Theory and Algorithms for Plan Merging

David E. Foulser [*] Ming Li [†] Qiang Yang [‡]

Abstract

Merging operators in a plan can yield significant savings in the cost to execute a plan. This paper provides a formal theory for plan merging and presents both optimal and efficient heuristic algorithms for finding minimum-cost merged plans. The optimal plan-merging algorithm applies a dynamic programming method to handle multiple linear plans and is extended to partially ordered plans in a novel way. Furthermore, with worst and average case complexity analysis and empirical tests, we demonstrate that efficient and well-behaved approximation algorithms are applicable for optimizing plans with large sizes.

[*]Silicon Graphics Computer Systems, M/S 7U-550, 2011 N. Shoreline Blvd., Mountain View, CA 94039, USA. Supported in part by Office of Naval Research grant N00014-86-J-1906 and by NIH Grant R01 LM05044 from the National Library of Medicine.

[†]Department of Computer Science, University of Waterloo. Waterloo, Ontario, Canada, N2L 3G1. Supported in part by NSERC operating grants OGP0036747 and OGP0046506.

[‡]Department of Computer Science, University of Waterloo. Waterloo, Ontario, Canada, N2L 3G1. Supported in part by NSERC operating grant OGP0089686 and an ITRC grant.

1 Introduction

1.1 The Plan Merging Problem

The value of helpful or positive interactions among the different parts of plans was recognized early in AI planning research [13, 16, 18]. A helpful goal interaction occurs in a plan, or among several plans, if certain portions of the plan can be modified to improve its quality. An important type of helpful goal interaction occurs when certain operators in a plan can be grouped, or *merged*, together in such a way as to make the resulting plan more efficient to execute. This happens often in domains where redundant setup and restore operations can be eliminated in the execution of consecutive tasks and where redundant journeys can be eliminated by fetching multiple objects at once.

Consider the following description given by Wilensky [17].

> John was planning to go camping for a week. He went to the supermarket to buy a week's worth of groceries.

The main character in this example, John, had a set of subgoals to achieve, each subgoal being to buy food for a meal during the camping week. However, instead of making several shopping trips separately for each individual meal, John was able to *merge* the plans for the subgoals, and achieve them simultaneously with a single trip to the market. The resultant *merged* plan is more efficient to execute than the separate ones.

Merging actions for reducing plan costs, as demonstrated above , is typical of the kind of optimization task people do for general transportation problems. For example, suppose that cargo items A and B both need delivered from location L_1 to location L_2 by planes. If the two delivery goals are planned separately, then two airplanes would be required for both A and B. For each item, a separate loading and unloading operation are needed. However, if A and B fit into one plane, then combining the two sub-plans can produce a more efficient overall plan for both goals. This *merged plan* requires one combined loading and unloading operation for A and B, as well as a single flying operation. The result is a plan that is considerably cheaper than the original ones.

Plan merging is equally important for minimizing the costs of robot task plans. As an example, consider a blocks world problem with variable-sized blocks. To pick up one block of a certain size, the robot arm has to mount a gripper of an appropriate size. Suppose that only one robot arm exists, and in order to grab a block of a different size, the robot has to unmount the current gripper and mount the gripper with the new size. In this case, it is more efficient to group block stacking operations that use the same type of grippers. In the example shown in Figure 1, the blocks are of two different sizes. Suppose that the robot can use two different grippers of type `A` and `B` to pick up a corresponding type of block. Then a minimal cost way for achieving the goal would be to move `A1` and `A2` to the table together, then move `B1` and `B2` to the the top of `A1` and `A2`. This plan saves two gripper changing operations as compared with moving A's and B's interleavingly.

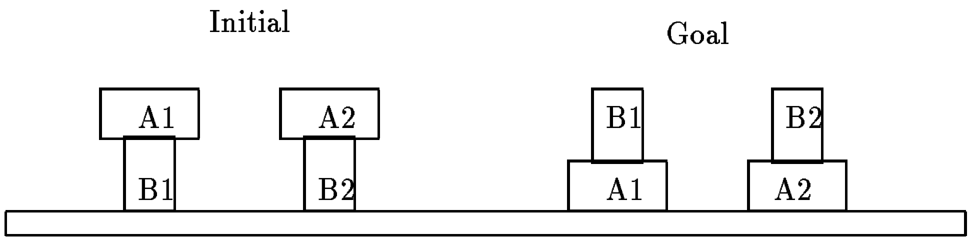


Figure 1: A blocks world where blocks are of different sizes.

The blocks world problem is illustrative of the role of operator merging in robotic assembly domains. Identical plan merging issues also arise in the domain of automated manufacturing where process plans for metal-cutting [6], set-up operations [4] and tool-approach directions [10] need to be optimized. Similarly, in the area of query optimization in database systems [15], as well as domains having multiple agents [2, 12], operator merging in multiple plans seems inevitable.

1.2 Previous Work

Sacerdoti's NOAH [13] system is one of the first planners to explicitly seek opportunities for plan merging. It relies on its set of Critics to handle possible interactions among the different parts a plan. Three critics are introduced for the purpose of improving the quality of plans, including "eliminate redundant preconditions," "use existing objects" and "optimize disjuncts." For example, the "eliminate redundant preconditions" critic can be considered as merging two or more operators that are used to achieve the same preconditions, as long as no precedence relation in the plan is violated. Wilkins' SIPE [18] and Tate's NONLIN [16] are other planning systems that have relatively more advanced capabilities in operator merging. For example, they are able to recognize that a goal is achievable by another operator in the current plan. In such situations, it will impose constraints on orderings and variable bindings so that the operator is used to achieve it. This process, known as *phantomization* of a goal, is also used in the plan reuse framework of Kambhampati's plan reuse framework for reducing plan cost [?]. At a meta-planning level, Wilensky [17] considers different types of *positive goal relationships*, in the context of cognitive modeling of human problem solving.

In contrast to the above domain-independent approaches to planning, Hayes [4] proposed a domain-dependent method for plan merging in *machining* domains. In her Machinist system, a greedy algorithm is implemented and compared with human machinists in terms of the quality of the plans produced. It was shown that it can out-perform an experienced human process planner in many instances.

As an intermediate approach that falls between domain-independent and domain-dependent extremes, Yang, Nau and Hendler [11] proposed a limited-interaction approach to planning, taking into consideration operator merging as one type of interactions allowed. In recognizing the complexity of the general problem, they imposed certain restrictions that work well for a number of domains. An interesting feature of their system is that they considered the case where there is more than one alternate plan for each goal, and proposed a branch and bound algorithm for selecting the plans that can be merged optimally from a set of alternative choices.

These previous research have had successes in various practical domains. But the understanding of plan merging has largely remained at a qualitative level. Most existing systems have adopted some heuristic methods for encouraging plan merging, but little has been known about the condition under which plans may be merged, the computational complexities of optimal and approximate algorithms for plan merging, as well as the quality of the merged plans that the approximation algorithms produce. This lack of knowledge is partly due to the fact that the majority of planning research so far has concentrated on methods for dealing with negative relationship among goals and plans, such as goal conflict and resource competitions. As a consequence, formal theories are appearing on methods for constructing sound and complete planners[1], for efficiently resolving conflicts [19, 1], for reusing past planning experiences[5], and for building "good" abstraction hierarchies [7, 21] for improving problem-solving efficiency. A common theme of these research has been on finding a solution that works. In contrast, this paper attempts to address the issues of finding solutions with optimal or "good" quality.

1.3 Overview

The purpose of this paper is to develop a computational theory on plan mering. This theory will attempt to address the following issues:

1. What are the conditions under which operators in a plan can be merged?
2. What is the computational complexity of optimal plan mering?
3. What are the optimal algorithms for plan merging? When are these algorithms feasible in practice?
4. Where the optimal algorithms are infeasible to apply, what are the approximation algorithms that perform fast? What are the worst and average-case complexities of these algorithms?

Our approach to addressing the above questions is to combine formal methods developed in Artificial Intelligence and computational techniques from Operations Research (OR). In particular, we first propose a formalization of plan merging using the STRIPS operator definitions. Based on the formalization, we present a dynamic programming algorithm for determining the optimal solution by reducing the problem to the shortest common supersequence problem, a variant of the longest common subsequence problem, and apply several known results from that area. While the dynamic programming algorithm has traditionally formulated for inputs corresponding to matrices of symbols, AI planning has been concerned about plans represented as partially ordered operator sets. To bridge the two different fields, we also extend the dynamic programming method to handling partially ordered plans in a novel way.

One drawback of the dynamic programming method is that it becomes computationally infeasible for problems of larger sizes. While we are able to phrase an optimal algorithm for general purpose, domain-independent plan merging, its run time requirements may be prohibitive for inputs of practical sizes. To make the planning problem more tractable, most existing systems that consider helpful interactions employ certain kinds of greedy algorithms for plan merging [4, 13, 16, 18]. Thus, we also describe four approximation algorithms for merging plans, and analyze the qualities of their outputs in average cases. The approximation algorithms all have linear time complexity in the number of operators of the input plan. Finally, we present a set of empirical results comparing the quality of the plans produced by the optimal and approximation algorithms under various conditions.

The remainder of the paper has the following organization. Section 2 gives a formal description of plan merging. Section 3 presents the optimal algorithm for merging linearly ordered plans. Section 4 extends the traditional dynamic programming algorithm to partially ordered plans as well. Section 5 develops four approximation plan merging algorithms and analyzes their behavior. Section 6 presents experimental results for the four heuristic methods. Conclusions are stated in section 7.

2 Formal Description of the Problem

2.1 A Formal Definition of Operator Merging

Given a set of goals to be achieved, a plan Π is a partially ordered set of operators, where each operator α is represented by preconditions P_α and effects E_α, which for the sake of simplicity are assumed to be sets of literals (positive or negative atomic sentences). No operator's preconditions or effects can contain both a literal *and* its negation. A plan Π may be represented as a graph $\Pi = (O, B)$ where the vertices O are the set of operators in Π and the edges B are the set of precedence relations in Π. It is also assumed that two special operators exist in every plan Π that represent the initial and goal states in Π. The initial state operator $\mathcal{I}$ precedes all other operators in Π, and the goal state operator $\mathcal{G}$ is preceded by all other operators. $\mathcal{I}$ has an empty set of preconditions, and has as its effects the set of initial conditions. Likewise, the goal operator $\mathcal{G}$ has an empty set of effects, and has as its preconditions the goal conditions to be achieved. In a least-commitment plan, there is usually a set of constraints on the binding of variables, known as codesignation and noncodesignation constraints [1], which we omit here for simplicity. Thus, plans are fully ground.

In addition to preconditions and effects, we also assume that each operator α has an associated cost cost(α), and that the cost for a plan Π, denoted cost(Π), is the *sum* of the costs of the operators in Π.

In a partially ordered plan, operator α necessarily precedes β if α precedes β under every consistent total ordering of the plan. We will use $\prec$ to denote *necessary* precedence in the partial order on Π.

We assume that the plan Π is justified, in that every operator in Π is useful in establishing some preconditions of other operators, or the goals themselves. Operator α necessarily establishes a precondition p for an operator β, denoted by Establishes(α, β, p) [20], if and only if p is a precondition of β and effect of α, and no operator necessarily between α and β has either p or $\neg p$ as an effect.

For a given plan, there may be some operators in the plan that can be grouped together, and replaced by a less costly operator that achieves all the useful effects of the grouped operators. In such a case, we say that the operators are *mergeable.* We formalize this notion below.

