Executing Parallel Plans Faster by Adding Actions

Christer Bäckström[1]

Abstract. If considering only sequential execution of plans, the execution time of a plan will increase if adding actions to it. However, if also considering parallel execution of plans, then the addition of actions to a plan may sometimes decrease the execution time. It is further shown that plans which are sub-optimal in this way are likely to be frequently generated by standard partial-order planners. Unfortunately, optimizing the execution time of a plan by adding actions is NP-hard and cannot even be approximated within a constant ratio.

1 INTRODUCTION

In many applications where plans, made by man or by computer, are executed, it is important to find plans that are optimal with respect to some cost measure, typically execution time. Examples of such applications are manufacturing and error-recovery for industrial processes. Many different kinds of computations can be made to improve the cost of a plan—only a few of which have been extensively studied in the literature. The most well-known and frequently used of these is *scheduling*.

Scheduling typically takes a set of actions[2] and some temporal constraints on these, for instance, their durations, earliest release times and deadlines. The problem can either be to order these actions into an optimal sequence or to find a schedule allowing also parallel execution of actions, subject to some constraint like resource limitations. The first of these problems is usually referred to as the *sequencing problem* and the latter as the *(multi-processor) scheduling problem*. The parameter to optimize is typically the total execution time or some penalty function for actions missing their deadlines. The *precedence constrained scheduling problem* additionally takes a partial order on the actions which must be satisfied by the resulting schedule, that is, essentially it takes a partial-order plan as input.

The scheduling problem concerns only computing the exact release times for the given actions; changing the set of actions or the given partial order on these is never considered. *Planning*, on the other hand, is the problem of finding the set of actions and an execution order on this set. Scheduling can be viewed as a post-processing step to planning, although these two computations are often more intimately connected in practice. The execution order on a plan may be either a total order or a partial order. In a *total order plan (linear plan)* the actions have to be executed in the specified total order—making scheduling a meaningless computation. In a *partial order plan (non-linear plan)* the actions may be executed in any sequence satisfying the partial order. Such a plan may be interpreted either as a *sequential plan* or as a *parallel plan*—in the first case two unordered actions may be executed in either order but not in parallel, while also parallel execution is allowed in the second case. In contrast to total-order plans, partial-order plans may be post-processed to find an optimal execution sequence or schedule.

Scheduling is a rather restricted way of optimizing plans since both the set of actions and the execution order are taken for granted and must not be changed. Nevertheless, saying that the scheduling problem has received much attention in the literature would be an understatement—scheduling is an important way of optimizing plans. However, more dramatic improvements are possible if also the execution order and/or the set of actions may be modified. Plan optimization of this type has not received much attention in the literature, though.

There are basically three ways of modifying the execution order of a plan: adding ordering constraints, deleting ordering constraints and a combination of these (mixed modification). The first of these three operations is hardly viable for optimizing plans, but both the latter are. Algorithms for these problems have been presented in the literature, but optimizing parallel plans by mixed modification of the execution order or even approximating this problem within a constant ratio is NP-hard [2].

The set of actions of a plan may similarly be modified by adding actions, deleting actions or both. However, while the actions can be reordered without modifying the action set it is not possible to modify the action set without also modifying the execution order. Optimizing a plan by removing actions is possible only if it contains redundant actions, since it will otherwise not remain valid. The problem of finding such redundant actions is NP-hard although some polynomial approximation algorithms have been suggested in the literature [4]. The complexity of mixed modification of sequential plans for the purpose of refitting a plan to solve another problem instance has been investigated by Nebel and Koehler [9]. However, no analysis seems to exist for mixed modification with the purpose of optimizing the execution time of a plan. Finally, modifying a plan by adding actions only is obviously not a viable operation for sequential plans since the total execution time of the plan is the sum of the execution times

[1] Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden

[2] The scheduling literature typically uses the term *task*, while the planning literature typically uses the term *plan step* or *action*, the latter being used in this paper.

ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

for the actions in the plan. However, if considering also parallel execution of plans, then the execution time can, perhaps somewhat counter to intuition, be improved by adding actions to it, as will be shown in the next section. After that, Section 3 introduces a formal framework for parallel plans, Section 4 analyzes the complexity of optimizing plans by action addition and Section 5 presents some results on approximation. The paper ends by a discussion of the practical relevance of the results and the conclusions.

