

The Role of Macros in Tractable Planning over Causal Graphs

Anders Jonsson

Departament de Tecnologia
Universitat Pompeu Fabra
Passeig de Circumval·lació, 8
08003 Barcelona, Spain
anders.jonsson@upf.edu

Abstract

The complexity of existing planners is bounded by the length of the resulting plan, a fact that limits planning to domains with relatively short solutions. We present a novel planning algorithm that uses the causal graph of a domain to decompose it into subproblems and stores subproblem plans in memory as macros. In many domains, the resulting plan can be expressed using relatively few macros, making it possible to generate exponential length plans in polynomial time. We show that our algorithm is complete, and that there exist special cases for which it is optimal and polynomial. Experimental results demonstrate the potential of using macros to solve planning domains with long solution plans.

1 Introduction

Most successful planners exploit structure in a planning domain to generate valid plans. In particular, factored planners exploit independence within a domain to decompose the domain into several subproblems that can be solved independently and combined to produce a global plan. Hierarchical planners factor domains into a hierarchy of increasingly abstract subproblems [8; 9]. Other factored planners solve each subproblem separately and coordinate the solutions to produce a coherent global plan [1; 4; 6; 10].

The causal graph underlying a planning domain appears to capture relevant information about domain structure. Jonsson and Bäckström [7] proved that plan existence is polynomial in a class of domains with acyclic causal graphs, although the plans may have exponential length. Brafman and Domshlak [3] designed a polynomial-time algorithm for plan generation in domains whose causal graphs are polytrees with bounded indegree. Helmert [5] used causal graph analysis to design an efficient heuristic for planning.

We present a novel algorithm for plan generation in a subclass of domains with unary operators and acyclic causal graphs. Just like other factored planners, the algorithm decomposes a planning domain into several subproblems. However, unlike other planners, partial plans generated in a subproblem are stored in memory as macros. In subsequent subproblems, the algorithm retrieves macros from memory and uses them as part of the solution. We show that the algorithm

is complete, i.e., it always generates a plan if one exists. In addition, the algorithm is optimal for a subclass of domains.

Our algorithm takes a dynamic programming approach to planning: instead of solving the same subproblem repeatedly, solve the subproblem once and cache the resulting partial plan in memory. Whenever the solution is needed, retrieve the partial plan from memory. In many domains, this approach dramatically reduces the complexity of generating valid plans. In addition, macros generated in one domain can be stored and reused in similar domains. If approximation techniques are used that require backtracking, macros can retain the solution to some subproblems when a deadend is reached.

Helmert [5] showed that the structure of a planning domain becomes much more explicit when the domain is represented using multi-valued variables. Since we share this view and want to exploit structure, we use the SAS⁺ formalism [2] to represent planning domains using multi-valued variables. We show that for a subset of domains in our class, our algorithm runs in polynomial time. Experimental results demonstrate the utility of our algorithm and verify theoretical results.

Brafman and Domshlak [4] suggested that factored planning is likely to work well in domains with limited local depth whose causal graph has limited tree-width. Informally, the local depth is the minmax number of times any state variable has to change its value on a valid plan. Our algorithm provably solves Tower of Hanoi in polynomial time, a domain whose local depth is exponential in the number of state variables and whose causal graph has maximal tree-width. The algorithm computes the transitive reduction of the causal graph, which often has much smaller tree-width. In addition, exponential length plans can be expressed using relatively few macros, drastically reducing the local depth.

2 Notation

We define a SAS⁺ planning domain instance as a tuple $\mathcal{P} = \langle \mathbf{V}, \mathbf{s}_0, \mathbf{c}_*, A \rangle$, where:

- $\mathbf{V} = \{v_1, \dots, v_n\}$ is a set of state variables, each with finite domain $\mathcal{D}(v_i)$. Let $\mathcal{D}_{\mathbf{C}} = \times_{v_i \in \mathbf{C}} \mathcal{D}(v_i)$ be the joint domain of a subset $\mathbf{C} \subseteq \mathbf{V}$ of variables. A context $\mathbf{c} \in \mathcal{D}_{\mathbf{C}}$ is an assignment of values to variables in \mathbf{C} ; \mathbf{c} assigns the value $\mathbf{c}[v_i] \in \mathcal{D}(v_i)$ to each $v_i \in \mathbf{C}$. A capitalized context denotes its associated subset of variables. A state $\mathbf{s} \in \mathcal{D}_{\mathbf{V}}$ assigns a value to each variable in \mathbf{V} .

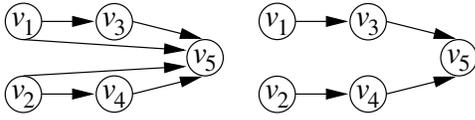


Figure 1: An acyclic causal graph and its transitive reduction

- s_0 is an initial state.
- c_* is a goal context.
- $A = \{a_1, \dots, a_m\}$ is a set of operators of the form $a_k = \langle \mathbf{pre}_k, \mathbf{post}_k, \mathbf{prv}_k \rangle$, where the contexts \mathbf{pre}_k , \mathbf{post}_k , and \mathbf{prv}_k denote the pre-, post- and prevail-condition of operator a_k , respectively. For each operator $a_k \in A$, $\mathbf{Pre}_k = \mathbf{Post}_k$ and $\mathbf{Pre}_k \cap \mathbf{Prv}_k = \emptyset$.