We start by defining what operators in a plan can be grouped together. A set of operators Σ in a plan $\Pi = (O, B)$ induces a subplan (Σ, B_Σ) within Π, where B_Σ is a maximal subset of B that are relations on Σ. Operators in Σ can be *grouped together* if and only if no other operator outside Σ is necessarily between any pair in Σ. More precisely,

$$\forall \alpha, \beta \in \Sigma, \neg\exists \gamma \in O - \Sigma, \text{ such that } \alpha \prec \gamma \prec \beta.$$

Let Π_Σ be a subplan of $\Pi = (O, B)$ induced by Σ, where Σ is a set of operators that can be grouped together. Consider the collective behavior of Π_Σ in Π. Some effects of the operators in Σ are *useful* in Π, because they establish the preconditions of some *other* operators in Π that are outside Σ, or the goals directly. Other effects are either side-effects of the operators, or are used to achieve the preconditions of the operators within Σ. We use Useful-Effects(Σ, Π) to denote the set of all useful effects of the operators in Σ. Likewise, we use Net-Preconds(Σ, Π) to denote the set of all preconditions of the operators in Σ not achieved by any operators in Σ. More formally,

$$\text{Useful-Effects}(\Sigma, \Pi) = \bigcup_{\alpha \in \Sigma} \{e \mid e \in E_\alpha \textit{ and } \exists \beta \in (O - \Sigma) \textit{ such that } \text{Establishes}(\alpha, \beta, e)\}.$$

$$\text{Net-Preconds}(\Sigma, \Pi) = \bigcup_{\alpha \in \Sigma} \{p \in P_\alpha \mid \exists \beta \in (O - \Sigma) \textit{ such that } \text{Establishes}(\beta, \alpha, p)\}.$$

We now give a more precise definition for plan merging. A set Σ of operators is *mergeable* in a plan $\Pi = (O, B)$ if and only if $\exists B', \mu$, where B' is a set of precedence relations on Σ , and μ is an operator, such that

1. Σ can be grouped together in Π,

2. $B \bigcup B'$ is consistent, and in plan $\Pi' = (O, B \bigcup B')$

$$P_\mu \subseteq \text{Net-Preconds}(\Sigma, \Pi'),$$

and

$$\text{Useful-Effects}(\Sigma, \Pi') \subseteq E_\mu.$$

That is, the operator μ can be used to achieve all the useful effects of the operators in Σ while requiring only a subset of their preconditions, after precedence constraints B' is imposed on plan Π.

3. $\text{cost}(\mu) < \text{cost}(\Sigma)$.

The operator μ is called a *merged* operator of Σ in the plan Π, and denoted by $\mu = \text{merge}_\Pi(\Sigma)$ (or simply merge(Σ) if it is clear about the plan Π).

We now explain why the set of precedence relations B' is needed in the above definition. Recall that the set of operators in Σ can be grouped together. Depending on different precedence relations (i.e., B') imposed upon Σ, the set Σ of operators will require different sets of overall preconditions to be achieved by operators outside Σ. Thus, a different choice of B' can give rise to a different merged operator μ. We choose the least expensive one to be our merged operator. For example, consider a plan with two operators α_1 and α_2 that are unordered. The preconditions and effects of the two operators are given in Table 1. The choices in μ are listed below:

Choice 1: $B' = \{\alpha_1 \prec \alpha_2\}$. Then α_1 achieves the precondition q_1 of α_2. Therefore, the net preconditions of Σ should be $\{q_1\}$.

Choice 2: $B' = \{\alpha_2 \prec \alpha_1\}$. Then α_2 achieves the precondition q_2 of α_1. Therefore, the net preconditions of Σ should be $\{q_2\}$.

In each choice above the precondition set is different from the other choice. Thus, each choice of B' may result in a different merged operator μ. In this case, the one with the lower cost is a merged operator.

Operator	Precondition	Effects
α_1	q_1	p_1, q_2
α_2	q_2	p_2, q_1

Table 1: Operator Definition.

The definition for operator merging clearly covers the examples given in the previous section. Below, we show three examples of plan merging according to the definition.

Example 1.

Consider again the multi-gripper blocks world problem with variable sized blocks (See Figure 1). The plan for solving the goals is shown in Figure 2. Suppose that *Can-Grab-A* is a precondition of the operator *moveA*, and *Can-Grab-B* is a precondition of the operator *moveB*. Then the following establishment relations hold in the plan:

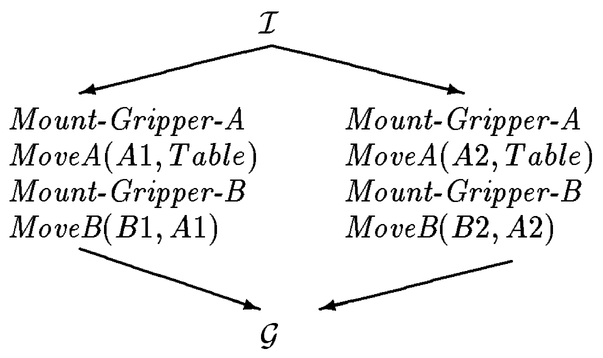


Figure 2: Plans for a blocks world problem.

Establishes(*Mount-Gripper-A*, *moveA*$(A1, Table)$, *Can-Grab-A*),
Establishes(*Mount-Gripper-A*, *moveA*$(A2, Table)$, *Can-Grab-A*),
Establishes(*Mount-Gripper-B*, *moveA*$(B1, A1)$, *Can-Grab-B*),
Establishes(*Mount-Gripper-B*, *moveA*$(B2, A2)$, *Can-Grab-B*), and

where each *Mount-Gripper-A* operator involves unmounting any previously mounted gripper and mount a gripper of an appropriate size. In the plan, the two *Mount-Gripper-A* operators can be merged into a single instance of the operator, so are the *Mount-Gripper-B* operators. This can be verified by the following facts:

1. The two *Mount-Gripper-A* operators can be *grouped* together in the plan, since no other operators are necessarily between them.

2. If we choose B' to be an empty set, then trivially $B \bigcup B'$ is consistent. Moreover, the set Useful-Effects($\{$*Mount-Gripper-A*, *Mount-Gripper-A*$\}$) is exactly $\{$*Can-Grab-A*$\}$, which is identical to the effect of a single *Mount-Gripper-A* operator. Similarly, the subset condition for net preconditions is also satisfied.

3. cost(*Mount-Gripper-A*) $< 2 *$ cost(*Mount-Gripper-A*).

After a similar merging of the two *Mount-Gripper-B* operators, the plan after merging is shown in Figure 3.

Example 2.

As another example, consider a plan for achieving two goals, going to school S from home H and back, and going from home to the grocery store G to get X and back. This plan consists of two subplans,

$$goto(H, S) \prec goto(S, H),$$

$$\begin{tikzcd}
 & \mathcal{I} \arrow[d] & \\
 & \textit{Mount-Gripper-A} \arrow[dl] \arrow[dr] & \\
\textit{MoveA}(A1, \textit{Table}) \arrow[dr] & & \textit{MoveA}(A2, \textit{Table}) \arrow[dl] \\
 & \textit{Mount-Gripper-B} \arrow[dl] \arrow[dr] & \\
\textit{MoveB}(B1, A1) \arrow[dr] & & \textit{MoveB}(B2, A2) \arrow[dl] \\
 & \mathcal{G} &
\end{tikzcd}$$

Figure 3: The merged plan for blocks world.

$$goto(H, G) \prec get(X, G) \prec goto(G, H).$$

The two subplans can be merged by replacing $goto(S, H)$ and $goto(H, G)$ by $goto(S, G)$. If G is located between home H and school S, then the resultant plan

$$goto(H, S) \prec goto(S, G) \prec get(X, G) \prec goto(G, H)$$

costs less than the original one.

In the above example, the set B' of precedence relations is $\{goto(S, H) \prec goto(H, G)\}$. Let Π' be the plan with B' imposed on the original plan, then

$$\text{Net-Preconds}(\Sigma, \Pi') = \{at(S)\},\ \text{Useful-Effects}(\Sigma, \Pi') = \{at(G)\}$$

where $\Sigma = \{goto(S, H), goto(H, G)\}$. Thus, Σ can be merged into the operator $\text{merge}(\Sigma) = goto(S, G)$. Notice that there is another way of merging the operators in this plan, i.e., $\{goto(G, H), goto(H, S)\}$ can be merged into $goto(G, S)$, but one has to decide which way to merge since it is impossible to merge both sets of operators. The ability to optimally choose between several inconsistent possible mergings is an important feature of our dynamic programming algorithm.

Example 3:

Consider the abstract example shown in Figure 4. In this plan, if the operators are defined as in Table 2, the following establishment relation holds:

Establishes(α_1, β_1, p), Establishes(α_2, β_2, p)
Establishes$(\beta_1, \mathcal{G}, q)$, and Establishes$(\beta_2, \mathcal{G}, r)$.

For the set $\Sigma = \{\alpha_1, \alpha_2\}$, the net preconditions are empty, while the useful effects are $\{p\} \cup \{p\} = \{p\}$. If $\text{cost}(\alpha_1) < \text{cost}(\alpha_2)$, then α_1 is the merged operator for Σ. The resultant merged plan is shown in Figure 5.

$$\begin{tikzcd}
 & \mathcal{I} \arrow[dl] \arrow[dr] & \\
\alpha_1 \arrow[d] & & \alpha_2 \arrow[d] \\
\beta_1 \arrow[dr] & & \beta_2 \arrow[dl] \\
 & \mathcal{G} &
\end{tikzcd}$$

Figure 4: An example plan.

$$\begin{tikzcd}
 & \mathcal{I} \arrow[d] & \\
 & \alpha_1 \arrow[dl] \arrow[dr] & \\
\beta_1 \arrow[dr] & & \beta_2 \arrow[dl] \\
 & \mathcal{G} &
\end{tikzcd}$$

Figure 5: A merged plan.

Operator	Precondition	Effects
α_1		p
α_2		p
β_1	p	q
β_2	p	r
$\mathcal{I}$		$\neg p, \neg q, \neg r$
$\mathcal{G}$	q, r	

Table 2: Operator Definition.

Let Π be a plan and Σ be a set of operators in Π that are mergeable. Then after merging Σ in Π, every precedence relation in Π that involves an operator $\alpha \in \Sigma$ and an operator $\beta \notin \Sigma$ is replaced by the relation with a modification: every occurrence of α is replaced by μ. Let the new set of precedence relations be *NewB*. Formally, let α, β be operators in $O(\Pi)$ such that $(\alpha \prec \beta) \in B$, then

$$NewB = \begin{cases} \alpha \prec \beta \text{ if } \alpha, \beta \notin \Sigma \\ \mu \prec \beta \text{ if } \alpha \in \Sigma, \beta \notin \Sigma \\ \alpha \prec \mu \text{ if } \alpha \notin \Sigma, \beta \in \Sigma \end{cases}$$

The plan $\Pi_1 = (O_1, B_1)$ with the operators in Σ merged into μ is defined as a plan $\Pi_2 = (O_2, B_2)$, in which

1. $O_2 = (O_1 - \Sigma) \bigcup \{\mu\}$, and
2. B_2 is *NewB*.

The merged plan is denoted as $\Pi_2 = \text{merge}(\Pi_1, \Sigma, \mu)$.