2 OPTIMIZING EXECUTION TIME BY ADDING ACTIONS

When considering only sequential execution of plans it is obvious that adding actions to a plan must increase its execution time since this equals the sum of the execution times of its constituent actions.[3] In contrast to this, adding actions to a parallel plan can improve its execution time. To understand why this somewhat counter-intuitive claim is true, consider the plan in Figure 1. This example assumes the 'standard' propositional STRIPS framework (see Definition 3.2). The plan in Figure 1 consists of $m + n + 1$ actions, named $a, b_1, \ldots, b_m, c_1, \ldots, c_n$ respectively. Each action is shown as a box labelled with the name of the action. The partial execution order is indicated by arrows (transitive arcs are omitted). The preconditions of an action are shown to the left of its box and the effects are shown to the right of the box, preceded by a $+$ or $-$ depending on whether they are added or deleted respectively. The actions $b_1, \ldots, b_m$ are intended to form a chain s.t. each action b_k produces a necessary precondition, q_k, for the succeeding action b_{k+1} in the chain and the actions $c_1, \ldots, c_n$ form a similar chain. Finally, the initial state is assumed to be empty and the goal state to be $\{q_m, r_n\}$.

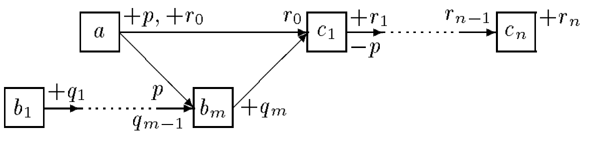

Figure 1. A partial-order plan.

Any sequential execution of the plan is a total ordering, that is, a permutation sequence of its actions that satisfies the given partial order. In this case the only ordering choice is the relative position of action a wrt. the actions $b_1, \ldots, b_m$. The only requirement is that a be executed before b_m, so there are m possible execution sequences for the plan. However, all these execution sequences take the same time to execute, $m + n + 1$ time units if assuming that all actions take unit time. A parallel execution could not do much better, on the other hand. In the best case, action a can be executed in parallel with some of the actions $b_1, \ldots, b_{m-1}$, resulting in a total execution time of $m + n$ time units—quite a modest improvement.

Now, make the somewhat optimistic assumption that two actions may be executed in parallel whenever their add- and delete-lists are not conflicting. Under this interpretation all the actions $b_1, \ldots, b_{m-1}$ can be executed in parallel with the actions $a, c_1, \ldots, c_n$. Obviously, the plan could be executed much faster, in parallel, if it were not for the bottleneck that action b_m must precede action c_1. This can be exploited by breaking the bottleneck, if actions may be added to the plan. A new action a' of the same type as a can be inserted and the partial order be modified as shown in Figure 2. This plan is a minor modification of the previous one. Yet, it can be executed in $\max\{4, m, n + 1\}$ time units—quite a dramatic improvement if m and n are large. Although such dramatic speed-ups should not be expected in practice, since many actions would probably not be executable in parallel, considerable improvements could still be expected in many cases.

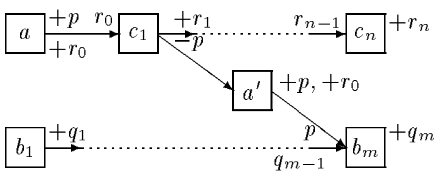

Figure 2. Modified version of the previous plan.

Even greater speed-up is possible for plans containing several structures of the kind shown in Figure 1. For instance, for arbitrary $k > 0$, it is possible to construct a plan consisting of k subplans of the type in Figure 1 cascaded in series. Such a plan can be made to execute k times faster by inserting k new actions. That is, there exist plans which can be modified to execute faster by an arbitrary factor.

Although a greater number of subplans of the type in Figure 1 may make it possible to reduce the execution time by a greater factor, this is not always the case; local improvement need not imply global improvement of the overall plan. Suppose, for instance, that the plan in Figure 1 is a subplan of another plan. If considering only this subplan, the modification in Figure 2 is obviously an improvement. However, if considering the whole plan, there may be further constraints forcing this subplan to be executed sequentially, resulting in a longer execution time, $m + n + 2$ time units. Further, two action additions which both improve the total execution time if considered in isolation may interfere and increase the execution time when considered together. Not surprisingly, this leads to computational difficulties, as will be shown in Section 4.

