A Quantitative Theory for Plan Merging *

David E. Foulser
Silicon Graphics Computer Systems
2011 N. Shoreline Blvd.
Mountain View, CA 94039.
foulser@sgi.com

Ming Li
University of Waterloo
Computer Science
Ont., Canada, N2L 3G1
mli@math.waterloo.edu

Qiang Yang
University of Waterloo
Computer Science
Ont., Canada, N2L 3G1
qyang@dragon.waterloo.edu

Abstract

Merging operators in a plan can yield significant savings in the cost to execute a plan. Past research in planning has concentrated on handling harmful interactions among plans, but the understanding of positive ones has remained at a qualitative, heuristic level. This paper provides a quantitative study for plan optimization and presents both optimal and approximate algorithms for finding minimum-cost merged plans. With worst and average case complexity analysis and empirical tests, we demonstrate that efficient and well-behaved approximation algorithms are applicable for optimizing general plans with large sizes.

Introduction

The value of helpful or positive interactions among the different parts of plans was recognized early in AI planning research [Sacerdoti, 1977; Wilensky, 1983; Wilkins, 1988]. 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.

Many examples of plan merging can be found in human problem solving in [Wilensky, 1983]. Operator merging is equally important in areas of robot task planning and scheduling. For example, if several grippers with different sizes exist in a blocks world domain, and if picking up one block requires the robot to change to a gripper of an approriate size, then it is more efficient to group block stacking operations that use the same type of grippers. Identical plan merging issues also arise in the domain of automated manufacturing where process plans for metal-cutting [Karinthi *et al.*, 1990], set-up operations [Hayes, 1989] and tool-approach directions [Mantyla and Opas, 1988] need to be optimized. Similarly, in the area of query optimization in database systems [Sellis, 1988], as well as domains having multiple agents [Derfee and Lesser, 1987; Rosenblitt, 1991], operator merging in multiple plans seems inevitable.

Despite the importance of this problem, few attempts have been made in conducting a systematic study on operator merging in planning. The majority of planning research so far has been aimed at dealing with negative interactions in plans such as conflicts and resource competitions [Sacerdoti, 1977; Wilkins, 1988; Chapman, 1987]. The strategies that deal with positive interactions such as operator merging, are usually heuristic in nature, and no merits or drawbacks have been evaluated in a systematic way.

In this paper, we provide a formalization for plan merging. In particular, we formalize the kinds of plan merging that have been considered in the previous systems, and discuss the complexity of the problem in general. 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. 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 runtime 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 [Hayes, 1989; Sacerdoti, 1977; Wilkins, 1988]. Thus, we also describe four polynomial time approximation algorithms for merging plans, and analyze their average case merging complexity.

*Foulser is supported in part by ONR grant N00014-86-J-1906 and by NIH Grant R01 LM05044 from the National Library of Medicine. Li is supported in part by NSERC operating grant OGP0046506. Yang is supported in part by NSERC operating grant OGP0089686.

Formal Description of the Problem

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). 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.

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.

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 of operators in Σ.

The following definition is adapted from [Yang *et al.*, 1991]. Let Σ be a set of operators in Π that can be grouped together. We use NetEffects(Σ, Π) to denote the set of all useful effects of the operators in Σ. Likewise, we use NetPrecond(Σ, Π) to denote the set of all preconditions of the operators in Σ not achieved by any operators in Σ. A set Σ of operators is *mergeable* in a plan $\Pi = (O, B)$ if and only if an operator μ exists, such that

1. Σ can be grouped together in Π, $P_\mu \subseteq$ NetPrecond(Σ, Π'), and NetEffects(Σ, Π') $\subseteq E_\mu$. 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; and
2. cost(μ) < cost(Σ).

The operator μ is called a *merged* operator of Σ in the plan Π. Thus, by merging operators in a plan, one can improve the quality of the plan.

Two issues exist in operator merging. One is related to how to find the mergeable operators. To solve this problem requires various kinds of domain knowledge and experience to help reduce the problem complexity. In this paper, we *assume* that knowledge is available about what operators can be merged, and concentrate on the second problem: finding and analyzing methods for computing the optimal and approximate plan. We start by discussing its complexities in the next section.