We define two operations on contexts. Let $f_{\mathbf{W}}(c)$ be the projection of context c onto the subset $\mathbf{W} \subseteq \mathbf{V}$ of state variables. The result of $f_{\mathbf{W}}(c)$ is a context \mathbf{x} such that $\mathbf{X} = \mathbf{C} \cap \mathbf{W}$ and $\mathbf{x}[v_i] = c[v_i]$ for each $v_i \in \mathbf{X}$. Also, let $c \oplus w$ be the composition of contexts c and w . The result of $c \oplus w$ is a context \mathbf{x} such that $\mathbf{X} = \mathbf{C} \cup \mathbf{W}$, $\mathbf{x}[v_i] = w[v_i]$ for each $v_i \in \mathbf{W}$ and $\mathbf{x}[v_i] = c[v_i]$ for each $v_i \in \mathbf{C} - \mathbf{W}$. Note the asymmetry; the right operand overrides the values of the left operand.

An operator $a_k \in A$ is applicable in state s if $f_{\mathbf{Pre}_k}(s) = \mathbf{pre}_k$ and $f_{\mathbf{Prv}_k}(s) = \mathbf{prv}_k$. The result of successfully applying a_k in state s is $s \oplus \mathbf{post}_k$. We add a dummy operator a_* to A whose prevail-condition equals the goal context c_* .

In this paper, we study the restricted class of SAS⁺ domains that satisfy the following two definitions:

Definition 2.1 A planning domain \mathcal{P} has unary operators if, for each operator $a_k \in A$, $|\mathbf{Pre}_k| = 1$.

For a planning domain with unary operators, we can form the set $A_i = \{a_k \in A \mid \mathbf{Pre}_k = \{v_i\}\}$ of operators that affect state variable $v_i \in \mathbf{V}$. The causal graph of a planning domain with unary operators is a graph with one node per state variable. There is a directed edge between state variables v_j and v_i if there is an operator $a_k \in A_i$ such that $v_j \in \mathbf{Prv}_k$.

Definition 2.2 A planning domain \mathcal{P} is tree-reducible if its causal graph is acyclic and the transitive reduction of the causal graph is a tree, i.e., weakly connected with no undirected cycles and with all edges pointing towards the root.

The transitive reduction of a graph only retains edges necessary to maintain connectivity, and is unique for acyclic graphs. Figure 1 illustrates the causal graph and its transitive reduction for a planning problem in our restricted class.

3 A planning algorithm

We present an algorithm that uses macros to solve planning domains in our restricted class. The algorithm divides a planning domain into n subproblems \mathcal{P}_i , $i = 1, \dots, n$. Let \mathbf{C}_i denote the set of state variables of subproblem \mathcal{P}_i . \mathbf{C}_i contains v_i as well as all ancestors of v_i in the causal graph. Without loss of generality, assume that v_1, \dots, v_n is a topological sort of the nodes. Since the transitive reduction graph is a tree,

v_n is a descendant of each other node, so $\mathbf{C}_n = \mathbf{V}$, and \mathcal{P}_n corresponds to the original domain \mathcal{P} . Our algorithm solves the subproblems bottom-up starting with \mathcal{P}_1 .

Let M_i be the set of macros that our algorithm generates in subproblem \mathcal{P}_i , and let $\mathbf{Pa}_i \subset \mathbf{V}$ be the set of parent nodes of v_i in the transitive reduction of the causal graph. The set of operators of subproblem \mathcal{P}_i is $O_i = A_i \cup (\cup_{v_j \in \mathbf{Pa}_i} M_j)$, i.e., operators that affect v_i and macros generated in parent subproblem \mathcal{P}_j for each parent node $v_j \in \mathbf{Pa}_i$.

Definition 3.1 In subproblem \mathcal{P}_i , a macro is a tuple $m_i = \langle c_i^x, op, c_i^y \rangle$, composed of an initial context c_i^x , an operator sequence $op \in O_i^*$, and a goal context c_i^y .

We use superscript to distinguish between contexts that assign values to the same subset of state variables, in this case \mathbf{C}_i . A macro is associated with a specific initial context even if its operator sequence could be applied in other contexts.

The idea of introducing macros in subproblem \mathcal{P}_i is to capture all relevant subplans involving state variable v_i and its ancestors. A subplan is relevant precisely when it satisfies the prevail-condition of an operator that affects a descendant of v_i . To determine relevant subplans, we project the prevail-conditions of such operators onto \mathbf{C}_i . Let Z_i be the set of contexts \mathbf{z} , $|Z_i| \geq 1$, that equal the projected prevail-condition $f_{\mathbf{C}_i}(\mathbf{prv}_k)$ of some operator $a_k \in A_j$, $j > i$.

Our algorithm constructs the domain graph of \mathbf{C}_i in which nodes are contexts in $\mathcal{D}_{\mathbf{C}_i}$. Let the initial context be $f_{\mathbf{C}_i}(s_0)$, the initial state projected onto \mathbf{C}_i . When we visit a context c_i , we add edges to the graph for each operator $a_k \in A_i$ such that $\mathbf{pre}_k = c_i[v_i]$. For each parent $v_j \in \mathbf{Pa}_i$, we determine the set of macros $m_j = \langle f_{\mathbf{C}_j}(c_i), op, c_j \rangle \in M_j$ such that $f_{\mathbf{Prv}_k}(c_j) = f_{\mathbf{C}_j}(\mathbf{prv}_k)$. Each such macro starts in c_i and ends in a context that satisfies the prevail-condition of a_k .

