In C. Bäckström and E. Sandewall (eds.), *Current Trends in AI Planning: EWSP'93—2nd European Workshop on Planning*, Vadstena, Sweden, Dec. 1993. IOS Press, 1994.

Finding Least Constrained Plans and Optimal Parallel Executions is Harder than We Thought

Christer Bäckström
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
email: cba@ida.liu.se

Abstract. It seems to have been generally assumed in the planning community that it is easy to compute a *least-constrained* partially ordered version of a total-order plan. However, it is not clear what this concept means. Five candidates for this criterion are defined in this paper, and it turns out that the only ones giving some reasonable optimality guarantee are NP-hard to compute. A related problem is to find a shortest parallel execution of a plan, also proven NP-hard. Algorithms can be found in the literature which are claimed to solve these problems optimally in polynomial time. However, according to the NP-hardness of the problems, this is impossible unless P=NP, and it is explained in this paper why the algorithms fail. The algorithms are, instead, reconsidered as approximation algorithms, but it is shown that neither algorithm gives any constant performance guarantee. This is not surprising, however, since both problems turn out not to be approximable within a constant ratio.

1. Introduction

A *total-order plan* (linear plan) is a totally ordered set of actions, that is, a sequence of actions, where the order is the intended execution order. A *partial-order plan* (non-linear plan), on the other hand, is a partially ordered set of actions; that is, some actions may be left unordered with respect to each other. That two actions are unordered does not imply that they can be executed in parallel—it only means that they can be executed in either order. The main advantage of a partial-order plan over a total-order one is that the former can be post-processed by a scheduler subject to additional information. For instance, the final execution order of two unordered actions may be chosen such that a deadline can be met and some of the unordered actions may be executable in parallel, thus decreasing the total execution time of the plan.

A *total-order planner* is a planner that maintains and works on a total-order plan, by modifying or extending it. Such a planner is *over-committed* in the sense that it mostly introduces stronger ordering constraints than is necessary. The early AI planners were total-order planners, *eg.* STRIPS [6]. A *partial-order planner* maintains and works on a partial-order plan, thus being less committed to ordering the actions until necessary, typically leaving many actions unordered in the final plan. Partial-order planning was first introduced in NOAH [22], with the motivation that less commitment would lead to less backtracking and, thus, efficiency gains. Most planners since NOAH have been partial-order planners and it has generally been assumed that partial-order planning is more efficient than total-order planning. For simple action languages this assumption seems to hold, in principle, [16, 17], although choosing a good commitment strategy for partial-order planners is non-trivial [4, 13, 15, 16, 17]. It has further been argued that partial-order planning becomes very hard if introducing context-dependent effects, in some cases leading to a revival of total-order planning.

At least two examples [20, 25] can be found in the literature where this argument motivates using a total-order planner and then convert the resulting plan into a partial-order plan—the purpose being to exploit possible parallelism according to some further criteria. In both these cases algorithms are presented for converting a total-order plan into a partial-order plan, claimed to be a *least constrained plan* or allowing a *shortest parallel execution* respectively. However, although often used in the literature, it is unclear what the term least constrained plan means; some candidate criteria for this concept are, thus, defined and analysed in this paper. Further, a concept of optimal parallel execution of plans is defined. The complexity results for these problems are disappointing, under reasonable optimality criteria, they are NP-hard. In the light of this, the algorithms from the literature, mentioned above, are analysed wrt. to these criteria. Not surprisingly, they are found not to guarantee optimality wrt. to these criteria. Hence, the algorithms are reconsidered as approximation algorithms for the same problems, unfortunately also with disappointing results. However, this is not only due to problems with the algorithms themselves, since it is also shown that both problems are intrinsically hard to approximate.

2. Basic Definitions

This section defines plans and related concepts. In order to make the definitions and results as general and formalism independent as possible, only a minimum of assumptions about the underlying formalism will be made. Any planning formalism may be used that defines some concept of a *planning problem*, a domain of entities called *actions* and a *validity* test. It will be assumed that the planning problem consists of *planning problem instances (ppi's)*[1], with no further assumptions about the inner structure of these. Given a ppi. Π and a sequence of actions $\langle a_1, \ldots, a_n \rangle$, the validity test may be true or false. The intuition behind the validity test is that if the test is true, then the action sequence $\langle a_1, \ldots, a_n \rangle$ *solves* Π. The inner structure of the ppi's and the exact definition of the validity test are, of course, crucial for any specific planning formalism. However, for the results to be proven in this paper it is sufficient to make assumptions only about the computational complexity of the validity test. For instance, some results will depend on whether this test is tractable or not. Based on these concepts, the notion of plans can be defined in the usual way.

Definition 2.1 *A* total-order plan (t.o. plan) *is a sequence $P = \langle a_1, \ldots, a_n \rangle$ of actions, which can alternatively be denoted by the tuple $\langle \{a_1, \ldots, a_n\}, \prec \rangle$ where for $1 \leq k, l \leq n$,*

[1]This is the complexity theoretic terminology for problems. What is called an instance of a planning problem in this paper is often called a planning problem in the planning literature.

$a_k \prec a_l$ *iff* $k < l$. *Given a ppi.* Π*,* P *is said to be* Π-valid *iff the validity test is true for* Π *and* P*.*

A partial-order plan (p.o. plan) *is a tuple* $P = \langle A, \prec \rangle$ *where* A *is a set of actions and* $\prec$ *is a strict (ie. irreflexive) partial order on* A*. A t.o. plan* $\langle A', \prec' \rangle$ *is a* linearization *of* P *iff* $A' = A$ *and* $\prec \subseteq \prec'$*,* ie. $\prec'$ *is a total order on* A *obeying the partial order* $\prec$*. The validity test is extended to p.o. plans s.t. given a ppi.* Π*,* P *is* Π*-valid iff all linearizations of* P *are* Π*-valid.*

The actions of a t.o. plan must be executed in the specified order, while unordered actions in a p.o. plan may be executed in either order. That is, a p.o. plan can be viewed as a compact representation for a set of t.o. plans. There is no implicit assumption that unordered actions can be executed in parallel; parallel plans will be defined in Section 4. P.o. plans will be viewed as *directed acyclic graphs* in figures with the transitive arcs tacitly omitted to enhance readability.