Operators can be merged in both correct and temporarily incorrect plans. A plan $\Pi = (O, B)$ is *correct* if B is a partial ordering relation, every precondition of every operator in O is necessarily established, and all goals are necessarily achieved. Operator merging, as defined here, is not a means of making an incorrect plan correct, but rather to make a plan more efficient. Therefore, in general, we do not require that an incorrect plan be made correct after merging. However, we would like to ensure that after merging, the correctness of a plan is preserved, given that certain conditions hold. We explain one such condition next.

Recall that after merging, the effects of the merged operator μ must include all useful effects of operators in set Σ. In addition to the useful effects, there may also be other side effects that μ achieves. To preserve plan correctness during merging, we could require that no side effects of μ negate the "establishment structure" of the original plan. This is stated formally as:

$\forall q \in (E_\mu - \text{Useful-Effects}(\Sigma, \Pi)), \forall \alpha_1, \alpha_2 \notin \Sigma$ *and a condition* p
such that $\text{Establishes}(\alpha_1, \alpha_2, p)$ *holds in* Π.
if $\neg(\mu \prec \alpha_1)$ *and* $\neg(\alpha_2 \prec \mu$, then $q \neq \neg p$. (1)

That is, if μ can possibly be between α_1 and α_2 after merging, and α_1 establishes a precondition p of α_2, then no extraneous effect of μ negates p.

The following theorem holds given that Condition (1) holds.

Theorem 2.1 *Let* Π_1 *be a correct plan and* Σ *be a set of operators in* Π *that are mergeable. Let* μ *be a merged operator of* Σ. *Suppose that* E_μ *satisfies Condition (1). Then* $\Pi_2 = \text{merge}(\Pi_1, \Sigma, \mu)$ *is also correct.*

Proof: We prove by contradiction, showing that if Π_2 is incorrect, then Π_1 is also incorrect.

Suppose that Π_2 is incorrect. Then from the definition of correctness of plans, it must have a total order $\overline{\Pi}_2$ that is also incorrect. From $\overline{\Pi}_2$, we construct a total order of operators in Π_1, by replacing the operator μ in $\overline{\Pi}_2$ by a total order of Σ consistent with the ordering constraints B' in the definition for operator merging. The resulting operator sequence $\overline{\Pi}_1$ is a total order of the operators in Π_1 because all operators of $\overline{\Pi}_1$ are also operators of Π_1, and B' is consistent with the precedence relations in Π_1. Let α_1 be the first operator, and α_n be the last operator, of the operators in Σ that appears in $\overline{\Pi}_1$. From the definition for $NewB$ above, the relationship between $\overline{\Pi}_1$ and $\overline{\Pi}_2$ is as follows:

1. The subsequence of operators of $\overline{\Pi}_1$ before α_1 is identical to that of $\overline{\Pi}_2$ before μ.

2. The subsequence of operators of $\overline{\Pi}_1$ after α_n is identical to that of $\overline{\Pi}_2$ after μ.

Now we show that $\overline{\Pi}_1$ is incorrect.

Because $\overline{\Pi}_2$ is assumed to be incorrect, there must be an operator β in $\overline{\Pi}_2$ with a precondition p such that no operators in $\overline{\Pi}_2$ *establish* p for β. From the definition of establishment, there can be two cases where this is true.
Case 1: there is no operator $\alpha \prec \beta$ in $\overline{\Pi}_2$ such that $p \in E_\alpha$. Case 1 can be further split into three possibilities:

Case 1(a): $\beta \prec \mu$ *in* $\overline{\Pi}_2$. Since the subsequence of operators before μ in $\overline{\Pi}_2$ is identical to that before α_1 in $\overline{\Pi}_1$, the precondition p of β in $\overline{\Pi}_1$ is also unestablished. That is, $\overline{\Pi}_1$ is incorrect.

Case 1(b): $\beta = \mu$ *in* $\overline{\Pi}_2$. Since $P_\mu \subseteq \text{Net-Preconds}(\Sigma, \Pi_1)$, this case implies that there is some operator $\beta' \in \Sigma$, such that a precondition p of β' is not asserted by any previous operators in $\overline{\Pi}_1$. Therefore, $\overline{\Pi}_1$ is also incorrect.

Case 1(c): $\mu \prec \beta$ *in* $\overline{\Pi}_2$. Since $\text{Useful-Effects}(\Sigma, \Pi_1) \subseteq E_\mu$, this case implies that no operator in Σ, or before α_1 in $\overline{\Pi}_1$ has an effect p either. Therefore, $\overline{\Pi}_1$ is also incorrect.

Case 2: For some β with precondition p in $\overline{\Pi}_2$, and for the last operator α before β with p as an effect, there is an operator γ such that $\alpha \prec \gamma$, $\gamma \prec \beta$, and $\neg p \in E_\gamma$.

Case 2 can be split into four different situations:

Case 2(a): *α, β and $\gamma \neq \mu$.* This implies that α, β and γ are also operators in $\overline{\Pi}_1$ that has exactly the same relative ordering as in $\overline{\Pi}_2$. Also, recall that α is the last operator in $\overline{\Pi}_2$ that has p as an effect. Thus, from the conditions that Useful-Effects$(\Sigma, \Pi) \subseteq E_\mu$, and that μ does not establish p for β in $\overline{\Pi}_2$, no operator in Σ can re-establish p for β in $\overline{\Pi}_1$ either. Thus, p is not established in $\overline{\Pi}_1$, and as a consequence, $\overline{\Pi}_1$ is incorrect.

Case 2(b): *$\alpha = \mu$.* Then there is an operator α' that is the last operator in $\overline{\Pi}_1$ such that (1) $\alpha' \prec \alpha_n$ or $\alpha' = \alpha_n$, and (2) $p \in E_{\alpha'}$. However, since $\mu \prec \gamma \prec \beta$ in $\overline{\Pi}_2$, $\alpha' \prec \gamma \prec \beta$ in $\overline{\Pi}_1$. Hence the precondition p of β is not established. Thus, $\overline{\Pi}_1$ is incorrect.

Case 2(c): *$\beta = \mu$.* Since $P_\mu \subseteq$ Net-Preconds(Σ, Π_1), in plan $\overline{\Pi}_1$, there is an operator β' in Σ with precondition p that is not established by operators within Σ. Since the ordering $\alpha \prec \gamma \prec \beta'$ also holds in $\overline{\Pi}_1$, the precondition p of β' is not established in $\overline{\Pi}_1$ either. Therefore, $\overline{\Pi}_1$ is incorrect.

Case 2(d): *$\gamma = \mu$.* Then $\neg p \in E_\mu$. Since μ satisfies Condition (1), $\neg p$ must be a member of Useful-Effects(Σ, Π_1). But this means that for some operator γ' in Σ, $\neg p \in E_{\gamma'}$. Thus, the ordering $\alpha \prec \gamma' \prec \beta$ in $\overline{\Pi}_1$ implies that the precondition p is not established for operator β. Therefore, $\overline{\Pi}_1$ is incorrect.

To sum up, both Case 1 and 2 implies that $\overline{\Pi}_1$ is incorrect. As a result, Π_1 is also incorrect, contradicting to the initial assumption of the theorem. □

Condition (1) trivially holds when the merged operator μ has no side effects. That is, when $E_\mu =$ Useful-Effects(Σ, Π). Thus, the following corollary holds:

Corollary 2.2 *If $E_\mu =$ Useful-Effects(Σ, Π), and Π is correct, then* merge(Σ, Π, μ) *is also correct.*

The merging examples 1, 2, and 3 presented earlier all satisfy the equality condition of the corollary.

A second special case of the theorem is when no side effect q of the merged operator μ deny any precondition p of any operator α_2 that are *possibly* after μ in merge(Σ, Π, μ). This special case satisfies the condition

$$\forall \alpha_2,\ \forall p \in P_{\alpha_2}.\ \text{if possibly } \mu \prec \alpha_2 \text{ then } q \neq \neg p. \qquad (2)$$

This condition logically implies the implication in Condition (1). Thus,

Corollary 2.3 *If Condition (2) is satisfied, and Π is correct, then* merge(Σ, Π, μ) *is also correct.*

To illustrate the application of the corollary, consider again the blocks world problems with variable sized blocks and grippers. If a new gripper GripperC is introduced that can grab any block of sizes A, B, C, then even though the problem at hand only concerns blocks of size A and B, the additional side effects of the new operator would include an additional side-effect *Can-Grab-C*. Since this side effect is harmless to the preconditions of operators for moving $A1$ and $B2$, *etc.*, correctness of a plan is preserved after merging.

It can be further shown by induction that the correctness of a plan is preserved by merging operators in a plan any number of times, as long as the conditions in the theorem or corollaries are satisfied.

2.2 Finding Mergeable Operators

There are two important issues in optimizing Π via plan merging. The first one is to find the set Σ of mergeable operators and, for each set of mergeable operators, find one or more merged operators μ. The second issue is computing the optimal way to merge the plan, given that several sets of merged operators are found.

The problem of finding operators that can be merged in a plan Π may be a computationally expensive process if no additional domain knowledge is given, since then it would be necessary to examine the useful effects and net preconditions of every subset of operators. To make the process more efficient, various domain knowledge can be employed. For example, one way for the operators in Σ to be merged is when they contain various sub-operators which cancel each other out, in which case the merged operator μ would correspond to the set of operators in Σ with these sub-operators removed. In manufacturing domains where it is desirable to minimize set-up costs, this situation occurs often and can be profitably employed [11]. This case also corresponds to what Wilensky calls "partial-plan merging" [17]. Another case is when all the operators in Σ share a common schema, and in this case, the goals of these operators can be achieved by executing the schema only once. This case corresponds to what Wilensky calls "common-schema merging."

In this paper, we assume that knowledge is available about what operators can be merged, and for each set of these operators, what the merged operators are. Given this knowledge, we concentrate on the second issue, that of finding and analyzing methods for computing the optimal plan. We will call this problem the *plan merging problem*. We start by discussing its complexities in the next section.

2.3 Complexity

Several complications exist that make the plan merging problem computationally expensive in general. First of all, for a given set Σ of operators to be merged, there may be several alternative merged operators, $\{\mu_1, \ldots, \mu_i\}$, to choose from, each with a different set of preconditions, effects and cost value. Second, an operator may lie in the intersection of several non-identical groups of operators, but not all operators in these groups may be merged, even though all the operators in question are unordered in a plan. For example, in the blocks world domain, there may be a gripper capable of picking up blocks of sizes A and B, and another gripper capable of picking up blocks of sizes A and C, but no gripper that can pick up a block of type B and C. Then a gripper-changing operator for picking up a block of type A may be merged with ones for either B or C, but not all three can be merged together. An optimization algorithm has to make a choice in such a situation. A third complication occurs because the partial order on Π may render inconsistent some pairs of mergings. For example, consider the two plans

$$a_1 \prec c_1 \prec b_1 \text{ and } a_2 \prec b_2 \prec c_2,$$

where operators can have three types, a, b or c. Thus, after operator merging, one merged plan is

$$\text{merge}(\{a_1, a_2\}) \prec b_2 \prec \text{merge}(\{c_1, c_2\}) \prec b_1,$$

another merged plan is

$$\text{merge}(\{a_1, a_2\}) \prec c_1 \prec \text{merge}(\{b_1, b_2\}) \prec c_2.$$

However, it is impossible to merge both pairs b_1, b_2 and c_1, c_2.