Since the transitive reduction of the causal graph is a tree, the parent nodes of v_i have no common ancestors. Consequently, applying a macro generated in parent subproblem \mathcal{P}_j has no impact on the values of state variables in other parent subproblems. To achieve the prevail-condition of a_k , we can apply the macros of parent subproblems in any order. For each combination of parent macros, we add an edge from c_i to c_i^x in the domain graph, where c_i^x is the context that results from applying each of the parent macros followed by a_k . The weight of the edge equals the total length of the operator sequences of the macros plus one, the cost of applying a_k .

We use Dijkstra's algorithm to traverse the domain graph in order of shortest paths. We only add nodes and edges to the graph as necessary, never representing the complete graph. Consequently, the algorithm only visits contexts that are reachable from $f_{\mathbf{C}_i}(s_0)$. For each context, we record the shortest operator sequence for reaching it from $f_{\mathbf{C}_i}(s_0)$. Dijkstra ensures that the algorithm always finds the shortest operator sequence in O_i^* for reaching a context.

At a context c_i , we use the projected prevail-conditions to determine macros. We find each projected prevail-condition $\mathbf{z} \in Z_i$ such that either $v_i \notin \mathbf{Z}$ or $\mathbf{z}[v_i] = c_i[v_i]$. For each parent $v_j \in \mathbf{Pa}_i$, we find each macro $m_j = \langle f_{\mathbf{C}_j}(c_i), op, c_j \rangle \in M_j$ such that $f_{\mathbf{Z}}(c_j) = f_{\mathbf{C}_j}(\mathbf{z})$. For each combination of parent macros, let c_i^x be the context that results from applying the parent macros starting in c_i , and let $op_m \in O_i^*$ be

Algorithm 1: MACROPLANNER(\mathcal{P})

```

1 for each  $i = 1, \dots, n$ 
2    $\mathbf{C}_i \leftarrow$  set of  $v_i$  and its ancestors
3    $M_i \leftarrow \emptyset$ 
4    $L \leftarrow$  list containing projected initial state  $f_{\mathbf{C}_i}(s_0)$ 
5   while  $L$  is non-empty
6      $\mathbf{c}_i \leftarrow$  remove first context in  $L$ 
7     construct the domain graph of  $\mathbf{C}_i$  starting at  $\mathbf{c}_i$ 
8      $H \leftarrow$  hash table mapping contexts to operator sequences
9     run Dijkstra and record macros from  $\mathbf{c}_i$  in  $H$ 
10    for each  $\langle \mathbf{c}_i^x, op \rangle \in H$ 
11      append the macro  $\langle \mathbf{c}_i, op, \mathbf{c}_i^x \rangle$  to  $M_i$ 
12      if  $\mathbf{c}_i^x$  not previously visited
13        append  $\mathbf{c}_i^x$  to  $L$ 
14    if no macro generated to projected goal context
15      stop; there is no valid plan

```

the corresponding sequence of macros. Let $op_c \in O_i^*$ be the shortest operator sequence for reaching \mathbf{c}_i from $f_{\mathbf{C}_i}(s_0)$, and let $\langle op_c, op_m \rangle$ denote op_c followed by op_m . If $\langle op_c, op_m \rangle$ is the shortest operator sequence for reaching \mathbf{c}_i^x so far, generate a macro $\langle f_{\mathbf{C}_i}(s_0), \langle op_c, op_m \rangle, \mathbf{c}_i^x \rangle$ in M_i , replacing any previous macro to \mathbf{c}_i^x . The algorithm may construct a macro from a context to itself with operator sequence length 0.

For each context \mathbf{c}_i such that \mathbf{c}_i is the goal context of any macro generated by our algorithm, we repeat Dijkstra with \mathbf{c}_i as the initial context. If it is not possible to reach the projected goal context $f_{\mathbf{C}_i}(\mathbf{c}_*)$ from \mathbf{c}_i , we can remove all macros whose goal context equals \mathbf{c}_i . If it is not possible to reach the projected goal context from the projected initial state, there exists no valid global plan.

Algorithm 1 summarizes our planner for domains in the restricted class. To ensure constant-time lookup, we use hash tables to store operators, projected prevail-conditions and macros. In subproblem \mathcal{P}_n , $\mathbf{C}_n = \mathbf{V}$, so the projected initial state $f_{\mathbf{V}}(s_0) = s_0$ corresponds to the initial state itself. The only projected prevail-condition is that of the dummy operator a_* , which equals $f_{\mathbf{V}}(\mathbf{c}_*) = \mathbf{c}_*$. Any macro in M_n starts in s_0 and ends in a state s such that $f_{\mathbf{C}_*}(s) = \mathbf{c}_*$. Thus, the macro in M_n with shortest operator sequence corresponds to the optimal global plan.

