

On the Hardness of Planning Problems with Simple Causal Graphs

Omer Giménez

Dept. of Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona, 1-3
08034 Barcelona, Spain
omer.gimenez@upc.edu

Anders Jonsson

Dept. of Information and Communication Technologies
Universitat Pompeu Fabra
Passeig de Circumval·lació, 8
08003 Barcelona, Spain
anders.jonsson@upf.edu

Abstract

We present three new complexity results for classes of planning problems with simple causal graphs. First, we describe a polynomial time algorithm that uses macros to generate plans for a class of planning problems with binary state variables and acyclic causal graphs. This implies that plan generation may not be intractable just because a planning problem has exponential length solution. We also prove that the problem of plan existence for planning problems with multi-valued variables and chain causal graphs is NP-hard. Finally, we show that plan existence for planning problems with binary state variables and polytree causal graphs is NP-complete.

Introduction

Planning in artificial intelligence is the problem of obtaining a sequence of transformations for moving a system from an initial state to a goal state. Researchers usually distinguish between plan generation, the problem of generating such a sequence, and plan existence, the problem of determining whether such a sequence exists. If the original STRIPS formalism is used, plan existence is undecidable in the first-order case (Chapman 1987) and PSPACE-complete in the propositional case (Bylander 1994). However, it is widely known that many real-life problems exhibit structure that can be exploited to solve these problems more efficiently.

One of the most fruitful tools that researchers have used to characterize structure in planning problems is the so called *causal graph* (Knoblock 1994). The causal graph of a planning problem is a graph that captures the degree of interdependence among the state variables of the problem. The causal graph has been used both as a tool for describing tractable subclasses of planning problems (Brafman & Domshlak 2003; 2006; Williams & Nayak 1997) and as the basis for domain-independent heuristics that guide the search for a valid plan (Helmert 2006).

In the present work we explore the computational complexity of solving planning problems with simple causal graphs. We present new results for three classes of planning problems studied in the literature: the class 3S (Jonsson & Bäckström 1998), the class \mathbb{C}_n (Domshlak & Dinitz 2001), and the class of planning problems with polytree

causal graphs (Brafman & Domshlak 2003). In brief, we show that plan generation for instances of the first class can be solved in polynomial time using macros, but that we cannot hope to solve plan existence in polynomial time for the remaining two classes, unless $P = NP$.

The class 3S was conceived to show that tractable plan existence does not always imply tractable plan generation. Jonsson and Bäckström (1998) described a polynomial time algorithm for determining whether or not a valid plan exists for instances of 3S. On the other hand, there are instances of 3S with optimal solutions that are exponential in the size of the input. This result is sometimes interpreted as being *stronger* than, say, being NP-hard, since it is not known whether problems in NP are intractable, but it is certain that we cannot generate exponential length output in polynomial time. To improve on this result, Jonsson and Bäckström (1998) showed that it is possible to generate valid plans in polynomial time in the size of the *output*.

However, it is not clear if plan generation is inherently hard, or if the difficulty just lies in the fact that plans may be very long. Consider the two functional problems

$$f_1(n) = w(1, 2^n),$$

$$f_2(F) = w(t(F), 2^{|F|}),$$

where n is a natural number, F is a 3-CNF formula, $|F|$ is the number of clauses of F , $w(\sigma, k)$ is a word containing k copies of the symbol σ , and $t(F)$ is 1 if F is satisfiable (i.e., F is in 3-SAT), and 0 if it is not. In both cases, the problem consists in generating the correct word. Observe that both f_1 and f_2 are provably intractable, since their output is at least exponential in the size of the input, and that both can be solved in polynomial time in the size of the output. Hence, on first sight, they appear to have the same hardness as plan generation for instances of 3S.

Nevertheless, it is intuitive to regard problem f_1 as *easier* than problem f_2 , even when, in fact, the output of f_1 is doubly exponential in the bit size of the input. One way to formalize this intuition is to allow programs to produce the output in some succinct notation. For instance, if we allow programs to write “ $w(\sigma, k)$ ” instead of a string containing k copies of the symbol σ , and “ $\exp(2, n)$ ” instead of the bit representation of 2^n , then problem f_1 becomes polynomial, but problem f_2 does not (unless $P = NP$).

We wanted to investigate the following question: regarding plan generation for 3S, is the source of intractability that solution plans are long, like f_1 , or that the problem is intrinsically hard, like f_2 ? The answer is that plan generation for 3S can be solved in polynomial time, provided that one is allowed to give the solution in terms of macros, where a macro is a simple substitution scheme: a sequence of operators and/or other macros. To back up this claim, we present an algorithm that solves plan generation for 3S in polynomial time.

The idea of using macros in planning is almost as old as planning itself (Fikes & Nilsson 1971). Minton (1985) and Korf (1987) were among the first to successfully apply macros to planning. Recent successful approaches include those of Vidal (2004) and Botea et al. (2005). Jonsson (2007) described an algorithm that uses macros to generate plans for planning problems with tree-reducible causal graphs. There exist planning problems for which the algorithm can generate exponentially long solutions in polynomial time, just like our algorithm for 3S. Unlike ours, the algorithm can handle multi-valued variables, but not all planning problems in 3S have tree-reducible causal graphs.

Other researchers have argued intractability using the fact that plans may have exponential length. Domshlak and Dinitz (2001) proved complexity results for several classes of planning problems with multi-valued state variables and simple causal graphs. They argued that the class \mathbb{C}_n of planning problems with chain causal graphs is intractable since plans may have exponential length. Brafman and Domshlak (2003) stated that plan existence for STRIPS planning problems with unary operators and acyclic causal graphs is intractable using the same reasoning. Our new result puts in question the actual hardness of these problems.

In the remaining sections of the paper, we investigate the complexity of the class \mathbb{C}_n of planning problems with multi-valued state variables and chain causal graphs. In other words, the causal graph is just a directed path. Even if instances of this class may have exponential length solutions, it is possible that the instances are inherently easy to solve, as is the case for 3S. However, even for this restricted class of problems, we show that the problem of plan existence is NP-hard.