The framework outlined above is very general and abstract and will be used mainly for definitions and tractability results, in order to make these as generally applicable as possible. Otherwise, the propositional STRIPS framework, sometimes further restricted, will be used, especially to make examples simple and to make hardness results as strong as possible. The usual propositional STRIPS model (see for instance Bylander [3]) will be used where actions have a precondition, an add-list and a delete-list. In figures, actions will be shown as boxes. Atoms to the left of an action box are members of its precondition and atoms to the right are members of its add-list (or the delete-list if negated). Furthermore, all problems considered in this paper concern reordering the actions in plans, not changing the set of actions Hence, a STRIPS ppi. will be written as a tuple $\langle I, G \rangle$ where I is the initial state and G is the goal state, tacitly omitting the implicit and, in this context irrelevant, atom and operator sets.

3. Least Constrained Plans

It seems to have been generally assumed in the planning community that there is no difference between t.o. plans and p.o. plans in the sense that a t.o. plan can easily be converted into a p.o. plan and *vice versa*. However, while a p.o. plan can be trivially converted into a t.o. plan in low-order polynomial time by topological sorting, it is less obvious that also the converse holds. At least two polynomial-time algorithms for converting t.o. plans into p.o. plans have been presented in the literature [20, 25] (both these algorithms will be analyzed later in this paper). The claim that a t.o. plan can easily be converted into a p.o. plan is of course, vacuously true since any t.o. plan is also a p.o. plan, by definition. Hence, no computation at all needs to be done. This is hardly what the algorithms were intended to compute, however. In order to be useful, such an algorithm must output a p.o. plan satisfying some interesting criterion, ideally some optimality criterion. In fact, both algorithms mentioned above are claimed to produce optimal plans according to certain criteria. For instance, Veloso *et al.* claim [25, p. 207] their algorithm to produce *least constrained* plans. They do not define what they mean by this term, however, and theirs is not the only paper in the literature using this term without further definition.

Unfortunately, it is by no means obvious what constitutes an intuitive or good criterion for when a p.o. plan is least constrained and, to some extent, this also depends on the purpose of achieving least-constrainment. The major motivation for producing p.o. plans instead of t.o. plans (see for instance Tate [24]) is that a p.o. plan can be post-processed by a scheduler according to further criteria, such as release times and deadlines or resource limits. Either the actions are ordered into an (ideally) optimal sequence or, given criteria for parallel execution, into a parallel plan that can be executed faster than if the actions were executed in sequence. In both cases, the less constrained the original plan is, the greater is the chance of arriving at an optimal schedule or optimal parallel execution respectively. Both of the algorithms mentioned above are motivated by the goal of exploiting possible parallelism to decrease execution time.

There is, naturally, an infinitude of possible definitions of least-constrainment. Some seem more reasonable than others, however. Five intuitively reasonable candidates are defined and analyzed below. Although other definitions are possible, it is questionable whether considerably better definitions, with respect to the purposes mentioned above, can be defined without using more information than is usually present in a t.o. or p.o. plan.

Given a p.o. plan $P = \langle A, \prec \rangle$, let the relation $?$ be implicitly defined s.t. for distinct $a, b \in A$, $a?b$ iff neither $a \prec b$ nor $b \prec a$. Further, let the function *length* be defined s.t. $length(P)$ is the number of actions in the longest action chain in P. The notation $P \sqsubseteq^{\Pi} P'$, will be used to denote that a p.o. plan P is less constrained than another p.o. plan P' wrt. a ppi. Π. In particular, the following five definitions of least constrainment will be considered.

Definition 3.1 *Given a ppi.* Π *and two* Π*-valid p.o. plans* $P = \langle A, \prec \rangle$ *and* $P' = \langle A', \prec' \rangle$*:*

LC1: $P \sqsubseteq_1^{\Pi} P'$ *iff* $A = A'$ *and* $\prec \subseteq \prec'$*,*

LC2: $P \sqsubseteq_2^{\Pi} P'$ *iff* $A = A'$ *and* $| \prec | \leq | \prec' |$*,*

LC3: $P \sqsubseteq_3^{\Pi} P'$ *iff* $A = A'$ *and* $?' \subseteq ?$*,*

LC4: $P \sqsubseteq_4^{\Pi} P'$ *iff* $A = A'$ *and* $|?'| \leq |?|$*,*

LC5: $P \sqsubseteq_5^{\Pi} P'$ *iff* $A = A'$ *and* $length(P) \leq length(P')$*.*

Obviously, the criteria LC3 and LC4 are redundant since $P \sqsubseteq_1^{\Pi} P'$ iff $P \sqsubseteq_3^{\Pi} P'$ and $P \sqsubseteq_2^{\Pi} P'$ iff $P \sqsubseteq_4^{\Pi} P'$. Furthermore, $\sqsubseteq_2$ is likely to be a reasonable substitute for $\sqsubseteq_5$ since when computing $\sqsubseteq_2$ the 'cost' of chains will be quadratic in their length and a long chain can, thus, dominate over many shorter chains. Hence, this paper will focus on least constrainment defined as minimality wrt. either LC1 or LC2.