3.1 Example

We illustrate our algorithm using an example domain. Let $\mathbf{V} = \{v_1, \dots, v_5\}$ with $\mathcal{D}(v_i) = \{0, 1\}$ for each $v_i \in \mathbf{V}$. Let the initial state be $s_0 = (0, 0, 0, 0, 0)$, and let the goal context be $\mathbf{c}_* = (v_3 = 0, v_4 = 0, v_5 = 1)$. Assume that the causal graph of the domain is the one illustrated in Figure 1, and that there are two operators in A_5 :

$$\begin{aligned}
a_5^1 &= \langle v_5 = 0, v_5 = 1, (v_3 = 1, v_4 = 1) \rangle, \\
a_5^2 &= \langle v_5 = 1, v_5 = 0, (v_1 = 0, v_2 = 0) \rangle.
\end{aligned}$$

Assume that M_3 and M_4 contain the following macros:

$$\begin{aligned}
m_3^1 &= \langle (v_1 = 0, v_3 = 0), op^1, (v_1 = 0, v_3 = 1) \rangle, \\
m_3^2 &= \langle (v_1 = 0, v_3 = 0), op^2, (v_1 = 1, v_3 = 1) \rangle,
\end{aligned}$$

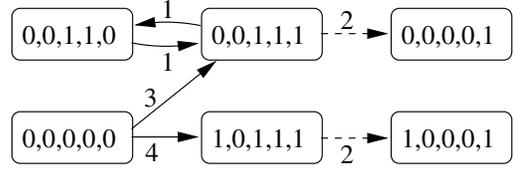


Figure 2: The domain graph for subproblem \mathcal{P}_5

$$\begin{aligned}
m_3^3 &= \langle (v_1 = 0, v_3 = 1), op^3, (v_1 = 0, v_3 = 0) \rangle, \\
m_3^4 &= \langle (v_1 = 1, v_3 = 1), op^4, (v_1 = 1, v_3 = 0) \rangle, \\
m_4^1 &= \langle (v_2 = 0, v_4 = 0), op^5, (v_2 = 0, v_4 = 1) \rangle, \\
m_4^2 &= \langle (v_2 = 0, v_4 = 1), op^6, (v_2 = 0, v_4 = 0) \rangle,
\end{aligned}$$

where $len(op^i) = 1$ for $i = 1, 3, \dots, 6$ and $len(op^2) = 2$.

Figure 2 illustrates the domain graph for subproblem \mathcal{P}_5 that our algorithm constructs. The algorithm starts in the projected initial state $(0, 0, 0, 0, 0)$. There is one operator in A_5 whose pre-condition equals $v_5 = 0$, namely a_5^1 . Two macros in M_3 , m_3^1 and m_3^2 , satisfy the prevail-condition $v_3 = 1$ of a_5^1 . One macro in M_4 , m_4^1 , satisfies the prevail-condition $v_4 = 1$ of a_5^1 . That gives us two possible combinations of parent macros: $\langle m_3^1, m_4^1 \rangle$ and $\langle m_3^2, m_4^1 \rangle$. Applying $\langle m_3^1, m_4^1 \rangle$ followed by a_5^1 has total length 3 and results in the state $(0, 0, 1, 1, 1)$. Applying $\langle m_3^2, m_4^1 \rangle$ followed by a_5^1 has total length 4 and results in the state $(1, 0, 1, 1, 1)$. The algorithm adds the corresponding edges and nodes to the domain graph.

The only projected prevail-condition in Z_5 is that of the dummy operator a_* , which equals the goal context \mathbf{c}_* . At $(0, 0, 1, 1, 1)$, there is one macro in M_3 , m_3^3 , that satisfies $v_3 = 0$, and one macro in M_4 , m_4^2 , that satisfies $v_4 = 0$. The only possible combination of parent macros is $\langle m_3^3, m_4^2 \rangle$ with total length 2 and resulting in state $(0, 0, 0, 0, 1)$. Consequently, the algorithm adds a macro from $(0, 0, 0, 0, 0)$ to $(0, 0, 0, 0, 1)$ with operator sequence length 5. At $(1, 0, 1, 1, 1)$, the only combination of parent macros that satisfy \mathbf{c}_* is $\langle m_3^4, m_4^2 \rangle$, resulting in a macro from $(0, 0, 0, 0, 0)$ to $(1, 0, 0, 0, 1)$ with length 6. At $(0, 0, 1, 1, 1)$, it is also possible to apply operator a_5^2 , resulting in the state $(0, 0, 1, 1, 0)$, but that does not lead to a state satisfying a projected prevail-condition. Note how the algorithm never represents most of the $2^5 = 32$ possible states.

4 Completeness and optimality

In this section, we prove that our algorithm is complete, i.e., it always generates a valid plan if one exists. In addition, our algorithm is optimal, i.e., it always generates the shortest possible valid plan.

Definition 4.1 A context \mathbf{c}_i matches a projected prevail-condition $\mathbf{z} \in Z_i$ if $f_{\mathbf{Z}}(\mathbf{c}_i) = \mathbf{z}$.

Definition 4.2 Let $\mathbf{c}_i^x \rightarrow_i \mathbf{c}_i^y$ denote that there exists an operator sequence in O_i^* from a context \mathbf{c}_i^x to a context \mathbf{c}_i^y .

