In proc. *Twelth National Conference of the American Association for Artificial Intelligence (AAAI-94)*, Seattle, WA, USA, Jul-Aug. 1994, pp. 998–1003

Tractable Planning with State Variables by Exploiting Structural Restrictions

Peter Jonsson and Christer Bäckström[1]
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
email: {petej,cba}@ida.liu.se
phone: +46 13 282429
fax: +46 13 282606

Abstract

So far, tractable planning problems reported in the literature have been defined by syntactical restrictions. To better exploit the inherent structure in problems, however, it is probably necessary to study also structural restrictions on the state-transition graph. Such restrictions are typically computationally hard to test, though, since this graph is of exponential size. Hence, we take an intermediate approach, using a state-variable model for planning and restricting the state-transition graph implicitly by restricting the transition graph for each state variable in isolation. We identify three such restrictions which are tractable to test and we present a planning algorithm which is correct and runs in polynomial time under these restrictions.

Introduction

Many planning problems in manufacturing and process industry are believed to be highly structured, thus allowing for efficient planning if exploiting this structure. However, a 'blind' domain-independent planner will most likely go on tour in an exponential search space even for tractable problems. Although heuristics may help a lot, they are often not based on a sufficiently thorough understanding of the underlying problem structure to guarantee efficiency and correctness. Further, we believe that if having such a deep understanding of the problem structure, it is better to use other methods than heuristics.

Some tractability results for planning have been reported in the literature lately (Bäckström & Klein 1991; Bäckström & Nebel 1993; Bylander 1991; Erol, Nau, & Subrahmanian 1992). However, apart from being very restricted, they are all based on essentially syntactic restrictions on the set of operators. Syntactic restrictions are very appealing to study, since they are typically easy to define and not very costly to test. However, to gain any deeper insight into what makes planning problems hard and easy respectively probably require that we study the structure of the problem, in particular the state-transition graph induced by the operators. To some extent, syntactic restrictions allow us this since they undoubtedly have implications for what this graph looks like. However, their value for this purpose seems somewhat limited since many properties that are easy to express as explicit structural restrictions would require horrendous syntactical equivalents. Putting explicit restrictions on the state-transition graph must be done with great care, however. This graph is typically of size exponential in the size of the planning problem instance, making it extremely costly to test arbitrary properties. In this paper, we take an intermediate approach. We adopt the state-variable model SAS$^+$(Bäckström & Nebel 1993) and define restrictions not on the whole state-transition graph, but on the domain-transition graph for each state variable in isolation. This is less costly since each such graph is only of polynomial size. Although not being a substitute for restrictions on the whole state-transition graph, many interesting and useful properties of this graph can be indirectly exploited. In particular, we identify three structural restrictions which makes planning tractable and which properly generalize previously studied tractable SAS$^+$ problems (Bäckström & Klein 1991; Bäckström & Nebel 1993). We present an algorithm for generating optimal plans under our restrictions. Despite being structural, our restrictions can be tested in polynomial time. Further, note that this approach would not be very useful for a planning formalism based on propositional atoms, since the resulting two-vertex domain-transition graphs would not allow for very interesting structure to exploit.

The SAS$^+$ Formalism

We use the SAS$^+$ formalism (Bäckström & Klein 1991; Bäckström & Nebel 1993), which is a variant of propositional STRIPS, generalizing the atoms to multi-valued state variables. Furthermore, what is called a precondition in STRIPS is here divided into

[1]This research was sponsored by *the Swedish Research Council for the Engineering Sciences (TFR)* under grants Dnr. 92-143 and Dnr. 93-00291.

two conditions, the precondition and the prevailcondition. Variables which are required and changed by an operator go into the precondition and those which remain unchanged, but are required, go into the prevailcondition.[2] We briefly recapitulate the SAS$^+$ formalism below, referring to Bäckström and Nebel (Bäckström & Nebel 1993) for further explanation. We follow their presentation, except for replacing the variable indices by variables and some other minor changes.