It is currently not known whether also other types of plan structures than the one shown in Figure 1 give similar opportunities for improving the parallel execution time of plans.

3 BASIC DEFINITIONS

This section will formally define planning problems, plans and related concepts. In order to make the definitions and results as general and formalism independent as possible, an axiomatic framework will be used. This framework makes only a minimum of assumptions about the underlying formalism.

Definition 3.1 *A* planning formalism *is defined as a triple $\mathcal{F} = \langle \mathcal{O}, \mathcal{S}, V \rangle$, where $\mathcal{O}$ is a domain of operators, $\mathcal{S}$ is a*

[3] This assumes that the execution time of an action is not context dependent, as in Sandewall's furniture assembly scenario [10], for instance.

domain of plan specifications and V *is a a* validity test, *defined as a binary relation* $V \subseteq Seqs(\mathcal{O}) \times \mathcal{S}$, *where* $Seqs(\mathcal{O}) = \{\langle\rangle\} \cup \{\langle o\rangle;\omega | o \in \mathcal{O}, \omega \in Seqs(\mathcal{O})\}$ *is the set of all sequences over* $\mathcal{O}$ *and* ; *denotes sequence concatenation. A* parallel planning problem instance (parallel ppi.) *over* $\mathcal{F}$ *is a triple* $\Pi = \langle O, \#, S\rangle$ *s.t.* $O \in \mathcal{O}$, $S \in \mathcal{S}$ *and* $\#$ *is a reflexive and symmetric binary relation on* O.

The relation $\#$ in parallel ppi's is referred to as a *non-concurrency relation* and it indicates which operators may not be executed in parallel. Furthermore, 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, by not making any further assumptions—except, maybe, for the time complexity of the validity test—it will be possible to prove widely applicable upper bounds on complexity in the next section. For hardness proofs, however, it is better to use a very restricted formalism—in this case, the Simple Propositional STRIPS formalism.

Definition 3.2 *The* Simple Propositional STRIPS formalism (SPS) *is defined as follows. Let* $\mathcal{P} = \{p_1, p_2, \ldots\}$ *be the set of propositional atoms. The SPS formalism is a triple* $\mathcal{F}_{SPS} = \langle \mathcal{O}, \mathcal{S}, V\rangle$, *where* $V \subseteq Seqs(\mathcal{O}) \times \mathcal{S}$ *is recursively defined s.t. for arbitrary* $\langle o_1, \ldots, o_n\rangle \in Seqs(\mathcal{O})$ *and* $\langle I, G\rangle \in \mathcal{S}$, $\langle\langle o_1, \ldots, o_n\rangle, \langle I, G\rangle\rangle \in V$ *iff either*

1. $n = 0$ *and* $G \subseteq I$ *or*
2. $n > 0$, $o_1 = \langle pre, add, del\rangle$, $pre \subseteq I$ *and* $\langle\langle o_2, \ldots, o_n\rangle, \langle I \cup add - del, G\rangle\rangle \in V$.

A parallel ppi. $\Pi = \langle O, \#, S\rangle$ *over* $\mathcal{F}_{SPS}$ *is said to use the* effect conflict criterion *if* $\#$ *is defined s.t. for all operators* $o = \langle pre, add, del\rangle$ *and* $o' = \langle pre', add', del'\rangle$ *in* O, $o\#o'$ *iff* $(add \cap del') \cup (add' \cap del) \neq \varnothing$.

For partial-order plans, it is necessary to distinguish the different occurrences of an operator, motivating the introduction of actions.

Definition 3.3 *Given an operator domain* $\mathcal{O}$, *an instantiation (or occurrence) of an operator in* $\mathcal{O}$ *is called an* action. *Given an action* a, *the notion* $type(a)$ *denotes the operator that* a *instantiates. A set* $\{a_1, \ldots, a_n\}$ *or sequence* $\langle a_1, \ldots, a_n\rangle$ *of actions is said to be* over *a set* O *of operators iff* $\{type(a_1), \ldots, type(a_n)\} \subseteq O$.

