Paper II

Tractable Planning for an Assembly Line

Inger Klein, Peter Jonsson and Christer Bäckström

Abstract

Industry wants formal methods for dealing with combinatorial dynamical systems that are provably correct and fast. One example is error recovery in industrial processes. We have used a provably correct, polynomial-time planning algorithm to plan for a miniature assembly line, which assembles toy cars. Although somewhat limited, this process has many similarities with real industrial processes. By exploring the structure of this assembly line we have extended a previously presented algorithm increasing the class of problems that can be handled in polynomial time.

1 Introduction

AI planning in its general form is known to be very hard. STRIPS planning is undecidable in the first-order case [Chapman, 1987; Erol *et al.*, 1992b], and PSPACE-complete in the unrestricted propositional case [Bylander, 1991]. Complexity results have also been reported for a number of restricted cases [Bäckström and Nebel, 1993; Bylander, 1991; Erol *et al.*, 1992a; Gupta and Nau, 1992], most of these being computationally difficult. However, these results apply to the complexity of planning in various restricted versions of certain formalisms. The results do not say anything about the inherent complexity of naturally arising planning problems. For instance, the ubiquitous *blocks-world problem* in its standard form can be encoded in the propositional STRIPS formalism, where plan existence is PSPACE-complete. Yet, plan existence for the blocks-world problem *per se* is only NP-complete [Gupta and Nau, 1992], for finding an optimal plan, or even tractable [Gupta and Nau, 1992], if we do not require an optimal plan. Furthermore, the modified problem where several blocks can have identical labels cannot be naturally modelled in propositional STRIPS, yet this problem is also NP-complete [Chenoweth, 1991]. That is, the problem lies within NP, but it cannot be reasonably encoded even in a PSPACE-complete (standard) planning formalism. Finding the restrictions on planning formalisms that capture the inherent structure and complexity of interesting applications problems is thus an interesting challenge.

The usual way to tackle the complexity of planning problems is to use a general-purpose domain-independent planner and add heuristics that are tailored specifically to the problem at hand. Unless the problem is inherently tractable, this method cannot guarantee improved performance, but only result in a better average-case complexity. Alternatively, it can improve the worst-case complexity figure, but at the expense of incompleteness, *ie.* possibly missing that a problem has a solution.

For many applications we may be satisfied with an incomplete algorithm or with a good average-case complexity, despite intractability in the worst case. The latter of these situations is also a form of incompleteness since the intractable worst cases can be reagarded as unsolvable in practice. On the other hand, there are also applications where we do need correct solutions promptly. Such applications arise, for instance, in the area of *sequential control* within *automatic control*, where many problems can be viewed as planning problems and we often need to find solutions correctly and promptly. Automatic control has a long tradition of using mathematically well-founded methods with provable properties. Researchers in this area see the lack of such theories as one of the major problems with AI planning [Passino and Antsaklis, 1989]. Furthermore, Åström and Benveniste [1993] present a study of how computer software is used in large-scale control applications such as the process industry and metro traffic networks. One of their findings was that industry requires more mathematical tools for modelling dynamical systems of combinatorial nature. Using formal methods is a way to improve formal guarantees and reduce the complexity of the resulting code. That is, we have to aim at more formal methods than heuristics whenever possible.

The manufacturing process industry is one example of application areas where AI planning can be useful. The problem is not primarily to find the plan for normal operation of the plant. That is usually performed once and for all and can probably be done better manually, since time is not critical in this case. Automated planning is more likely to enter the scene when something goes wrong. Since there are many ways in which a large process may go wrong, we can end up in any of a very large number of states. It is not realistic to have pre-compiled plans for recovering from any such state, so it would be useful to find a plan automatically for how to get back to a safe state where normal operation can resume. It is important that such a plan is correct and found quickly since costs accumulate rapidly when large-scale industrial processes are non-operational. Automated plan generation is also important if the initial state is not fully specified until the plan is neeeded. As in the error recovery case, it is important that the plan is correct and that it is found reasonably quickly.

What we ideally need is a provably correct planner that runs in polynomial time. No general-purpose such planner can exist, however, but we may be able to find planners for certain restricted planning problems with practical use. We have run a project for over five years trying to identify restricted, tractable planning problems. Starting with a test problem in sequential control, we have identified a number of inherent restrictions of this problem which, taken together, result in tractability. We have defined a formal planning problem, the SAS-PUBS problem, based on these restrictions, proved this problem tractable and devised a provably correct, polynomial-time algorithm for it [Bäckström and Klein, 1991b; Klein and Bäckström, 1991]. We have removed restrictions successively which resulted in more general yet tractable planning problems [Bäckström, 1992; Bäckström and Klein, 1991a; Bäckström and Nebel, 1993; Jonsson and Bäckström, 1994]. Our general research methodology has been to to use a bottom-up strategy, starting with a tractable problem and remove or replace restrictions so that the resulting problem is either more expressive than or different from the original problem, but still tractable.