Definition 3.2 *Given a ppi.* Π *and a* Π*-valid p.o. plan* $P = \langle A, \prec \rangle$*,*

1. *P is $\sqsubseteq_1^{\Pi,A}$-minimal iff there is no other Π-valid plan $P' = \langle A, \prec' \rangle$ s.t. $P' \sqsubseteq_1^{\Pi} P$;*

2. *P is $\sqsubseteq_2^{\Pi,A}$-minimal iff $P \sqsubseteq_2^{\Pi} P'$ for all Π-valid p.o. plans $P' = \langle A, \prec' \rangle$.*

The definitions for $\sqsubseteq_1$-minimality and $\sqsubseteq_2$-minimality look different since $\sqsubseteq_1$ is a (reflexive) partial order while $\sqsubseteq_2$ is a pre-order. It should also be noted that the $\sqsubseteq_1^{\Pi}$-minimal plans for some plan P are incomparable wrt. $\sqsubseteq_1$, while all $\sqsubseteq_2$-minimal plans form an equivalence class under $\sqsubseteq_2$. Furthermore, $\sqsubseteq_2$-minimality is a stronger criterion than $\sqsubseteq_1$-minimality, but is harder to compute.

Theorem 3.3 *$\sqsubseteq_2$-minimality implies $\sqsubseteq_1$-minimality, but not vice versa.*

Proof: Trivial. $\square$

Theorem 3.4 *If validity can be tested in polynomial time for p.o. plans, then any Π-valid p.o. plan $\langle A, \prec \rangle$ for some ppi. Π can be converted into a $\sqsubseteq_1^{\Pi,A}$-minimal Π-valid p.o. plan $\langle A, \prec' \rangle$ in polynomial time.*

Proof sketch: Given a p.o. plan $\langle A, \prec \rangle$, repeatedly select some non-transitive 'arc' in $\prec$ that can be removed without making the plan invalid and remove this arc. Terminate when no more such arc exists in $\prec$. $\square$

Definition 3.5 *The problem LC2-MINIMAL PLAN (LC2MP) is defined as follows: Given a ppi. Π, a Π-valid p.o. plan $\langle A, \prec \rangle$ and a positive integer k, decide whether there exists some Π-valid p.o. plan $P' = \langle A, \prec' \rangle$ s.t. $| \prec' | \leq k$.*

Theorem 3.6 *The problem LC2-MINIMAL PLAN is NP-hard if the action language is at least as expressive as propositional STRIPS with empty delete lists and it is further NP-complete if also p.o. plans can be tested for validity in polynomial time.*

Proof: Proof by transformation from the MINIMUM COVER problem [7, p. 222], which is NP-complete. Let $S = \{p_1, \ldots, p_n\}$ be a set of atoms, $C = \{C_1, \ldots, C_m\}$ a set of subsets of S and $k \leq |C|$ a positive integer. A *cover* of size k for S is a subset $C' \subseteq C$ s.t. $|C'| \leq k$ and $S \subseteq \cup_{T \in C'} T$. Using propositional STRIPS, construct, in polynomial time, the ppi. $\Pi = \langle \varnothing, \{r\} \rangle$ and the Π-valid t.o. plan $P = \langle a_1, \ldots, a_m, a_S \rangle$ where $addlist(a_k) = C_k$ for $1 \leq k \leq m$, $precond(a_S) = S$ and $addlist(a_S) = \{r\}$. Obviously, S has a minimum cover of size k iff there exists some Π-valid p.o. plan $P' = \langle \{a_1, \ldots, a_m, a_S\}, \prec \rangle$ s.t. $| \prec | = k$. Membership in NP is immediate in the case where p.o. plans can be tested for validity in polynomial time. $\square$

It follows immediately that the corresponding search problem, that is, the problem of *generating* an $\sqsubseteq_2^{\Pi,A}$-minimal plan is also NP-hard (and even NP-equivalent [7] if validity testing is tractable).

4. Parallel Plans

Most of the results in this paper are about or are motivated by the problem of finding efficient parallel executions of plans. Hence, the concept of parallel plans must be added to the previous planning framework.

Definition 4.1 *A* parallel t.o. plan *is a sequence $P = \langle S_1, \ldots, S_n \rangle$, where $S_1, \ldots, S_n$ are disjoint sets of actions. For $1 \leq k \leq n$, the set S_k is referred to as the kth* slice *of P. A* linearization *of P is a t.o. plan $P' = \alpha_1;\ldots;\alpha_n$, where the symbol ; denotes sequence concatenation and for $1 \leq k \leq n$, α_k is a permutation sequence of S_k. There is assumed to exist a validity test also for parallel t.o. plans with the restriction that if a parallel t.o. plan P is Π-valid for some ppi. Π, then all linearizations of P are also Π-valid.*

The intuition behind the definition of parallel t.o. plans is that the actions within a slice can be executed yielding the same result independently of whether the actions are executed in sequence, overlapping or in any other temporal order (more precisely, the intervals occupied by the actions may be related by any set of relations in Allen's interval algebra [2]).

Definition 4.2 *A* parallel p.o. plan *is a triple $P = \langle A, \prec, \# \rangle$, where $\langle A, \prec \rangle$ is a p.o. plan and $\#$ is an irreflexive, symmetric relation on A. A parallel t.o. plan $P' = \langle S_1, \ldots, S_n \rangle$ is an* n-step parallel execution *of P iff $S_1, \ldots, S_n$ is a partitioning of A and for any two actions $a \in S_k$ and $b \in S_l$, where $1 \leq k, l \leq n$, if $a \prec b$, then $k \leq l$ and if $a \# b$, then $k \neq l$. P' is further an* optimal parallel execution *of P if it is a parallel execution of P and P has no parallel execution in less than n steps. The validity test is extended to parallel p.o. plans s.t. given a ppi. Π, P is Π-valid iff all parallel executions of P are Π-valid.*

Intuitively, a parallel p.o. plan is a p.o. plan extended with an extra relation, $\#$, (a *non-concurrency* relation) expressing which of the unordered actions must not be executed in parallel, *ie.* must not belong to the same slice in any parallel execution. There are no restrictions on the relation $\#$ in addition to those in Definition 4.2. For instance, $a \prec b$ does not imply that $a \# b$.