Considering only sequential execution, the notion of plans can be defined in the usual way as follows.

Definition 3.4 *Let* $\Pi = \langle O, \#, S\rangle$ *be a parallel ppi. over some formalism* $\mathcal{F} = \langle \mathcal{O}, \mathcal{S}, V\rangle$. *A* total-order plan (t.o. plan) *over* O *is a sequence* $P = \langle a_1, \ldots, a_n\rangle$ *of actions over* O. P *can alternatively be denoted by the tuple* $\langle\{a_1, \ldots, a_n\}, \prec\rangle$ *where for all* $1 \leq k, l \leq n$, $a_k \prec a_l$ *iff* $k < l$. *The t.o. plan* P *is* Π-valid *iff* $\langle\langle type(a_1), \ldots, type(a_n)\rangle, S\rangle \in V$. *A* partial-order plan (p.o. plan) *over* O *is a tuple* $P' = \langle A, \prec\rangle$ *where* A *is a set of actions over* O *and* $\prec$ *is a strict (ie. irreflexive) partial order on* A. *The p.o. plan* P' *is* Π-valid *iff all topological sortings of it 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, however. Parallel execution of plans is defined as follows, assuming that two unordered actions a and b in a valid p.o. plan can be executed in parallel unless $type(a)\#type(b)$

Definition 3.5 *Let* $\Pi = \langle O, \#, S\rangle$ *be a parallel ppi. over some formalism* $\mathcal{F}$ *and let* $P = \langle A, \prec\rangle$ *be a* Π*-valid p.o. plan over* O. *An* n-step parallel execution *of* P *is a sequence* $\langle A_1, \ldots, A_n\rangle$ *of subsets, referred to as* slices, *of* A *satisfying the following conditions*

1. $A_1, \ldots, A_n$ *is a partitioning of* A;
2. *if* $a \in A_k$, $b \in A_l$ *and* $a \prec b$, *then* $k < l$.
3. *if* $a \in A_k$, $b \in A_l$, $a \neq b$ *and* $type(a)\#type(b)$, *then* $k \neq l$.

A parallel execution of P *is a* shortest parallel execution *of* P *if there is no parallel execution of* P *in fewer steps.*

This framework admits expressing possible parallelism only; necessary parallelism is out of the scope of this paper and it requires a planner having access to and being able to make use of additional information. Further, it will be assumed that all actions take unit time and that all actions within a slice are executed exactly in parallel, that is, each slice takes unit time. Since this is a special case of more general parallelism, the hardness results in the following sections carry over implicitly to more general formalisms, like metric time and Allen's interval algebra [1].

4 COMPLEXITY RESULTS

In this section it will be proven that it is computationally hard to optimize the execution time of parallel plans by adding actions. The closely related problem of reordering the actions in a plan to optimize the parallel execution time—the OPTIMAL PARALLEL PLAN REORDERING problem—has previously been analyzed by Bäckström [2] and proven computationally difficult.[4]

Definition 4.1 *The decision problem OPTIMAL PARALLEL PLAN REORDERING (OPPR) is defined as follows for arbitrary planning formalism* $\mathcal{F}$.
Given: *A parallel ppi.* $\Pi = \langle O, \#, S\rangle$ *over* $\mathcal{F}$, *a* Π*-valid p.o. plan* $P = \langle A, \prec\rangle$ *over* O *and an integer* $k \leq |A|$.
Question: *Is there an order* $\prec'$ *over* A *s.t.* $\langle A, \prec'\rangle$ *is a* Π*-valid p.o. plan having a* k*-step parallel execution?*

Theorem 4.2 (Bäckström [2, Theorem 4.5]) *The problem OPTIMAL PARALLEL PLAN REORDERING is NP-hard, even when restricted to the SPS formalism with operators restricted to have empty preconditions, the effect conflict criterion is used and all actions take unit time.*

The more general problem, discussed in Section 2, where also the addition of actions to the plan is allowed can be defined analogously.

[4] This problem was originally called OPTIMAL PARALLEL PLAN (OPP), but is here specialized to OPTIMAL PARALLEL PLAN REORDERING since different ways of optimizing plans are considered. Furthermore, the formalism is slightly changed in this paper, but the theorems recapitulated below still hold, requiring only minor changes to their proofs.