Let G_i , $i = 1, \dots, n$, be the subset of contexts that appear either as the initial context or the goal context of at least one macro in M_i . In other words, for each context $\mathbf{c}_i^x \in G_i$, there

exist $op \in O_i^*$ and $\mathbf{c}_i^y \in \mathcal{D}_{C_i}$ such that $\langle \mathbf{c}_i^x, op, \mathbf{c}_i^y \rangle \in M_i$ or $\langle \mathbf{c}_i^y, op, \mathbf{c}_i^x \rangle \in M_i$.

First, we prove that the macros generated by our algorithm correspond to the shortest operator sequences for reaching a context that matches a projected prevail-condition.

Lemma 4.3 *For each context $\mathbf{c}_i^x \in G_i$ and each context \mathbf{c}_i^y , if $\mathbf{c}_i^x \rightarrow_i \mathbf{c}_i^y$ and \mathbf{c}_i^y matches a projected prevail-condition $\mathbf{z} \in Z_i$, our algorithm generates a macro $\langle \mathbf{c}_i^x, op, \mathbf{c}_i^y \rangle \in M_i$ such that \mathbf{c}_i^y matches \mathbf{z} and such that $op \in O_i^*$ is part of a shortest operator sequence from \mathbf{c}_i^x to \mathbf{c}_i^y .*

Proof By induction on i . For $i = 1$, the set of applicable operators is $O_1 = A_1$. Since $|Z| \geq 1$ for each $\mathbf{z} \in Z_1$, a context \mathbf{c}_1^y matches \mathbf{z} if and only if $\mathbf{c}_1^y = \mathbf{z}$. Dijkstra is guaranteed to find a shortest path to any context that is reachable from \mathbf{c}_1^x , corresponding to an operator sequence in A_1^* . Hence, if $\mathbf{c}_1^x \rightarrow_1 \mathbf{c}_1^y$ and \mathbf{c}_1^y matches \mathbf{z} , our algorithm generates a macro $\langle \mathbf{c}_1^x, op, \mathbf{c}_1^y \rangle \in M_1$ such that $op \in O_1^*$ is a shortest operator sequence from \mathbf{c}_1^x to \mathbf{c}_1^y .

For $i > 1$, if $|\mathbf{Pa}_i| = 0$, the proof is analogous to $i = 1$. For $|\mathbf{Pa}_i| > 0$, we first treat the case that $v_i \in \mathbf{Z}$ and $\mathbf{c}_i^x[v_i] = \mathbf{z}[v_i]$. If \mathbf{c}_i^x matches \mathbf{z} , the algorithm generates a macro $\langle \mathbf{c}_i^x, \emptyset, \mathbf{c}_i^x \rangle \in M_i$, and trivially \emptyset is part of a shortest operator sequence from \mathbf{c}_i^x to \mathbf{c}_i^y . Otherwise, for some parent nodes $v_j \in \mathbf{Pa}_i$, it must be that $f_{C_j}(\mathbf{z}) \in Z_j$. Since our algorithm only uses macros in M_j to change the values of state variables in C_j , it follows that $f_{C_j}(\mathbf{c}_i^x) \in G_j$ for each such $v_j \in \mathbf{Pa}_i$. We can remove any operators not affecting state variables in C_j from the operator sequence $\mathbf{c}_i^x \rightarrow_i \mathbf{c}_i^y$ to obtain $f_{C_j}(\mathbf{c}_i^x) \rightarrow_j f_{C_j}(\mathbf{c}_i^y)$. Since \mathbf{c}_i^y matches \mathbf{z} , it must be that $f_{C_j}(\mathbf{c}_i^y)$ matches $f_{C_j}(\mathbf{z})$.

By hypothesis of induction, for each such parent node $v_j \in \mathbf{Pa}_i$, our algorithm generates a macro $\langle f_{C_j}(\mathbf{c}_i^x), op_j, \mathbf{c}_j \rangle \in M_j$ such that \mathbf{c}_j matches $f_{C_j}(\mathbf{z})$ and op_j is part of a shortest operator sequence from $f_{C_j}(\mathbf{c}_i^x)$ to $f_{C_j}(\mathbf{c}_i^y)$. As a consequence, our algorithm generates a macro $\langle \mathbf{c}_i^x, op, \mathbf{c}_i^y \rangle \in M_i$, where \mathbf{c}_i^y is the context that results from applying all such parent macros in sequence. Since the goal context \mathbf{c}_j of each parent macro matches $f_{C_j}(\mathbf{z})$, it follows that \mathbf{c}_i^y matches \mathbf{z} . Since op_j is part of a shortest operator sequence from $f_{C_j}(\mathbf{c}_i^x)$ to $f_{C_j}(\mathbf{c}_i^y)$, it follows that op is part of a shortest operator sequence from \mathbf{c}_i^x to \mathbf{c}_i^y .

In case $v_i \in \mathbf{Z}$ and $\mathbf{c}_i^x[v_i] \neq \mathbf{z}[v_i]$, the operator sequence $\mathbf{c}_i^x \rightarrow_i \mathbf{c}_i^y$ must include operators in A_i to change the value of v_i from $\mathbf{c}_i^x[v_i]$ to $\mathbf{z}[v_i]$. Our algorithm adds edges from \mathbf{c}_i^x for each operator $a_k \in A_i$ such that $\mathbf{prv}_k = \mathbf{c}_i^x[v_i]$. For some parent nodes $v_j \in \mathbf{Pa}_i$, $f_{C_j}(\mathbf{prv}_k) \in Z_j$, and we know that $f_{C_j}(\mathbf{c}_i^x) \in G_j$. For each context \mathbf{c}_j such that $f_{C_j}(\mathbf{c}_i^x) \rightarrow_j \mathbf{c}_j$ and that matches $f_{C_j}(\mathbf{prv}_k)$, by induction our algorithm generates a macro $\langle f_{C_j}(\mathbf{c}_i^x), op_j, \mathbf{c}_j^w \rangle \in M_j$ such that \mathbf{c}_j^w matches $f_{C_j}(\mathbf{prv}_k)$ and op_j is part of a shortest operator sequence from $f_{C_j}(\mathbf{c}_i^x)$ to \mathbf{c}_j .