This framework for parallel plans admits expressing possible parallelism only; necessary parallelism is out of the scope of this paper and requires a planner having access to and being able to make use of additional information. It will be further assumed that all actions take unit time and that all actions within a slice are executed perfectly in parallel, that is, each slice takes unit time. Since this is a special case of more general parallelism, the hardness results in this paper carry over implicitly to more general formalisms.

An interesting computation for a parallel plan is to find its shortest parallel execution. If the ultimate goal for finding a least-constrained plan is to exploit possible parallelism to minimize the execution time, then it is better to find the plan admitting the shortest parallel execution rather than the least-constrained plan. This is the only way to guarantee optimality, but it requires access to the additional information supplied by the non-concurrency relation.

Definition 4.3 *Given a ppi. Π, a parallel p.o. plan $P = \langle A, \prec, \# \rangle$ is* Π-reordering *of a parallel p.o. plan $P' = \langle A', \prec', \#' \rangle$ iff $A = A'$, $\# = \#'$ and both P and P' are Π-valid. P is further an n-*step Π-reordering *of P' if it also admits a parallel execution in n steps and it is an* optimal Π-reordering *of P' if no other Π-reordering of P' admits a parallel execution in fewer steps.*

Definition 4.4 *The decision problem OPTIMAL PARALLEL PLAN (OPP) is defined as follows. Given a ppi. Π, a parallel p.o. plan P and an integer k, decide whether P has a k-step Π-reordering.*

Theorem 4.5 *The problem OPTIMAL PARALLEL PLAN is NP-hard and it is further NP-complete if parallel t.o. plans can be tested in polynomial time.*

Proof: Hardness is proven by transformation from the GRAPH K-COLOURABILITY problem (GC) [7, p. 191], which is NP-complete. Let $G = \langle V, E \rangle$ be an arbitrary undirected graph. Construct, in polynomial time,[2] the parallel plan $P = \langle V, \varnothing, E\rangle$ that is valid for some ppi. Π. It is obvious that G is k-colourable iff P has a k-step Π-reordering since each colour of G will correspond to a unique slice in the parallel execution of P (note that $\langle V, \prec', E\rangle$ could not have a shorter execution than P for any order $\prec'$). It remains to prove membership in NP in the case parallel t.o. plans can be validated in polynomial time. Given a ppi. Π and a Π-valid parallel p.o. plan $P = \langle A, \prec, \#\rangle$, P has a k-step Π-reordering iff there exists a Π-valid parallel t.o. plan $\langle S_1 \cup \ldots S_k\rangle$ satisfying the # relation and having $S_1 \cup \ldots S_k = A$. Since such a plan can be guessed and validated in polynomial time, it follows that OPP is in NP. $\Box$

5. Analysis of the VPC Algorithm

Veloso *et al.* [25] have presented an algorithm (called VPC below) for converting t.o. plans into p.o. plans. The algorithm is used in the following context: first a total-order planner is used to produce a t.o. plan, then the VPC algorithm converts this plan into a p.o. plan and, finally, the resulting p.o. plan is analyzed to determine which actions can be executed in parallel. The action language used is a STRIPS-style language allowing quantifiers and context-dependent effects. However, the plans input to the VPC algorithm must be fully instantiated and they can hardly be allowed to contain context-dependent effects either,[3] thus being propositional STRIPS plans. The VPC algorithm is presented in Figure 1.[4]

VPC is a greedy algorithm which constructs an entirely new partial order by analysing the action conditions, using the original total order only to guide the greedy strategy. The algorithm is claimed [25, p. 207] to produce a least-constrained p.o. plan, although

[2]It is assumed that the formalism at hand use reasonable encoding schemes [7].

[3]At least, there is nothing in the paper or the algorithm suggesting that context-dependent effects can be handled by VPC. Furthermore, it is questionable whether the algorithm would still be polynomial, were this the case; plan validation is NP-hard for conditional plans in the general case [5, 18, 19].

[4]Veloso (personal communication, Oct. 1993) has pointed out that the published version of the VPC algorithm is incorrect and that a corrected version exists. This does not affect the problem analysed in this paper, however.

```
**procedure** VPC;
Input: a ppi. $\Pi$ and a $\Pi$-valid t.o. plan $\langle a_1, \ldots, a_n\rangle$
Output: A $\Pi$-valid p.o. plan
**for** $1 \leq i \leq n$ **do**
  **for** $p \in \mathit{precond}(a_i)$ **do**
    Find max $k < i$ s.t. $p \in \mathit{addlist}(a_k)$;
    **if** such a $k$ exists **then**
      order $a_k \prec a_i$;
    **for** $p \in \mathit{deletelist}(a_i)$ **do**
      **for** $1 \leq k < n$ s.t. $p \in \mathit{precond}(a_k)$ **do**
        Order $a_k \prec a_i$;
  **return** $\langle\{a_1, \ldots, a_n\}, \prec\rangle$;
```

Figure 1: The VPC algorithm

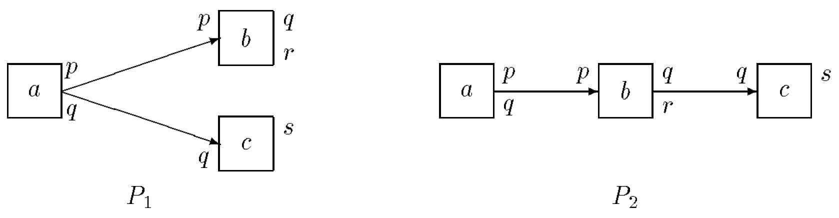

Figure 2: The p.o. plans in the failure example for VPC.

no definition is given of what this means.[5] In the absence of such a definition from its authors, the algorithm will be analysed with respect to $\sqsubseteq_1$-minimality and $\sqsubseteq_2$-minimality. It is then immediate from Theorem 3.6 that VPC cannot produce $\sqsubseteq_2$-minimal plans, unless P=NP. What is more surprising is that it does not even produce $\sqsubseteq_1$-minimal plans although this is a trivial tractable problem.