We also show that plan existence for planning problems whose causal graph is a polytree (i.e., the underlying undirected graph is acyclic) is NP-complete, even if we restrict to problems with binary variables. This result closes the complexity gap that appears in Brafman and Domshlak (2003), where it is shown that plan existence is NP-complete for planning problems with singly connected causal graphs, and that plan generation is polynomial for planning problems with polytree causal graphs of bounded indegree.

Notation

Let V be a set of state variables, and let $D(v)$ be the finite domain of state variable $v \in V$. We define a state s as a function on V that maps each state variable $v \in V$ to a value $s(v) \in D(v)$ in its domain. A partial state p is a function on a subset $V_p \subseteq V$ of state variables that maps each state variable $v \in V_p$ to $p(v) \in D(v)$. For a subset $C \subset V$ of state

variables, $p \upharpoonright C$ is the partial state obtained by restricting the domain of p to $V_p \cap C$. Sometimes we use the notation $(v_1 = x_1, \dots, v_k = x_k)$ to denote a partial state p defined by $V_p = \{v_1, \dots, v_k\}$ and $p(v_i) = x_i$ for each $v_i \in V_p$.

A planning problem is a tuple $P = \langle V, \text{init}, \text{goal}, A \rangle$, where V is the set of variables, init is an initial state, goal is a partial goal state, and A is a set of operators. An operator $a = \langle \text{pre}(a); \text{post}(a) \rangle \in A$ consists of a partial state $\text{pre}(a)$ called the *pre-condition* and a partial state $\text{post}(a)$ called the *post-condition*. Operator a is applicable in any state s such that $s \upharpoonright V_{\text{pre}(a)} = \text{pre}(a)$, and applying operator a in state s results in a new state s' such that $s'(v) = \text{post}(a)(v)$ for each $v \in V_{\text{post}(a)}$ and $s'(v) = s(v)$ otherwise.

The causal graph of a planning problem P is a graph (V, E) with state variables as nodes. There is an edge $(u, v) \in E$ if and only if there exists an operator $a \in A$ such that $u \in V_{\text{pre}(a)} \cup V_{\text{post}(a)}$ and $v \in V_{\text{post}(a)}$. We define a macro m as an operator augmented with an operator sequence. In other words, m has a pre-condition $\text{pre}(m)$ and a post-condition $\text{post}(m)$, as well as an associated operator sequence $op(m)$ composed of either operators in A or other macros. For a valid macro m , the operator sequence $op(m)$ is applicable in $\text{pre}(m)$ and results in the partial state $\text{post}(m)$ when executed. However, $op(m)$ may very well be applicable in states that do not coincide with $\text{pre}(m)$.

3S

Jonsson and Bäckström (1998) introduced the 3S class of planning problems with binary state variables and acyclic causal graphs. In addition to these restrictions, state variables belong to one of three categories that we describe below. Since state variables are binary, the domain of each state variable $v \in V$ is $D(v) = \{0, 1\}$. The fact that the causal graph is acyclic implies that operators are unary, i.e., for each operator $a \in A$, $|V_{\text{post}(a)}| = 1$. Also, for each state variable v , we can form the subset $\text{Anc}(v) \subset V$ of the ancestors of v in the causal graph.

For each state variable $v \in V$, let A_1^v be the subset of operators whose pre-condition specifies $v = 1$, i.e., $A_1^v = \{a \in A \mid v \in V_{\text{pre}(a)} \wedge \text{pre}(a)(v) = 1\}$. Let Q_1^v be the set of state variables in the post-condition of operators in A_1^v , i.e., $Q_1^v = \{u \mid \exists a \in A_1^v \text{ s.t. } V_{\text{post}(a)} = \{u\}\}$. Let (V, E_1^v) be the subgraph of (V, E) whose edges exclude operators in A_1^v , i.e., $E_1^v = \{(u, w) \mid \exists a \in A - A_1^v \text{ s.t. } u \in V_{\text{pre}(a)} \wedge w \in V_{\text{post}(a)}\}$. Let V_1^v be the set of state variables w such that there is an undirected path from any state variable u in Q_1^v to w in the subgraph (V, E_1^v) . Define V_0^v in the same way for $v = 0$.

For planning problems in the 3S class, each state variable $v \in V$ belongs to one of the following three categories:

1. **static**: the value of v must not or cannot change,
2. **symmetrically reversible**: for each operator $a \in A$ such that $V_{\text{post}(a)} = \{v\}$, there exists a symmetric operator $a' \in A$ such that $V_{\text{pre}(a')} = V_{\text{pre}(a)}$, $V_{\text{post}(a')} = \{v\}$, $\text{post}(a')(v) = \text{pre}(a)(v)$, $\text{pre}(a')(v) = \text{post}(a)(v)$, and $\text{pre}(a') \upharpoonright \text{Anc}(v) = \text{pre}(a) \upharpoonright \text{Anc}(v)$,

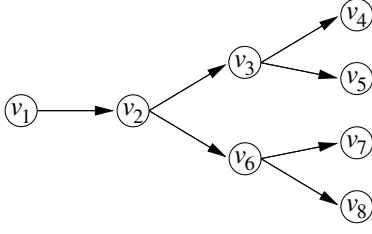


Figure 1: Causal graph of a planning problem in 3S

3. splitting: v is not static, not reversible, and the two sets V_1^v and V_0^v are disjoint.

Inclusion in each category can be checked in polynomial time; we refer to Jonsson and Bäckström (1998) for details. We illustrate the 3S class using an example planning problem. The set of state variables is $V = \{v_1, \dots, v_8\}$ where v_1 is static, v_2 and v_8 are splitting, and v_3 – v_7 are symmetrically reversible. The initial state is defined by $init(v_i) = 0$ for each $v_i \in V$ and the goal state is defined by $goal = (v_4 = 1, v_5 = 1, v_8 = 1)$. Figure 1 shows the causal graph (V, E) of the planning problem.

For each $v_i \in \{v_3, v_4, v_5, v_7, v_8\}$ there is an operator $a_i = \langle v_j = 1, v_i = 0; v_i = 1 \rangle$, where v_j is the parent of v_i in the causal graph. For each $v_i \in \{v_2, v_6\}$ there is an operator $a_i = \langle v_j = 0, v_i = 0; v_i = 1 \rangle$. Since v_3 – v_7 are symmetrically reversible, there is a corresponding symmetric operator a'_i for each of these state variables. For splitting state variable v_2 , $Q_1^{v_2} = \{v_3\}$, the graph $(V, E_1^{v_2})$ excludes the edge (v_2, v_3) , and $V_1^{v_2} = \{v_3, v_4, v_5\}$. Likewise, $V_0^{v_2} = \{v_6, v_7, v_8\}$. For splitting state variable v_8 , $V_1^{v_8} = V_0^{v_8} = \emptyset$.