Complexity

Several complications exist that in plan merging. First, 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. A third complication occurs because the partial order on Π may render inconsistent some pairs of mergings.

To remove the first complication, we assume that for each set Σ of mergeable operators, there is a unique merged operator μ. The second complication is easily handled by imposing a distance metric on the operators. The distance metric induces a cost function for unmerged and merged sets of operators. With this metric, the plan merging problem can be directly solved via the algorithms we will present below.

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. This problem is equivalent to the problem of finding the shortest common supersequence (SCS), that is, finding the shortest sequence S such that every sequence in Π is a subsequence of S. The problem is generally known to be NP-complete. But for the naturally occurring case of a fixed number of input sequences, the SCS may be simply calculated in polynomial time. We show this result in the next section.

Optimal Plan Merging via SCS

Optimal Algorithm

For simplicity, we assume that operators are of m different types, and that two unordered operators in a plan can be merged iff they are of the same type. These assumptions can be easily relaxed later. Also, let there be k input plans S^1, S^2, ..., S^k, each of which is a linear sequence of operators drawn from a finite assortment (or alphabet) $\mathcal{A} = \{\alpha_1, \ldots, \alpha_m\}$ of operator types. Let $|S^i|$ denote the length of S^i. We write a sequence $S^i = s^i_1 s^i_2 \ldots s^i_{|S^i|}$.

Consider a k-dimensional array A. To dimension i of A assign the operator sequence S^i, giving a size $|S^1| \times |S^2| \times \ldots \times |S^k|$. A will be used to represent the SCS lengths of partial inputs, so that $A(i_1, \ldots, i_k)$ is the length of the SCS of $S^1_{1\ldots i_1}$, ..., $S^k_{1\ldots i_k}$. Define also the identically-sized array R, which will be used to represent the components of the actual SCS.

From the dynamic programming principle, the minimal length path is easily computed using the following recurrence relation :

$$A(i_1, \ldots, i_k) = \min_{\alpha_1, \ldots, \alpha_m} 1 + A(i_1 - \delta_1, \ldots, i_k - \delta_k),$$

where the "min" ranges through all operator types α_l, and each δ_i is 1 iff the operator ${i_j}^{th}$ operator of the jth sequence S_j is of type α_l. Otherwise δ_i is 0. This recurrence forms the basis for the inner loop of the SCS algorithm.

We use the array R to store the reverse links necessary to reconstruct the actual SCS on completion of the algorithm, which we now present.

```
Compute A(i_1, ..., i_k) and R(i_1, ..., i_k):
for i_1 = 0 to |S^1| do
    :
    for i_k = 0 to |S^k| do
        minval = ∞
        for α_l = α_1 to α_m do
            for all non-empty subsets σ ⊂
{ set of sequence indices for which s^j_{i_j} = α_l} do
                for j = 1 to k do
                    δ_j = { 1  if j ∈ σ
                          { 0  otherwise
                endfor
                if minval > 1 + A(i_1 − δ_1, ..., i_k − δ_k) then
                    minval = 1 + A(i_1 − δ_1, ..., i_k − δ_k)
                    R(i_1, ..., i_k) = σ
                endif
            endfor
        endfor
        A(i_1, ..., i_k) = minval
    endfor
    :
endfor
```

The key steps are the location of the path segment yielding the minimal path and the contributing sequence indices σ, updating the cost to reflect the minimal path, and setting the reverse link to that path segment. The reverse links R are indicated by storing the vector of sequence indices whose operators match and are to be merged. In order to reconstruct the SCS from R, one simply follows path segments starting from $R(|S^1|, \ldots, |S^k|)$ to $R(0, \ldots, 0)$, which yields the reverse of T.

The cost of the method is $O(\prod_{i=1}^{k} |S^i|)$, with fixed number k of input sequences and fixed alphabet size m.

Extensions

A number of extensions can be easily made to the dynamic programming algorithm. We give a brief summary of the extensions. For a more detailed discussion, see [Foulser *et al.*, 1990].