As long as the projection onto C_j remains within G_j for each parent node $v_j \in \mathbf{Pa}_i$, our algorithm is guaranteed to add edges for each applicable operator $a_k \in A_i$. If an operator sequence achieves the prevail-condition \mathbf{prv}_k of an operator $a_k \in A_i$ at a context $\mathbf{c}_j^w \notin G_j$, it follows by induction that our algorithm generates a macro to a context that matches \mathbf{prv}_k and that is on the shortest path to \mathbf{c}_j^w . Thus, it is never

possible to construct a shorter path from \mathbf{c}_i^x to \mathbf{c}_i^y by leaving G_j . In particular, whatever context we use to achieve the prevail-condition of the last operator in A_i , there is a context on the shortest path whose projection onto C_j is in G_j . As a result, our algorithm generates a path in the domain graph from \mathbf{c}_i^x to a context \mathbf{c}_i^w such that $\mathbf{c}_i^w[v_i] = \mathbf{z}[v_i]$ and \mathbf{c}_i^w is on the shortest path from \mathbf{c}_i^x to \mathbf{c}_i^y . Since $f_{C_j}(\mathbf{c}_i^w) \in G_j$ for each parent node $v_j \in \mathbf{Pa}_i$, we can use the reasoning from the case $\mathbf{c}_i^x[v_i] = \mathbf{z}[v_i]$ to prove the lemma.

In case $v_i \notin \mathbf{Z}$, it is not necessary to change the value of v_i to satisfy \mathbf{z} . This is analogous to the case $v_i \in \mathbf{Z}$, $\mathbf{c}_i^x[v_i] = \mathbf{z}[v_i]$. However, it is possible to construct tasks such that the optimal plan satisfies \mathbf{z} in a context that assigns a different value to v_i . For this reason, our algorithm constructs macros to contexts that match \mathbf{z} for each value of v_i . In this case, the proof is analogous to the case $v_i \in \mathbf{Z}$, $\mathbf{c}_i^x[v_i] \neq \mathbf{z}[v_i]$.

Theorem 4.4 (Completeness and optimality) *If there exists a valid global plan, our algorithm generates the optimal global plan.*

Proof The set of state variables of subproblem \mathcal{P}_n is $\mathbf{C}_n = \{v_1, \dots, v_n\} = \mathbf{V}$. Unless M_n is empty, at least one macro must start at the projected initial state $f_{\mathbf{V}}(s_0) = s_0$, in which case $s_0 \in G_n$. The only projected prevail-condition of subproblem \mathcal{P}_n is that of the dummy operator a_* , which equals $f_{\mathbf{V}}(\mathbf{c}_*) = \mathbf{c}_*$. Any valid global plan is a sequence of operators from s_0 to a state \mathbf{s} that matches \mathbf{c}_* , which means that $s_0 \rightarrow_n \mathbf{s}$. For each such state \mathbf{s} , it follows from Lemma 4.3 that our algorithm generates a macro $\langle s_0, op, \mathbf{s}^w \rangle \in M_n$ such that \mathbf{s}^w matches \mathbf{c}_* and op is part of a shortest operator sequence from s_0 to \mathbf{s} . One of these macros must correspond to the optimal global plan.

5 Complexity

In subproblem \mathcal{P}_i , the complexity of generating macros is polynomial in the size of the domain graph that our algorithm constructs. In the worst case, the size of the domain graph is exponential in the number of ancestors of v_i . In many cases, however, the size of the domain graph is much smaller. In particular, there is one case for which the complexity of our algorithm is provably polynomial.

Theorem 5.1 *Assume that for each $i = 1, \dots, n$, the prevail-condition of each operator in A_i specifies a value for each ancestor of v_i . Then the proposed algorithm runs in polynomial time with complexity $O(|\mathbf{V}||A|^3)$.*

Proof For each value $d \in \mathcal{D}(v_i)$ in the domain of v_i , let $A_i^d = \{a_k \in A_i \mid \text{post}_k = d\}$ be the set of operators that change the value of v_i to d . Since prevail-conditions specify a value for each ancestor, it follows that $v_i \in \mathbf{Z}$ for each $\mathbf{z} \in Z_i$. Let $Z_i^d = \{\mathbf{z} \in Z_i \mid \mathbf{z}[v_i] = d\}$ be the set of projected prevail-conditions that specify the value d for v_i . Any context in the domain graph that specifies the value d for v_i has to either match the prevail-condition of an operator in A_i^d or a projected prevail-condition in Z_i^d . Since each prevail-condition specifies values for each preceding state variable, only a single context matches each prevail-condition. The number of projected prevail-conditions is bounded by the number of operators, so the number of nodes in the domain

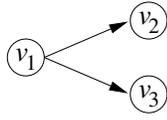


Figure 3: Transitive reduction graph with outdegree > 1

graph is $O(\sum_{d \in \mathcal{D}(v_i)} |A_i^d| + |Z_i^d|) = O(|A_i| + |Z_i|) = O(|A_i| + |A|) = O(|A|)$.