Definition 4.3 *The decision problem OPTIMAL PARALLEL PLAN EXTENSION (OPPE) is defined as follows for arbitrary planning formalism $\mathcal{F}$.*
Given: *A parallel ppi. $\Pi = \langle O, \#, S \rangle$ over $\mathcal{F}$, a Π-valid p.o. plan $P = \langle A, \prec \rangle$ over O and two integers $k \leq |A|$ and $l \geq 0$.*
Question: *Is there a Π-valid p.o. plan $P' = \langle A', \prec' \rangle$ over O s.t. $A \subseteq A'$, $|A'| \leq |A| + l$ and P' has a k-step parallel execution?*

The special case where $l = 0$ coincides with the OPPR problem, so the NP-hardness result for OPPR carries over immediately to OPPE. Furthermore, the problem is NP-complete if plans can be validated in polynomial time.

Theorem 4.4 *The problem OPTIMAL PARALLEL PLAN EXTENSION is NP-hard, even if the formalism is SPS with operators restricted to have empty preconditions, the effect conflict criterion is used, all actions take unit time and $l = 0$. The problem is furthermore NP-complete if p.o. plans can be validated for parallel ppi's in polynomial time and all actions take unit time.*

Proof: Hardness is immediate from Theorem 4.2 by choosing $l = 0$. For membership in NP, first note that since $\#$ is reflexive, no slice in a parallel execution of a p.o. plan can contain more than $|O|$ actions. Hence, no plan of length $|A| + l$ can contain more than $|O|(|A| + l)$ actions. It follows that a solution can be guessed and verified in polynomial time so membership in NP follows. □

This gives little hope for computing efficiently the length of the optimal parallel execution for a plan in the worst case, even if not adding any actions. An alternative question to ask then is whether the shortest parallel execution of a plan can be decreased at all by adding some number of actions,[5] which can be formalized as follows.

Definition 4.5 *The problem SHORTER PARALLEL PLAN EXTENSION (SPPE) is defined as follows:*
Given: *A parallel ppi. $\Pi = \langle O, \#, S \rangle$ over $\mathcal{F}$, a Π-valid p.o. plan $P = \langle A, \prec \rangle$ over O and an integer $l \geq |A|$.*
Question: *Is there a Π-valid p.o. plan $P' = \langle A', \prec' \rangle$ over O s.t. $A \subseteq A'$, $|A'| \leq |A| + l$ and P' has a strictly shorter shortest parallel execution than P has?*

Unfortunately, even this problem is very difficult.

Theorem 4.6 *The problem SHORTER PARALLEL PLAN EXTENSION is NP-hard, even if the formalism is SPS with operators restricted to have empty preconditions, the effect conflict criterion is used and all actions take unit time. The problem is furthermore NP-complete if p.o. plans can be validated for parallel ppi's in polynomial time and all actions take unit time.*

Proof: For hardness it is sufficient to prove the restricted special case. Proof by transformation from the MINIMUM COVER problem [5, p. 322], which is NP-complete. Let $S = \{p_1, \ldots, p_n\}$ be a set of atoms, $C = \{S_1, \ldots, S_m\}$ 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. $|C'| = k$ and $S \subseteq \cup_{T \in C'} T$.

Construct the operator set

$$O = \{o_1^0, \ldots, o_4^0\} \cup \{o_1^i, o_2^i \mid 1 \leq i \leq n\},$$

with operators defined according to Table 1. Also define the

Table 1. Operators for the proof of Theorem 4.6.