Definition 1 *An instance of the SAS$^+$ planning problem is given by a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ with components defined as follows:*

- $\mathcal{V} = \{v_1, \ldots, v_m\}$ *is a set of* **state variables**. *Each variable $v \in \mathcal{V}$ has an associated* **domain** $\mathcal{D}_v$, *which implicitly defines an* **extended domain** $\mathcal{D}_v^+ = \mathcal{D}_v \cup \{\mathsf{u}\}$, *where* u *denotes the* **undefined value**. *Further, the* **total state space** $\mathcal{S} = \mathcal{D}_{v_1} \times \ldots \times \mathcal{D}_{v_m}$ *and the* **partial state space** $\mathcal{S}^+ = \mathcal{D}_{v_1}^+ \times \ldots \times \mathcal{D}_{v_m}^+$ *are implicitly defined. We write $s[v]$ to denote the value of the variable v in a state s.*
- $\mathcal{O}$ *is a set of* **operators** *of the form* $\langle \mathsf{b}, \mathsf{e}, \mathsf{f} \rangle$, *where* $\mathsf{b}, \mathsf{e}, \mathsf{f} \in \mathcal{S}^+$ *denote the* **pre-**, **post-** *and* **prevailcondition** *respectively. If $o = \langle \mathsf{b}, \mathsf{e}, \mathsf{f} \rangle$ is a SAS$^+$ operator, we write $\mathsf{b}(o)$, $\mathsf{e}(o)$ and $\mathsf{f}(o)$ to denote b, e and f respectively. $\mathcal{O}$ is subject to the following two restrictions*
  - **(R1)** *for all $o \in \mathcal{O}$ and $v \in \mathcal{V}$ if $\mathsf{b}(o)[v] \neq \mathsf{u}$, then $\mathsf{b}(o)[v] \neq \mathsf{e}(o)[v] \neq \mathsf{u}$,*
  - **(R2)** *for all $o \in \mathcal{O}$ and $v \in \mathcal{V}$, $\mathsf{e}(o)[v] = \mathsf{u}$ or $\mathsf{f}(o)[v] = \mathsf{u}$.*
- $s_0 \in \mathcal{S}^+$ *and* $s_* \in \mathcal{S}^+$ *denote the* **initial state** *and* **goal state** *respectively.*

We write $s \sqsubseteq t$ if the state s is subsumed (or satisfied) by state t, ie. if $s[v] = \mathsf{u}$ or $s[v] = t[v]$. We extend this notion to whole states, defining

$$s \sqsubseteq t \text{ iff for all } v \in \mathcal{V}, s[v] = \mathsf{u} \text{ or } s[v] = t[v].$$

$Seqs(\mathcal{O})$ denotes the set of operator sequences over $\mathcal{O}$ and the members of $Seqs(\mathcal{O})$ are called **plans**. *Given two states $s, t \in \mathcal{S}^+$, we define for all $v \in \mathcal{V}$,*

$$(s \oplus t)[v] = \begin{cases} t[v] & \text{if } t[v] \neq \mathsf{u}, \\ s[v] & \text{otherwise.} \end{cases}$$

The ternary relation $Valid \subseteq Seqs(\mathcal{O}) \times \mathcal{S}^+ \times \mathcal{S}^+$ is defined recursively s.t. for arbitrary operator sequence $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ and arbitrary states $s, t \in \mathcal{S}^+$, $Valid(\langle o_1, \ldots, o_n \rangle, s, t)$ iff either

1. *$n = 0$ and $t \sqsubseteq s$ or*
2. *$n > 0$, $\mathsf{b}(o_1) \sqsubseteq s$, $\mathsf{f}(o_1) \sqsubseteq s$ and $Valid(\langle o_2, \ldots, o_n \rangle, (s \oplus \mathsf{e}(o_1)), t)$.*

Finally, a plan $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ **solves** Π *iff $Valid(\langle o_1, \ldots, o_n \rangle, s_0, s_*)$.*

[2] Drummond & Currie (1988) make the same distinction.

To define partially ordered plans, we must introduce the concept of *actions*, *ie.* instances of operators. Given an action a, $type(a)$ denotes the operator that a instantiates. Furthermore, given a set of actions $\mathcal{A}$, we define $type(\mathcal{A}) = \{type(a) \mid a \in \mathcal{A}\}$ and given a sequence $\alpha = \langle a_1, \ldots, a_n \rangle$ of actions, $type(\alpha)$ denotes the operator sequence $\langle type(a_1), \ldots, type(a_n) \rangle$.