To see this, consider the following propositional STRIPS example. Suppose a total-order planner is given the ppi. $\Pi = \langle \varnothing, \{r, s\}\rangle$ as input. It may then return either of the Π-valid t.o. plans $\langle a, b, c\rangle$ and $\langle a, c, b\rangle$, with action conditions as shown in Figure 2. If given as input to the VPC algorithm, the two t.o. plans will give quite different results. VPC will convert the plan $\langle a, c, b\rangle$ into the p.o. plan P_1 in Figure 2, a plan which is both $\sqsubseteq_1$-minimal and $\sqsubseteq_2$-minimal. If instead given the plan $\langle a, b, c\rangle$, it will output the p.o. plan P_2 in Figure 2, a plan which is neither $\sqsubseteq_1$-minimal nor $\sqsubseteq_2$-minimal (note that transitive arcs are not shown in the figure). The reason that VPC is not guaranteed to find $\sqsubseteq_1$-minimal plans is that it is greedy; whenever it needs to find an operator a achieveing an effect required by the precondition of another operator b it chooses the last such action ordered before b in the input t.o. plan. However, there may be other actions having the same effect and being a better choice.

6. Analysis of the RF Algorithm

Based on the motivation that it is easier to generate a t.o. plan than a p.o. plan when using complex action representations, Regnier and Fade [20, 21] have presented an algorithm (called RF below) for converting a t.o. plan into a p.o. plan. The resulting plan has the property that all unordered actions can be executed in parallel. The authors of the algorithm further claim that the algorithm finds all pairs of actions that can be executed in parallel and, hence, the plan can be post-processed to find an optimal parallel execution. The RF algorithm is presented in Figure 3, with all implementation-dependent details irrelevant to analyzing the algorithm removed. The original algorithm returns a p.o. plan $\langle A, \prec\rangle$, but since this plan has the property that all unordered actions can be executed in parallel it corresponds to the parallel p.o. plan $\langle A, \prec, \varnothing\rangle$.

It seems that the RF algorithm is intended to compute a restriction of the problem OPTIMAL PARALLEL PLAN, with the non-concurrency relation implicitly defined by the predicate *parallelizable*. This predicate determines whether two actions may be

[5]Veloso (personal communication, Oct. 1993) has confirmed that the term “least constrained plan” was used in a “loose sense” and no optimality claim was intended. However, if this term is not defined, then we do not even know what problem the algorithm is supposed to solve.

```
1 **procedure** RF;
2 Input: a ppi. $\Pi$ and a $\Pi$-valid t.o. plan $\langle a_1, \ldots, a_n \rangle$
3 Output: A $\Pi$-valid parallel p.o. plan
4 **for** $1 \le i < j \le n$ **do**
5   **if** not $parallelizable(a_i, a_j)$ **then**
6     Order $a_i \prec a_j$;
7 **return** $\langle A, \prec, \varnothing \rangle$;
```

Figure 3: The RF algorithm

executed in parallel or not, that is, whether they interfere or not. Regnier and Fade use STRIPS operators where the conditions are sets of possibly negated object-attribute-value triples. For the results in this paper it is sufficient to consider the special case of ground triples, *ie.* propositional literals. Two sets S and T of atoms are *contradictory* iff there exists some atom p s.t. $p \in S$ and $\neg p \in T$. Regnier and Fade defined the *parallelizability* criterion to be true of two actions a_i, a_j in a t.o. plan $\langle a_1, \ldots, a_n \rangle$ iff

1. both $pre(a_i)$ and $pre(a_j)$ hold in the state preceding the execution of a_i and a_j,

2. all the pairs $(pre(a_i), pre(a_j))$; $(pre(a_i), post(a_j))$; $(post(a_i), pre(a_j))$ and $(post(a_i), post(a_j))$ are non-contradictory,

3. if $i < j + 2$, then for some $i \le k < j$, a_k and a_{k+1} are parallelizable.

Unfortunately, there is a serious problem with this definition, *viz.* that, because of the third requirement, it can only be applied to t.o. plans. Hence, it is not possible to use this criterion to decide whether two actions in a p.o. plan can be executed in parallel or not. This implies that the parallelizability criterion cannot be used to define a non-concurrency relation. It is, thus, impossible to decide whether a parallel p.o. plan is an optimal reordering according to this criterion. The claim that the RF algorithm finds all pairs of parallel actions is, thus, somewhat questionable since it can only refer to the actions that are parallelizable according to a criterion on the original t.o. plan; it is not possible to verify that no ordered actions in the output plan can be executed in parallel, which seems a more reasonable interpretation of the claim.

The third condition in the parallelizability criterion is not really necessary for deciding whether two actions can be executed in parallel, however. Conditions one and two together coincide with Horz' [9] *independence criterion*, which is intended to express when two actions in a p.o. plan can be executed in parallel. This criterion is trivially extendible to sets of unordered actions. Furthermore, any two unordered actions in a parallel p.o. plan output by the RF algorithm are independent. It thus follows that the parallelizability criterion is stronger than independence, but is only applicable to t.o. plans. Even for the purpose of deciding whether two actions in a t.o. plan can be executed in parallel or not the independence criterion seems the more appropriate of the two; the parallelizability criterion is simply over-restrictive.

In the plan input to the RF algorithm, all pairs of actions are ordered since it is a t.o. plan. It is easy to see that the algorithm can only remove orderings. That is, it can make two previously ordered actions unordered if these are parallelizable. However, it cannot reorder actions, something which is necessary for producing an optimal reordering, as will be seen in the following example.