To remove the first complication, we assume that for each set Σ of mergeable operators, there is a unique merged operator μ. In addition, we assume that operator merging is an atomic action, so that the merged operator μ may not be combined with any other operators. The second complication is naturally resolved by our definition of operator merging; since each merged operator has an associated cost, at all times we know which set of merging will result in a least-cost plan. This set of merging then is all we are interested in.

We now consider the computational complexity as a result of the third complication. The problem is to decide which set of mergeable operators to merge, if temporal orderings prevent all of them from being merged together. To see the complexity involved, consider a simplified plan Π, consisting of a set of linear sequences of operators. Let the k linear input plans S^1, S^2, ..., S^k, each of which is a linear sequence of operators. Each sequence S^i is denoted as $s^i_1 s^i_2 \dots s^i_{|S^i|}$. A supersequence $U = u_1 \dots u_P$ of $S = s_1 \dots s_M$ has the property that there exist $i_1 < \cdots < i_M$ such that $u_{i_1} = s_1, \dots, u_{i_M} = s_M$. In other words, the operators of S can be found in order in the sequence U. We say that U is a common supersequence of S and T if it is separately a supersequence of S and of T. Elements of S and T may overlap in a common supersequence U, so that $|U| \leq |S| + |T|$. For any set of input sequences, there exists at least one SCS.

As a special case, let an operator sequence of length N have cost N. In this case, the optimal operator merging for an input plan is the shortest common supersequence of the linear input operator sequences, S^1, S^2, ..., S^k. Under some formulations the problem of finding the SCS is NP-complete [3]. However, in planning the total number of final goals to be achieved usually stays constant. As a result, for the naturally occurring case of a fixed number of input sequences, the SCS may be simply calculated in polynomial time.

We next consider the problem of finding an optimally merged plan using dynamic programming. For simplicity, we first consider in the next section plans that are linear sequences of operators and apply dynamic programming directly to solve the problem. In Section 4 we extend the algorithm to handle general partially ordered plans as well.

3 Optimal Plan Merging for Linearly Ordered Plans

Assume that the input plan Π consists of k linear sequences of operators. If S^i is a sequence, then $S^i_{1\dots i_l}$ denotes the subsequence of S^i from the first to the l^{th} operator. Consider also a k-dimensional array M. To dimension i of M assign the subsequence costs of S^i. That is,

$$M(0,0,\dots,0,j,0,\dots,0) = \text{cost}(S^i_{1,\dots,j}),\ j = 1,2,\dots|S_i|$$

where the index j appears on the i^{th} dimension. The matrix A has a size of $(|S^1|+1)\times(|S^2|+1)\times\dots\times(|S^k|+1)$. M will be used to represent the least cost plan costs of partial inputs, so that $M(i_1,\dots,i_k)$ is the optimal cost of merging $S^1_{1\dots i_1},\dots,S^k_{1\dots i_k}$. Define also the identically-sized array R, which will be used to represent the components of the actual index set where merging occurs. Therefore, after the optimal computation, the element $M(|S_1|,|S_2|,\dots|S_k|)$ contains the optimal cost of merging all plans, and $R(|S_1|,|S_2|,\dots,|S_k|)$ contains a set of elements, where each element is a subset of indices denoting where an operator merging should occur in the optimally merged plan.

For a set of indices $i_1,i_2,\dots,i_k$, let Σ be the set of index pairs $\{(1,i_1),(2,i_2),\dots,(k,i_k)\}$. Let $\sigma\subseteq\Sigma$ be the index pairs of operators, such that all operators in $ops_\sigma = \{s^j_{i_j}|(j,i_j)\in\sigma\}$ can be merged, The minimal costs in M is computed using the dynamic programming principle, which states the following:

$$M(i_1,\dots,i_k) = \min_{\sigma\in\Sigma}\{\text{cost}(\text{merge}(\sigma)) + M(i_1-\delta_1,\dots,i_k-\delta_k)\},$$

where each δ_i is 1 if and only if i_j appears in σ. Otherwise δ_i is 0. This recurrence forms the basis for the inner loop of the optimal plan merging algorithm. As initial conditions, let $M(0,\dots,0)=0$ and assume that $M(i_1,\dots,i_k)=\infty$ if any $i_j<0$. Also, $R(0,\dots,0)=\emptyset$.

```
Compute the cost $M(i_1,\dots,i_k)$ and indices $R(i_1,\dots,i_k)$ where merging occurs:
for $i_1 = 0$ to $|S^1|$ do
    ⋮
    for $i_k = 0$ to $|S^k|$ do
        minval = $\infty$
        for all $\sigma \subseteq \{(1,i_1),(2,i_2),\dots,(k,i_k)\}$ such that
        the operators $ops_\sigma = \{s^j_{i_j}|(j,i_j)\in\sigma\}$ can be merged,
            for $j = 1$ to $k$ do
                $\delta_j = \begin{cases} 1 & \textbf{if } j\in\sigma \\ 0 & \text{otherwise}\end{cases}$
            endfor
            if minval > cost(merge(operators_σ)) + $M(i_1 - \delta_1, \ldots, i_k - \delta_k)$ then
               minval = cost(merge(operators_σ)) + $M(i_1 - \delta_1, \ldots, i_k - \delta_k)$
               $R(i_1, \ldots, i_k) = R(i_1 - \delta_1, \ldots, i_k - \delta_k) \bigcup \{\sigma\}$
            endif
         endfor
         $A(i_1, \ldots, i_k)$ = minval
      endfor
      ⋮
endfor
```

The key steps are the location of the contributing sequence indices σ, updating the cost to reflect cost of the minimal cost merged plan, and setting the optimal mergeable indices in R. The matrix cell $R(|S^1|, \ldots, |S^k|)$ contains a set of index sets, where each index set $\{(j_1, i_{j_1}), (j_2, i_{j_2}), \ldots, (j_m, i_{j_m})\}$ indicates that the operators in $ops_\sigma = \{s^j_{i_j} | (j, i_j) \in \sigma\}$ should be merged in the final optimally merged plan. Notice that once all operators are merged as indicated, the resultant plan is a partially ordered one in general, since after merging the newly imposed ordering constraints for operator merging only linearizes a portion of the plan Π, but leaving all other parts unordered as before.

The cost of the algorithm is $O(\prod_{i=1}^k |S^i|)$, assuming a fixed number k of input sequences.

4 Dynamic Programming Methods for Partially Ordered Inputs

Just as the dynamic programming method can be used to compute an optimal plan from two or more linear input sequences, a similar computation determines the optimal merged plan from a partially ordered input plan. We consider as input a plan Π, partially ordered by $\prec$. The method we present creates the optimal merged plan from Π.

Recall that the dynamic programming method for linear plans pushes forward a frontier that ranges across all plans. The frontier marks a set of operators that could be merged (See Figure 6). The method then decides on the best merge up to the frontier by comparing different subsets of possible merges and previously computed best merges. With a partially ordered plan Π, we could push a frontier forward in a similar manner. As well, the frontier at all times crosses a set of plan segments that are also linear. Notice that the frontier at any time crosses a *maximally unordered* of linear plan segments. As an example, the current frontier in the partially ordered plan Π of Figure 7 crosses plan segments $\{A_1, B_2, C_2\}$, each of which is a linear sequence of operators. Thus, we could apply directly the dynamic programming principle to compute the optimal cost up to the frontier based on possible merges along the frontier as well as computation result just before the frontier. However, with partially ordered plans, a complication arises when two or more branches of linear plan

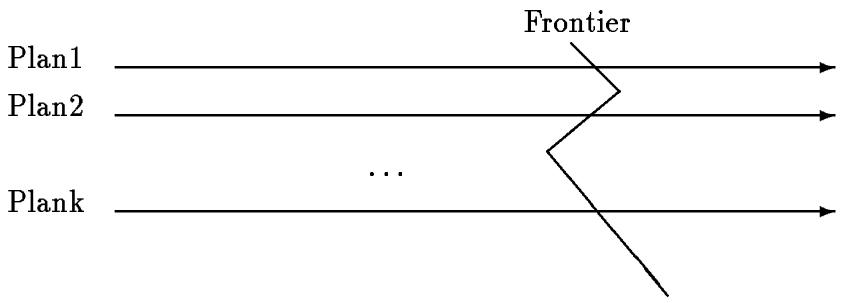


Figure 6: A set of linear plans.

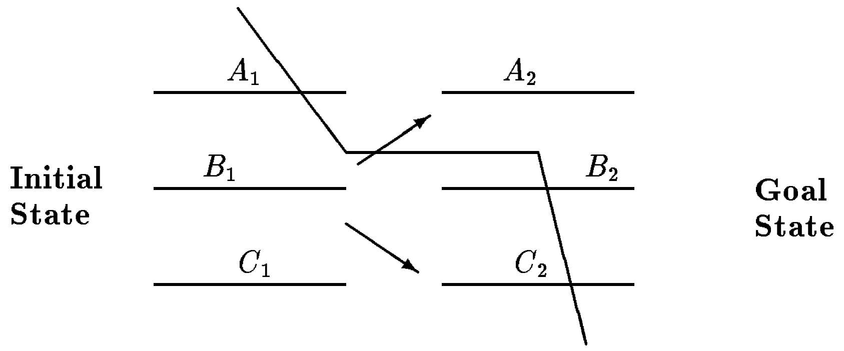


Figure 7: A partially ordered plan.

segments directly precede another set of segments. For example, in Figure 7 the segment B_1 precedes not only B_2, but also A_2 and C_2. To ensure optimality, the partially computed results has to be transferred from one set of segments to the next. The method below deals mainly with this difficulty.

Informally, the new method first converts a set of partially ordered plans into a set of linear plan segments that are partially ordered. Then it computes the set of maximally unordered sets of the plan segments, and transform the original plan into a dual graph whose nodes are the maximally unordered sets. The dual graph is a directly acyclic graph. Finally, it applies the dynamic programming algorithm for linear plans as a subroutine, by systematically computing the optimally merged plan in a topological order of the dual graph. In the process, care is taken to transfer the result of merging the subplans from one frontier to the next. Below, we discuss each step in detail.

4.1 Conversion to Linear Plan Segments

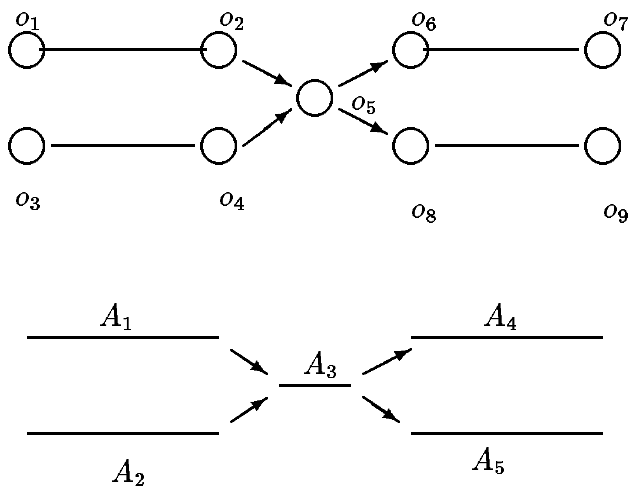


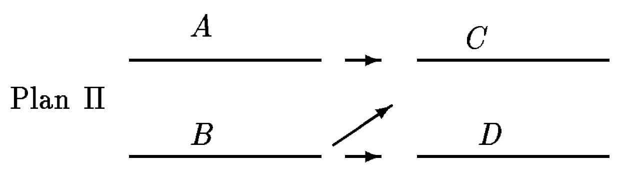
Figure 8: Conversion to linear plan segments.