Definition 2 *A* **partial-order plan** *is a tuple $\langle \mathcal{A}, \prec \rangle$ where $\mathcal{A}$ is a set of actions, ie. instances of operators, and $\prec$ is a strict partial order on $\mathcal{A}$. A partial-order plan $\langle \mathcal{A}, \prec \rangle$ solves a SAS$^+$ instance Π iff $\langle type(a_1), \ldots, type(a_n) \rangle$ solves Π for each topological sort $\langle a_1, \ldots, a_n \rangle$ of $\langle \mathcal{A}, \prec \rangle$.*

Further, given a set of actions $\mathcal{A}$ over $\mathcal{O}$, and a variable $v \in \mathcal{V}$, we define $\mathcal{A}[v] = \{a \in \mathcal{A} \mid \mathsf{e}(a)[v] \neq \mathsf{u}\}$, *ie.* the set of all actions in $\mathcal{A}$ affecting v.

Structural Restrictions

In this section we will define three structural restrictions (I, A and O) on the state-transition graph, or, rather, on the domain-transition graphs for each state variable in isolation. We must first define some other concepts, however. Most of these concepts are straightforward, possibly excepting the set of requestable values, which plays an important role for the planning algorithm in the following section. Unary operators is one of the restrictions considered by Bäckström and Nebel (1993), but the others are believed novel. For the definitions below, let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance.

Definition 3 *An operator $o \in \mathcal{O}$ is* **unary** *iff there is exactly one $v \in \mathcal{V}$ s.t. $\mathsf{e}(o)[v] \neq \mathsf{u}$.*

A value $x \in \mathcal{D}_v$ where $x \neq \mathsf{u}$ for some variable $v \in \mathcal{V}$ is said to be *requestable* if there exists some action $o \in \mathcal{O}$ such that o needs x in order to be executed.

Definition 4 *For each $v \in \mathcal{V}$ and $\mathcal{O}' \subseteq \mathcal{O}$, the set $\mathcal{R}_v^{\mathcal{O}'}$ of* **requestable values** *for $\mathcal{O}'$ is defined as*

$$\begin{aligned} \mathcal{R}_v^{\mathcal{O}'} \quad = \quad & \{\mathsf{f}(o)[v] \mid o \in \mathcal{O}'\} \cup \\ & \{\mathsf{b}(o)[v], \mathsf{e}(o)[v] \mid o \in \mathcal{O}' \text{ and } o \text{ non-unary}\} \\ & -\{\mathsf{u}\}. \end{aligned}$$

Similarly, for a set $\mathcal{A}$ of actions over $\mathcal{O}$, we define $\mathcal{R}_v^{\mathcal{A}} = \mathcal{R}_v^{type(\mathcal{A})}$.

Obviously, $\mathcal{R}_v^{\mathcal{O}} \subseteq \mathcal{D}_v$ for all $v \in \mathcal{V}$. For each state variable domain, we further define the graph of possible transitions for this domain, without taking the other domains into account, and the reachability graph for arbitrary subsets of the domain.

Definition 5 *For each $v \in \mathcal{V}$, we define the corresponding* **domain transition graph** G_v *as a directed labelled graph $G_v = \langle \mathcal{D}_v^+, \mathcal{T}_v \rangle$ with vertex set $\mathcal{D}_v^+$ and arc set $\mathcal{T}_v$ s.t. for all $x, y \in \mathcal{D}_v^+$ and $o \in \mathcal{O}$, $\langle x, o, y \rangle \in \mathcal{T}_v$ iff $\mathsf{b}(o)[v] = x$ and $\mathsf{e}(o)[v] = y \neq \mathsf{u}$. Further, for* *each* $X \subseteq \mathcal{D}_v^+$ *we define the* **reachability graph** *for* X *as a directed graph* $G_v^X = \langle X, \mathcal{T}_X \rangle$ *with vertex set* X *and arc set* $\mathcal{T}_X$ *s.t. for all* $x, y \in X$, $\langle x, y \rangle \in \mathcal{T}_X$ *iff there is a path from* x *to* y *in* G_v.

Alternatively, G_v^X can be viewed as the restriction to $X \subseteq \mathcal{D}_v^+$ of the transitive closure of G_v, but with unlabelled arcs. When speaking about a path in a domain-transition graph below, we will typically mean the sequence of labels, *ie.* operators, along this path. We say that a path in G_v is *via* a set $X \subseteq \mathcal{D}_v$ iff each member of X is visited along the path, possibly as the initial or final vertex.

