is also desirable that we can minimise automata, but this is not essential for correctness or completeness of the method. These operations are well known on ordinary finite automata; here we briefly sketch how they are extended to the weighted case. We refer to Mohri (2009) for details.

The product, $A_1 \times A_2$, of two weighted automata is obtained by forming the classical, unweighted, parallel product[4] and assigning transition and state weights the sum of their weights in $A_1$ and $A_2$. Figures 1(a), (c) and (d) show an example. Product preserves determinism: if $A_1$ and $A_2$ are both deterministic, then so is $A_1 \times A_2$.

The projection of $A$ on a subset of actions $\Sigma' \subseteq \Sigma$ is obtained by replacing the label of any transition labelled with some $a \notin \Sigma'$ by the silent label $\epsilon$, followed by weighted version of standard $\epsilon$-removal. Figures 1(b)–(c) shows an example of projection. Projection may produce a non-deterministic automaton. For efficiency, we want to minimise the automata representing messages. The determinisation and minimisation procedures for WA are essentially analogues of those for ordinary automata, but the former is more complicated due to the management of weights. We refer to Mohri's text (2009) for details.

Unlike ordinary automata, not every WA is determinisable. Briefly, the reason is that a non-deterministic WA may accept an arbitrarily long string along paths with different weights, and deciding which path is the cheaper (and therefore the right one) may require looking at the entire string. However, as noted above, determinisation is not essential for either correctness or completeness of the factored algorithm; it is only a tool that may improve efficiency.

The product (for two WA), projection and minimisation procedures all run in time polynomial in the size of their input. As a convention, we assume that automata are simplified ("trimmed") by removing states that are not reachable from any initial state, and states from which no accepting state is reachable. This is also a polynomial time operation. Determinisation, when it is possible, may as usual produce an exponentially larger automaton as output (and thus take exponential time). An important step in proving the complexity results in the next section will be to show that under the right conditions, determinisation is either not required, or possible without the exponential blow-up.

**Implementation** Applying the MPA using weighted automata to represent sets of plans, we obtain a cost optimal factored planner. The construction of initial automata representing sets of locally valid plans for each component is the same as computing the (constrained) abstraction onto the set of variables that belong to the component (just as in the construction of a PDB). The extraction of a tuple of compatible local plans works like the second pass of the MPA, where each component sends to those below only a single plan. This plan is extracted by direct graph search on its final automaton. We have implemented this algorithm, using the OpenFST library[5] for automata operations, in a planner called Distoplan.

[4]The *parallel product* synchronises only transitions with labels that the automata share, leaving each automaton free to take non-shared transitions. Thus, this product mirrors the one on languages: $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \times \mathcal{L}(A_2)$. This is unlike the *synchronous product*, which implements language intersection. The two coincide when $A_1$ and $A_2$ work on the same alphabet.

## Conditions for Polynomial Time Complexity

When the interaction graph forms a tree, the message passing algorithm converges to a solution, using a linear number of messages. In applications like finite domain constraint optimisation, the size of messages, and the complexity of computing the product of a component and its incoming messages, are bounded by the size of the components themselves. Therefore, bounding component size is sufficient to make the MPA run in polynomial time (assuming basic operations are polynomial time). When the interaction graph is not a tree but has tree-width bounded by $w$, it can be transformed into a tree with an increase in component size that is exponential only in $w$.

But planning is a harder problem. Even when the interaction graph is a tree, limiting the size of components is not sufficient to prevent messages from growing exponentially.[6] This issue has not been observed in previous factored planning methods, due to the choice of message representations that are by their very nature bounded. But the use of such representations also limits the planner to finding only solutions within the bound, forcing it to use iterative deepening on those parameters to achieve even completeness.

Yet, it is possible to find conditions that are sufficient to guarantee polynomial runtime also for the factored planning method using the unbounded weighted automaton representation, because the necessary operations (i.e., product and projection) are polynomial in the size of their input. It is simply a question of finding conditions which limit the size of automata handled by the MPA. We will examine one set of conditions, closely related to those assumed by Brafman & Domshlak (2008), and show that it suffices. The key is bounding the number of shared action occurrences in any locally valid plan. Later we show that this condition is not essential, using an example of a problem where it does not hold but message growth is still polynomial.