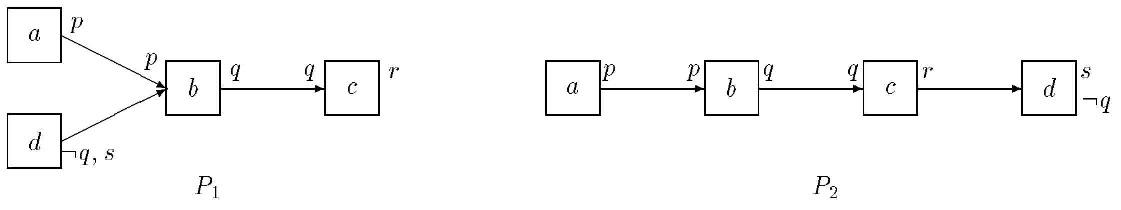

Figure 4: Plans in the RF failure example (the relation # is not shown).

Let the STRIPS ppi. $\Pi = \langle \varnothing, \{r, s\}\rangle$ be given. A total-order planner may return either of the three Π-valid t.o. plans, $\langle a, b, c, d\rangle$, $\langle a, d, b, c\rangle$ and $\langle d, a, b, c\rangle$ with action conditions as shown in Figure 4. A total-order planner may output either of these plans. If RF is given either of the two latter plans as input, it will produce the parallel p.o. plan P_1 in Figure 4 since actions a and d are parallelizable. On the other hand, if given the first plan as input, then it will return this plan unchanged, that is, the plan P_2 in Figure 4. This is because neither of the action pairs (a, b); (b, c) and (c, d) is parallelizable and, hence, they remain in the same order as in the input plan. P_1 admits a 3-step parallel execution and is an optimal reordering of each of the possible input plans. P_2, on the other hand, admits no shorter execution than 4 steps, and is thus not an optimal reordering of any of the input plans. The reason that the algorithm fails to produce an optimal plan in the second case is that it never considers reordering action d wrt the actions b and c. Since d is ordered after b and c in the input, it can either remain ordered after these or be unordered wrt to them in the output; the algorithm can never order d before b and c, unless this was already the case in the input.

It is obvious from this example that the RF algorithm is intrinsically dependent on in which order the actions appear in the input plan. This is a problem with the algorithm, however, not with the parallelizability criterion, since it would give the same result if using the independence criterion instead. Nevertheless, there are important differences between the two criteria. Any algorithm using the parallelizability criterion would return the plan P_2, while there can exist algorithms (not necessarily polynomial ones) using the independence criterion which find the plan P_1 on input $\langle a, b, c, d\rangle$. Further, P_1 in Figure 4 is an optimal parallel version of the t.o. plan $\langle a, b, c, d\rangle$ under the independence criterion. No similar claim can be made if using the parallelizability criterion, however, since the concept optimal reordering is then undefined. As remarked by Regnier and Fade [20], it is trivial to find an optimal parallel execution of the plan output by the RF algorithm using standard algorithms. However, this is something very different from optimality wrt. the t.o. plan input to the algorithm.

What is positive about the RF algorithm, however, is that it runs in polynomial time, while the problem of finding an optimal reordering is NP-complete (Theorem 4.5[6]). That is, no polynomial-time algorithm can exist that finds an optimal reordering. The RF algorithm should, thus, be viewed as some kind of approximation algorithm for OPP. However, it is unclear what it computes and to be useful as an approximation algorithm, it should give some kind of guarantee on its output—which is the topic of the next section.

[6]It is easy to see that the theorem holds also for propositional STRIPS even if the plan P is restricted s.t. $\prec$ is a total order on A and the # relation is defined via the independence criterion.

7. Approximation

Since $\sqsubseteq_2$-minimality seems to be a reasonable least-constrainment criterion and is NP-hard to compute, it is interesting to ask whether VPC may at least serve as an approximation algorithm for finding $\sqsubseteq_2$-minimal plans.

The terminology for approximation of minimization problems can be briefly recast as follows (for a more elaborate discussion, see Garey and Johnson [7]). Given some cost function *cost* on candidate solutions, the *performance ratio* of an algorithm A on a problem instance Π is defined as $R_A(\Pi) = cost_A(\Pi)/cost_{opt}(\Pi)$, where $cost_A(\Pi)$ is the cost of the solution returned by algorithm A on instance Π and $cost_{opt}(\Pi)$ is the cost of the optimal solutions to the instance Π. The *absolute performance ratio* R_A is the value of $R_A(\Pi)$ in the worst case, measured over all instances Π of the problem. The *asymptotic performance ratio* R_A^∞ is the value of $R_A(\Pi)$ in the worst case, measured over all instances Π s.t. $cost_{opt}(\Pi) > c$ for some constant c. Finally, the *best achievable performance ratios* $R_{MIN}(\Pi)$, R_{MIN} and R_{MIN}^∞ denote the minimum values for $R_A(\Pi)$, R_A and R_A^∞ respectively, measured over all possible algorithms.

Let the cost of a p.o. plan $\langle A, \prec\rangle$ be defined as $|\prec|$, which is the parameter to minimize. The question, then, is whether there exists some constant c s.t. $R_{VPC} < c$ or, at least, $R_{VPC}^\infty < c$. Unfortunately, this turns out not to be the case and, hence, VPC gives no constant performance guarantee, even asymptotically.