Definition 6 *An operator* $o \in \mathcal{O}$ *is* **irreplaceable** *wrt. a variable* $v \in \mathcal{V}$ *iff removing an arc labelled with* o *from* G_v *splits some component of* G_v *into two components.*

In the remainder of this paper we will be primarily interested in SAS$^+$ instances satisfying the following restrictions.

Definition 7 *A SAS*$^+$ *instance* $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ *is:*

(I) Interference-safe *iff every operator* $o \in \mathcal{O}$ *is either unary or irreplaceable wrt. every* $v \in \mathcal{V}$ *it affects.*

(A) Acyclic *iff* $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ *is acyclic for each* $v \in \mathcal{V}$.

(O) prevail-Order-preserving *iff for each* $v \in \mathcal{V}$, *whenever there are two* $x, y \in \mathcal{D}_v^+$ *s.t.* G_v *has a shortest path* $\langle o_1, \ldots, o_m \rangle$ *from* x *to* y *via some set* $X \subseteq \mathcal{R}_v^{\mathcal{O}}$ *and it has any path* $\langle o'_1, \ldots, o'_n \rangle$ *from* x *to* y *via some set* $Y \subseteq \mathcal{R}_v^{\mathcal{O}}$ *s.t.* $X \subseteq Y$, *there exists some subsequence* $\langle \ldots, o'_{i_1}, \ldots, o'_{i_m}, \ldots \rangle$ *s.t.* $\mathsf{f}(o_k) \sqsubseteq \mathsf{f}(o'_{i_k})$ *for* $1 \leq k \leq m$.

We will be mainly concerned with SAS$^+$-IA and SAS$^+$-IAO instances, that is, SAS$^+$ instances satisfying the two restrictions I and A and SAS$^+$ instances satisfying all three restrictions respectively. Both restrictions I and A are tractable to test. The complexity of testing O in isolation is currently an open issue, but the combinations IA and IAO are tractable to test.

Theorem 8 *The restrictions I and A can be tested in polynomial time for arbitrary SAS*$^+$ *instances. Restriction O can be tested in polynomial time for SAS*$^+$ *instances satisfying restriction A.*

Proof sketch: [3] Testing A is trivially a polynomial time problem. Finding the irreplaceable operators wrt. a variable $v \in \mathcal{V}$ can be done in polynomial time by identifying the maximal strongly connected components in G_v, collapsing each of these into a single vertex and perform a reachability analysis. Since it is further polynomial to test whether an operator is unary, it follows that also I can be tested in polynomial time.

Furthermore, given that the instance satisfies A, O can be tested in polynomial time as follows. For each pair of vertices $x, y \in \mathcal{D}_v^+$, find a shortest path in G_v from x to y. If the instance satisfies O, then for each operator o along this path, $\mathcal{D}_v^+$ can be partitioned into two disjoint sets X, Y s.t. every arc from some vertex in X to some vertex in Y is labelled by an operator o' satisfying that $\mathsf{f}(o) \sqsubseteq \mathsf{f}(o')$. This can be tested in polynomial time by a method similar to finding shortest-paths in G_v. Hence, O can be tested in polynomial time if A holds. □

Furthermore, the SAS$^+$-IAO problem is strictly more general than the SAS$^+$-PUS problem (Bäckström & Nebel 1993).

Theorem 9 *All SAS*$^+$*-PUS instances are SAS*$^+$*-IAO instances, while a SAS*$^+$*-IAO instance need not satisfy either P, U or S.*

Planning Algorithm

Before describing the actual planning algorithm, we make the following observations about the solutions to arbitrary SAS$^+$ instances.

Theorem 10 *Let* $\langle \mathcal{A}, \prec \rangle$ *be a partial-order plan solving some SAS*$^+$ *instance* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$. *Then for each* $v \in \mathcal{V}$ *and for each action sequence* α *which is a total ordering of* $\mathcal{A}[v]$ *consistent with* $\prec$, *the operator sequence type*(α) *is a path in* G_v *from* $s_0[v]$ *to* $s_*[v]$ *via* $\mathcal{R}_v^{\mathcal{A}}$.

This is a declarative characterization of the solutions and it cannot be immediately cast in procedural terms—the main reason being that we cannot know the sets $\mathcal{R}_v^{\mathcal{A}}$ in advance. These sets must, hence, be computed incrementally, which can be done in polynomial time under the restrictions I and A. We have devised an algorithm, *Plan* (Figure 1), which serves as a plan generation algorithm under these restrictions.