Algorithm

We present a polynomial time algorithm for plan generation in 3S. The idea is to construct macros that change the value of a single state variable. The macros may change the values of other state variables during execution, but always reset them before terminating. Once the macros have been generated, the goal can be achieved one state variable at a time. We show that the algorithm generates a valid plan if and only if one exists. The solution is a sequence of macros that completely describe the steps needed to achieve the goal.

First, perform a topological sort of the state variables, which is possible since the causal graph is acyclic. During this process, take special care of each splitting state variable v . Since V_1^v and V_0^v are disjoint, state variables in one set are independent of state variables in the other, so their mutual order in the topological sort can be arbitrarily selected. If $init(v) = 1$, order all state variables in V_0^v before state variables in V_1^v in the topological sort. If $init(v) = 0$, order V_1^v before V_0^v .

For each state variable v , the algorithm maintains a start state $start^v$, initially equal to the initial state. The role of the start state is to detect possible changes in the values of splitting state variables, which is later used to determine whether

it is possible to satisfy the pre-condition of some operators. The algorithm generates macros for each state variable v in topological order. Depending on the category to which v belongs, perform one of the following steps:

1. If v is static, do nothing.
2. If v is symmetrically reversible, attempt to generate a macro m_0^v such that $V_{pre(m_0^v)} = Anc(v) \cup \{v\}$, $V_{post(m_0^v)} = \{v\}$, $pre(m_0^v)(u) = start^v(u)$ for each $u \in Anc(v)$, $pre(m_0^v)(v) = 0$, and $post(m_0^v)(v) = 1$. If this is possible, also generate a symmetric macro m_1^v with $pre(m_1^v) \upharpoonright Anc(v) = pre(m_0^v) \upharpoonright Anc(v)$, $pre(m_1^v)(v) = 1$, and $post(m_1^v)(v) = 0$.
3. If v is splitting, attempt to generate the macro $m_{init(v)}^v$ in the same way as for symmetrically reversible state variables. In other words, if $init(v) = 0$, generate the macro m_0^v , otherwise generate m_1^v . In case $init(v) = 0$ and m_0^v is successfully generated, modify the start state of each state variable $u \in V_1^v$ to be $start^u(v) = 1$. In case $init(v) = 1$ and m_1^v is generated, modify the start state of each $u \in V_0^v$ to be $start^u(v) = 0$.

To generate the macro $m_{init(v)}^v$ for state variable v , go through each operator $a \in A$ such that $V_{post(a)} = \{v\}$, $pre(a)(v) = init(v)$, and $post(a)(v) \neq init(v)$. Attempt to satisfy the pre-condition $pre(a)$ of a . Let $Z(a) = \{u_1, \dots, u_k\} \subseteq Anc(v)$ be the set of ancestors of v such that $start^v(u_i) \neq pre(a)(u_i)$, preserving their topological order. In other words, $Z(a)$ is the set of state variables whose values need to be changed to satisfy $pre(a)$. It is possible to satisfy $pre(a)$ if and only if each state variable $u_i \in Z(a)$ is symmetrically reversible and it was possible to generate the macros $m_0^{u_i}$ and $m_1^{u_i}$ (if u_i is splitting the value of $start^v(u_i)$ would have been altered as a result of generating a macro for u_i). Since each u_i is symmetrically reversible, it follows that $start^v(u_i) = init(u_i)$.

Note that the net result of executing a macro is changing the value of a single state variable, even though the values of other state variables may change during the course of its execution. In particular, the macro $m_{init(u_k)}^{u_k}$ changes the value of state variable u_k from $init(u_k)$ to $pre(a)(u_k)$, leaving the values of u_1, \dots, u_{k-1} unchanged. Since u_k comes after u_1, \dots, u_{k-1} in the topological sort, it cannot be an ancestor of any of these state variables. Consequently, executing $m_{init(u_k)}^{u_k}$ does not interfere with the pre-condition of the macros that were generated for these state variables.

It is easy to show that the sequence of macros $S_1 = \langle m_{init(u_k)}^{u_k}, \dots, m_{init(u_1)}^{u_1} \rangle$ changes the values of state variables in $Z(a)$ from $init$ to $pre(a)$. Conversely, the sequence $S_2 = \langle m_{pre(a)(u_1)}^{u_1}, \dots, m_{pre(a)(u_k)}^{u_k} \rangle$ changes the values of state variables in $Z(a)$ back to $init$. A valid operator sequence of $m_{init(v)}^v$ is $op(m_{init(v)}^v) = \langle S_1, a, S_2 \rangle$. If v is symmetrically reversible, there exists an operator a' such that $V_{post(a')} = \{v\}$, $pre(a')(v) \neq init(v)$, $post(a')(v) = init(v)$, and $pre(a') \upharpoonright Anc(v) = pre(a) \upharpoonright Anc(v)$. If it is possible to generate $m_{init(v)}^v$, it is always possible to generate a symmetric macro with operator sequence $\langle S_1, a', S_2 \rangle$.

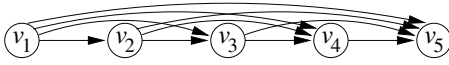


Figure 2: Causal graph of the planning problem P_5

To generate the final plan, go through the state variables in reverse topological order. For each state variable v , if $v \in V_{goal}$ and $goal(v) \neq init(v)$, append the macro $m_{init(v)}^v$ to the current plan. Recall that this macro does not interfere with the values of the ancestors of v . Consequently, the preconditions of macros that change the values of the ancestors are preserved. If the macro $m_{init(v)}^v$ was not generated, there exists no valid plan. For a splitting state variable v , if a macro was generated, the macro needs to be inserted in the plan between V_0^v and V_1^v in the appropriate place.