The subsequent computation requires that Π is represented by an augmented plan consisting of a set of linearly ordered plan segments, such that the plan segments are partially ordered and no two adjacent segments share an operator. An ordering relation exists between two linear plan segments if every operator of the first segment precedes that of the second. Every plan can be easily converted to one with this property. Consider the plan in Figure 8, where a complication arises because operator o_5 has two predecessors and two successors. However, o_5 by itself can be considered as a single plan segment. Thus, the whole plan in Figure 8 has five linear plan segments that are partially ordered: A_1, A_2, A_3 which contains o_5 alone, A_4 and A_5 (See Figure 8). The converted plan is hereafter referred simply as Π.

4.2 Conversion to Dual Graph

As mentioned earlier, the dynamic programming algorithm for linear plans will be applied to the individual linearly ordered plans. Any operator merging can only take place between unordered operators, which must come from unordered linear plans. Conversely, any set of unordered operators is a candidate for merging. After the conversion from last section, ordering constraints exist between the set of linear plan segments. Let ω represent a set of linear plan segments from Π, such that all plan segments in ω are unordered with respect to one another. We say that ω is *maximally unordered* if every linear segment in $\Pi - \omega$ is ordered by $\prec$ with respect to ω. For example, in the plan Π in Figure 7, the maximally unordered sets of plan segments are $\{A_1, B_1, C_1\}$, $\{A_1, B_2, C_1\}$, etc.

For a given plan Π the set of all maximally unordered sets, along with the ordering relations among them, defines a *dual graph* corresponding to Π. Each node in the dual graph *DualGraph*(Π) corresponds to a maximally unordered set of plan segments, and an



$$\textit{DualGraph}(\Pi)\colon \{A, B\} \longrightarrow \{A, D\} \longrightarrow \{C, D\}$$

Figure 9: A plan and its dual graph.

arc between two nodes n_1 and n_2 in the graph exists if a plan segment in n_1 precedes a plan segment in n_2. Figure 9 shows the dual graph of the plan in the same figure. As a special case, a plan Π is a set of linear sequences S_i, $i = 1, 2, \ldots, n$. Then the dual graph has only one node, consisting of the linear sequences themselves.

The size of the dual graph can be exponential in the worst case, in which the plan Π takes the shape of a tree. But in general, the worst case doesn't happen frequently. For example, if each plan segment in a plan Π is unordered with at most C other plan segments, then the time complexity for constructing the dual graph is $O(N)$, where N is the total number of plan segments in Π. This is a realistic assumption in nonlinear planning, since the number of unordered linear plan segments generally corresponds to the total number of goals to be achieved, or the total number of preconditions of an operator. The goals are fixed for each set of plans, and the number of preconditions of each operator is also bounded by a constant. Furthermore, in a planning domain, interactions among operator preconditions and effects usually require the imposition of ordering constraints, which greatly reduces the number of unordered plan segments. Therefore on the average, the complexity of converting a plan to its dual graph is polynomial in the number of plan segments.

4.3 Boundary Conditions

Consider Figure 9, where there are four linear plan segments A, B, C, and D that are partially ordered. The dynamic programming algorithm for partially ordered plans will apply its linear plan version to the nodes of this dual graph in a topological order. The nodes of the graph are sets of linear plan segments. Suppose that instead of returning an optimal cost of merging, the dynamic programming algorithm for linear inputs returns an entire matrix of computation. For a given set of unordered linear plan segments n, let the output of the dynamic programming algorithm be *DP*(n), which is a multi-dimensional matrix. The first

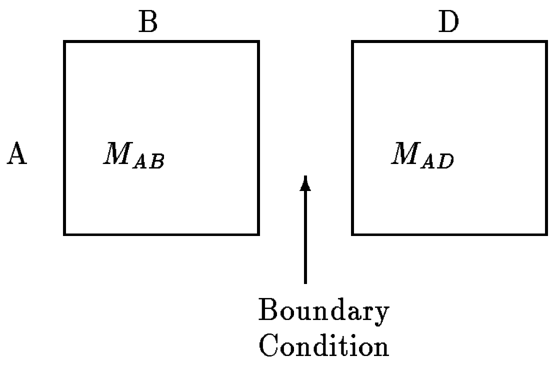


Figure 10: Boundary Conditions.

application to the dual graph in Figure 9 is straightforward, since it involves only merging A and B. Let the result of the first merging be $DP(\{A, B\})$, which is a two dimensional matrix. When the next merging starts with A and D, however, the algorithm must take into account that B is before D in the chronological ordering of Π, and thus results of the previous merging must be used to ensure optimality. Recall that merging A and B will produce a two dimensional matrix M_{AB} of size $(|A| + 1) \times (|B| + 1)$, in which each location (i, j) contains the cost of the lowest cost merged plan segments $A_{1 \ldots i}$ and $B_{1,\ldots,j}$. If we're only seeking to merge A and B, then the output is the value at the lower right corner of M_{AB}, plus the matrix R_{AB} which records the indices where merging happened. However, if we seek to continue merging beyond this set of plans, then we take the entire lower or right side of the matrix as initial conditions for the next match. This is because when doing the next merging, all possible combinations of the previous merging must be taken into account. In the previous example, if we intend to compute $DP(\{A, D\})$ after $DP(\{A, B\})$, then we first form a two dimensional matrix $M_{AD}(i, j), i = 0, 1, 2, \ldots |A|, j = 0, 1, 2, \ldots |D|$. Along the D direction we assign the costs of subplans in D without merging with any of A's operators. That is, $M(0, j) = \text{cost}(|D(j)|$. But along the A direction we assign the results of merging $A_{1,\ldots,i}$ and B without also merging with any of D's operators. That is,

$$M_{AD}(i, 0) = M_{AB}(i, |B|).$$

This corresponds to taking the right side of the first matrix as the initial condition for merging A and D. Likewise, when B and C are merged, the initial conditions along the B direction is provided by the bottom side of the M_{AB} matrix.

In general, let n be a node in *DualGraph*(Π) corresponding to the merging of plan segments $\{A_i, i = 1, 2, \ldots k\}$. Let $M(i_1, i_2, \ldots i_k)$ be the resultant matrix after merging the plans by the dynamic programming algorithm, with $0 \leq i_j \leq N_j$. Let $Subset = \{A_{u_1}, A_{u_2}, \ldots A_{u_l}\}, l \leq k$, be a subset of n. We define the projection of n on $Subset$ as

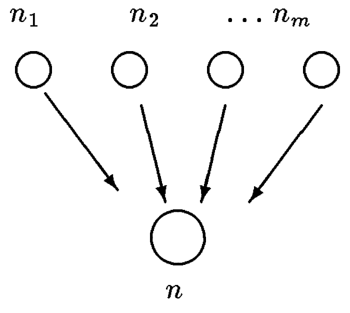


Figure 11: A node n and its predecessors in a dual graph.

a matrix:

$$Proj(n, Subset) = M_n(N_1, N_2, \ldots, i_{u_1}, N_{u_1+1}, \ldots, i_{u_l}, N_{u_l+1}, \ldots N_k), \ where \ 0 \leq i_{u_j} \leq N_{u_j}.$$

That is, the projection is obtained from M by replacing all indices outside the subset by their limits, while letting all indices within the subset free. This is a matrix of dimension $(N_{u_1} + 1) \times (N_{u_2} + 1) \times \ldots \times (N_{u_l} + 1)$. In the example of Figure 10, $Proj(M_{AB}, \{A\}) = M_{AB}(i, |B|)$, which corresponds to the right column.

With the definition of projections, we now explain how the results of merging is passed from one set of nodes to the next, in the dual graph *DualGraph*(Π) of plan Π. Let $n_i, i = 1, 2, \ldots m$ be the predecessor of node n in the dual graph *DualGraph*(Π) (see Figure 11). Then the initial condition assignment for computation at node n is given as follows:

```
Algorithm Set-Initial-Condition$(n, \{n_i, i = 1, 2, \ldots m\})$.
for each $n_i, i = 1, 2, \ldots m$ do
    Subset$_i$ := $n \bigcap n_i$
    Let Subset$_i$ be $\{A_{i_1}, A_{i_2}, \ldots, A_{i_l}\}$,
    $M_n(0, 0, \ldots, i_1, 0, \ldots, i_2, \ldots, i_l, 0, \ldots, 0) := Proj(n_i, \text{Subset}_i)$,
      where $0 \leq i_j \leq N_{i_j}$, for $j = 1, 2, \ldots l$.
endfor
```

As an example, consider the node $\{C, D\}$ in Figure 9. According to the above algorithm, the initial condition is $M_{CD}(i, 0) = Proj(M_{BC}, \{C\}) = M_{BC}(i, |B|)$.

4.4 Computation of Plan Merging

Given the above method for computing the initial conditions, a dynamic programming algorithm scans the dual graph of Π by computing the merging of plan segments in a topological order. At each node, the initial condition for the dynamic programming computation is set using previous results by projection. The algorithm is presented formally below.

```
Comment: $M_n = \mathrm{DP}(n)$ is a matrix after each processing of node $n$.
1. Perform a topological sort of $DualGraph(\Pi)$. Let the resultant sorted list be $L$.
while $L$ is not empty do
        2. Let $n$ be the first node in $L$. Remove $n$ from $L$.
        3. Let $n_i, i = 1, 2, \dots m$ be the parents of $n$ in $DualGraph(\Pi)$
        4. Apply Set-Initial-Condition$(n, \{n_i, i = 1, 2, \dots m\})$.
        5. Apply dynamic programming algorithm: $M_n := DP(n)$.
endwhile
```

For simplicity of discussion, we have so far assumed that our computation only concerns the minimal cost of optimal merging. Let r be the last node in the topologically sorted list L. Then this minimal cost is stored in $M_r(N_1, N_2, \dots, N_k)$, where $N_i, i = 1, 2, \dots k$ are the lengths of the plan segments in r. In a similar fashion, we could extend our algorithms to compute the optimal merged plan as well. Recall that in the linear input case, a matrix R is used to record a set of index sets, indicating the subsets of operators that should be merged to yield an optimal plan. With partially ordered inputs, we could likewise associate with each node n in the dual graph $DualGraph(\Pi)$ a matrix R_n, which records sets of operator identifiers where merging should occur. Then the boundary conditions of R_n can be computed in exactly the same manner as M_n, that is, via projection computation from previously computed R_{n_i}. Suppose that we augment the function $DP(n)$ so it returns a pair of matrices: (M_n, R_n), then in the dynamic programming algorithm above, we can modify step 5 to:

5. $(M_n, R_n) := DP(n)$.

When the algorithm terminates, the operators to be merged optimally in plan Π can be found in the matrix cell $R_r(N_1, N_2, \dots, N_k)$. If the set of indices $\{i_1, i_2, \dots i_l\}$ is associated with node $n_i = \{A_1, A_2, \dots A_l\}$, then we know that in the optimal merged plan, the operators $\{A_j(i_j), j = 1, 2, \dots, l\}$ should be merged to yield the optimal cost. In this way, we could find out all operators that are to be merged in the optimal merged plan. In the merged plan no additional ordering constraints are imposed except those required by merging. As a result, the merged plan is also partially ordered.