Operator Weightings. The algorithm assumes a uniform cost for all operator types. Now suppose that operator types are of different costs, $\text{cost}(\alpha_l)$, for $\alpha_1 \leq \alpha_l \leq \alpha_m$. The "minval" calculation step in the algorithm can be correspondingly updated by:

$$\text{minval} = \text{cost}(\alpha_l) + A(i_1 - \delta_1, \ldots, i_k - \delta_k).$$

In this setting, those operators with large weights are to be preferentially merged, even if their number of occurrences is small.

Non-identical Overlaps. In practice, operators of different types may need to be merged. To make the extension, one might therefore consider assigning a fixed matching cost to each combination of two or more operators, on the assumption that a set of j non-identical operators could be combined to effect some savings in cost.

In this case, one defines the cost function $\text{cost}(k_1, \ldots, k_m)$ based on k_i occurrences of operator α_i, for $\alpha_1 \leq \alpha_i \leq \alpha_m$. The inner loop of the SCS computation is then taken over all 2^k subsets of the operators $s^1_{i_1}, \ldots, s^k_{i_k}$.

Partially Ordered Inputs. One can also extend the algorithm to handle partially ordered plans. The basis of the computation is a multi-dimensional grid created using the notion of "maximally unordered" operators. A set ω of operators in plan Π is maximally unordered iff all operators in the set is unordered, and no other operator in Π is unordered with every operator in ω. One can also define an ordering among the maximally unordered sets ω_1 and ω_2, if all operators in the former precedes all other operators in the latter.

The computation of an optimal plan proceeds as in the linearly ordered case, starting from the subset of A containing elements with no predecessors and proceeding through their descendants to the elements with no successors. As with the case of linear inputs, the optimal plan is constructed as a monotone sequence of elements in A, where the transition from ω_1 to an adjacent ω_2 indicates the merging of two or more plan operators in Π, or the inclusion of a single unmerged operator in the resulting optimal plan.

Similarly, one can extend so that the algorithm outputs a partially ordered set of operators.

Approximation Algorithms

For problems of large sizes, the complexity of the dynamic programming methods may be too high. Below, we develop and analysis a set of four approximate algorithms that all have linear worst-case time complexity, but returns plans with different costs. It is assumed that every operator has a unit cost, so the cost of a plan is the total number of operators in it. This assumption can be easily relaxed similar to with the dynamic programming algorithm.

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. The notation $\text{Start}(\Pi)$ refers to the set of operators with no operator preceding them. $\text{remove}(\Sigma, \Pi)$ refers to the plan Π with operators in Σ removed. Inputs are assumed to be k linear sequences of operators, where any pair of sequences is unordered, and each sequence has a length n. All algorithms below basically operate by sweeping through the input plans in a left to right manner, where op-

erators are merged only when they don't violate the existing precedence relations.

Due to space limitation, we omit all proofs of subsequent theorems. Interested readers can find full proofs and relevant references in [Foulser *et al.*, 1990].

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, which forms a "thread" that partitions the plan Π into three subplans, Π_{11} on the left, Π_{12} on the right, and Π_2 those not touched by the thread.

Algorithm M1.
(1) If $\Pi = \emptyset$ then return $\emptyset$. Otherwise, arbitrarily find $\alpha \in \text{Start}(\Pi)$. 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 S in Π_1, let α' be the operator in Σ. Split S at α' into S_1 and S_2, so that $S = S_1\alpha' S_2$. Let Π_{11} be the set of all S_1, and Π_{12} be the set of all S_2.
(4) Return $\text{M1}(\Pi_{11} \bigcup \Pi_2); \mu; \text{M1}(\Pi_{12})$, where ";" stands for concatenation.

Theorem 1 *Given n plans, each of length n, algorithm M1 returns a merged plan with a worst case cost of $\Theta(n^2)$.*

Algorithm M2

Algorithm M2 is less greedy, and is the most straightforward algorithm. In each iteration, it merges all of the leftmost operators into the m types of merged operators in the supersequence, and terminates till no operators are left to be merged in the original plan.