Examples

We illustrate the algorithm on the planning problem from the previous section. For splitting state variable v_2 , $init(v_2) = 0$, $V_1^{v_2} = \{v_3, v_4, v_5\}$, and $V_0^{v_2} = \{v_6, v_7, v_8\}$. The algorithm orders $V_1^{v_2}$ before $V_0^{v_2}$, so v_1, \dots, v_8 is a valid topological sort of the state variables. State variable v_1 is static so the algorithm does nothing. Operator a_2 satisfies $pre(a_2)(v_2) = 0 = init(v_2)$ and $post(a_2)(v_2) = 1 \neq init(v_2)$. Since $pre(a_2)(v_1) = 0$ and $start^{v_2}(v_1) = init(v_1) = 0$, $Z(a_2)$ is empty, so the algorithm generates the macro $m_0^{v_2}$ with operator sequence $op(m_0^{v_2}) = \langle a_2 \rangle$. In addition, the start state of each state variable $v_i \in V_1^{v_2}$ is modified to be $start^{v_i}(v_2) = 1$.

The algorithm proceeds to generate macros $m_0^{v_i}$ for each v_3-v_8 and $m_1^{v_i}$ for each v_3-v_7 . The operator sequences of macros $m_0^{v_i}$ are $op(m_0^{v_3}) = \langle a_3 \rangle$, $op(m_0^{v_4}) = \langle m_0^{v_3}, a_4, m_1^{v_3} \rangle$, $op(m_0^{v_5}) = \langle m_0^{v_3}, a_5, m_1^{v_3} \rangle$, $op(m_0^{v_6}) = \langle a_6 \rangle$, $op(m_0^{v_7}) = \langle m_0^{v_6}, a_7, m_1^{v_6} \rangle$, and $op(m_0^{v_8}) = \langle m_0^{v_6}, a_8, m_1^{v_6} \rangle$. To generate the final plan, the algorithm achieves the goal state of state variables in reverse topological order. This results in the macro sequence $\langle m_0^{v_8}, m_0^{v_5}, m_0^{v_4} \rangle$. However, since the macro $m_0^{v_2}$ was generated, it needs to be inserted between $V_1^{v_2}$ and $V_0^{v_2}$, resulting in the final plan $\langle m_0^{v_8}, m_0^{v_2}, m_0^{v_5}, m_0^{v_4} \rangle$. This is equivalent to the operator sequence $\langle a_6, a_8, a_6', a_2, a_3, a_5, a_3', a_3, a_4, a_3' \rangle$. This plan is not optimal; an example of an optimal plan is $\langle a_6, a_8, a_2, a_3, a_5, a_4 \rangle$.

The following example was introduced by Jonsson and Bäckström (1998) to show that there are instances of 3S with exponentially sized minimal solutions. Let $P_n = \langle V, init, goal, A \rangle$ be the planning problem, where n is a natural number and $V = \{v_1, \dots, v_n\}$. Let the initial state be defined by $init(v_i) = 0$ for each state variable $v_i \in V$, and let the goal state be defined by $V_{goal} = V$, $goal(v_i) = 0$ for each $v_i \in \{v_1, \dots, v_{n-1}\}$, and $goal(v_n) = 1$.

For each state variable v_i , there are two operators, $a_i = \langle v_1 = 0, \dots, v_{i-2} = 0, v_{i-1} = 1, v_i = 0; v_i = 1 \rangle$ and $a_i' = \langle v_1 = 0, \dots, v_{i-2} = 0, v_{i-1} = 1, v_i = 1; v_i = 0 \rangle$. In other words, each state variable is symmetrically reversible. To change the value of v_i it is necessary to set v_{i-1} to 1 and the

remaining ancestors to 0. The causal graph of the planning problem P_5 is shown in Figure 2. Bäckström and Nebel (1995) showed that the length of the shortest plan solving P_n is $2^n - 1$, i.e., exponential in the number of state variables.

For each state variable v_i , our algorithm generates two macros $m_0^{v_i}$ and $m_1^{v_i}$. Since each state variable is symmetrically reversible, the start state of v_i is equal to the initial state. There is a single operator, a_i , that changes the value of v_i from 0 to 1. The only state variable whose value in $init$ differs from that in $pre(a_i)$ is v_{i-1} , so $Z(a_i) = \{v_{i-1}\}$. Thus, the operator sequence of $m_0^{v_i}$ is $op(m_0^{v_i}) = \langle m_0^{v_{i-1}}, a_i, m_1^{v_{i-1}} \rangle$. Similarly, the operator sequence of $m_1^{v_i}$ is $op(m_1^{v_i}) = \langle m_0^{v_{i-1}}, a_i', m_1^{v_{i-1}} \rangle$.

To generate the final plan, the algorithm appends macros for state variables whose value in the goal state differs from that in the initial state. There is a single such state variable, v_n , so the solution simply consists of the sequence $\langle m_0^{v_n} \rangle$. The operator sequence of the macro $m_0^{v_n}$ consists of the sequence $\langle m_0^{v_{n-1}}, a_n, m_1^{v_{n-1}} \rangle$. If we continue to expand these macros, we end up with a sequence of $2^n - 1$ operators. However, no individual macro has operator sequence length greater than 3. Together, the macros recursively specify a complete solution to the planning problem.

Completeness and Complexity

In this section we prove that our algorithm is sound and complete, i.e., it generates a valid plan if and only if one exists. In addition, we prove that the algorithm runs in polynomial time.

Lemma 1 *The algorithm generates the macro $m_{init(v)}^v$ for state variable v if and only if there exists an operator sequence that starts in $init$ and changes the value of v .*

Proof By induction on state variables v . If $|Anc(v)| = 0$, there exists an operator sequence that starts in $init$ and changes the value of v if and only if there exists an operator a such that $pre(a)(v) = init(v)$ and $post(a)(v) \neq init(v)$. In this case, the algorithm generates the macro $m_{init(v)}^v$ with $op(m_{init(v)}^v) = \langle a \rangle$. Otherwise, $m_{init(v)}^v$ is not generated.

If $|Anc(v)| > 0$, there exists an operator sequence that starts in $init$ and changes the value of v if and only if there exists an operator a such that $pre(a)(v) = init(v)$, $post(a)(v) \neq init(v)$, and it is possible to satisfy the precondition $pre(a)$ of a . If no such operator exists, $m_{init(v)}^v$ is not generated. Otherwise, it is possible to change the value of each state variable in the set $Z(a) = \{u \mid u \in V_{pre(a)} \wedge init(u) \neq pre(a)(u)\}$. By hypothesis of induction, the algorithm generates the macro $m_{init(u)}^u$ for each $u \in Z(a)$.