We report in this paper how we can plan for a semi-realistic miniature version of an industrial process, the LEGO[1] car factory [Strömberg, 1991]. This is a realistic miniature version of real industrial processes in many respects. Modelling this process as a tractable planning problems has thus been one of our primary goals. We show that by exploiting the inherent structure of the problem, we can model it and use a polynomial-time planning algorithm to solve it. More precisely, we use a simple and provably correct modification of our previously presented algorithm for the SAS$^+$-IAO problem [Jonsson and Bäckström, 1994]. The basic idea is to partition the problem so that one part fits in the SAS$^+$-IAO class and its solution becomes a plan skeleton that can be straightforwardly filled in with operators from the second part.

The rest of this paper is structured as follows. In Section 2 we describe the formalism employed and in Section 3 we introduce restrictions forming the SAS$^+$-IAO class of planning problems. The planning algorithm is described in Section 4. The LEGO car factory is described in Section 5 and we show how to model it using the SAS formalism. In Section 6 we apply the planning algorithm to the LEGO car factory. Section 7 contains the conclusions.

2 The SAS$^+$ Formalism

We use the SAS$^+$ planning formalism [Bäckström and Nebel, 1993; Jonsson and Bäckström, 1994] which can be viewed as a variation on the propositional version of the STRIPS formalism. The SAS$^+$ formalism is, in fact, equivalent, under polynomial reduction, to most other common variants of propositional STRIPS [Bäckström, 1995]. Yet, the formalisms have different modelling properties, making them conceptually different. Some problems are more naturally expressed in one of the formalisms than in the other. For instance, control engineers seem to find the SAS$^+$ formalism much more appealing than the STRIPS formalism.

First we define the SAS$^+$ planning problem.

Definition 2.1 An instance of the SAS$^+$ planning problem is given by a quadruple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ with components defined as follows:

[1] LEGO is a trademark of the LEGO company.

- $\mathcal{V} = \{v_1, \ldots, v_m\}$ is a set of *state variables*. Each variable $v \in \mathcal{V}$ has an associated *domain* of values $\mathcal{D}_v$, 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 $\langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle$, where $\mathsf{pre}, \mathsf{post}, \mathsf{prv} \in \mathcal{S}^+$ denote the *pre-, post-* and *prevail-condition*, respectively. $\mathcal{O}$ is subject to the following two restrictions

  (R1) for all $\langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle \in \mathcal{O}$ and $v \in \mathcal{V}$ if $\mathsf{pre}[v] \neq \mathsf{u}$, then $\mathsf{pre}[v] \neq \mathsf{post}[v] \neq \mathsf{u}$,

  (R2) for all $\langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle \in \mathcal{O}$ and $v \in \mathcal{V}$, $\mathsf{post}[v] = \mathsf{u}$ or $\mathsf{prv}[v] = \mathsf{u}$.

- $s_0 \in \mathcal{S}^+$ and $s_* \in \mathcal{S}^+$ denote the *initial* and *goal state*, respectively.

The prevail-condition can be thought of as the part of the pre-condition that is not changed by the operator. Restriction R1 essentially says that a state variable can never be made undefined, once made defined by some operator. Restriction R2 says that the post- and prevail-conditions of an operator can never define the same variable. We further write $s \sqsubseteq t$ if the state s is subsumed (or satisfied) by state t, *i.e.* if $s[v] = \mathsf{u}$ or $s[v] = t[v]$. We extend this notion to states, defining

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

For $o = \langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle$ is a SAS$^+$ operator, we write $\mathsf{pre}(o)$, $\mathsf{post}(o)$ and $\mathsf{prv}(o)$ to denote pre, post and prv respectively. A sequence $\langle o_1, \ldots, o_n \rangle \in Seqs(\mathcal{O})$ of operators is called a *SAS$^+$ plan* (or simply a plan) over Π.

A plan is a solution to a planning problem if it is executable in the initial state, and it leads to the goal state.

Definition 2.2 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. $n > 0$, $\mathsf{pre}(o_1) \sqsubseteq s$, $\mathsf{prv}(o_1) \sqsubseteq s$ and $\mathit{Valid}(\langle o_2, \ldots, o_n\rangle, (s \oplus \mathsf{post}(o_1)), t)$.