The heart of the algorithm is the procedure *Extend*, which operates on the global variables $X_1, \ldots, X_m$, extending these monotonically. It also returns operator sequences in the global variables $\omega_1, \ldots, \omega_m$, but only their value after the last call are used by *Plan*. For each i, *Extend* first finds a shortest path ω_i in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via X_i. (The empty path $\langle\rangle$ is considered as the shortest path from any vertex x to u, since $\mathsf{u} \sqsubseteq x$). If no such path exists, then *Extend* fails and otherwise each X_i is set to $\mathcal{R}_{v_i}^{\mathcal{O}'}$, where $\mathcal{O}'$ is the set of all operators along the paths $\omega_1, \ldots, \omega_m$. The motivation for this is as follows: If $\mathsf{f}(o)[v_i] = x \neq \mathsf{u}$ for some i and some operator o in some ω_j, then some action in the final plan must achieve this value, unless it holds initially. Hence, x is added to X_i to ensure that *Extend* will find a path via x in the next iteration. Similarly, each non-unary operator occurring in some ω_i must also appear in ω_j for all other j such that o affects v_j.

Starting with all $X_1, \ldots, X_m$ initially empty, *Plan* calls *Extend* repeatedly until nothing more is added to these sets or *Extend* fails. Viewing *Extend* as a function *Extend* : $\mathcal{S}^+ \rightarrow \mathcal{S}^+$, *ie.* ignoring the side effect on

[3] The full proofs of all theorems can be found in (Jonsson & Bäckström 1994).

1 **procedure** *Plan*($\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$);
2 $\langle X_1, \ldots, X_m \rangle \leftarrow \langle \varnothing, \ldots, \varnothing \rangle$;
3 **repeat**
4 *Extend*;
5 **until** no X_i is changed;
6 *Instantiate*;
7 **for** $1 \leq i \leq m$ and $a, b \in \alpha_i$ **do**
8 Order $a \prec b$ iff a precedes b in α_i;
9 **for** $1 \leq i \leq m$ and $a \in \mathcal{A}$ s.t. $\mathsf{f}(a)[v_i] \neq \mathsf{u}$ **do**
10 *Assume* $\alpha_i = \langle a_1, \ldots, a_k \rangle$
11 **if** $\mathsf{e}(a_l)[v_i] = \mathsf{f}(a)[v_i]$ for some $1 \leq l \leq k$ **then**
12 Order $a_l \prec a$;
13 **if** $l < k$ **then** Order $a \prec a_{l+1}$;
14 **else** Order $a \prec a_1$;
15 $\mathcal{A} \leftarrow \{a \in \alpha_i \mid 1 \leq i \leq m\}$;
16 **if** $\prec$ is acyclic **then return** $\langle \mathcal{A}, \prec \rangle$;
17 **else fail**;

procedure *Extend*; (*Modifies* $X_1, \ldots, X_m$ *and* $\omega_1, \ldots, \omega_m$)
for $1 \leq i \leq m$ **do**
$\omega_i \leftarrow$ any shortest path from $s_0[v_i]$ to $s_*[v_i]$ in G_v via X_i;
if no such path exists **then fail**;
for $1 \leq i, j \leq m$ and $o \in \omega_j$ **do**
$X_i \leftarrow X_i \cup \{\mathsf{f}(o)[v_i]\} - \{\mathsf{u}\}$;
if o not unary **then**
$X_i \leftarrow X_i \cup \{\mathsf{b}(o)[v_i], \mathsf{e}(o)[v_i]\} - \{\mathsf{u}\}$;

procedure *Instantiate*; (*Modifies* $\alpha_1, \ldots, \alpha_m$)
for $1 \leq i \leq m$ **do**
Assume $\omega_i = \langle o_1, \ldots, o_k \rangle$
for $1 \leq l \leq k$ **do**
if o_l not unary **and** there is some a of type o_l in α_j for some $j < i$ **then** $a_l \leftarrow a$;
else Let a_l be a new instance of *type*(a_l);
$\alpha_i \leftarrow \langle a_1, \ldots, a_k \rangle$