If u is splitting, the algorithm changes the start state of v such that $start^v(u) = pre(a)(u)$. Let u_1, \dots, u_k be the remaining state variables in $Z(a)$, preserving their topological order. These state variables have to be symmetrically reversible, so in addition to $m_{init(u_i)}^{u_i}$, the algorithm also generates the symmetric macro $m_{pre(a)(u_i)}^{u_i}$ for each u_i . Consequently, the algorithm generates the macro $m_{init(v)}^v$ with operator sequence $op(m_{init(v)}^v) =$

$\langle S_1, a, S_2 \rangle$, where $S_1 = \langle m_{init(u_k)}^{u_k}, \dots, m_{init(u_1)}^{u_1} \rangle$ and $S_2 = \langle m_{pre(a)(u_1)}^{u_1}, \dots, m_{pre(a)(u_k)}^{u_k} \rangle$.

Theorem 2 *The proposed algorithm generates a valid plan for planning problems in 3S if and only if one exists.*

Proof It should be clear from the description of the algorithm that each macro generated by the algorithm is valid. It remains to be shown that the final plan generated by the algorithm is valid. For each state variable v such that $v \in V_{goal}$ and $goal(v) \neq init(v)$, it follows from Lemma 1 that the algorithm generates a macro $m_{init(v)}^v$ if and only if there exists an operator sequence that starts in $init$ and changes the value of v . If this is not the case, no valid plan exists, and the algorithm does not generate a final plan.

Note that the macro $m_{init(v)}^v$ only changes the value of v , leaving the values of all other state variables unchanged. In particular, this preserves the pre-condition of the macro $m_{init(u)}^u$ of each state variable u that precedes v in the topological sort. Thus, we can apply macros to change the values of state variables in inverse topological order. Since some macros assume that the value of a splitting state variable v is different from that in the initial state, it is necessary to insert the macro that changes the value of v in the correct place in the plan. This is the procedure followed by the algorithm.

Applying a macro that changes the value of a splitting state variable v never invalidates the plan. From the definition of splitting it follows that v is neither static nor reversible. In other words, it is possible to change the value of v once from its initial value (otherwise v would be static) but it is not possible to change the value of v back to its initial value (otherwise v would be reversible).

Theorem 3 *The above algorithm for plan generation in 3S runs in polynomial time with complexity $O(|V|^2 + |V||A|)$.*

Proof For each state variable v , the algorithm performs one of the steps 1-3 above. If v is static, the algorithm does nothing. If v is symmetrically reversible, the algorithm attempts to generate two macros. For each operator a that changes the value of v , determining $Z(a)$ and checking whether each state variable in $Z(a)$ is symmetrically reversible and has the two required macros can be done in time linear in the number of state variables in $Anc(v)$. If a is applicable, the resulting macro is composed of two macros for each state variable in $Z(a)$ plus a , which is also linear in $|Anc(v)|$. Assuming v is the i th variable in the topological sort, v has $O(i)$ ancestors, so the complexity of this step is $O(i|A_i|)$, where A_i is the subset of operators that change the value of v . If v is splitting, the algorithm generates a macro for v in the same way as for symmetrically reversible state variables. In addition, the algorithm changes the start state of some of its descendents. In the worst case, this is linear in $O(|V| - i)$, the number of descendents of v , so the complexity of this step is $O((|V| - i) + i|A_i|)$. The complexity of assembling the final plan is $O(|V|)$, since we just need to go through the state variables once, and determining whether or not a required macro exists can be done in constant time. In total, the complexity of the algorithm is $O(|V| + \sum_{i=1}^{|V|} [(|V| - i) + i|A_i|]) = O(|V|^2 + |V||A|)$.

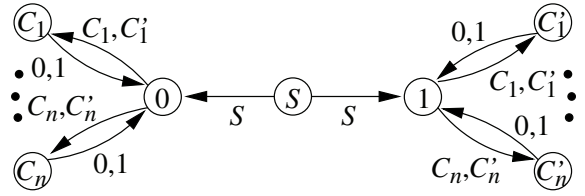


Figure 3: Domain transition graph for v_i

C_n

Domshlak and Dinitz (2001) defined the class C_n of planning problems with multi-valued state variables and chain causal graphs. Since chain causal graphs are acyclic, it follows that operators are unary. Moreover, let v_i be the i th state variable in the chain. If $i > 1$, for each operator a such that $V_{post(a)} \subseteq \{v_i\}$ it holds that $V_{pre(a)} = \{v_{i-1}, v_i\}$. In other words, each operator that changes the value of a state variable v_i may only have pre-conditions on v_{i-1} and v_i .

The authors showed that there are instance of C_n with exponentially sized minimal solutions, and therefore argued that the class is intractable. In light of the previous section, this argument on the length of the solutions does not discard the possibility that instances of the class can be solved in polynomial time using macros. We show that this is not the case, unless $P = NP$.

We define the decision problem $PLAN-EXISTENCE-C_n$ as follows. A valid input of $PLAN-EXISTENCE-C_n$ is a planning instance P of C_n . The input P belongs to $PLAN-EXISTENCE-C_n$ if and only if P is solvable. We show in this section that the problem $PLAN-EXISTENCE-C_n$ is NP-hard. This implies that, unless $P = NP$, solving instances of C_n is a truly intractable problem, namely, no polynomial time algorithm can distinguish between solvable and unsolvable instances of C_n . In particular, no polynomial time algorithm can solve C_n instances by using macros or any other kind of output format.¹

We prove that $PLAN-EXISTENCE-C_n$ is NP-hard by a reduction from CNF-SAT, that is, the problem of determining whether a CNF formula F is satisfiable. Let C_1, \dots, C_n be the clauses of the CNF formula F , and let v_1, \dots, v_k be the variables that appear in these clauses. Define a planning problem $P(F) = \langle V, init, goal, A \rangle$ as follows. The variable set V is $\{v_1, \dots, v_k, w\}$, where $D(v_i) = \{S, 0, 1, C_1, C_1', \dots, C_n, C_n'\}$ for each v_i and $D(w) = \{S, 1, \dots, n\}$. The initial state defines $init(v) = S$ for each $v \in V$ and the goal state defines $goal(w) = n$.