A plan $\langle o_1, \ldots, o_n\rangle \in \mathit{Seqs}(\mathcal{O})$ is a *solution* to Π iff $\mathit{Valid}(\langle o_1, \ldots, o_n\rangle, s_0, s_*)$. The function *Result* returns the state resulting from executing a plan, *i.e.*, $t = \mathit{Result}(\langle o_1, \ldots, o_n\rangle, s)$ if either $t \in \mathcal{S}$ and $Valid(\langle o_1, \ldots, o_n\rangle, s, t)$ or $t = \langle \mathsf{u} \ldots \mathsf{u}\rangle$.

3 Restrictions

To be able to design polynomial-time algorithms we must somehow restrict the problem class. This paper uses the IAO restriction [Jonsson and Bäckström, 1994] and we repeat its definition in this section. For the definitions below, let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ be a SAS$^+$ instance.

Definition 3.1 An operator $o \in \mathcal{O}$ is *unary* iff there is exactly one $v \in \mathcal{V}$ s.t. $\mathsf{post}(o)[v] \neq \mathsf{u}$.

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 3.2 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

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

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 3.3 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 $\mathsf{pre}(o)[v] = x$ and $\mathsf{post}(o)[v] = y \neq \mathsf{u}$. 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 3.4 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.

Definition 3.5 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. $\mathsf{prv}(o_k) \sqsubseteq \mathsf{prv}(o'_{i_k})$ for $1 \leq k \leq m$.

A SAS$^+$ instance fulfilling the restrictions above is denoted as a SAS$^+$-IAO instance. Whether a SAS$^+$ instane satisfies the IAO restriction or not can be tested in polynomial time. Further, we have previously presented a polynomial time plan generation algorithm for the SAS$^+$-IAO class [Jonsson and Bäckström, 1994] which is proven to be correct.

4 Planning algorithm

In this section we present a formally correct extension of the SAS$^+$-IAO algorithm [Jonsson and Bäckström, 1994], which is sufficient for modelling the LEGO car factory. The basic idea is to partition the original SAS$^+$ instance into two separate instances, both being SAS$^+$-IAO instances. This instance can then be solved in polynomial time and its solution constitutes a skeleton to be filled in by solving subproblems from the second instance. This process is referred to as *interweaving* and can be viewed as a restricted variant of the more general concept *refinement,* as used in hierarchical state abstraction [Knoblock, 1991]. In fact, the whole method we use can be viewed as a restricted variant of two-level state abstraction. However, while state abstraction is a general method which is not formally well understood—it can on some occasions speed up planning considerably [Knoblock, 1991] and on other occasions be disastrous [Bäckström and Jonsson, 1995]—our, more restricted method, is provably correct, guaranteed not to make things worse and runs in polynomial-time.

First we show how a SAS$^+$ problem instance can be restricted to take only a subset of the variables into account.

Definition 4.1 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance, $s \in \mathcal{S}^+$ and $\mathcal{V}' \subseteq \mathcal{V}$. Then, the *restrictions* for states and operators are defined as follows:

- $s \Cap \mathcal{V}' = \langle s[v_{i_1}], \ldots, s[v_{i_n}] \rangle$ where $\mathcal{V}' = \{v_{i_1}, \ldots v_{i_n}\}$
- Let $o \in \mathcal{O}$. Then, $o \Cap \mathcal{V}' = \langle \mathsf{pre}(o) \Cap \mathcal{V}', \mathsf{post}(o) \Cap \mathcal{V}', \mathsf{prv}(o) \Cap \mathcal{V}' \rangle$.
- $\mathcal{O} \Cap \mathcal{V}' = \{o \Cap \mathcal{V}' | o \in \mathcal{O}\}$.

The restriction $\Pi \Cap \mathcal{V}'$ can now be defined as $\Pi \Cap \mathcal{V}' = \langle \mathcal{V}', \mathcal{O} \Cap \mathcal{V}', s_0 \Cap \mathcal{V}', s_* \Cap \mathcal{V}' \rangle$.

This can be used to partition a SAS$^+$ problem instance into two independent problems based on a partition of $\mathcal{V}$ in two disjoint sets.