Finally, we discuss the properties of the algorithm. The algorithm enumerates all maximally unordered sets of operators in a partially ordered plan in a systematic way. By setting the initial conditions of each matrix during the operation of the algorithm, all previously computed best merges are considered when deciding the current set of best merging operations. Therefore, the merged plan produced by the algorithm will have an optimal cost, according to the dynamic programming principle.

The algorithm also degenerates to the dynamic programming algorithm for linear inputs in a trivial way. If the plan Π consists of several linear sequences of operators that are unordered with each other, then the dual graph $DualGraph(\Pi)$ simply has one node, containing the set of all sequences. The dynamic programming algorithm DP for linear plans is applied only once to this node, equivalent to an application of DP to the sequences directly.

5 Approximation Algorithms

For problems of large sizes, the complexity of the dynamic programming methods may be too high to be useful. An alternative choice is to use approximation algorithms that operate in low-order polynomial time but output a suboptimal supersequence. In the past, many planning systems [13, 18, 4] have resorted to the application of greedy algorithms for plan merging. Thus, a unresolved but important issue is to determine the qualities of these algorithms in worst and average cases. In this section, we consider a spectrum of algorithms and their analysis, and in the next section, we show the experimental results.

For simplicity of mathematical analysis, we first assume the following input and output of each algorithm:

Input: A set Π of plans, which are assumed to be a set of k linear sequences of operators, each sequence has n operators.

Output: A merged plan S for Π.

In addition, it is also assumed that every operator in plan Π is of one of m different types, $\alpha_1, \alpha_2, \ldots, \alpha_m$. A set of operators can be merged if and only if they are of the same type. In blocks world where blocks can have different sizes, this assumption requires that for a block of a given size, there is a corresponding gripper for picking it up, and no gripper can pick up blocks of different sizes. Finally, we assume that the cost of a set of operators equals the total number of operators in that set. Therefore, if N operators of the same type are merged together, the cost reduces from N to one. Thus, the merged plan S is a supersequence of the plans in Π. The above assumptions are for the purpose of simplifying the presentation of the algorithms and their analysis. In the next section, we relax these assumptions.

As stated above, our key motivation in this section is to develop and analyze approximation algorithms that perform in low-order polynomial time. In fact, the algorithms to be presented all have linear time complexity in the number of operators of the input plans. Thus, the main concern here is not the time complexity of the algorithms, but the quality of the merged plan produced by each algorithm.

Given a plan $\Pi = (O, B)$, one can find a set of operators that can be performed in the initial situation. Denote this set by Start(Π). Formally,

$$\text{Start}(\Pi) = \{\alpha \in O \mid \neg\exists\beta \in O,\ \beta \prec \alpha.\}$$

Let Σ be a set of operators in a plan $\Pi = (O, B)$. The updated plan remove(Σ, Π) is the plan with all operators in Σ removed, and all the precedence relations relevant to operators in Σ removed.

For ease of exposition, it is assumed that all plans are arranged in a "left to right" way, so that they start from the left and end at right. All algorithms below basically operate by sweeping through the input plans in a left to right manner. In the sweeping process, operators that are mergeable are merged and collected into a superplan. Although the description of the algorithms and subsequent analysis assume the left to right way of merging, all the results apply equally to merging in the opposite direction, *i.e.*, in a right to left way. Such a process does not violate any existing precedence constraint in Π.

The algorithms we will discuss are all simple. They even look very similar. But there are subtle differences. These subtle differences give very different results. Analyzing these subtle differences helps us to choose good algorithms in practice wisely. Below, the algorithms are discussed in order of increasing sophistication.

5.1 Algorithm M1

Our first algorithm, M1, is the most greedy one. It looks for as many merges as possible in each iteration. In particular, it takes an operator on the left side of the remaining plan Π, and looks for nearest merges by searching through each of the next plans from left to right for operators that can be merged with α. The operators that are mergeable with α can be considered as forming a "thread" that partitions the plan Π into three subplans, where Π_{11} is on the left of the thread, Π_{12} is on the right of the thread, and Π_2 is the set of those sequences not touched by the thread. Below is a recursive version of the algorithm.

Algorithm M1.

1. If $\Pi = \emptyset$ then return $\emptyset$. Otherwise, arbitrarily find $\alpha \in \mathrm{Start}(\Pi)$. Let Σ be a leftmost maximal set of operators in Π mergeable with α. Let μ be the merged operator of Σ.
2. Partition Π into two sets of sequences, Π_1 and Π_2, such that each sequence in Π_1 contains an operator in Σ, and no operator in any sequence of Π_2 is a member of Σ.
3. For each operator sequence T in Π_1, let α' be the operator in Σ. Split T at α' into T_1 and T_2, so that $T = T_1\alpha' T_2$. Let Π_{11} be the set of all T_1, and Π_{12} be the set of all T_2.
4. Return $\mathrm{M1}(\Pi_{11} \bigcup \Pi_2); \mu; \mathrm{M1}(\Pi_{12})$, where ";" stands for concatenation.

Theorem 5.1 *Algorithm M1 has worst case cost* $\Theta(n^2)$.

Proof: We will consider plans with just two operators α, β. We will construct n plans each being a sequence of n operators from $\{\alpha, \beta\}$, arranged as rows in an $n \times n$ matrix M. We will show that M1 will construct a supersequence of length $\Omega(n^2)$ for some M. Let the first row of M contain entirely α. Row i of M contains $n-i$ initial operators α, followed by $i-1$ operators β, and then a terminal α. At step i of the algorithm, M1 removes column i from the first $n-i$ rows, and the entirety of row $n+1-i$. M1 is then applied to an equivalent submatrix of size $n-i \times n-i$. Each merge requires $n+1-i$ operators in the merged plan, for a total cost of $n(n+1)/2$. Concatenation of the input plan gives n^2 operators, so the worst case behavior is indeed $\Theta(n^2)$. □

5.2 Algorithm M2

Algorithm M2 is less greedy, and is the most straightforward algorithm. It merges the operators in a plan Π in a left to right scanning process. In each iteration, it merges all of the leftmost operators into the m types of merged operators in the supersequence, removes the leftmost operators from the remaining plan and continues until no operators are left in the original plan.

Algorithm M2.

1. $S := \emptyset$,

2. let $\Sigma :=$ Start(Π). Partition Σ into m classes, such that each class Σ_i contains operators that are mergeable. Let μ_i be the merged operator, for each class i.

3. $\Pi := \text{remove}(\Sigma, \Pi)$,

4. For $i = 1, 2, \ldots, m$, $S := S; \mu_i$.

5. If Π is empty, then return S, otherwise, goto 2.

Theorem 5.2 *Given a set of plans, each of length n. The worst case and average case costs for M2 are both mn.*

Proof: At each of the n columns, there are m different types of operators in general. Algorithm M2 simply merges each column into m operators in the superplan, regardless of what follows these operators in plan Π. Thus the total length of the superplan is mn, in both worst and average cases. □

5.3 Algorithm M3

The next algorithm, algorithm M3, is slightly more sophisticated than M2 in that during each iteration, it only merges the operators in the partitioned subclass Σ_i with the greatest cardinality.

Algorithm M3.

1. $S := \emptyset$,

2. let $\Sigma :=$ Start(Π), Partition Σ into m classes, such that each class Σ_i contains operators that are mergeable. Let Σ_l be the subclass with the largest cardinality, and let μ be the merged operator for Σ_l.

3. $\Pi := \text{remove}(\Sigma_l, \Pi)$,

4. $S := S; \mu$,

5. If Π is empty, then return S, otherwise, goto 2.

We now analyze the worst case and average case complexity of M3. M3 appeals to our intuition as a more aggressive algorithm than M2. However, as the following theorem shows, it actually performs worse than the trivial algorithm M2 in the worst case. This is of course counterintuitive since we expect M3 performs better in general. Such intuition is captured in our average case analysis: for a random instance, M3 does perform provably better than M2. We first give the worst case analysis and then give the average case analysis (under uniform distribution).

Theorem 5.3 *Given a set of n plans each of length n, the worst case cost of M3 is* $\Theta(n \log n)$.

Proof: We first give a set of n plans each containing n operators in $\{\alpha, \beta\}$, from which algorithm M3 will produce a superplan of size $\Omega(n \log n)$. We conveniently arrange the plans into an n by n matrix M such that each row of M corresponds to a plan, one operator per entry. We recursively construct M: Its first $\lfloor n/2 \rfloor + 1$ rows are all α. The first column of the rest of the rows contains β. Therefore, according to its description, algorithm M3 always chooses the first column of the first $\lfloor n/2 \rfloor + 1$ rows to merge at the first n steps. At step $n+1$, M3 chooses the first column of β from the rest of rows. After the execution of the first $n+1$ steps, we are left with a matrix of roughly $\lfloor n/2 \rfloor - 1$ by $n-1$. If we recursively apply above construction to the remaining matrix, it is apparent that the merge process will go on for $\Omega(n \log n)$ steps.

We now show that M3 always produces a superplan no more than size $O(n \log n)$ on plans drawn from a binary operator alphabet (for fixed alphabet size $m > 2$ the proof is similar). The minimum number of operators possibly merged by M3 in successive steps is a monotonically decreasing function $f(k)$ of step number k. The worst case behavior of M3 maximizes the step number at which $f(k)$ vanishes. (We approximate $f(k)$ by its continuous analog $f(x)$.) All realizations of $f(x)$ have $\int_0^\infty f(x)dx = n^2$, that is, M3 merges all n^2 operators into a superplan. As well, $f(x) \geq (2n)^{-1} \int_x^\infty f(y)dy$, for M3 is able to merge at least half of the number of plans not yet exhausted by step k. The worst case behavior of M3 is obtained by setting $f(x)$ to its minimum value for all x. In this case, $f(0) = n/2$ and $f(x) = (2n)^{-1} \int_x^\infty f(y)dy$, with solution $f(x) = (n/2)e^{-x/2n}$. For large n, this function first vanishes when $x = cn \log n$ for some $c > 0$.

□

We now examine the average case cost of M3. We employ a new tool, Kolmogorov complexity, in the proof of the following theorem. An equivalent proof is obtainable by probabilistic argument, but the Kolmogorov argument is simpler. The Kolmogorov complexity $K(x)$ of a string x, over alphabet $\{\alpha_1, \ldots, \alpha_m\}$, is the length of shortest PASCAL program (encoded over the same alphabet) that prints x with empty input. We call x Kolmogorov random (or, simply, random) if $K(x) \geq |x| - \log |x|$. Easy counting shows that there are at least $(1 - 1/n)2^n$ random strings of length n. We refer the reader to Li and Vitanyi [8] for an introduction to Kolmogorov complexity.

Theorem 5.4 *Given n random plans each containing at most n operators from the set $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, and for any small positive constant ϵ, the average case cost of M3 is no greater than $n(m+1)/2 + O(n^{1/2+\epsilon}\log n)$.*

Proof: We first give an informal description of the ideas. We will consider a fixed set of Kolmogorov random plans, which includes nearly all plans of a given length. The advantage of considering such set is that if we can show that on this *particular* set of plans M3 has certain complexity then M3 has the same complexity on average over all plans. Without loss of generality, we assume that we merge n plans, each containing n operators from $\{\alpha_1, \ldots, \alpha_m\}$[1]. Arrange a single Kolmogorov random string of length n^2 such that $K(x) \geq |x| - \log|x|$ into a matrix M row by row, giving n plans. Each of these plans is also Kolmogorov random. Then the claim is that, in the first column, there are almost exactly n/m operators of each type α_i. And if we merge operators of type α_j, then the operators ordered after the merged α_j must be again almost evenly divided into m groups with n/m^2 operators for each α_i; in the Kolmogorov case there are no realizations which deviate significantly from the mean. This nice property continues to hold until some row is completely removed, but by then, the rest of rows are all pretty short (about $O(n^{1/2+\epsilon})$). Further we show any Kolmogorov random string (over some m alphabet) of length n is a subsequence of $\pi^{n(m+1)/(2m)+o(n)}$ for any permutation π of the m operator alphabet.