Figure 1: Planning Algorithm

$\omega_1, \ldots, \omega_m$, this process corresponds to constructing the minimal fixed point for *Extend* in $\mathcal{S}^+$. The paths $\omega_1, \ldots, \omega_m$ found in the last iteration contain all the operators necessary in the final solution and procedure *Instantiate* instantiates these as actions. This works such that all occurrences of a non-unary operator are merged into one unique instance while all occurrences of a unary operator are made into distinct instances. It remains to compute the action ordering on the set $\mathcal{A}$ of all such operator instances (actions). For each v_i, the total order implicit in the operator sequence ω_i is kept as a total ordering on the corresponding actions. Finally, each action a s.t. $\mathsf{f}(a)[v_i] = x \neq \mathsf{u}$ for some i must be ordered after some action a' providing this condition. There turns out to always be a unique such action, or none if $v_i = x$ initially. Similarly, a must be ordered before the first action succeeding a' that destroys its prevailcondition. Finally, if $\prec$ is acyclic, then $\langle \mathcal{A}, \prec \rangle$ is returned and otherwise *Plan* fails. Observe that the algorithm does not compute the transitive closure $\prec^+$ of $\prec$ since this is a costly operation and the transitive closure is not likely to be of interest for executing the plan.

Procedure *Plan* is sound for SAS$^+$-IA instances and it is further optimal and complete for SAS$^+$-IAO instances.

Theorem 11 *If Plan returns a plan $\langle \mathcal{A}, \prec \rangle$ when given a SAS$^+$-IA instance Π as input, then $\langle \mathcal{A}, \prec \rangle$ solves Π and if Π is a SAS$^+$-IAO instance, then $\langle \mathcal{A}, \prec \rangle$ is also minimal. Further, if Plan fails when given a SAS$^+$-IAO instance Π as input, then there exists no plan solving Π.*

Proof outline: The proofs for this theorem are quite long, but are essentially based on the following observations. Soundness is rather straightforward from the algorithm and Theorem 10. Further, let $\langle \mathcal{A}, \prec \rangle$ be the plan returned by *Plan* and let $\langle \mathcal{A}', \prec' \rangle$ be an arbitrary solution to Π. Minimality follows from observing that $\mathcal{R}_v^{\mathcal{A}} \subseteq \mathcal{R}_v^{\mathcal{A}'}$ for all $v \in \mathcal{V}$. The completeness proof essentially builds on minimality and proving that if $\prec$ contains a cycle (*ie. Plan* fails in line 16), then there can exist no solution to Π. □

Furthermore, *Plan* returns LC2-minimal plans (Bäckström 1993) which means that there does not exist any strict (*ie.* irreflexive) partial order $\prec'$ on $\mathcal{A}$ such that $|\prec'| < |\prec^+|$ and $\langle \mathcal{A}, \prec' \rangle$ is a valid plan. Finally, *Plan* runs in polynomial time.

Theorem 12 *Plan has a worst-case time complexity of $O(|\mathcal{O}|^2(|\mathcal{V}| \max_{v \in \mathcal{V}} |\mathcal{D}_v|)^3)$.*

Example

In this section, we will present a small, somewhat contrived example of a manufacturing workshop and show how the algorithm handles this example. We assume that there is a supply of rough workpieces and a table for putting finished products. There are also two workstations: a lathe and a drill. To simplify matters, we will consider only one single workpiece. Two different shapes can be made in the lathe and one type of hole can be drilled. Furthermore, only workpieces of shape 2 fit in the drill. This gives a total of four possible combinations for the end product: rough (*ie.* not worked on), shape 1, shape 2 without a hole and shape 2 with a hole. Note also that operator Shape2 is tougher to the cutting tool than Shape1 is—the latter allowing us to continue using the cutting tool afterwards. Finally, both the lathe and the drill require that the power is on. This is all modelled by five state variables, as shown in Table 1, and nine operators, as shown in Table 2. This example is a SAS$^+$-IAO instance, but it does not satisfy either of the P, U and S restrictions in Bäckström and Nebel (1993).[4]

[4]Note in particular that since we do no longer require the S restriction, we can model sequences of workstations, which was not possible under the PUS restriction (Bäckström & Klein 1991).

variable	domain	denotes
1	{Supply,Lathe,Drill,Table}	Position of workpiece
2	{Rough,1,2}	Workpiece shape
3	{Mint,Used}	Condition of cutting tool
4	{Yes,No}	Hole in workpiece
5	{Yes,No}	Power on