Theorem 7.1 *There exists no constant c s.t. $R_{VPC}^\infty \leq c$.*

Proof: For arbitrary $n > 0$ let the set $A = \{a_0, \ldots, a_n\}$ of actions be defined s.t. $addlist(a_0) = \{p_1, \ldots, p_n\}$ and for $1 \leq k \leq n$, $precond(a_k) = \{p_k\}$ and $addlist(a_k) = \{p_k, q_k\}$. Consider the ppi. $\Pi = \langle \varnothing, \{q_1, \ldots, q_n\}\rangle$ and the t.o. plan $P = \langle a_0, \ldots, a_n\rangle = \langle A, \prec\rangle$. Any $\sqsubseteq_2^{\Pi,A}$-minimal Π-valid plan $P' = \langle A, \prec'\rangle$ obviously satisfies $|\prec'| = n$, while VPC would 'rediscover' and output the plan P, *ie.* the input. Since $|\prec| = (n^2 - n)/2$ it follows that R_{VPC}^∞ is $O(n)$. □

Since the RF algorithm fails to solve optimally the problem it was designed for it is interesting to ask also for this algorithm whether it may serve as an approximation algorithm. The cost for a parallel p.o. plan in this case is defined as the number of steps in its shortest parallel execution. Unfortunately, the situation is just as bad as for the VPC algorithm; the RF algorithm does not exhibit a constant performance guarantee, even asymptotically.

Theorem 7.2 *There exists no constant c s.t. $R_{RF}^\infty \leq c$.*

Proof: For arbitrary $n > 0$, construct the ppi. $\Pi_n = \langle \varnothing, \{r_1, r_2, \ldots, r_n\}\rangle$ and the t.o. plan $P = \langle a_1, b_1, c_1, a_2, b_2, c_2, \ldots, a_n, b_n, c_n\rangle$, where for $1 \leq i \leq n$, $pre(a_i) = \varnothing$, $post(a_i) = \{p_i, \neg q_{i-1}\}$, $pre(b_i) = \{p_i\}$, $post(b_i) = pre(c_i) = \{q_i\}$ and $post(c_i) = \{r_i\}$. Obviously there exists a parallel version of P admitting the 3-step execution $\langle S_1, S_2, S_3\rangle$, where for $1 \leq i \leq n$, $a_i \in S_1$, $b_i \in S_2$ and $c_i \in S_3$ (see Figure 5). The RF algorithm, on the other hand, will fail to reorder the actions and will return its input, *ie.* the plan P, admitting no parallel execution in fewer than $3n$ steps. Hence, the absolute performance guarantee for RF is $\geq n$. For the asymptotic performance guarantee it is sufficient to observe that the above construction can be repeated s.t. for arbitrary $m, n > 0$, there exists a t.o. plan $P' = \alpha_1;\ldots;\alpha_m$ where each α_i is isomorphic to the

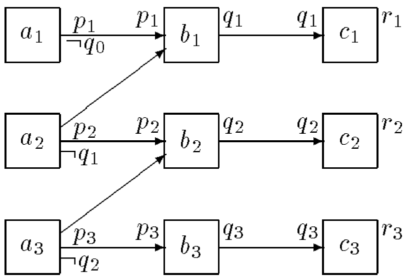

Figure 5: Optimal reordering of the input plan for $n = 3$ in the proof of Theorem 7.2.

plan P above, but extended with extra conditions s.t. for $1 \leq i < m$, all actions in α_i must be executed before the actions in α_{i+1}. Obviously, the shortest parallel execution for P' is of length $3m$, while RF may return the plan P' having no shorter parallel execution than length $3mn$. It follows that $R^{\infty}_{RF} \geq m$ and the theorem follows. □

Neither of the two results above are surprising, however, since there can exist no algorithms solving these problems with constant performance guarantee.

Theorem 7.3 *There exists no constant c s.t. $R^{\infty}_{MIN}(LC2MP) \leq c$, unless $NP \in$ DTIME$(n^{\text{poly} \log n})$ and there exists no constant c s.t. $R^{\infty}_{MIN}(OPP) \leq c$, unless P=NP.*

Proof: Suppose there exists a polynomial-time approximation algorithm A for LC2MP (OPP) and there exists a constant c s.t. $R^{\infty}_{A} \leq c$. It is then immediate from the proof of Theorem 3.6 (4.5) that it is possible to construct an approximation algorithm A' for MINIMUM COVER (GRAPH K-COLOURABILITY) s.t. $R^{\infty}_{A'} \leq c$. However, it was proven by Lund and Yannakakis [14] that no such algorithm can exist, unless $NP \in$ DTIME$(n^{\text{poly} \log n})$ (P=NP). The theorem follows by contradiction. □

8. Discussion

In this paper, the problems of reordering a plan to become least constrained or having an optimal (shortest) parallel execution were considered. For the first problem, two criteria called $\sqsubseteq_1$-minimality and $\sqsubseteq_2$-minimality were defined. $\sqsubseteq_1$-minimality is tractable, but gives no reasonable guarantee of optimality, while $\sqsubseteq_2$-minimality is NP-hard, but possibly has some useful flavour of optimality. The problem of finding a reordering of a plan admitting an optimal parallel execution was also proven NP-hard. Both these problems may seem somewhat related to precedence constrained scheduling, but there is an important difference. While a scheduler is only allowed to add ordering relations, the problems considered in this paper allow any reordering of the actions in a plan. Hence, these problems are closer related to planning than to scheduling, since they require reasoning about the validity of plans.

Two algorithms have been presented in the literature for converting a total-order plan into a least-constrained plan [25] and into a parallel plan having an optimal parallel execution [20] respectively. In the absence of a definition of least-constrainment from its authors, the first algorithm was analysed wrt. the least-commitment criteria defined in this paper. Obviously it cannot produce $\sqsubseteq_2$-minimal plans, unless P=NP, since this is an NP-hard problem. More surprisingly, however, it does not even guarantee finding $\sqsubseteq_1$-minimal plans. It is unclear what the algorithm computes, but is unlikely to compute anything significantly stronger than $\sqsubseteq_1$-minimality, which is a very weak least-constrainment criterion. The second algorithm was similarly found not to guarantee producing a plan having an optimal parallel execution, which is obvious in the light of the NP-hardness result for this problem. Also here it remains unclear what the algorithm really computes. Both algorithms were then reconsidered as approximation algorithms for finding $\sqsubseteq_2$-minimal plans and plans with optimal parallel executions respectively. However, it was shown that neither algorithm gives a constant performance guarantee—both algorithms occasionally return the worst possible solution, namely the input unchanged (when far better solutions exist).