It is now sufficient to show that, for this random M, M3 constructs a supersequence of length at most $n(m+1)/2 + O(n^{1/2+\epsilon}\log n)$. This is because a fraction $1 - \frac{1}{n}$ of all matrices are Kolmogorov random within a logarithmic additive factor (*i.e.*, $K(x) \geq |x| - \log|x|$). M3 constructs merged plans of length $n(m+1)/2 + o(n)$ on these inputs and of length $n\log n$ in the worst case for the other $1/n$ fraction of inputs. On average, our bound follows. We need the following fact about Kolmogorov complexity.

Lemma 5.5 *Let x be a Kolmogorov random string over an m operator alphabet with $|x| = n$. Define an ij-block to be adjacent operators of types α_i followed by α_j. Fix any $\epsilon > 0$. Let $K(x) \geq n - \log n$. Let s be a subsequence of x. We write $x - s$ to denote the new string formed by deleting s from its corresponding places from x. We say s is an enumerable subsequence of x if there is a program, p, such that $K(p|x-s) \leq \log n$, that outputs s together with the locations of each character of s in x. Then for large n,*

(1) For each $\alpha_i \in \{\alpha_1, \ldots, \alpha_m\}$, α_i appears in x at most $\frac{n}{m} + n^{1/2+\epsilon}$ times, and at least $\frac{n}{m} - n^{1/2+\epsilon}$ times.

(2) For each $\alpha_i, \alpha_j \in \{\alpha_1, \ldots, \alpha_m\}$ x contains at most $\frac{n}{m^2} + n^{1/2+\epsilon}$, and at least $\frac{n}{m^2} - n^{1/2+\epsilon}$, ij-blocks.

(3) For any enumerable subsequence s of x of length $n' = n^\delta$, for $\delta > 0$, (1) and (2) are true with x and n replaced by s and n' respectively.

[1]In case some plan contains less than n operators, we can always add to it a (uniform) random sequence (over our alphabet) so that it has length n. Then in the superplan, we remove these added operators. We consider plans of same length only in order to provide a clean analysis.

Remark. Intuitively, this lemma simply states that every operator appears equally likely in a random sequence and in an "easily enumerable" subsequence of the random sequence (since this subsequence also has to be random). In particular, the first row of M is of course "enumerable"; Also the sequence of operators (say, in top down order) that appeared behind the group merged by M3 at some step is an enumerable subsequence. This important fact will be used later, and will be simply referred to as Lemma 5.5 (1)(3).

Proof: We prove the second statement in (1) by contradiction. Suppose operator $\alpha \in \{\alpha_1, \ldots, \alpha_m\}$ appears in x at most $d = \frac{n}{m} - n^{1/2+\epsilon}$ times. Then there are at most $\binom{n}{d}(m-1)^{n-d}$ strings of length n with d occurrences of α. Using a standard estimation[2], $\log_m \binom{n}{d}(m-1)^{n-d} \leq n - \delta n^\epsilon$ for some $\delta > 0$. Hence x can be coded in $n - \delta n^\epsilon$ digits, implying that x is not random and a contradiction. A similar argument proves the first half of (1).

In order to prove (2), one only needs to observe that by pairing the operators two ways: x_{2i-1} with x_{2i} for $i = 1, 2, \ldots, n/2$, and x_{2i} with x_{2i+1} for $i = 1, 2, \ldots, n/2 - 1$, we reduce the problem to two half size problems in form of (1) with alphabet size m^2. Then (2) follows immediately.

Part (3) of the lemma automatically follows from (1), (2), and from the fact that every enumerable subsequence of x is also random [8].

□

Lemma 5.6 *For Kolmogorov random inputs x, M3 will keep outputting π^*, where π is a fixed permutation of $\alpha_1, \alpha_2, \ldots, \alpha_m$, until some row of M is completely deleted (*i.e.*, some plan is totally merged).*

Proof: We prove by induction, on steps of M3, with the following induction hypothesis from which the lemma follows. Before some row is completely removed,

(a) For each operator α_i, Σ always contains at least $n/m^2 - n^{1/2+\epsilon}$ operators α_i.

(b) If at step s an operator α_i is put into the supersequence, then from the end of step s (after deleting the α_i from Σ), for each operator α_j there will be always at least $n/m^2 - mn^{1/2+\epsilon}$ more α_j than α_i in Σ until α_j is chosen to be in the supersequence (*i.e.*, α_j in Σ are deleted).

At the first step, hypothesis (a) is true by Lemma 5.5 (1)(3). Hypothesis (b) is trivially true since nothing is deleted yet. Assume (a) and (b) are true until the end of step s and operator α_i is put into the supersequence at step s. To prove (a) we note that at step s we must have deleted at least n/m α_i by algorithm M3. It is easy to see that the subsequence appearing behind the deleted operators is enumerable. By Lemma 5.5 (1)(3), each operator appears at least $n/m^2 - n^{1/2+\epsilon}$ times in this subsequence of length at least n/m, from which (a) follows. In order to prove (b), notice that the number of each element is increased by $d/m \pm n^{1/2+\epsilon}$ uniformly, where $d \geq n/m$ is the number of α_i deleted from Σ in step s. But

[2]An estimation can be found in M. Li and P. Vitanyi, *An introduction to Kolmogorov complexity and its applications*, Addison-Wesley, forthcoming.

only the operator α_i starts from 0, and all others had at least $O(n/m^2)$ elements in Σ already. At each of the next m rounds, all operators are increased by almost equal amount with a variance of up to $n^{1/2+\epsilon}$, which will never make up the discrepancy. (Note the difference here between Kolmogorov random strings, from which we have excluded the set of strings having variance that could reverse the order of some (i,j) pair, and probabilistically random strings, which contain such inputs with small probability.) So operator α_i will only become the most frequent operator after every other operator has appeared in the supersequence exactly once. We thus have proved (b).

By (a) and (b), it is clear that M3 will continue to output π^* until some row of M (some plan) is completely merged, where π is a fixed permutation of $\alpha_1, \alpha_2, \ldots, \alpha_m$. $\Box$

Lemma 5.7 *Again let $K(x) \geq n - \log n$ where $x \in \{\alpha_1, \ldots, \alpha_m\}^*$ and $n = |x|$. Let π be any permutation of $\alpha_1 \ldots \alpha_m$. Let $h(n,\epsilon) = n^{1/2+\epsilon}$ for notational convenience. Then*

(1) x is a subsequence of $\pi^{n(m+1)/(2m)+h(n,\epsilon)}$ for any $\epsilon > 0$.

(2) x is not a subsequence of $\pi^{n(m+1)/(2m)-h(n,\epsilon')}$ for any fixed ϵ'.

Proof: Consider the $n-1$ operator pairs $x_i x_{i+1}$ in x, for $1 \leq i < n$. By Lemma 5.5 (2), x_i appears in a given π before x_{i+1} in at least L of these pairs, where $L = (\frac{m(m-1)}{2}\frac{n}{m^2}) - O(n^{1/2+\epsilon}) = (\frac{1}{2} - \frac{1}{2m})n - O(n^{1/2+\epsilon})$. For each such pair, x_{i+1} is fitted into π for "free". The other $n(\frac{1}{2} + \frac{1}{2m}) + o(n)$ pairs require an additional copy of π for x_{i+1}. Also, every sequence x can fit into n copies of π, *i.e.*, π^n. Since L of those π can be deleted because of the above free-fitting scheme, x can fit into a sequence

$$\pi^{n-L} = \pi^{cn+o(n)}$$

for $c = \frac{1}{2} + \frac{1}{2m}$, which proves (1). But there are also at least L operator pairs $x_j x_{j+1}$ such that x_j appears in π after x_{j+1}. For such a pair x_{j+1} needs to be put into a next π block. By letting ϵ in Lemma 5.5 (2) be less than the above ϵ', we have proved (2). $\Box$

We complete the proof of main theorem. By Lemma 5.6, M3 continues to output a prefix of π^*, where π is a permutation of $\alpha_1 \ldots \alpha_m$, until some row r of M is completely deleted. However, by Lemma 5.7, every row of M is a subsequence of $\pi^{n(m+1)/(2m)+h(n,\epsilon)}$ for any $\epsilon > 0$, but not a subsequence of $(\pi)^{n(m+1)/(2m)-h(n,\epsilon')}$ for any fixed ϵ'. So when row r of M is completely deleted, all the rest of rows are of length at most $O(n^{1/2+\epsilon})$. From this point on, the worst case analysis (Theorem 5.3) of M3 guarantees that M3 will finish within $O(n^{1/2+\epsilon}\log n)$ steps. When r is totally deleted, the partial supersequence has length at most $n(m+1)/2 + O(n^{1/2+\epsilon})$, by Lemma 5.7 (1). Therefore the total length of the supersequence constructed for M is $n(m+1)/2 + O(n^{1/2+\epsilon}\log n)$ for any small $\epsilon > 0$.

Then since M is formed from a Kolmogorov random string, this is also the average length, for the remaining (non-Kolmogorov random) matrices contribute to the total average at most an $n \log n$ term times their average $1/n$ occurrence. $\Box$

Remark. We provide another point of view. Since the Kolmogorov random inputs have the behavior that M3 outputs a repeating pattern (Lemma 5.6), we can model its merging behavior by a simple Markov chain on m states. Let state σ_i refer to the operator group Σ_l with i^{th} greatest cardinality. When operators are merged, all operators in state σ_1 are randomly distributed to other states, while operators in $\sigma_2, \ldots, \sigma_m$ move to states $\sigma_1, \ldots, \sigma_{m-1}$, respectively. The probability transition matrix for the process is just

$$\begin{pmatrix} 1/m & 1/m & 1/m & \cdots & 1/m \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix}$$

and its stationary vector is $2/(m(m+1))(m, m-1, \ldots, 1)$. At each step of M3, there is thus a compression ratio of $2n/(m+1)$, so that the n^2 operators can be merged to form a superplan of expected length $n(m+1)/2$.

5.4 Algorithm M4

Algorithm M3 has the best average case cost among the three algorithms we discussed so far ($\approx n(m+1)/2$), but M2 has the best worst case cost (mn). Algorithm M4 combines the advantages of both M2 and M3. As M3, it looks for the largest set of mergeable operators to merge from the left frontier Start(Π) of the remaining plan. However, it carefully avoids the worst case behavior of M3. Similar to M2, it collects all the other operators on the left frontier Start(Π) as well, and merges them before looking for new operators to merge in the next iteration.