Definition 4.2 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance and let $\mathcal{V}_1$, $\mathcal{V}_2$ be disjoint subsets of $\mathcal{V}$. Then, $\mathcal{V}_1$ is *independent* of $\mathcal{V}_2$ iff

- every operator $o \in \mathcal{O}$ affecting some variable in $\mathcal{V}_1$ satisfies $\mathsf{pre}(o) \Cap \mathcal{V}_2 = \mathsf{post}(o) \Cap \mathcal{V}_2 = \mathsf{prv}(o) \Cap \mathcal{V}_2 = \langle \mathsf{u}, \ldots, \mathsf{u} \rangle$.
- every operator $o \in \mathcal{O}$ affecting some variable in $\mathcal{V}_2$ satisfies $\mathsf{pre}(o) \Cap \mathcal{V}_1 = \mathsf{post}(o) \Cap \mathcal{V}_1 = \langle \mathsf{u}, \ldots, \mathsf{u} \rangle$.

Reachability means that every problem instance can be solved regardless of the initial and goal states.

Definition 4.3 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance. Then Π is *reachable* if for any two states $s_1, s_2 \in \mathcal{S}$ the planning problem $\langle \mathcal{V}, \mathcal{O}, s_1, s_2 \rangle$ is solvable.

Thus, reachability is the same as a strongly connected state transition graph. Note that even if reachability only depends on $\mathcal{V}$ and $\mathcal{O}$ we say that Π is reachable for convenience.

We can now show that if a planning problem Π is split into two independent problems, then the original problem can be solved if the two new problem instances can be solved.

Theorem 4.4 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance and let $\mathcal{V}_1, \mathcal{V}_2$ be disjoint subsets of $\mathcal{V}$ such that $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$. If $\mathcal{V}_1$ is independent of $\mathcal{V}_2$, $\Pi \Cap \mathcal{V}_1$ is reachable and $\Pi \Cap \mathcal{V}_2$ is solvable then Π is solvable.

Proof (Sketch): Suppose there exists a plan $\omega = \langle o_1; \ldots; o_n \rangle$ solving $\Pi \Cap \mathcal{V}_2$. Let ω_0 be a plan solving the SAS$^+$ instance $\langle \mathcal{V}_1, \mathcal{O}, s_0, \mathsf{prv}(o_1) \rangle \Cap \mathcal{V}_1$. Define recursively ω_k, $1 \leq k \leq n-1$ such that ω_k is a solution to the instance

$$\langle \mathcal{V}_1, \mathcal{O}, \mathit{Result}((\omega_0; \ldots; \omega_{k-1}), s_0 \Cap \mathcal{V}_1), \mathsf{prv}(o_{k+1}) \rangle \Cap \mathcal{V}_1.$$

Finally, let ω_n be a solution to the instance

$$\langle \mathcal{V}_1, \mathcal{O}, \mathit{Result}((\omega_0; \ldots; \omega_{n-1}), s_0 \Cap \mathcal{V}_1), s_* \rangle \Cap \mathcal{V}_1.$$

Since $\Pi \Cap \mathcal{V}_1$ is reachable, $\omega_0, \ldots, \omega_n$ exists. Now, let us consider the plan $\omega' = (\omega_0; o_1; \omega_1; \ldots; \omega_{n-1}; o_n; \omega_n)$. By the construction of $\omega_0, \ldots, \omega_{n-1}$, all pre- and prevail-conditions of $o_1, \ldots, o_n$ are satisfied because $\omega = (o_1; \ldots; o_n)$ is a valid plan by assumption. Furthermore, all pre- and prevail-conditions of the operators in $(\omega_0; \ldots; \omega_n)$ are satisfied because $(\omega_0; \ldots; \omega_n)$ is a valid plan and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. It remains to show that $s_* \sqsubseteq \mathit{Result}(\omega', s_0)$. This follows immediately since we know that $(o_1; \ldots; o_n)$ is a valid plan for $\Pi \Cap \mathcal{V}_2$ and $(\omega_0; \ldots; \omega_n)$ is a valid plan for $\Pi \Cap \mathcal{V}_1$ and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$.

An algorithm that works as indicated in the proof of Theorem 4.4 is shown in Figure 1. The sets $\mathcal{V}_1$ and $\mathcal{V}_2$ are such that $\Pi \Cap \mathcal{V}_1$ and $\Pi \Cap \mathcal{V}_2$ satisfies the IAO restriction in the previous section, and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. The procedure *PlanIAO* solves the SAS$^+$-IAO planning problem [Jonsson and Bäckström, 1994].

It is obvious from the proof of Theorem 4.4 that the algorithm is correct, and since *PlanIAO* is polynomial, the resulting algorithm is polynomial.