First, we state a general condition for tractability of factored planning using our representation. In the next two subsections we examine two special cases that imply this condition. Let $C_i$ and $M_{i,j}$ denote the weighted automata representing component $i$ and the message from $i$ to $j$, respectively, let $\Sigma_{\text{pub}(i)}$ be the subset of actions that component $i$ shares with any neighbour, and let $C_{\text{pub}(i)} = \Pi_{\Sigma_{\text{pub}(i)}}(C_i)$. $|A|$ denotes the number of states in automaton $A$.

In this analysis, we need to make a few more specific assumptions about how the MPA is implemented. First, we assume that it follows the two pass scheme, so no more than one message is sent in each direction along each edge. Second, when computing the outgoing message $\Pi_{\Sigma_{i,j}}(C_i \times M_{j_1,i} \times \ldots \times M_{j_k,i})$ of component $i$, we exploit that

[5]`http://www.openfst.org/`

[6]A simple example demonstrating this can be constructed using a factored binary counter.

$\Pi_{\Sigma_{i,j}}(C_i) = \Pi_{\Sigma_{i,j}}(C_{\text{pub}(i)})$ and that product is associative, by computing $C'_{\text{pub}(i)} = (\ldots(C_{\text{pub}(i)} \times M_{j_1,i}) \times \ldots) \times M_{j_k,i}$, incrementally, using the messages received by $i$ so far, and then each outgoing message as $\Pi_{\Sigma_{i,j}}(C'_{\text{pub}(i)})$. This way, $M_{j_1,i} \times \ldots \times M_{j_k,i}$ is never computed explicitly. The result $C'_{\text{pub}(i)}$ computed in the first pass is kept, and updated with the additional message in the second pass. Finally, as noted earlier, a factored planner must only extract a minimum cost element (plan) of $\mathcal{L}(C_i \times (\times_{k \in N(i)} M_{k,i}))$, which does not, in principle, require $C_i$ to be made explicit.

**Theorem 2** *Let $n$ be the number of components, and **poly**$(n)$ a polynomial in $n$. If the component interaction graph is a tree, and*

*(a) $|C_{pub(i)}| \leq$ **poly**$(n)$, and $C_{pub(i)}$ is computable in time polynomial in $n$, for each component $i$;*
*(b) $|M_{i,j}| \leq$ **poly**$(n)$, for each pair $i, j$;*
*(c) $|C'_{pub(i)}| \leq$ **poly**$(|C_{pub(i)}|+|M_{j_1,i}|+\ldots+|M_{j_k,i}|)$, where $C'_{pub(i)} = C_{pub(i)} \times M_{j_1,i} \times \ldots \times M_{j_k,i}$, for each component $i$ and subset of neighbours $\{j_1, \ldots, j_k\} \subseteq N(i)$;*
*(d) $M_{i,j} = \Pi_{\Sigma_{i,j}}(C'_{pub(i)})$ is computable in time polynomial in $|C'_{pub(i)}|$; and*
*(e) given a WA $A$ over $\Sigma_{pub(i)}$, a minimum cost plan in $\mathcal{L}(C_i \times A)$ can be found in time polynomial in $n$;*

*then the run time of the factored planner is polynomial in $n$.*
**Proof (sketch):** The two-pass MPA requires only a linear number of messages to be sent and processed. Conditions (a)–(d) imply that each step of generating and processing each message is polynomial. Condition (e) ensures that the final plan extraction is also polynomial. □

Determinisation is the only potentially non-polynomial step in processing a message: in cases where it is not needed, or does not produce an exponential blow-up (as, for example, in the proof of lemma 5), condition (d) is met.