The complexity of Dijkstra is quadratic in the number of nodes in the domain graph, or $O(|A|^2)$. We may have to repeat Dijkstra for each context that satisfies a projected prevail-condition. There can be at most $|A|$ such contexts. Consequently, the complexity of generating macros in sub-problem \mathcal{P}_i is $O(|A||A|^2) = O(|A|^3)$. The total complexity of our algorithm is $O(\sum_{i=1}^n |A|^3) = O(|V||A|^3)$.

6 Extending the algorithm

So far, we presented an algorithm that generates optimal plans in tree-reducible planning domains. In this section, we discuss an extension to domains whose transitive reduction graphs are polytrees. Unlike trees, polytrees may not be weakly connected and have unbounded outdegree. We can apply the algorithm to each weakly connected group of state variables, but unbounded outdegree is more challenging.

Consider the following domain, with $\mathbf{V} = \{v_1, v_2, v_3\}$ and $\mathcal{D}(v_1) = \{0, 1, 2\}$, $\mathcal{D}(v_2) = \mathcal{D}(v_3) = \{0, 1\}$. The initial state is $\mathbf{s}_0 = (0, 0, 0)$, the goal context is $\mathbf{c}_* = (v_2 = 1, v_3 = 1)$, and A contains the following operators:

$$\begin{aligned} a_1^1 &= \langle v_1 = 0, v_1 = 1, \emptyset \rangle, \\ a_1^2 &= \langle v_1 = 0, v_1 = 2, \emptyset \rangle, \\ a_2^1 &= \langle v_2 = 0, v_2 = 1, v_1 = 1 \rangle, \\ a_3^1 &= \langle v_3 = 0, v_3 = 1, v_1 = 2 \rangle. \end{aligned}$$

Figure 3 shows the causal graph of the domain, identical to its transitive reduction in this case. From the point of view of state variable v_2 , it is possible to set v_2 to 1: apply a_1^1 to change v_1 from 0 to 1, followed by a_2^1 to change v_2 from 0 to 1. From the point of view of v_3 , it is possible to set v_3 to 1 by applying a_1^2 followed by a_3^1 . Nevertheless, there is no valid plan that solves the domain, and our algorithm is currently unable to handle this case.

Williams and Nayak [10] required that operators are reversible, i.e., that other operators can reverse their effect.

Definition 6.1 A polytree-reducible planning domain \mathcal{P} is branch-decomposable if, for each v_i with outdegree > 1 in the transitive reduction graph, each $m_j \in O_i$ is reversible.

Let v_i be a state variable with outdegree > 1 in the transitive reduction graph. The definition requires each operator that affects v_i as well as each parent macro of v_i to be reversible. In other words, changing the values of state variables in \mathbf{C}_i does not prevent some future context \mathbf{c}_i from occurring. Thus, it is safe to decompose the planning domain as before and

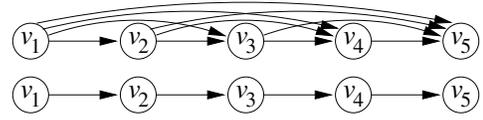


Figure 4: The causal graph and transitive reduction of ToH

solve each branch downstream from v_i separately. The context \mathbf{c}_i that results from solving one branch becomes the initial context of the solution to the next branch. This guarantees completeness, but not optimality.

As an example, consider the same planning domain as above, but with the following two additional operators:

$$\begin{aligned} a_1^3 &= \langle v_1 = 1, v_1 = 0, \emptyset \rangle, \\ a_1^4 &= \langle v_1 = 2, v_1 = 0, \emptyset \rangle. \end{aligned}$$

Since there are no parent macros, each operator $a_k \in O_1$ is reversible, so the new domain is branch-decomposable. First, consider the solution to the branch containing v_2 . This solution is the same as before: apply a_1^1 to change v_1 from 0 to 1, followed by a_2^1 to change v_2 from 0 to 1. The resulting context $v_1 = 1$ becomes the initial context of the solution to the branch containing v_3 . To change v_3 to 1, it is necessary to apply a_1^3 and a_1^2 to change v_1 from 1 to 0 to 2, followed by a_3^1 to change v_3 from 0 to 1. Note that it is perfectly possible to solve the branches in opposite order.

7 Experimental results

To test our algorithm, we ran experiments in two domains: Tower of Hanoi (ToH) and a modified version of GRIPPER. ToH can be modeled as a SAS⁺ instance with one state variable per disc. The domain of each state variable is the three pegs that each disc can occupy. For each disc, there are six operators for moving the disc between each pair of pegs, each with a prevail-condition on the location of each smaller disc. Figure 4 shows the causal graph and transitive reduction of ToH with 5 discs. The transitive reduction is a chain from the smallest to largest disc, i.e., ToH is tree-reducible. In addition, each operator affecting a state variable v_i has a prevail-condition on each ancestor of v_i , so it follows from Theorem 5.1 that our algorithm solves ToH in polynomial time.