Table 1: State variables for the workshop example

Operator	Precondition	Postcondition	Prevailcondition
MvSL	$v_1 = S$	$v_1 = L$	
MvLT	$v_1 = L$	$v_1 = T$	
MvLD	$v_1 = L$	$v_1 = D$	$v_2 = 2$
MvDT	$v_1 = D$	$v_1 = T$	
Shape1	$v_2 = R$	$v_2 = 1$	$v_1 = L, v_3 = M, v_5 = Y$
Shape2	$v_2 = R, v_3 = M$	$v_2 = 2, v_3 = U$	$v_1 = L, v_5 = Y$
Drill	$v_4 = N$	$v_4 = Y$	$v_1 = D, v_5 = Y$
Pon	$v_5 = N$	$v_5 = Y$	
Poff	$v_5 = Y$	$v_5 = N$	

Table 2: Operators for the workshop example. (Domain values will typically be denoted by their initial characters only).

After iteration 1:

$\omega_1 = \langle$MvSL,MvLT$\rangle$	$X_1 = \{L, D\}$
$\omega_2 = \langle$Shape2$\rangle$	$X_2 = \{R, 2\}$
$\omega_3 = \langle\rangle$	$X_3 = \{M, U\}$
$\omega_4 = \langle$Drill$\rangle$	$X_4 = \{\}$
$\omega_5 = \langle\rangle$	$X_5 = \{Y\}$

After iteration 2:

$\omega_1 = \langle$MvSL,MvLD,MvDT$\rangle$	$X_1 = \{L, D\}$
$\omega_2 = \langle$Shape2$\rangle$	$X_2 = \{R, 2\}$
$\omega_3 = \langle$Shape2$\rangle$	$X_3 = \{M, U\}$
$\omega_4 = \langle$Drill$\rangle$	$X_4 = \{\}$
$\omega_5 = \langle$Pon,Poff$\rangle$	$X_5 = \{Y\}$

Table 3: The variables ω_i and X_i in the example.

Suppose we start in $s_0 = \langle S, R, M, N, N\rangle$ and set the goal $s_* = \langle T, 2, \mathsf{u}, Y, N\rangle$, that is, we want to manufacture a product of shape 2 with a drilled hole. We also know that the cutting tool for the lathe is initially in mint condition, but we do not care about its condition after finishing. Finally, the power is initially off and we are required to switch it off again before leaving the workshop. Procedure *Plan* will make two calls to *Extend* before terminating the loop successfully, with variable values as in Table 3.

The operators in the operator sequences $\omega_1, \ldots, \omega_5$ will be instantiated to actions, where both occurrences of Shape2 are instantiated as the same action, since Shape2 is non-unary. Since there is not more than one action of each type in this plan, we will use the name of the operators also as names of the actions. The total orders in $\omega_1, \ldots, \omega_m$ is retained in $\alpha_1, \ldots, \alpha_m$. Furthermore, Shape2 must be ordered after MvSL and before MvLD, since its prevailcondition on variable 1 equals the postcondition of MvLD for this variable. Similarly, Drill must be ordered between MvLD and MvDT because of its prevailcondition on variable 1 and both Shape2 and Drill must be ordered between Pon and Poff because of their prevailcondition on variable 5. Furthermore, MvLD must (once again) be ordered after Shape2 because of its prevailcondition on variable 2. The final partial-order plan is shown in Figure 2.

Discussion

Several attempts on exploiting structural properties on planning problems in order to decrease complexity have been reported in the literature, but none with the aim of obtaining polynomial-time planning problems. Korf (1987) has defined some structural properties of planning problems modelled by state-variables, for instance serial operator decomposability. However, this property is PSPACE-complete to test (Bylander 1992), but does not guarantee tractable planning. Mädler (1992) extends Sacerdoti's (1974) essentially syntactic state abstraction technique to *structural abstraction*, identifying bottle-neck states (*needle's eyes*) in the state-transition graph for a state-variable formalism. Smith and Peot (1993) use an *operator graph* for preprocessing planning problem instances, identifying potential threats that can be safely postponed during planning—thus, pruning the search tree. The operator graph can be viewed as an abstraction of the full state-transition graph, containing all the information relevant to analysing threats.