The domain transition graph for each state variable v_i is shown in Figure 3. Each node represents a value in $D(v_i)$, and an edge from x to y means that there exists an operator a such that $pre(a)(v_i) = x$ and $post(a)(v_i) = y$. Edge labels represent the pre-condition of such operators on state variable v_{i-1} , and multiple labels indicate that several operators

¹A valid output format is one that enables efficient distinction between an output representing a valid plan and an output representing the fact that no solution was found.



Figure 4: Domain transition graph for w

are associated with an edge. We enumerate the operators acting on v_i using the notation $a = \langle pre(a); post(a) \rangle$ (when $i = 1$ any mention of v_{i-1} is understood to be void):

- (1) Operators $\langle v_{i-1} = S, v_i = S; v_i = 0 \rangle$ and $\langle v_{i-1} = S, v_i = S; v_i = 1 \rangle$ allow v_i to move from S to either 0 or 1.
- (2) Only when $i > 1$. For every clause C_j , four operators $\langle v_{i-1} = X, v_i = 0; v_i = C_j \rangle$ and $\langle v_{i-1} = X, v_i = 1; v_i = C'_j \rangle$, where $X \in \{C_j, C'_j\}$. These operators allow v_i to move to C_j or C'_j if v_{i-1} has done so.
- (3) For every clause C_j , two operators $\langle v_{i-1} = X, v_i = 0; v_i = C_j \rangle$ if \bar{v}_i occurs in clause C_j , and two operators $\langle v_{i-1} = X, v_i = 1; v_i = C'_j \rangle$ if v_i occurs in clause C_j , where $X \in \{0, 1\}$. These operators allow v_i to move to C_j or C'_j even if v_{i-1} has not done so.
- (4) For every clause C_j , four operators $\langle v_{i-1} = X, v_i = C_j; v_i = 0 \rangle$ and $\langle v_{i-1} = X, v_i = C'_j; v_i = 1 \rangle$, where $X = \{0, 1\}$. These operators allow v_i to move back to 0 or 1.

The domain transition graph for state variable w is shown in Figure 4. For every clause C_j the only two operators acting on w are $\langle v_k = X, w = j-1; w = j \rangle$, where $X \in \{C_j, C'_j\}$. (If $j = 1$, the pre-condition $w = j - 1$ must be replaced by $w = S$.)

Proposition 4 *A CNF formula F is satisfiable if and only if the planning instance $P(F)$ is solvable.*

Proof The proof follows from a relatively straightforward interpretation of the variables and values of the planning instance $P(F)$. For every state variable v_i , we must initially choose either 0 or 1. We can move a state variable v_i to C_j or C'_j if v_{i-1} has done so. However, state variable v_1 cannot move freely to C_j or C'_j , since variable v_1 has no operators of type (2). Hence, for every clause C_j , we are forced to use an operator of type (3) to move some state variable v_i to C_j or C'_j ; once this has been done, the remaining variables v_{i+1}, \dots, v_k can follow by using operators of type (2), and we can make w advance one step towards its goal. Note that the existence of an operator of type (3) acting on v_i implies that the initial choice of 0 or 1 for state variable v_i , when applied to the formula variable v_i , makes the clause C_j become true in the formula F .

Theorem 5 *PLAN-EXISTENCE- C_n is NP-hard.*

Proof Producing a planning instance $P(F)$ from a CNF formula F can be easily done in polynomial time, so we have a polynomial time reduction $CNF-SAT \leq_p PLAN-EXISTENCE-C_n$.

Polytree Causal Graphs

In this section, we study the class of planning problems with binary state variables and polytree causal graphs (Brafman

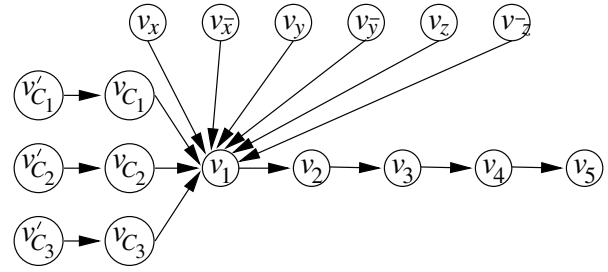


Figure 5: Causal graph of P_F when $F = C_1 \wedge C_2 \wedge C_3$ on three variables x, y, z .

& Domshlak 2003). We prove NP-hardness by demonstrating a reduction from 3SAT to this class of planning problems. As an example of the reduction, Figure 5 shows the causal graph of the planning problem P_F that corresponds to a formula F with three variables and three clauses (the precise definition of P_F is given in Proposition 7).

Let us describe briefly the idea behind the reduction. The planning problem P_F has two different parts. The first part (state variables $v_x, v_{\bar{x}}, \dots, v_{C_1}, v'_{C_1}, \dots$, and v_1) depends on the formula F and has the property that a plan may change the value of v_1 from 0 to 1 as many times as the number of clauses of F that a truth assignment can satisfy. However, this condition on v_1 cannot be stated as a planning problem goal. We overcome this difficulty by introducing a second part (state variables v_1, v_2, \dots, v_t) that translates it to a regular planning problem goal.

We first describe the second part. Let P be the planning problem $\langle V, init, goal, A \rangle$ where V is the set of state variables $\{v_1, \dots, v_{2k-1}\}$ and A is the set of $4k - 2$ operators $\{\alpha_1, \dots, \alpha_{2k-1}, \beta_1, \dots, \beta_{2k-1}\}$. For $i = 1$, the operators are defined as $\alpha_1 = \langle v_1 = 1; v_1 = 0 \rangle$ and $\beta_1 = \langle v_1 = 0; v_1 = 1 \rangle$. For $i > 1$, the operators are $\alpha_i = \langle v_{i-1} = 0, v_i = 1; v_i = 0 \rangle$ and $\beta_i = \langle v_{i-1} = 1, v_i = 0; v_i = 1 \rangle$. The initial state is $init(v_i) = 0$ for all i , and the goal state is $goal(v_i) = 0$ if i is even and $goal(v_i) = 1$ if odd.