operator	*pre*	*add*	*del*
o_1^0	$\varnothing$	$\{q_1\}$	$\varnothing$
o_2^0	$\{q_1\}$	$\{q_2\}$	$\varnothing$
o_3^0	$\{q_2\}$	$\{q_3\}$	$\varnothing$
o_4^0	$S \cup \{q_3\}$	$\{q_4\}$	$\varnothing$
o_1^1	$\varnothing$	$S_1 \cup \{r_1^1\}$	$\varnothing$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
o_1^n	$\varnothing$	$S_n \cup \{r_1^n\}$	$\varnothing$
o_2^1	$\{r_1^1\}$	$\{r_2^1\}$	S_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
o_2^n	$\{r_1^n\}$	$\{r_2^n\}$	S_n

plan specification $S = \langle \varnothing, \{q_4, r_2^1, \ldots, r_2^n\} \rangle$. The triple $\Pi = \langle O, \varnothing, S \rangle$ is obviously a parallel SPS instance. Now, define the action set $A = \{a_1^0, \ldots, a_4^0\} \cup \{a_1^i, a_2^i \mid 1 \leq i \leq n\}$ containing one action for each operator in O in the obvious way. Further, define the order $\prec$ on A s.t.

1. $a_1^i \prec a_4^0 \prec a_2^i$ for $1 \leq i \leq n$;
2. $a_i^0 \prec a_{i+1}^0$ for $1 \leq i < 4$

and $\prec$ is transitive. The tuple $P = \langle A, \prec \rangle$ is obviously a Π-valid plan having a shortest parallel execution of 5 steps.

It remains to prove that the shortest parallel execution for P can be decreased by adding k actions to P iff S has a cover of size k.

If: Suppose $C' = \{S_{i_1}, \ldots, S_{i_k}\}$ is a cover for S of size k. Modify the plan P as follows. Reorder a_4^0 after a_2^k for $1 \leq k \leq n$. Further, create a new instance $b_1^{i_l}$ of operator $o_1^{i_l}$ for each $S_{i_l} \in C'$ and order $a_2^{i_l} \prec b_1^{i_l} \prec a_4^0$. This new plan is Π-valid and has a parallel execution in 4 steps.

Only if: Suppose P can be modified to have a shorter parallel execution than 5 steps by adding actions. This must then be a 4-step execution since the actions $a_1^0, \ldots, a_4^0$ must be executed in sequence. This further requires that all a_2^k are ordered before a_4^0 (they cannot be unordered). The only way to make this plan valid is to insert actions of types $o_1^{i_1}, \ldots, o_1^{i_k}$ s.t. $pre(o_4^0) \subseteq \{q^3\} \cup (\cup_{i_1 \leq l \leq i_k} add(o_1^{i_l}))$ between the actions $a_2^1, \ldots, a_2^n$ and a_4^0. This, however, implies that $\{S_{i_1}, \ldots, S_{i_k}\}$ is a cover of size k for S.

Finally, membership in NP is immediate in the case of tractable validity testing, since there can never be any reason to add more than n new actions. □

5 APPROXIMATION RESULTS

Now that all three problems considered in the previous section are known to be NP-hard, one may ask whether they can, at least, be approximated. Unfortunately, it turns out that it is hard even to compute good approximations for these problems. The OPPR problem is already known to be impossible to approximate within a constant ratio.

[5] Note that this does not imply knowing the length of the optimal parallel execution either before or after adding actions.

Theorem 5.1 (Bäckström [2, Theorem 7.3]) *OPPR cannot be approximated within a constant ratio c for any $c > 0$, unless $P = NP$.*

Since the special case of OPPE where $l = 0$ coincides with the OPPR problem, the result carries over immediately also to OPPE.

Corollary 5.2 *OPPE cannot be approximated within a constant ratio c for any $c > 0$, unless $P = NP$.*

It turns out that even the SPPE problem is difficult to approximate, however.

Theorem 5.3 *SPPE cannot be approximated within a constant ratio c for any $c > 0$, unless $NP \subseteq \mathrm{DTIME}\,(n^{\mathrm{poly}\,\log n})$.*

Proof: Suppose there exists a polynomial-time algorithm A approximating SPPE within some constant ratio c. It is then immediate from the proof of Theorem 4.6 that this algorithm can be used to approximate SET COVER within ratio c. However, this is impossible unless $NP \subseteq \mathrm{DTIME}\,(n^{\mathrm{poly}\,\log n})$ [7], which contradicts the existence of A. □

6 PRACTICAL RELEVANCE

It has been shown in this paper that there exist plans which are optimal for sequential execution, but not for parallel execution. A question which remains to be addressed then is how frequent such plans are in practice. Although this question may be hard to answer in general, a partial answer is that standard partial-order planners like TWEAK [3] or McAllester and Rosenblitt's planner [8] can easily produce such plans.