The results of the experiments in ToH appear in Table 1. We varied the number of discs in increments of 10 and recorded the running time of our algorithm across 100 trials. The table also shows the number of macros generated by our algorithm, and out of those, how many were used to represent the resulting global plan. For example, 27 out of 82 generated macros formed part of the solution to ToH with 10 discs. Since the solution to ToH requires a number of operators exponential in the number of state variables, any algorithm that generates a complete operator sequence has exponential complexity. In contrast, our algorithm is able to quickly generate a macro that compactly represents the optimal plan. Naturally, executing the resulting plan has exponential complexity.

GRIPPER was one of the domains used at the first planning competition at AIPS 1998. Since our algorithm requires

DISCS	TIME (ms)	MACROS	LENGTH
10	31 ± 14	27/82	> 10 ³
20	84 ± 29	57/172	> 10 ⁶
30	161 ± 32	87/262	> 10 ⁹
40	279 ± 37	117/352	> 10 ¹²
50	457 ± 40	147/442	> 10 ¹⁵
60	701 ± 20	177/532	> 10 ¹⁸

Table 1: Results in Tower of Hanoi

unary operators, we assume unlimited load capacity of the robot. In this case, GRIPPER can be modeled as a SAS⁺ instance with one state variable, v_1 , representing the location of the robot and one state variable for each ball, representing the location of that ball. Operators for picking and dropping balls have prevail-conditions on the robot’s location, so the transitive reduction graph is a polytree (cf. Figure 3 for two balls). In other words, GRIPPER is polytree-reducible, and only v_1 has outdegree > 1. Assuming that the robot can reach any location from any other location, GRIPPER is also branch-decomposable. Thus, it is possible to solve GRIPPER using macros by transporting one ball at a time to the goal location.

We modified GRIPPER such that instead of two rooms, the environment consists of a maze with 967 rooms. To transport balls, the robot must navigate through the maze to the goal location. The robot can only pick balls at the initial location and drop them at the goal location. The results of the experiments appear in Figure 2. The resulting plans are not optimal since the robot could carry all balls at once. However, the experiments illustrate that although the resulting solution length is linear in the number of balls, our algorithm only needs to generate macros once for moving through the maze. If the algorithm is extended to handle non-unary operators, macros can generate a large advantage in domains such as this.

8 Conclusion

We have presented an algorithm that uses macros to generate valid plans for a subclass of planning domains with unary operators and acyclic causal graphs. The algorithm is complete for all domains in the class and optimal for a subclass of domains. Macros make it possible to generate exponential length plans in polynomial time by storing partial plans in memory that can be reused as many times as necessary. This opens up new possibilities for solving classes of planning domains that were previously thought to be intractable.

The complexity of the algorithm depends on the size of local transition graphs constructed by the algorithm, which in turn depends on the number of macros generated in other subproblems. The current version of the algorithm generates macros indiscriminately, many of which are never used as part of a global plan. A possible improvement of the algorithm would be to use inference to distinguish useful macros from superfluous ones.

Another possible extension is to consider domains with non-unary operators and whose causal graphs are not acyclic. Helmert [5] describes a strategy for approximating acyclic causal graphs by excluding some prevail-conditions of oper-

BALLS	TIME (ms)	MACROS	LENGTH
10 ⁰	1528 ± 86	2/5	≈ 300 × 10 ⁰
10 ¹	1535 ± 86	12/14	≈ 300 × 10 ¹
10 ²	1570 ± 86	102/104	≈ 300 × 10 ²
10 ³	2906 ± 143	1002/1004	≈ 300 × 10 ³

Table 2: Results in GRIPPER

ators. In some cases, this leads to deadends, in which case it is necessary to backtrack and include some of the excluded prevail-conditions. Macros are useful in this case since they retain partial plans in subproblems, obviating the need to generate a completely new plan from scratch. If the SAS⁺ representation included a notion of objects, it would be possible to generate macros for one object and share those macros among identical objects. Finally, macros generated in one domain could be stored and reused in other domains.

Acknowledgements

The author would like to thank Hector Geffner, Blai Bonet, Omer Gimenez and the anonymous reviewers for their valuable comments and feedback.

References

- [1] E. Amir and B. Engelhardt. Factored planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 18, pages 929–935, 2003.
- [2] C. Bäckström and B. Nebel. Complexity results for SAS⁺ planning. *Computational Intelligence*, 11(4):625–655, 1995.
- [3] R. Brafman and C. Domshlak. Structure and Complexity in Planning with Unary Operators. *Journal of Artificial Intelligence Research*, 18:315–349, 2003.
- [4] R. Brafman and C. Domshlak. Factored Planning: How, When, and When Not. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 2006.
- [5] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 2006.
- [6] P. Jonsson and C. Bäckström. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, 100(1–2):125–176, 1998.
- [7] P. Jonsson and C. Bäckström. Tractable plan existence does not imply tractable plan generation. *Annals of Mathematics and Artificial Intelligence*, 22(3–4):281–296, 1998.
- [8] C. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.
- [9] A. Lansky and L. Getoor. Scope and abstraction: Two criteria for localized planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 14, pages 1612–1618, 1995.
- [10] B. Williams and P. Nayak. A reactive planner for a model-based executive. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 15, pages 1178–1185, 1997.