If the number of state variables in each component is at most a logarithmic function of $n$, then the size of each component automaton, $C_i$, is polynomial in $n$, and plan extraction can be done in polynomial time by simply searching $C_i \times (\times_{k \in N(i)} M_{k,i})$. Thus, in this case, conditions (a) and (e) of theorem 2 are easily met. However, as pointed out by Brafman and Domshlak (2008), this restriction can be waived if each component has some other property that allows a (minimum-cost) plan, compatible with constraints on the shared parts of the plan, to be found in polynomial time. Conversely, if components have no such property, the following holds:

**Theorem 3** *If extracting a plan in polynomial time requires $|C_i|$ to be polynomially bounded, then any problem that the factored planning method solves in polynomial time has a polynomial length plan, if it has any plan at all.*

Theorem 3 implies that for problems known to have a plan, when we do not care about optimality, the unbounded automata representation offers no complexity theoretic advantage over using a representation with bounded local plan length in an iterative deepening search. But the unbounded factored method can also – under the right conditions – prove unsolvability and optimality in polynomial time, whereas a method using a length-bounded representation can never prove anything more than that there is no (better) plan within the length bounds examined so far. In this respect, such methods are similar to encoding bounded planning into, e.g., SAT.

## Bounded Shared Sequence Length

Brafman and Domshlak (2008) consider a setting where the number of shared action occurrences in any local plan for any component is bounded by a constant $K$, and each local planning problem, taking account of constraints imposed by incomming messages, can be solved in polynomial time. Under these restrictions, they show that the problem can be encoded as a CSP which can be decided in polynomial time. (To actually solve the planning problem, they repeat this for increasing values of $K$, up to the smallest that allows a plan to be found, or up to $2^{|V|}$, where $V$ is the set of state variables, if no solution exists.)

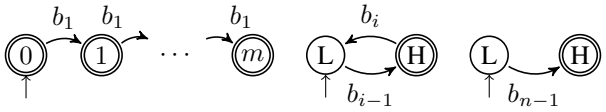
Next, we show that this assumption implies part of our tractability condition. Like Brafman and Domshlak (2008), we leave the mechanism by which local component plans, compatible with constraints imposed by neighbour components, are extracted open, assuming only that it runs in polynomial time.[7] Thus, if we impose this bound artificially and apply iterative deepening, we obtain the same complexity guaratees for non-optimal planning (on solvable instances).

**Theorem 4** *If condition (e) of theorem 2 holds, and if for each component $i$, $|\Sigma_{pub(i)}| \leq M$ and every locally valid plan contains at most $K$ shared action occurrences, where $M$ and $K$ are constant, then conditions (a)–(d) are also met.*
**Proof:** Lemma 5 (below) gives that $|C_{\text{pub}(i)}|$ is polynomially bounded. Moreover, when condition (e) holds $C_{\text{pub}(i)}$ can be constructed in polynomial time, by enumerating sequences of at most $K$ shared actions. Conditions (b) and (d) also follow from lemma 5, and condition (c) from lemma 6. □

**Lemma 5** *Let $A$ be a weighted automaton over alphabet $\Sigma$, and $\Sigma' \subset \Sigma$. If for any word in $w \in \mathcal{L}(A)$, $|w_{|\Sigma'}| \leq K$, i.e., $w$ contains at most $K$ letters from $\Sigma'$, then $\Pi_{\Sigma'}(A)$ is a determinisable WA, and determinisation of this automaton results in a WA that has no more than $K|\Sigma'|^K$ states.*
**Proof:** The automaton obtained by projecting $A$ on $\Sigma'$ may be non-deterministic, but it is determinisable. In short, the reason for this is that it cannot accept strings of arbitrary length (Mohri 2009).

Since the length of words in $\mathcal{L}(\Pi_{\Sigma'}(A))$ is bounded by $K$, there are at most $|\Sigma'|^K$ distinct words in the weighted language, and at most $K|\Sigma'|^K$ distinct prefixes of words in this language. Therefore, any deterministic automaton representing this language can reach at most $K|\Sigma'|^K$ distinct states. Determinisation does not produce an automaton containing states that cannot be reached, nor dead end states

[7] Note, however, that our *implementation* does compute component automata explicitly, and uses a plain search for plan extraction.