Algorithm M2.
(1) $S := \emptyset$,
(2) let $\Sigma := \text{Start}(\Pi)$.
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) $\Pi := \text{remove}(\Sigma, \Pi)$. For $i = 1, 2, \ldots, m$, $S := \mu_i; S$.
(4) If Π is empty, then return S, else goto (2).

Theorem 2 *Given n plans, each of length n, and m operator types, algorithm M2 returns a merged plan with the worst and average case costs of mn.*

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 $\Sigma := \text{Start}(\Pi)$,
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) $S := \mu; S$. If Π is empty, then return S, otherwise, goto (2).

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 give the worst case and the average case analysis (under uniform distribution).

Theorem 3 *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 worst case complexity of M3 is $\Theta(n \log n)$. The average case complexity of M3 is no greater than $n(m+1)/2 + O(n^{1/2+\epsilon} \log n)$.*

Algorithm M4

Algorithm M4 combines the advantages of both M2 and M3. It is careful to avoid the worst case behavior of M3, but collects all the other operators on the left frontier of the remaining operators as well, and merges them all before looking for new operators to merge in the next iteration.

Algorithm M4.
(1) $S := \emptyset$,
(2) let $\Sigma := \text{Start}(\Pi)$, $T := \{\text{Types}(\alpha) \mid \alpha \in \Sigma\}$.
(3) Until T is empty, do
(a) 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.
(b) $\Pi := \text{remove}(\Sigma_l, \Pi)$, $S := \mu; S$, $T := T - \text{Types}(\mu)$.
(c) $\Sigma := \{\alpha \mid \alpha \in \text{Start}(\Pi)$ *and* $\text{Types}(\alpha) \in T\}$.
(5) If Π is empty, then return S, else, goto 2.

Theorem 4 *The worst case cost of the merged plan returned by M4 is mn and the average case cost is the same as M3.*

As is expected, the algorithms which merge operators using the least amount of information (M1 and M2) perform the worst, while the algorithms using

more global information (M3 and M4) perform better on the average. As in the case for the optimal dynamic programming method, it is also possible to extend our four algorithms to handling as well as outputing partially ordered plans.

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.

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 1 shows the length of the supersequences generated by the approximation and optimal algorithms as a function of the length of each input sequences. Figure 2 shows the results as a function of alphabet size. It is clear 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 1), 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 2, 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 algoirthms 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 3 and 4 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 is again clear 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. 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 theorectical average case analysis in the previous section.

In Section 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 5.

In summary, we conclude that algorithm M4 has 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.

Conclusion

In this paper, we have presented a formalism as well as a quantitative study for optimal and approximate 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 arbitrary weights and dynamic programming merging of 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 four approximation algorithms and have shown, through theoretical and empirical analysis, that the algorithm M4 performs the best.

Acknowledgement. The authors would like to thank W.P. Tang for bringing us together, and to D. Wood and U. Manber for useful comments.

References

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

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

[Foulser *et al.*, 1990] Foulser, D.E.; Li, M.; and Yang, Q. 1990. Theory and algorithms for plan merging. Technical Report cs-90-40, University of Waterloo, Computer Science.

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

[Karinthi *et al.*, 1990] Karinthi, Raghu; Nau, Dana S.; and Yang, Qiang 1990. Handling feature interactions in process planning. Submitted for Publication.

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

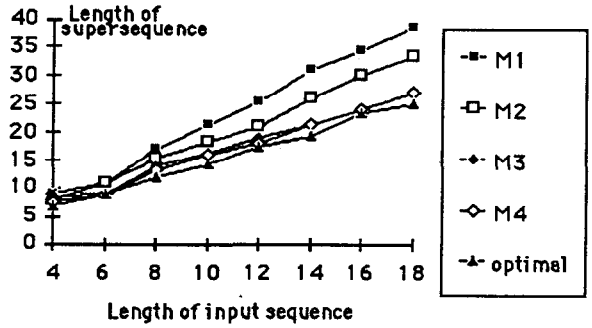

Figure 1: Tests with fixed number of sequences ($k = 4$) and alphabet size ($m = 2$). Each datum is an average over 5 random inputs.