Lemma 6 *Any valid plan for planning problem P changes state variable v_i from 0 to 1 at least k times. There is a valid plan that achieves this minimum.*

Proof Let A_i and B_i be, respectively, the sequences of operators $\langle \alpha_1, \dots, \alpha_i \rangle$ and $\langle \beta_1, \dots, \beta_i \rangle$. It is easy to verify that the plan $\langle B_{2k-1}, A_{2k-2}, B_{2k-3}, \dots, B_3, A_2, B_1 \rangle$ is valid: after finishing a sequence of operators A_i or B_i , the variable v_i is in its goal state (0 if i is even, 1 if i is odd). Subsequent operators in the plan do not modify v_i , so the variable remains in its goal state until the end. The operator β_1 appears k times in the plan (one for each sequence of type B_i), thus the value of v_1 changes k times from 0 to 1.

We proceed to show that k is the minimum. Consider a valid plan π , and let λ_i be the number of times that operators α_i and β_i appear in π (in other words, λ_i is the number of times that the value of v_i changes, either from 0 to 1 or from 1 to 0). Note that the number of occurrences of β_i has

to be equal to or precisely one more than the number of occurrences of α_i . We will show that $\lambda_{i-1} > \lambda_i$. Since λ_{2k-1} has to be at least one, $\lambda_{i-1} > \lambda_i$ implies that $\lambda_1 \geq 2k - 1$. In consequence, there are at least k operators β_i in plan π , finishing the proof.

We show that $\lambda_{i-1} > \lambda_i$ for valid plans. To begin with, let π be any plan (not necessarily a valid one) and consider only the subsequence consisting of operators α_i and β_i in π . It starts with β_i (since the initial state is $v_i = 0$), and the same operator cannot appear twice consecutively in the sequence. Thus the sequence alternates between β_i and α_i . Moreover, since β_i (for $i > 1$) has $v_{i-1} = 1$ as a pre-condition, and α_i has $v_{i-1} = 0$, there must be at least one operator α_{i-1} in plan π between the two operators β_i and α_i . For the same reason we must have at least one operator β_{i-1} between the two operators α_i and β_i , and one operator β_{i-1} before the first operator β_i . This shows that, in any plan π , not necessarily valid, we have $\lambda_{i-1} \geq \lambda_i$. If, in addition, π is valid, we require an extra operator: when v_i changes its value for the last time and attains its goal state, we have that $v_{i-1} = v_i$, so v_{i-1} is not in its goal state by parity. Hence a valid plan must have an extra operator α_{i-1} or β_{i-1} after all occurrences of α_i and β_i . Thus $\lambda_{i-1} > \lambda_i$ for valid plans.

Proposition 7 *3SAT reduces to plan existence for planning problems with binary variables and polytree causal graphs.*

Proof Let F be a CNF formula with k clauses and n variables. We produce a planning problem P_F with $2n + 4k - 1$ state variables and $2n + 14k - 3$ operators. The planning problem has two state variables v_x and $v_{\bar{x}}$ for every variable x in F , two state variables v_C and v'_C for every clause C in F , and $2k - 1$ additional variables v_1, \dots, v_{2k-1} . All variables are 0 in the initial state. The (partial) goal state is defined by $V_{goal} = \{v_1, \dots, v_{2k-1}\}$, $goal(v_i) = 0$ when i is even, and $goal(v_i) = 1$ when i is odd, like in problem P of Lemma 6. The operators are:

- (1) Operators $\langle v_x = 0; v_x = 1 \rangle$ and $\langle v_{\bar{x}} = 0; v_{\bar{x}} = 1 \rangle$ for every variable x of F .
- (2) Operators $\langle v'_C = 0; v'_C = 1 \rangle$, $\langle v'_C = 0, v_C = 0; v_C = 1 \rangle$ and $\langle v'_C = 1, v_C = 1; v_C = 0 \rangle$ for every clause C of F .
- (3) Seven operators for every clause C , one for each partial assignment that satisfies C . Without loss of generality, let x, y , and z be the three variables that appear in clause C . Then for each operator a among these seven, $V_{pre(a)} = \{v_x, v_{\bar{x}}, v_y, v_{\bar{y}}, v_z, v_{\bar{z}}, v_C, v_1\}$, $V_{post(a)} = \{v_1\}$, $pre(a)(v_C) = 1$, $pre(a)(v_1) = 0$, and $post(a)(v_1) = 1$. The pre-condition on state variables $v_x, v_{\bar{x}}, v_y, v_{\bar{y}}, v_z, v_{\bar{z}}$ depends on the corresponding satisfying partial assignment. For example, the operator corresponding to the partial assignment $\{x = 0, y = 0, z = 1\}$ of the clause $C = x \vee \bar{y} \vee \bar{z}$ has the pre-condition $\langle v_x = 0, v_{\bar{x}} = 1, v_y = 0, v_{\bar{y}} = 1, v_z = 1, v_{\bar{z}} = 0 \rangle$.
- (4) An operator $\langle \bigvee C, v_C = 0, v_1 = 1; v_1 = 0 \rangle$.
- (5) Operators $\alpha_i = \langle v_{i-1} = 0, v_i = 1; v_i = 0 \rangle$ and $\beta_i = \langle v_{i-1} = 1, v_i = 0; v_i = 1 \rangle$ for $2 \leq i \leq 2k - 1$ (the same operators as in problem P except for α_1 and β_1).

We note some simple facts about problem P_F . For any variable x , state variables v_x and $v_{\bar{x}}$ in P_F start at 0, and by

applying the operators in (1) they can change into 1 but not back to 0. In particular, a plan π cannot reach both of the partial states $\langle v_x = 1, v_{\bar{x}} = 0 \rangle$ and $\langle v_x = 0, v_{\bar{x}} = 1 \rangle$ during the course of its execution.

Similarly, if C is a clause of F , state variable v_C can change from 0 to 1 and, by first changing v'_C into 1, v_C can change back to 0. No further changes are possible, since no operator brings back v'_C to 0.

Now we interpret operators in (3) and (4), which are the only operators that affect v_1 . To change v_1 from 0 to 1 we need to apply one of the operators in (3), thus we require $v_C = 1$ for a clause C . But the only way to bring back v_1 to 0 is applying the operator in (4) which has as pre-condition that $v_C = 0$. We deduce that every time that v_1 changes its value from 0 to 1 and then back to 0 in plan π , at least one of the k state variables v_C is *used up*, in the sense that v_C has been brought from 0 to 1 and then back to 0, and cannot be used again for the same purpose.