5 The LEGO Car Factory

Our application example is an automated assembly line for LEGO cars [Strömberg, 1991], which is used for undergraduate laboratory sessions in

procedure $Plan(\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle)$;
$\langle o_1, \ldots, o_n \rangle \leftarrow PlanIAO(\langle \mathcal{V}_2, \mathcal{O}, s_0, s_* \rangle \Cap \mathcal{V}_2)$
$\omega_0 \leftarrow PlanIAO(\langle \mathcal{V}_1, \mathcal{O}, s_0, \mathsf{prv}(o_1) \rangle \Cap \mathcal{V}_1)$
for $k = 1, \ldots, n-1$ **do**
  $\omega_k \leftarrow PlanIAO(\langle \mathcal{V}_1, \mathcal{O}, Result(\langle \omega_0; \ldots; \omega_{k-1} \rangle, s_0 \Cap \mathcal{V}_1), \mathsf{prv}(o_{k+1}) \rangle \Cap \mathcal{V}_1)$
end for
$\omega_n \leftarrow PlanIAO(\langle \mathcal{V}_1, \mathcal{O}, Result(\langle \omega_0; \ldots; \omega_{n-1} \rangle, s_0 \Cap \mathcal{V}_1), s_* \rangle \Cap \mathcal{V}_1)$
return $(\omega_0; o_1; \omega_1; o_2; \ldots; \omega_{n-1}; o_n; \omega_n)$

Figure 1: Planning algorithm.

digital control at the Department of Electrical Engineering at Linköping University. The students are faced with the task of writing a program to control this assembly line using the graphical language Grafcet [IEC, 1988]. Grafcet is tailored to implementing industrial sequential control and it resembles Petri Nets.

The main operations for assembling a LEGO car are shown in Figure 2. The assembly line consists of two similar halves, the first mounting the chassis parts on the chassis (see Figure 3) and the second mounting the top (see Figure 4).

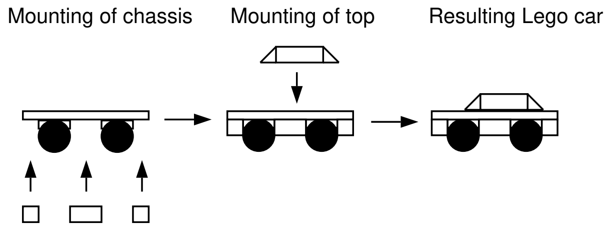

Figure 2: Assembling a LEGO car.

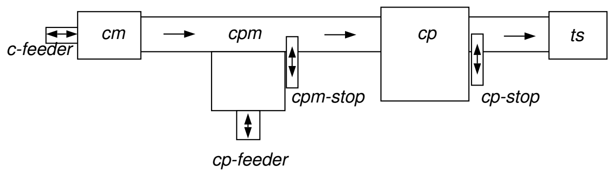

Figure 3: The first half of the LEGO car factory.

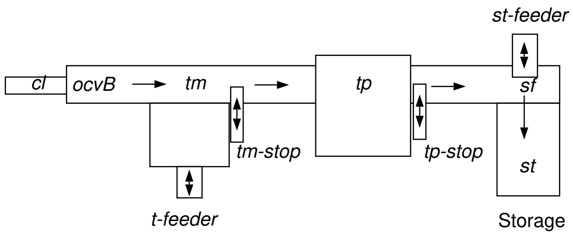

Figure 4: The second half of the LEGO car factory.

The first half of the LEGO car factory is presented in Figure 3. The chassis is initially stored upside down in the chassis magazine (*cm*). It enters the conveyor belt by using the chassis feeder (*c-feeder*), and is transported to the chassis parts magazine (*cpm*) where the chassis parts are fed onto the chassis using the chassis parts feeder (*cp-feeder*). The chassis is then transported to the chassis press (*cp*), where the chassis is pressed together. It is then transported to the turn station (*ts*) where the chassis is turned upright and enters the second half of the factory (Figure 4) where it is placed on the chassis lift (*cl*). It is lifted up, placed on the conveyor belt (*ocvB*) and transported to the top magazine (*tm*) where a top is fed onto the chassis by the top feeder (*t-feeder*). The chassis is then transported to the top press (*tp*) where the top is pressed tight onto the chassis. From there it is transported to the end of the conveyor belt (*sf*) and placed so that the storage feeder (*st-feeder*) can push the chassis into a buffer storage (*st*).

The conveyor belt used to transport the chassis runs continuously. Hence, stopper bars (*cpm-stop, cp-stop, tm-stop, tp-stop*) are pushed out in front of the chassis at the four work-stations *cpm, cp, tm* and *tp*, holding the car fixed, sliding on the belt (see Figure 3 and Figure 4).