The results in this paper do, however, not necessarily imply that the method of using a total-order planner and convert its output into a partial-order plan is bad. Few, if any, partial-order planners in the literature produce plans that are optimal wrt. to any of the criteria defined in this paper, so the conversion problem remains. Furthermore, a partial-order planner guaranteeing such optimality for its solution could not escape from the complexity problem, it would be inherent in the already difficult planning process.

To summarize, there are tractable least-commitment criteria, but no such criterion is likely to give any useful guarantee for putting a scheduler in a better position to arrive at an optimal schedule. The $\sqsubseteq_2$-minimality criterion gives a considerably better result, although still not giving any such guarantee—yet it is NP-hard to compute. The problem of finding an reordered plan having an optimal parallel execution, however, goes one step further by producing a parallel plan with optimality guarantees for the execution time. Since this problem is of the same complexity as $\sqsubseteq_2$-minimality it is probably better to attack it directly than to first try achieveing $\sqsubseteq_2$-minimality or anything similar. However, because of the NP-hardness of the problem nothing but an approximation can be computed. Even worse, it was shown in this paper that neither this problem nor the problem of finding $\sqsubseteq_2$-minimal plans can be approximated within a constant ratio. If insisting on approximations with worst-case guarantees, it could be worthwhile to exploit the intimate relationship between graph colouring and parallel executions. The best known graph colouring approximation algorithm to date is due to Halldórsson [8], giving a performance guarantee of $O(n(\log\log n)^2/\log^3 n)$. If not requiring completeness or worst-case guarantees there is also the possibility of using local search methods for graph colouring [12, 23], sometimes giving good results in practice.

Acknowledgements

Bernhard Nebel, Erik Sandewall and the anonymous referees provided helpful comments on this paper. The research was supported by the Swedish Research Council for Engineering Sciences (TFR) under grant Dnr. 92-143.

References

[1] *Proc. 10th (US) Nat'l Conf. on Artif Intell. (AAAI-92)*, San José, CA, USA, July 1992.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26(11):832–843, 1983.

[3] T. Bylander. Complexity results for planning. In IJCAI [11], pages 274–279.

[4] D. Chapman. Planning for conjunctive goals. *Artif. Intell.*, 32:333–377, 1987.

[5] T. Dean and M. Boddy. Reasoning about partially ordered events. *Artif. Intell.*, 36:375–399, 1988.

[6] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2:189–208, 1971.

[7] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[8] M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Inf. Process. Lett.*, 45:19–23, Jan. 1993.

[9] A. Horz. Relating classical and temporal planning. In C. Bäckström and E. Sandewall, editors, *2nd Eur. WS. Planning*, Vadstena, Sweden, Dec. 1993. IOS Press. This volume.

[10] *Proc. 4th Int'l Joint Conf. on Artif. Intell. (IJCAI-75)*, Tbilisi, USSR, Sept. 1975.

[11] *Proc 12th Int'l Joint Conf. on Artif. Intell. (IJCAI-91)*, Sydney, Australia, Aug. 1991.

[12] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Oper. Res.*, 39(3):378–406, May–June 1991.

[13] S. Kambhampati. On the utility of systematicity: Understanding tradeoffs between redundancy and commitment in partial-order planning. In *Proc 13th Int'l Joint Conf. on Artif. Intell. (IJCAI-93)*, Chamberý, France, Aug.–Sept. 1993.

[14] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *25th ACM Symp. Theory Comput. (STOC-93)*, pages 286–293. ACM, 1993.

[15] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th (US) Nat'l Conf. on Artif. Intell. (AAAI-91)*, pages 634–639, Anaheim, CA, USA, July 1991.

[16] S. Minton, J. Bresina, and M. Drummond. Commitment strategies in planning: A comparative analysis. In IJCAI [11], pages 259–265.

[17] S. Minton, M. Drummond, J. L. Bresina, and A. B. Philips. Total order *vs.* partial order planning: Factors influencing performance. In *Proc. 3rd Int'l Conf. on Principles of Knowledge Repr. and Reasoning (KR-92)*, pages 83–92, Cambridge, MA, USA, Oct. 1992.

[18] B. Nebel and C. Bäckström. On the computational complexity of temporal projection and plan validation. In AAAI [1], pages 748–753.

[19] B. Nebel and C. Bäckström. On the computational complexity of temporal projection, planning and plan validation. *Artif. Intell.*, 65(2), 1994. To appear.

[20] P. Regnier and B. Fade. Complete determination of parallel actions and temporal optimization in linear plans of action. In *Eur. WS. Planning*, volume 522 of *Lecture Notes in Artificial Intelligence*, pages 100–111, Sankt Augustin, Germany, Mar. 1991.

[21] P. Regnier and B. Fade. Détermination du parallélisme maximal et optimisation temporelle dans les plans d'actions linéaires. *Rev. Intell. Artif.*, 5(2):67–88, 1991.

[22] E. D. Sacerdoti. The non-linear nature of plans. In IJCAI [10].

[23] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In AAAI [1], pages 440–446.

[24] A. Tate. Interacting goals and their use. In IJCAI [10], pages 215–218.

[25] M. M. Veloso, M. A. Pérez, and J. G. Carbonell. Nonlinear planning with parallel resource allocation. In *WS. Innovative Approaches to Planning, Scheduling and Control*, pages 207–212, San Diego, CA, USA, Nov. 1990.