(a) (b) (c)

Figure 2: Automata representations of components (a) $C_1$, (b) $C_i$, $i = 2, \ldots, n-1$, and (c) $C_n$.

if the input automaton does not have them. Hence, determinisation of $\Pi_{\Sigma'}(A)$ results in an automaton with at most $K|\Sigma'|^K$ states. □

**Lemma 6** *Let $j_1, \ldots, j_d$ be neighbours of $i$, and for each $j_k$, $\Sigma_{i,j_k}$ their set of shared actions. If $C_{pub(i)}$ and each $M_{j_k,i}$ are deterministic, and any plan accepted by $C_{pub(i)}$ contains at most $K$ (shared) actions, then $C'_{pub(i)} = C_{pub(i)} \times M_{j_1,i} \times \ldots \times M_{j_d,i}$ is a deterministic WA and minimisation of this automaton results in a WA that has no more than $K|\Sigma_{pub(i)}|^K$ states. Furthermore, any plan accepted by $C'_{pub(i)}$ contains no more than $K$ (shared) actions.*

**Proof:** Product preserves determinism: hence $C'_{\text{pub}(i)}$ is deterministic, and therefore minimisable. $\mathcal{L}(C_{\text{pub}(i)})$ contains at most $|\Sigma_{\text{pub}(i)}|^K$ distinct sequences. The language accepted by $C_{\text{pub}(i)} \times M_{j_1,i} \times \ldots \times M_{j_d,i}$ is, modulo weights, the intersection of $\mathcal{L}(C_{\text{pub}(i)})$ and $\mathcal{L}(M_{j_1,i}) \cap \ldots \cap \mathcal{L}(M_{j_d,i})$, so it too cannot accept more than $|\Sigma_{\text{pub}(i)}|^K$ distinct sequences. Thus, there is a deterministic prefix tree automaton (like that in the proof of lemma 5) with at most $K|\Sigma_{\text{pub}(i)}|^K$ states accepting $\mathcal{L}(C_{\text{pub}(i)} \times M_{j_1,i} \times \ldots \times M_{j_d,i})$. □

## Bounded Shared Sequence Length Is Not Essential

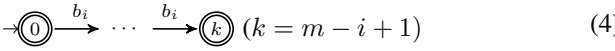
Above we have shown that bounding the number of shared action occurrences in any local plan is sufficient to ensure polynomial runtime. Next we show that it is not necessary, i.e., that the factored planner using the weighted automata representation provably runs in polynomial time also on certain problems where no such bound holds.

We examine a scaling family of problems. Each consists of $n$ components in a line ($n \geq 3$), whose automata are as shown in figure 2. Component $C_1$ can take transition $b_1$ (shared with $C_2$) at most $m$ times, where $m \leq$ **poly**$(n)$. A plan exists iff $m \geq n$. Note that components $C_i$, $1 < i < n$, have locally valid plans that contain an unbounded number of shared action occurrences; hence theorem 4 cannot be used to show tractability on this problem set.

**Theorem 7** *The factored planner decides this problem in time polynomial in $n$.*

**Proof:** The interaction graph is a tree (a simple chain). The size of each component automaton is bounded by $m$, so conditions (a) and (e) of theorem 2 are met.

Pick any component $r$ to be the root. In the first pass, messages are sent from $C_1$ to $C_2$ to $C_3$ etc. up to $C_r$, and from $C_n$ to $C_{n-1}$ to $C_{n-2}$ etc. down to $C_r$. We show, by induction, that message $M_{i,i+1}$ is




$(k = m - i + 1)$ (4)

and, by another induction, that message $M_{i,i-1}$ is




$(k = n - i + 1)$ (5)