Figure 5 shows one of the work-stations in more detail, namely the one where the top is put onto the chassis (*tm* in Figure 4). The chassis is held fixed at the top storage (A) by the stopper bar (B). The tops are stored in a pile and the feeder (C) is used to push out the lowermost top onto the chassis. When the top is on the chassis, the feeder is withdrawn and then the stopper bar is withdrawn, thus allowing the chassis to move on to the next work-station.

We continue by modelling the LEGO car factory as a SAS$^+$ instance. The state variables are shown in Table 1. The variable *pos* gives the position of the chassis, and the corresponding positions are given in Figure 3 and Figure 4. The stopper bars and the corresponding variable names are also

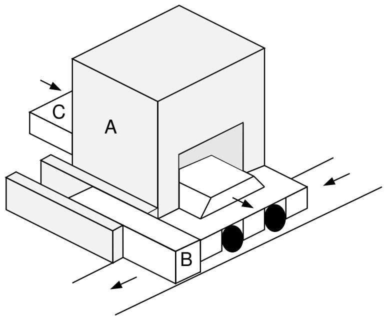

Figure 5: Putting the top onto the chassis.

variable	values
pos	cm, cpm, cp, ts, cl, ocvB, tm, tp, sf, st
turner	A, B
cp-status	off, on, pressed
t-status	off, on, pressed
c-status	prepared, not-prepared
cp-press	down, up
t-press	down, up
clift	down, up
c-feeder, cp-feeder, t-feeder, st-feeder	ext, rtr
cpm-stop, cp-stop, tm-stop, tp-stop	ext, rtr

Table 1: State variables $\mathcal{V}$ and their associated domains of values $\mathcal{D}_v$.

marked in these figures, as well as the variable names for the feeders. For the feeders and the stopper bars the value *ext* means that the feeder (or stopper bar) is extended, while *rtr* means that it is retracted. The variable *turner* tells if the turner (*ts* in Figure 3) is turned towards the first half of the factory (A) or towards the second half of the factory (B). The two variables *cp-status* and *t-status* denote the status of the chassis parts and the top, respectively, while the variable *c-status* denotes the status of the chassis and is mainly needed since we have no sensor detecting whether the chassis is just outside the chassis magazine. The other variables should be obvious from the table and the figures.

Using the variables defined in Table 1 we can define operators as in Tables 2 and 3. Additionally there are two operators for each feeder and each stopper bar for restracting and ectending the feeder or stopper bar. The operators corresponding to the chassis feeder are denoted *extend-c-feeder* and *retract-c-feeder*. The pre-condition is that *c-feeder* = *rtr*, the post-condition is that *c-feeder* = *ext* and there is no prevail-condition. The other operators corresponding to the feeders and stopper bars are denoted in a similar manner.

6 Planning for the LEGO car factory

We can now apply the planning algorithm in Figure 1 to the LEGO car factory. Using the variables defined in Table 1 we can define two subsets as follows:

$$\mathcal{V}_1 = \{\text{turner, cp-press, t-press, clift, c-feeder, cp-feeder, t-feeder, st-feeder, cpm-stop, cp-stop, tm-stop, tp-stop}\}$$

$$\mathcal{V}_2 = \{\text{pos, cp-status, t-status, c-status}\}$$

This results in that the operators represented in Table 3 together with all operators for extending or retracting feeders or stopper bars are the operators that affect variables in $\mathcal{V}_1$. The operators given in Table 2 are the operators that affect variables in $\mathcal{V}_2$, and these two sets define the partition of the problem.

It is easy to see that $\Pi \Cap \mathcal{V}_1$ and $\Pi \Cap \mathcal{V}_2$ satisfy the IAO restriction and that $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. The set $\mathcal{V}_1$ is such that for every $o \in \mathcal{O} \Cap \mathcal{V}_1$ and $v \in \mathcal{V}_1$ $\mathsf{prv}(o)[v] = \mathsf{u}$. Furthermore, for every $v \in \mathcal{V}_1$ there are operators for setting each value (for example there is one operator for extending the c-feeder and one operator for retracting it). Obviously $\Pi \Cap \mathcal{V}_1$ is reachable, and the algorithm given in Figure 1 can be applied.