We show that F is in 3-SAT if and only if there is a valid plan for problem P_F . Let σ be a truth assignment that satisfies F . By Lemma 6 we can extend a plan π' that switches variable v_1 from 0 to 1 at least k times to a plan π that sets all variables v_i to their goal values. Plan π' starts by setting the value of all state variables v_x and $v_{\bar{x}}$ to the corresponding value of the truth assignment σ using the operators in (1). Then, for each of the k state variables v_C , we set $v_C = 1$, we apply the operator of (3) that corresponds to σ restricted to the variables of clause C , and we change v_C back to 0 so that we can apply the operator in (4). By repeating this process for every clause C of F we are switching the state variable v_1 exactly k times from 0 to 1.

We show the converse, namely, that the existence of a valid plan π in P_F implies that F is satisfiable. By Lemma 6 the state variable v_1 has to change from 0 to 1 at least k times. This implies that k operators of (3), all of them corresponding to different clauses, have been used to move v_1 from 0 to 1. Hence we can define a satisfying assignment σ by setting $\sigma(x) = 1$ if the partial states $\{v_x = 1, v_{\bar{x}} = 0\}$ appears during the execution of π , and $\sigma(x) = 0$ otherwise.

Theorem 8 *Plan existence for planning problems with a polytree causal graph is NP-complete.*

Proof Due to Proposition 7 we only need to show that the problem is in NP. But Brafman and Domshlak (2003) showed that this holds in the more general setting of planning problems with causal graphs where each component is singly connected. Their proof uses the non-trivial result that solvable planning problems on binary variables with a singly connected causal graph have plans of polynomial length (the same is not true for non-binary variables, or unrestricted causal graphs).

Conclusion

We have presented three new complexity results for planning problems with simple causal graphs. First, we provided a polynomial time algorithm that uses macros to generate solution plans for the class 3S. Jonsson and Bäckström (1998) showed that plan existence for 3S can be solved in polynomial time, while plan generation is intractable in the sense

that solution plans may have exponential length. Our work casts new light on this result: even when solution plans have exponential length, it is possible to generate a representation of the solution in polynomial time. Thus, it appears as if for the class 3S, plan generation is not inherently harder than plan existence. We are not aware of any other work that determines the relative complexity of plan existence and plan generation, so the question of whether plan generation is harder than plan existence remains unanswered.

Jonsson and Bäckström (1998) also showed that the bounded plan existence problem (does there exist a solution plan of length at most k ?) is NP-hard for the class 3S. Consequently, optimal plan generation for 3S is NP-hard as well; otherwise, bounded plan existence could be solved by generating an optimal solution plan and checking whether or not it is longer than k . As we have demonstrated, even though our algorithm generates a solution plan in polynomial time, it does not generate an optimal plan in general.

If all you want is execute a solution plan once, our algorithm does not offer a great advantage over the incremental algorithm of Jonsson and Bäckström (1998). Neither algorithm is guaranteed to produce a polynomial length solution if one exists, and may produce an exponential length solution instead. However, we believe that there are several benefits of compiling a solution in the form of macros. First, the solution plan can be generated and validated in polynomial time, and the plan can be stored and reused using polynomial memory. Also, the plan length can be calculated in polynomial time using dynamic programming. Finally, during plan execution, no additional work is required except retrieving the next operator to execute from memory, whereas the incremental approach performs a constant number of operations at each step.

Since they are relatively simple, the class \mathbb{C}_n and the class of planning problems with binary state variables and polytree causal graphs could be seen as promising candidates for proving the relative complexity of plan existence and plan generation. However, we have shown that plan existence for \mathbb{C}_n is NP-hard, and that plan existence for planning problems with polytree causal graphs is NP-complete. Consequently, these classes cannot be used to show that plan generation is harder than plan existence, since plan existence is already difficult. Our work also closes the complexity gaps that appear in the literature regarding these two classes.

It is however possible that there exist subsets of planning problems in these classes for which plan existence can be solved in polynomial time. In fact, for polytree causal graphs we know that this is the case, since Brafman and Domshlak (2003) presented a polynomial time algorithm for solving planning problems in this class, provided that the indegree of nodes in the causal graph is bounded. Our reduction from 3SAT has a node in the causal graph, v_1 , with unbounded indegree, which causes the problem of plan existence to be NP-complete. In the case of \mathbb{C}_n , the state variables in our reduction has domains whose size depends on the number of clauses of the corresponding CNF formula. It would be interesting to investigate whether the problem of plan existence for the class \mathbb{C}_n is easier if the size of the state variable domains is bounded by a constant.

Acknowledgements

This work was partially funded by MEC grant TIN2006-15387-C03-03 and grant TIN2004-07925-C03-01 (GRAMMARS).

References

- Bäckström, C., and Nebel, B. 1995. Complexity Results for SAS⁺ Planning. *Computational Intelligence* 11(4):625–655.
- Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research* 24:581–621.
- Brafman, R., and Domshlak, C. 2003. Structure and Complexity in Planning with Unary Operators. *Journal of Artificial Intelligence Research* 18:315–349.
- Brafman, R., and Domshlak, C. 2006. Factored Planning: How, When, and When Not. In *Proceedings of the 21st National Conference on Artificial Intelligence*.
- Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:165–204.
- Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.
- Domshlak, C., and Dinitz, Y. 2001. Multi-Agent Off-line Coordination: Structure and Complexity. In *Proceedings of the 6th European Conference on Planning*, 277–288.
- Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5(2):189–208.
- Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.
- Jonsson, P., and Bäckström, C. 1998. Tractable plan existence does not imply tractable plan generation. *Annals of Mathematics and Artificial Intelligence* 22(3-4):281–296.
- Jonsson, A. 2007. The Role of Macros in Tractable Planning Over Causal Graphs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1936–1941.
- Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.
- Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.
- Minton, S. 1985. Selectively generalizing plans for problem-solving. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 596–599.
- Vidal, V. 2004. A Lookahead Strategy for Heuristic Search Planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, 150–159.
- Williams, B., and Nayak, P. 1997. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 1178–1185.