Algorithm M4.

```
1. $S := \emptyset$,
2. let $\Sigma :=$ Start($\Pi$), and $T := \{\text{Type}(\alpha) \mid \alpha \in \Sigma\}$.
3. Until $T$ is empty, do
    a. Partition $\Sigma$ into $m$ classes, such that each class $\Sigma_i$ contains operators that are
       mergeable. Let $\Sigma_l$ be the subclass with the largest cardinality, and let $\mu$ be the
       merged operator for $\Sigma_l$.
    b. $\Pi :=$ remove($\Sigma_l, \Pi$), $S := S; \mu$, $T := T -$ Type($\mu$).
    c. $\Sigma := \{\alpha \mid \alpha \in \text{Start}(\Pi)$ and $\text{Type}(\alpha) \in T\}$.
5. If $\Pi$ is empty, then return $S$, otherwise, goto 2.
```

Theorem 5.8 *The worst case cost of M4 is mn and the average case cost of M4 is the same as M3.*

Proof:

The reader should notice the subtle difference between M3 and M4. In M3, we always choose an operator with maximum cardinality. But in M4, we consider set T which contains operator types of all leftmost operators; M4 chooses the best operator *from* T such that we would merge more operators from the current frontier. M4 will use up all operator types in T and then take new operator types from current frontier. This in fact is very similar to M2 since M2 simply merges every set of operators of the same type in the current frontier (column). If all input plans are of length at most n, and there are m operators, then, as for M2, M4's worst case is mn.

However, M4 is also more aggressive than M2. In fact, it tries to merge as many operators as possible, not restricting itself to the current column as M2. From the analysis of M3, it turns out that if the input plans are random, then, if we merge as M3, the output will be precisely a cyclic alternation of operators. This is precisely what we are doing with M4. Thus Lemma 5.6 in the proof of M3 implies that M4 has the same average case complexity as that of M3. □

From above analysis, an algorithm that is too greedy (M1) or an algorithm that is not greedy enough (M2) performs worse. On the other hand algorithms that compromise the two extremes (M3 and M4) perform better on average.

So far, our analyses have not compared with the behavior of optimal plan merging, as this is still an open problem. Also, the analyses assumed large input sizes. To verify our theoretical analysis with small input sizes, we have also conducted a series of experiments in Section 6.

5.5 Generalizations

For mathematical simplicity, so far we have restricted our attention to simple cases of linear input plans and disjoint mergeable operator types in analyzing the approximation algorithms. In this section, we generalize our results in two ways, by inputing and outputting partially ordered plans and by merging arbitrarily mergeable sets of operators. In blocks world, this relaxation allows a single gripper to be able to pick up blocks of different sizes.

As noted in the previous section, M3 and M4 have the same average case performance. Since M3 is slightly simpler than M4, we present a generalization of M3 to handle partially ordered inputs. Recall that Algorithm M3 sweeps through the input plan Π by merging operators on each frontier. After each merging step, a new frontier is formed, and thus merging continues until the whole plan is exhausted. With partially ordered plans, the frontier at each step is still well-defined by Start(Π). Let Mergeables(Start(Π)) $= \{\Sigma_i, i = 1, 2, \ldots\}$ be the sets of operators in the frontier that can be merged. Let Σ_l be a set of operators among Mergeables(Start(Π)) such that cost(Σ_l) $-$ cost(merge(Σ_l)) is maximum. Then in this iteration Σ_l will be chosen to be merged. This is the same with M3 in spirit, because at each iteration the most profitable merging is chosen along the current frontier.

To output a partially ordered plan after merging is done, we only need to keep track of what operators are merged by the algorithm in each iteration. Then after the whole plan is merged, these operator sets are merged in the original plan with no more ordering constraints imposed, which leaves a partially ordered resultant plan in general.

Algorithm M5.

1. $R := \emptyset$. Each element of R is a set of operator identifiers indicating where merging should occur.

2. Let $\Sigma_i, i = 1, 2, \ldots, m$ be sets of operators that can be merged, such that

   1. $\Sigma_i \subseteq \text{Start}(\Pi)$,
   2. no Σ_i is also a proper subset of Σ_j, for $i \neq j$.

   That is, Σ_i contains operators that are mergeable. Let Σ_l be the set such that $\text{cost}(\Sigma_l) - \text{cost}(\text{merge}(\Sigma_l))$ is minimal.

3. $\Pi := \text{remove}(\Sigma_l, \Pi)$,

4. $R := R \bigcup \{\Sigma_l\}$,

5. If Π is not empty, goto 2.

6. Reset Π to be the original input plan. For each set Σ_i in R, merge Σ_i in Π.

Note that when operators belong to disjoint classes partitioned by operator types, and when the subplans are linear sequences of operators, M5 degenerates to M3. M4 can be similarly generalized.

6 Experimental Results

In this section, we compare the empirical behavior of the algorithms over several sets of randomly generated test cases. These empirical tests are important because they reveal the behavior of the algorithms when input sizes are small, a situation not covered by the theoretical analysis in the previous section. Each random test case is a set of linearly ordered sequences of operators with equal lengths. Each sequence is generated by assuming a uniform distribution of operator types. Test cases are distinguished by three parameters: the size of the operator alphabet, the length of each input sequence and the number of input sequences. Test programs were written in Kyoto Common Lisp. Figures showing the test results can be found at the end of the paper.

The tests are grouped into two classes. The first class aims at comparing each approximation algorithm with the optimal solution generated using the dynamic programming method.

Each test datum obtained in this class corresponds to the average result over five inputs. Figure 12 shows the length of the supersequences generated by the approximation and optimal algorithms as a function of the length of each input sequences. Figure 13 shows the results as a function of alphabet size. It appears from these tests that algorithm M4 performs the best on the average among all four algorithms, while M1 performs the worst. As the length of each input sequence increases (Figure 12), algorithms M1 and M2 perform increasingly worse when compared with the optimal, while M3 and M4 stay fairly close to the optimal solutions. In Figure 13, M3 and M4 perform much better than M1 and M2 with small alphabet sizes. But as the size of the alphabet increases, all four approximation algorithms deviate from the optimal. Since the dynamic programming algorithm has a higher time and space complexity, no tests were done with the changing number of input sequences.

The second group consists of tests comparing the performance of the approximation algorithms with large input sizes. Each test in this group corresponds to the average over 10 randomly generated data. For those input sequences, the optimal dynamic programming algorithm becomes infeasible to execute. Figures 14, 15 and 16 show the performance of each algorithm as a function of the length of each input sequence, the size of the alphabet, and the number of input sequences, respectively. It again appears from these tests that algorithms M3 and M4 perform increasingly better than M2, which in turn performs increasingly better than M1 with the length of input sequence and the size of alphabet. Also note that no algorithm is affected by the number of input sequences (Figure 16), as might be inferred from our average complexity theorems. Note that in Figure 16 algorithms M3 and M4 has identical performance. Further, it is worth noting that as the length and number of input sequences gets larger, the empirical behavior of the algorithms converges closer to our theoretical average case analysis in the previous section.

In Section 5 we stated that algorithm M1 not only has the highest worst case complexity, it also has the worst average case behavior. It is our conjecture that M1 has a average case complexity of $O(n \log n)$, where n is the length of input sequences, taking plans with binary alphabet as input, and taking the length and the number of input sequences being equal. In support of our conjecture, we conducted an experiment on M1, with results presented in Figure 17. The figure shows that the average case behavior of M1 is indeed worse than $2n \log n$.

In summary, we conclude that algorithm both M3 and M4 have the best empirical performance among all approximation algorithms when the input size gets large. However, with small enough input sizes, the difference between the for algorithms is not significant.

7 Conclusion

In this paper, we have presented a formalism as well as a computational theory for optimal and approximate plan merging. Using the STRIPS operator definition, we have formally defined when operators in a plan can be merged, and discussed the complexity for optimal plan merging. With plans of relatively small sizes, our dynamic programming method can be used to compute the optimal solution. Various extensions of the algorithm are considered, including plan merging with partially ordered input plans. For plans with large sizes, the optimal algorithm is no longer feasible. In such cases, approximation algorithms can be utilized to compute high quality plans at low cost. We have presented several approximation algorithms and have shown, through theoretical and empirical analysis, that the algorithms M3 and M4 perform the best among the greedy algorithms in terms of the worst and average case complexities.

Acknowledgement

The authors would like to thank W.P. Tang for bringing us together, and to D. Wood and U. Manber for many useful comments. The authors would also like to thank the referees for their constructive comments.

References

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

[2] E.H. Derfee and V.R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the 10th IJCAI*, pages 875–883, 1987.

[3] Michael R. Garey and David S Johnson. *Computers and Intractability.* W. H. Freeman and Company, 1979.

[4] Caroline C Hayes. A model of planning for plan efficiency: Taking advantage of operator overlap. In *Proceedings of the 11th IJCAI*, Detroit, Michigan, 1989.

[5] Subbarao Kambhampati and James A. Hendler. A validation structure based on theory of plan modification and reuse. In *Artificial Intelligence (To appear)*, 1990.

[6] Raghu Karinthi, Dana S. Nau, and Qiang Yang. Handling feature interactions in process planning. to appear, Journal of Applied Artificial Intelligence, 1991.

[7] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the 9th AAAI*, Anaheim, CA, 1991.

[8] M. Li and P. Vitanyi. Two decades of applied kolmogorov complexity. In *3rd Structure in Complexity Theory*, pages 80–101, 1988.

[9] U. Manber. *Introduction to Algorithms: A Creative Approach.* Addison-Wesley, Reading, Massachusetts, 1989.

[10] M. Mantyla and J. Opas. Hutcapp—a machining operations planner. In *Proceedings of the Second International Symposium on Robotics and Manufacturing systems*, 1988.

[11] Dana S. Nau, Qiang Yang, and James Hendler. Optimization of multiple-goal plans with limited interaction. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 160–165, San Diego, CA., November 1990.

[12] D.A. Rosenblitt. *Supporting Collaborative Planning: The Plan Integration Problem.* PhD thesis, MIT, Cambridge, MA, Feb. 1991.

[13] Earl Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, 1977.

[14] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, Massachusetts, 1983.

[15] T Sellis. Multiple query optimization. *ACM Trasactions on Database Systems*, 13(1), March 1988.

[16] Austin Tate. Generating project networks. In *Proceedings of the 5th IJCAI*, pages 888–893, 1977.

[17] R. Wilensky. *Planning and Understanding.* Addison Wesley, 1983.

[18] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, CA, 1988.

[19] Qiang Yang. An algebraic approach to conflict resolution in planning. In *Proceedings of the 8th AAAI*, Boston, MA, August 1990.

[20] Qiang Yang and Josh Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of the 8th AAAI*, pages 204–209, Boston, MA, August 1990.

[21] Qiang Yang, Josh Tenenberg, and Steve Woods. Abstraction in nonlinear planning. University of Waterloo Technical Report CS91-65, 1991.

Figure 12: Tests with fixed number of sequences ($k = 4$) and alphabet size ($m = 2$). Each datum is an average over 5 random inputs.

Figure 13: Tests with fixed number of sequences ($k = 4$) and sequence length ($n = 10$). Each datum is an average over 5 random inputs.

Figure 14: Tests with fixed number of sequences ($k = 4$) and alphabet size ($m = 2$). Each datum is an average over 10 tests.

Figure 15: Tests with fixed number of sequences ($k = 40$) and sequence length ($n = 40$). Each datum is an average over 10 tests.

Figure 16: Tests with fixed alphabet size ($m = 2$) and sequence length ($n = 10$). Each datum is an average over 10 tests.

Figure 17: Tests with fixed alphabet size ($m = 2$). Each datum is an average over 10 tests.