Depending on how we choose the initial state and the goal state, we can plan for various cases. Here we show a plan for normal operation, *i.e.* the goal is to assemble a LEGO car. It is straightforward to modify this to plan for error recovery or for a before-execution unknown initial state. The goal state is that the chassis should be in the buffer storage (*pos* = *st*) and the top and chassis parts should be pressed onto the chassis (*cp-status* = *pressed* and *t-status* = *pressed*). All other state variables are undefined and can have any value. Suppose that the initial state is as follows. The chassis is placed in the chassis magazine (*pos* = *cm*, *c-status* = *not-prepared*), there are no chassis parts on the chassis (*cp-status* = *off*) and there is no top on the chassis (*t-status* = *off*). Furthermore the turner is turned towards the first

Operator	Pre	Post	Prevail
cm2cpm	pos = cm	pos = cpm	c-feeder = ext cp-feeder = rtr cpm-stop = ext
cpm2cp	pos = cpm	pos = cp	cpm-stop = rtr cp-stop = ext cp-press = up
cp2ts	pos = cp	pos = ts	turner = A cp-stop = rtr cp-press = up
ts2cl	pos = ts	pos = cl	turner = B clift = down
cl2ocvB	pos = cl	pos = ocvB	clift = up
ocvB2tm	pos = ocvB	pos = tm	tm-stop = ext t-feeder = rtr
tm2tp	pos = tm	pos = tp	tm-stop = rtr tp-stop = ext t-press = up
tp2sf	pos = tp	pos = sf	tp-stop = rtr st-feeder = rtr t-press = up
sf2st	pos = sf	pos = st	st-feeder = ext
prepare-ch	c-status = not-prep	c-status = prep	c-feeder = rtr
put-cp	cp-status = off	cp-status = on	pos = cpm cp-feeder = ext
press-cp	cp-status = on	cp-status = pressed	cp-press = down pos = cp
put-top	t-status = off	t-status = on	pos = tm t-feeder = ext
press-top	t-status = on	t-status = pressed	pos = tp t-press = down

Table 2: Operators with prevail-conditions.

Operator	Pre	Post	Prevail
A2B	turner = A	turner = B	-
B2A	turner = B	turner = A	-
cl-down	clift = up	clift = down	-
cl-up	clift = down	clift = up	-
cp-press-down	cp-press = up	cp-press = down	-
cp-press-up	cp-press = down	cp-press = up	-
t-press-down	t-press = up	t-press = down	-
t-press-up	t-press = down	t-press = up	-

Table 3: Operators without prevail-conditions.

half of the factory (*turner* = *A*), all feeders and stopper bars are retracted and the chassis press, the top press and the chassis lift are in their down position.

Applying the algorithm given in Figure 1 results in a plan as in Figure 6.

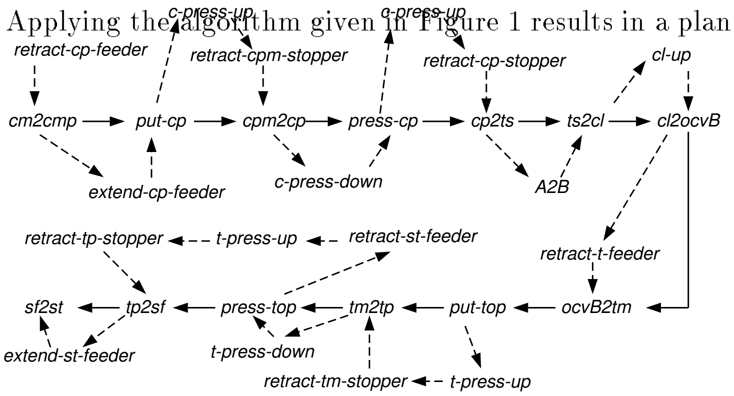

Figure 6: Resulting plan. The solid arrows are the output from the SAS$^+$-IAO algorithm solving $\Pi \Cap \mathcal{V}_2$, and the dashed arrows are the result from the interweaving process.

7 Conclusions

Tsatsoulis and Kashyap [Tsatsoulis and Kayshap, 1988] call planning "one of the most underused techniques of AI" in the context of manufacturing. They list a number of areas within industry where planning could be applied, but where no or very few attempts have been made at such applications.

We have applied our previous results on tractable planning to an application example in automatic control—an assembly line for LEGO cars. This example does not quite fit within the restriction of any previoulsy presented tractable planning class. The closest class is the SAS$^+$-IAO class, which is sufficient except for the requirement that there may be no cycles between requestable states. The assembly line does not have this property. This fact provided feedback for modifying the theory and we have presented an extended variant of the previous SAS$^+$-IAO class and its associated algorithm. Limited forms of such cycles are now allowed, which is sufficient for modelling the assembly line. The modified algorithm is also provably correct and runs in polynomial time.