The base cases are simple: $M_{1,2}$ equals $C_1$ and $M_{n,n-1}$ equals $C_n$, since all their actions in shared. $M_{i,i+1} = \Pi_{\{b_i\}}(C_i \times M_{i-1,i})$. By inductive assumption, the product $C_i \times M_{i-1,i}$ is


where $k = m - i + 1$. Trimming state $k$,L and projecting on $\{b_i\}$ gives $M_{i,i+1}$ as in (4). Similarly, $C_i \times M_{i+1,i}$ is


where $k = n - i + 1$. Projecting on $\{b_{i-1}\}$ gives (5). If $m - r + 1 \geq n - r + 1$, the product $C_r \times M_{r-1,r} \times M_{r+1,r}$ is the same as $M_{r,r-1}$ shown above. If $m - r + 1 < n - r + 1$, it has no reachable accepting state, as expected since in this case the problem is unsolvable. The messages sent from $C_r$ to $C_{r-1}$ etc. down to $C_1$ continue the same pattern, while the messages from $C_r$ to $C_{r+1}$ etc. up to $C_n$ are identical to those sent in the opposite direction.

Thus, the size of all intermediate results is bounded by $n$ or $m$, giving conditions (b) and (c) of theorem 2. Condition (d) is met since no determinisation is needed. □

## Planning Benchmarks

We believe that the majority of existing planning benchmarks are not suited to factoring, but since no adequate measure of "factorability" is yet known, we can only offer anecdotal evidence of this.

We have examined indicative measures of the interaction graphs, and, of course, tried to come up with working decompositions but failed. As an example, many domains involve objects moving, or being transported, on a "roadmap" or network. These can be decomposed along the network structure, but in this decomposition, the size of each component grows with the total number of moving objects in the problem, since all components must distinguish for each object if it is or is not present. Even in planning domains where the underlying problem intuitively *ought* to be amenable to factoring, we often find that the particular PDDL encodings of those benchmarks do not factor well, or at least not well enough for factored planning to pay off.

Amir & Engelhardt (2003) tested their factored planner on a simple "robot & rooms" domain. We have done the same, and found that our planner scales polynomially with problem size. More relevantly, we have found three standard (IPC) planning benchmark domains for which we were able to formulate new PDDL encodings of their underlying problems in such a way that growing problem size is primarily reflected in the number of components (i.e., size of individual components remains either constant, or tends to grow more slowly). The domains are the two Promela domains (Philosophers and Optical-Telegraph) and Pipesworld (with and without tankage restrictions).[8] Results on these domains are mixed.

[8]The alternative PDDL encodings are available from us on request. The fact that all domains hail from IPC4 is a coincidence.

**The Promela Domains** The IPC4 Promela domains are PDDL encodings deadlock detection problems, generated by automatic translation from models in the Promela language (Hoffmann et al. 2006). The two domains model the classic "dining philosophers" example, and a communications protocol for an optical telegraph system. Both are deadlockable, but can be made deadlock free by a small change to the model (in the case of the dining philosophers, make one of the philosophers pick up his forks in opposite order; in the optical telegraph model, arrange the stations in a line instead of a circle).

Promela models are made up of processes communicating via message channels (queues). In both domains models, the network of processes and channels forms a ring, i.e., each process communicates only with two neighbouring processes (via one or two channels for each). Thus, if each process and channel is made into a component, we would expect to find a very sparse interaction graph. However, the IPC4 PDDL encoding – because it is based on a general, automatic translation from Promela – enforces a global synchronisation between processes and channels, which makes the interaction graph effectively a clique.

We have devised alternative PDDL encodings of both domains which yield the expected interaction structure. (The tree-width of the interaction graph is 2 for Philosophers problems, and at most 4 in the Optical-Telegraph domain.) The encodings are straightforward; the only difficulty is in expressing the global deadlock condition (which is the goal) locally. This is done by allowing each process to *conditionally block* when none of its normal transitions in the current state are applicable (e.g., when the fork it needs to pick up is already taken). The conditional blocking action marks the related channel so that from this point on no other action which would unblock the process (e.g., by returning the missing fork) can take place. When all processes are conditionally blocked, the system is deadlocked.

**Experiments and Results** Figure 3 summarises experiment results in the Promela domains. The results are mostly expected: The factored planner scales polynomially with increasing problem size, and it is totally unaffected by whether the problem has a solution (deadlock) or not. In the Optical-Telegraph domain, the planner spends most of its time (around 90%) constructing initial automata.