A number of issues are on our research agenda for the future. Firstly, we should perform a more careful analysis of the algorithm to find a tighter upper bound for the time complexity. Furthermore, the fixpoint for the *Extend* function is now computed by Jacobi iteration, which is very inefficient. Replacing this by some strategy for *chaotic iteration* (Cousot & Cousot 1977)

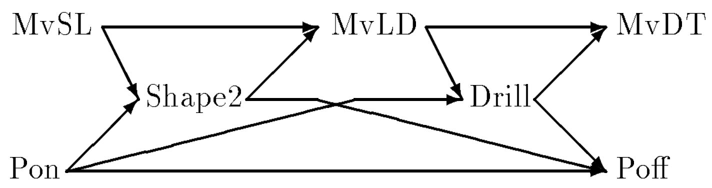

Figure 2: The final partial-order plan.

would probably improve the complexity considerably, at least in the average case. Furthermore, the algorithm is likely to be sound and complete for less restricted problems than SAS$^+$-IA and SAS$^+$-IAO respectively. Finding restrictions that more tightly reflect the limits of the algorithm is an issue for future research. We also plan to investigate some modifications of the algorithm, *eg.*, letting the sets $X_1, \ldots, X_m$ be partially ordered multisets, allowing a prevailcondition to be produced and destroyed several times—thus relaxing the A restriction. Another interesting modification would be to relax some of the restrictions and redefine *Extend* as a non-deterministic procedure. Although requiring search and, thus, probably sacrificing tractability, we believe this to be an intriguing alternative to ordinary search-based planning.

Conclusions

We have identified a set of restrictions allowing for the generation of optimal plans in polynomial time for a planning formalism, SAS$^+$, using multi-valued state variables. This extends the tractability borderline for planning, by allowing for more general problems than previously reported in the literature to be solved tractably. In contrast to most restrictions in the literature, ours are structural restrictions. However, they are restrictions on the transition graph for each state variable in isolation, rather than for the whole state space, so they can be tested in polynomial time. We have also presented a provably correct, polynomial time algorithm for planning under these restrictions.

References

American Association for Artificial Intelligence. 1992. *Proceedings of the 10th (US) National Conference on Artificial Intelligence (AAAI-92)*, San José, CA, USA.

Bäckström, C., and Klein, I. 1991. Parallel non-binary planning in polynomial time. In Reiter and Mylopoulos (1991), 268–273.

Bäckström, C., and Nebel, B. 1993. Complexity results for SAS$^+$ planning. In Bajcsy, R., ed., *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*. Chambéry, France: Morgan Kaufmann.

Bäckström, C. 1993. Finding least constrained plans and optimal parallel executions is harder than we thought. In Bäckström, C., and Sandewall, E., eds., *Current Trends in AI Planning: EWSP'93—2nd European Workshop on Planning*, Frontiers in AI and Applications. Vadstena, Sweden: IOS Press.

Bylander, T. 1991. Complexity results for planning. In Reiter and Mylopoulos (1991), 274–279.

Bylander, T. 1992. Complexity results for serial decomposability. In AAAI-92 (1992), 729–734.

Cousot, P., and Cousot, R. 1977. Automatic synthesis of optimal invariant assertions: Mathematical foundations. *SIGPLAN Notices* 12(8):1–12.

Drummond, M., and Currie, K. 1988. Exploiting temporal coherence in nonlinear plan construction. *Computational Intelligence* 4(4):341–348. Special Issue on Planning.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1992. On the complexity of domain-independent planning. In AAAI-92 (1992), 381–386.

Jonsson, P., and Bäckström, C. 1994. Tractable planning with state variables by exploiting structural restrictions. Research report, Department of Computer and Information Science, Linköping University.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.

Mädler, F. 1992. Towards structural abstraction. In Hendler, J., ed., *Artificial Intelligence Planning Systems: Proceedings of the 1st International Conference*, 163–171. College Park, MD, USA: Morgan Kaufmann.

Reiter, R., and Mylopoulos, J., eds. 1991. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*. Sydney, Australia: Morgan Kaufmann.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5(2):115–135.

Smith, D. E., and Peot, M. A. 1993. Postponing threats in partial-order planning. In *Proceedings of the 11th (US) National Conference on Artificial Intelligence (AAAI-93)*, 500–506. Washington DC, USA: American Association for Artificial Intelligence.