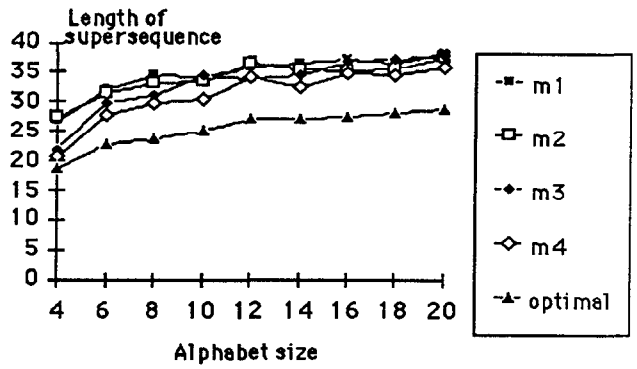

Figure 2: Tests with fixed number of sequences ($k = 4$) and sequence length ($n = 10$). Each datum is an average over 5 random inputs.

[Rosenblitt, 1991] Rosenblitt, D.A. 1991. *Supporting Collaborative Planning: The Plan Integration Problem.* Ph.D. Dissertation, MIT, Cambridge, MA.

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

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

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

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

[Yang *et al.*, 1991] Yang, Qiang; Nau, Dana S.; and Hendler, James 1991. Merging separately generated plans in limited domains. submitted for publication.

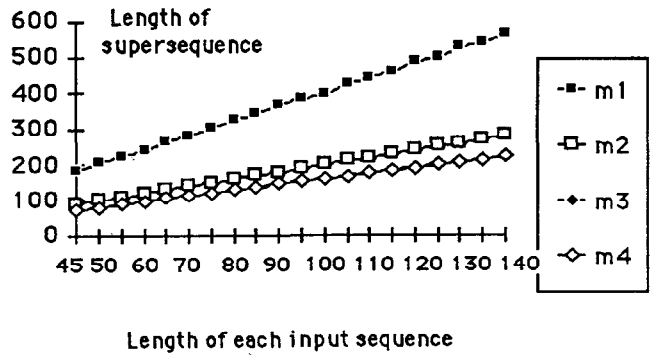

Figure 3: Tests with fixed number of sequences ($k = 4$) and alphabet size ($m = 2$). Each datum is an average over 10 tests.

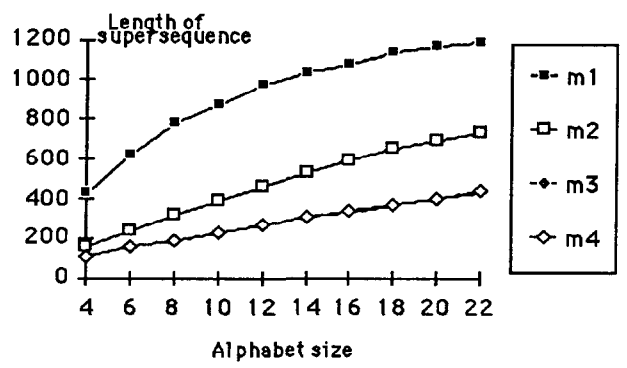

Figure 4: Tests with fixed number of sequences ($k = 40$) and sequence length ($n = 40$). Each datum is an average over 10 tests.

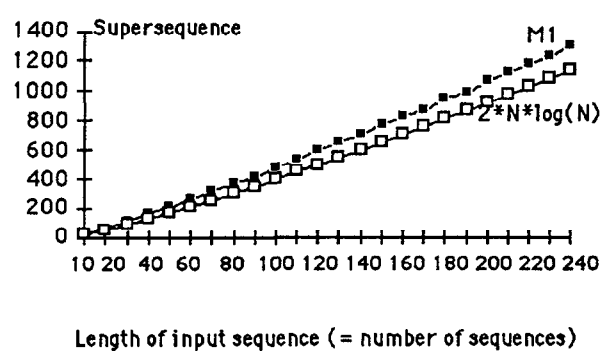

Figure 5: Tests with fixed alphabet size ($m = 2$). Each datum is an average over 10 tests.