As points of comparison, we tested SATPLAN (the IPC 2006 version; Kautz, Selman, and Hoffmann 2006) and the Fast Downward implementation of state space search with the recent landmark cut heuristic (Helmert and Domshlak 2009). As expected, SATPLAN is lightning-fast at finding solutions in the deadlockable problems, but it is completely unable to prove unsolvability of even the tiniest deadlock free instance. The heuristic search based planner scales exponentially, except on solvable instances of the Philosophers domain, where the landmark cut heuristic turns out to achieve perfect accuracy.

**Pipesworld** The Pipesworld domain models the problem of transporting (liquid) products through a network of pipelines and transit areas. The main simplification of the planning benchmark, compared to the real application, is that the continuous flow of liquid is divided into discrete,

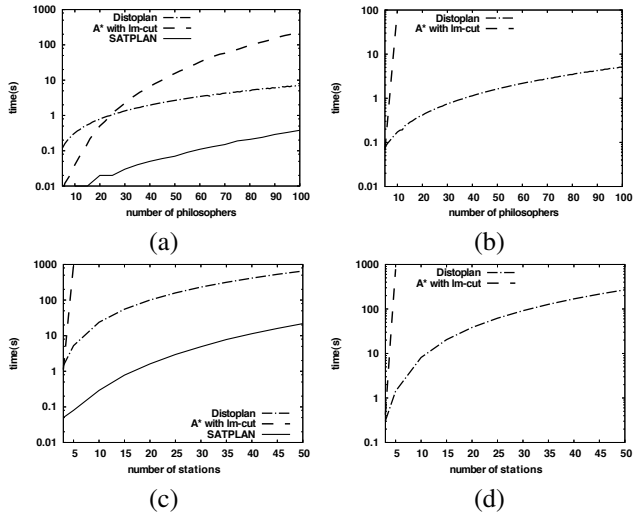



Figure 3: Planner runtimes (logarithmic scale) on the alternative PDDL encodings of Promela domains: (a) Philosophers, deadlockable; (b) Philosophers, deadlock free; (c) Optical-Telegraph, deadlockable; (d) Optical-Telegraph, deadlock free.

unit-sized "batches" (Hoffmann et al. 2006).

The pipeline networks are fairly sparse, and operations on each pipeline affect only the adjacent transit areas, so the problem can be decomposed along the network structure. However, the IPC4 PDDL encoding gives every batch a unique name (whereas in the real application, only the type of product it is made up of matters). Just as in other "named object" movement domains, this makes components grow in size with the total number of batches in the system, which grows, quite fast, with network size.

We replace the named batches by (bounded) counters keeping track of the number of batches of each product type in each area. For each pipeline segment, we use a number of ordered "slots", equal to the segment length, which record what type of product is at each position (this is needed to model the FIFO behaviour of a pipe). In this encoding, component automata grow with the maximum number of batches of each type, which is typically a smaller quantity (across the IPC4 problem set, the maximum is 9). Limitations on storage space in areas, which are modelled in the "tankage" version of the domain, further limit the range of counters.

Mapping problem instances from the IPC4 Pipesworld encoding to our formulation is not as straightforward as in the Promela domains. Depending on whether we interpret the goal as absolute (i.e., "have N units of type X at A") or relative (i.e., "have N units *more* of type X at A") we can end up with problems that are much easier, or problems that are unsolvable. We experimented with both mappings.

**Results and Analysis** The reformulated Pipesworld domain seems like a good candidate for a problem suited to factored planning, but the performance of the planner is disappointing: it fails to solve even the smallest IPC instances in reasonable time. To understand why, we examine a family of simple problem instances, shown schematically in figure

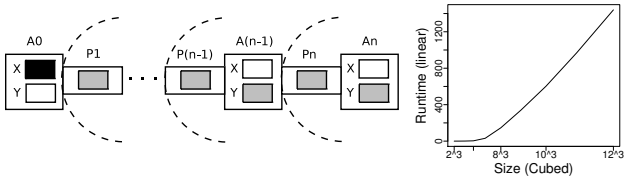



Figure 4: (a) Layout of the Pipesworld problem (two product types). Dashed lines mark component boundaries. A$i$'s are storage areas, P$i$'s are pipes; white slots are empty. The goal is to have one batch of type X at A$n$. (b) Runtime vs. $n^3$.