When planning for the conjunctive goal $\{q_m, r_n\}$ such a planner will attempt solving the two subgoals separately. At some point in the search space it is likely to encounter the plan in Figure 1, but with actions b_m and c_1 yet unordered. Such a plan arises from generating separately the two subplans $a, c_1, \ldots, c_n$ and $b_1, \ldots, b_m$ solving the subgoals q_m and r_n respectively, reusing a to produce preconditions both for b_m and for c_1. In this plan the actions a, b_m and c_1 form a *right-fork conflict* [6], which can only be resolved by ordering c_1 after b_m (unless new actions are inserted), resulting in the plan in Figure 1.

For the planners mentioned above, all valid plans appear in the search space, so they could, in principle, just as well happen to return the optimal plan in Figure 2. Further, in the case above, TWEAK may choose to solve the right-fork conflict by inserting a white knight between c_1 and b_m. However, typical termination criteria for planners prefer plans with fewer actions, making the planner return the suboptimal plan in Figure 1. The termination criterion could, of course, be chosen to favour plans having optimal parallel executions, but, as was shown in the previous sections, this criterion is NP-hard to compute and it is, thus, probably too costly to use as termination criterion. Thus, although a planner can be tuned to return optimal parallel plans, this is not necessarily any better than post-processing a plan that is optimal for sequential execution—it just moves the computational problem from the post-processing step to the termination criterion.

7 CONCLUSIONS

It has been shown in this paper that while the sequential execution time of a plan necessarily increases when adding actions to it, the parallel execution time may occasionally decrease. That is, a plan that is optimal for sequential execution need not be optimal for parallel execution. Furthermore, standard partial-order planners are likely to return such plans not infrequently. Post-processing plans by adding actions to optimize the parallel execution time thus seems to be an operation which is not only of theoretical interest, but also useful in practice; quite dramatic improvement may occur in the ideal case. Unfortunately, determining how fast a plan can be executed by adding actions to it or how many actions need be added to give any improvement at all is NP-hard. Even worse, these problems cannot even be approximated within a constant ratio. To make practical use of the speed-up phenomenon reported, one must, thus, probably turn to other methods, either settling for approximation with weaker performance guarantees or use randomized methods like simulated annealing or randomized local search.

ACKNOWLEDGEMENTS

The author would like to thank the anonymous referees for their comments. The research presented in this paper was supported by the Swedish Research Council for Engineering Sciences (TFR) under grant Dnr. 92-143.

REFERENCES

[1] James F Allen, 'Maintaining knowledge about temporal intervals', *Comm. ACM*, **26**(11), 832–843, (1983).

[2] Christer Bäckström, 'Finding least constrained plans and optimal parallel executions is harder than we thought', in *Current Trends in AI Planning: EWSP'93—2nd Eur. WS. Planning*, pp. 46–59, Vadstena, Sweden, (December 1993). IOS Press.

[3] David Chapman, 'Planning for conjunctive goals', *Artif. Intell.*, **32**, 333–377, (1987).

[4] Eugene Fink and Qiang Yang, 'Formalizing plan justifications', in *Proc. 9th Conf. of the Can. Soc. Comput. Stud. Intell. (CSCSI'92)*, pp. 9–14, Vancouver, BC, Canada, (May 1992).

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

[6] Joachim Hertzberg and Alexander Horz, 'Towards a theory of conflict detection and resolution in nonlinear plans', in *Proc. 11th Int'l Joint Conf. on Artif. Intell. (IJCAI-89)*, pp. 937–942, Detroit, MI, USA, (August 1989).

[7] Carsten Lund and Mihalis Yannakakis, 'On the hardness of approximating minimization problems', in *25th ACM Symp. Theory Comput. (STOC-93)*, pp. 286–293. ACM, (1993).

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

[9] Bernhard Nebel and Jana Koehler, 'Plan modification versus plan generation: A complexity-theoretic perspective', in *Proc 13th Int'l Joint Conf. on Artif. Intell. (IJCAI-93)*, pp. 1436–1441, Chamberý, France, (August–September 1991).

[10] Erik Sandewall, *Features and Fluents*, Oxford University Press, 1994. To appear.