We believe this limited form of cycle to be sufficient for modelling many other similar applications. However, the modification suggests generalizing the extension in this paper, allowing more complex cyclic structures, which is a topic for future research.

References

[AAAI-91, 1991] American Association for Artificial Intelligence. *Proceedings of the 9th (US) National Conference on Artificial Intelligence (AAAI-91)*, Anaheim, CA, USA, July 1991. AAAI Press/MIT Press.

[Åström and Benveniste, 1993] Karl-Johan Åström and Albert Benveniste. Meeting the challenge of computer science in the industrial applications of control: An introductory discussion to the special issue. *IEEE Transactions on Automatic Control*, 38(7):1004–1010, July 1993.

[Bäckström and Jonsson, 1995] Christer Bäckström and Peter Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In Chris Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montréal, PQ, Canada, August 1995. Morgan Kaufmann.

[Bäckström and Klein, 1991a] Christer Bäckström and Inger Klein. Parallel non-binary planning in polynomial time. In Reiter and Mylopoulos [1991], pages 268–273.

[Bäckström and Klein, 1991b] Christer Bäckström and Inger Klein. Planning in polynomial time: The SAS-PUBS class. *Computational Intelligence*, 7(3):181–197, August 1991.

[Bäckström and Nebel, 1993] Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1430–1435, Chambéry, France, August–September 1993. Morgan Kaufmann.

[Bäckström, 1992] Christer Bäckström. Equivalence and tractability results for SAS$^+$ planning. In Bill Swartout and Bernhard Nebel, editors, *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning (KR-92)*, pages 126–137, Cambridge, MA, USA, October 1992. Morgan Kaufmann.

[Bäckström, 1995] Christer Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1–2):17–34, 1995.

[Bylander, 1991] Tom Bylander. Complexity results for planning. In Reiter and Mylopoulos [1991], pages 274–279.

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Chenoweth, 1991] Stephen V. Chenoweth. On the NP-hardness of blocks world. In AAAI-91 [1991], pages 623–628.

[Erol *et al.*, 1992a] Kutluhan Erol, Dana S Nau, and V S Subrahmanian. On the complexity of domain-independent planning. In *Proceedings of the 10th (US) National Conference on Artificial Intelligence (AAAI-92)*, pages 381–386, San José, CA, USA, July 1992. American Association for Artificial Intelligence.

[Erol *et al.*, 1992b] Kutluhan Erol, Dana S Nau, and V S Subrahmanian. When is planning decidable? In James Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the 1st International Conference (AIPS'92)*, pages 222–227, College Park, MD, USA, June 1992. Morgan Kaufmann.

[Gupta and Nau, 1992] Naresh Gupta and Dana S Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56:223–254, 1992.

[IEC, 1988] IEC. Preparation of function charts for control systems. Publication 848:1988, IEC, 1988.

[Jonsson and Bäckström, 1994] Peter Jonsson and Christer Bäckström. Tractable planning with state variables by exploiting structural restrictions. In *Proceedings of the 12th (US) National Conference on Artificial Intelligence (AAAI-94)*, pages 998–1003, Seattle, WA, USA, July–August 1994. American Association for Artificial Intelligence.

[Klein and Bäckström, 1991] Inger Klein and Christer Bäckström. On the planning problem in sequential control. In *Proceedings of the 30th IEEE Conference on Decision and Control (CDC-91)*, pages 1819–1823, Brighton, UK, December 1991.

[Knoblock, 1991] Craig A Knoblock. Search reduction in hierarchical problem solving. In AAAI-91 [1991], pages 686–691.

[Passino and Antsaklis, 1989] K M Passino and P J Antsaklis. A system and control theoretic perspective on artificial intelligence planning systems. *Applied Artificial Intelligence*, 3(1):1–32, 1989.

[Reiter and Mylopoulos, 1991] Ray Reiter and John Mylopoulos, editors. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, August 1991. Morgan Kaufmann.

[Strömberg, 1991] Jan-Erik Strömberg. Styrning av LEGO-bilfabrik. Andra omarbetade upplagan. Department of Electrical Engineering, Linköping University, 1991.

[Tsatsoulis and Kayshap, 1988] Costas Tsatsoulis and R L Kayshap. Planning and its application to manufacturing. In Soundar T Kumara, Rangasami L Kashyap, and Allen L Soyster, editors, *Artificial Intelligence, Manufacturing Theory and Practice*, chapter 7, pages 193–223. Institute of Industrial Engineers, 1988.