4(a). On these instances, the planner scales roughly as $n^3$, as shown in figure 4(b).

We can show a lower bound on the size of messages sent, which is polynomial in the length of the component chain (actually, in the total amount of storage spaces in it) but exponential in the number of different product types. The main reason for this is that the message sent from the $i$th component to its neighbour describes all possible plans for the subsystem to one side of it, *assuming no constraints from the other*. We conjecture that there is a similar upper bound. Even though the smaller instances in the IPC4 set have networks that are short lines like this, the number of products and storage capacities are much higher than in our problem, causing message sizes to rise very rapidly.

## Conclusions

There are still significant gaps in our understanding of factored planning. What are adequate measures of factorability, and how can we use them to automatically detect and decompose problems amenable to factoring? What restrictions are essential to guarantee polynomial complexity? Bounded tree-width (of some interaction graph) is known to be necessary, but not sufficent; we have shown that additionally limiting the number of shared action occurrences is sufficient, but not necessary.

Previous factored planning methods have used bounded representations in combination with iterative deepening, essentially running the factored solver many times, whereas we use an unbounded representation to compute all valid plans in one run. Both methods have their flaws: The iterative method cannot prove unsolvability nor optimality without exhausting the bound space. Our method requires stronger conditions to guarantee polynomial runtime, which is overkill for solvable instances of non-optimal planning.

There is a space of other possibilities to explore: intelligent combinations of backtracking search and computing entire sets of local plans, and alternative message passing strategies such as iteratively tightening constraints across the whole system using multiple passes.

Analysis of the cases where the factored planner fails suggest we should perhaps be looking at *more* powerful representations, instead of the (less powerful) intrinsically bounded ones. For example, there are regular expressions with exponentiation, which have a corresponding automata representation. And, of course, the most compact representation we have of a set of plans is a planning problem. The question is how to perform projection on it?

Finally, we have observed that current planning benchmarks do not factor well. In some cases, this can be blamed on the encoding, but in most, it appears to us that no amount of reformulation is going to help. We conjecture that a reason for this is that planning benchmarks usually model only one narrow aspect of an application problem. If we were tackling an integrated planning problem, e.g., involving logistics, inventory management, production planning, etc., opportunities for decomposition would naturally arise.

**Acknowledgements** This work was supported by the Franco-Australian program for Science and Technology, Grants 18616NL and FR080033. E. Fabre is also supported by the European 7th FP project DISC (DIstributed Supervisory Control of large plants), Grant INFSO-ICT-224498, and by the INRIA/ALU-Bell joint research lab. S. Thiébaux and P. Haslum are supported by the Australian Research Council discovery project DP0985532 "Exploiting Structure in AI Planning".

## References

Amir, E., and Engelhardt, B. 2003. Factored planning. In *Proc. IJCAI'03*.

Brafman, R., and Domshlak, C. 2006. Factored planning: How, when and when not. In *Proc. AAAI'06*.

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proc. ICAPS'08*.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Fabre, E., and Jezequel, L. 2009. Distributed optimal planning: an approach by weighted automata calculus. In *Proc. CDC'09*.

Fabre, E. 2003. Convergence of the turbo algorithm for systems defined by local constraints. Research report PI 4860, INRIA.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS'09*.

Hoffmann, J.; Edelkamp, S.; Thiébaux, S.; Englert, R.; Liporace, F.; and Trüg, S. 2006. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of AI Research* 26:453–541.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SATPLAN: Planning as satisfiability. In *5th International Planning Competition Booklet*. `http://zeus.ing.unibs.it/ipc-5/`.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored planning using decomposition trees. In *Proc. IJCAI'07*.

Mohri, M. 2009. Weighted automata algorithms. In *Handbook of Weighted Automata*. Springer. chapter 6, 213–255.

Pearl, J. 1986. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence* vol. 29:241–288.