Artificial Intelligence Review **14:** 421–446, 2000.
Issues on the Application of Data Mining.

© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

PLANMINE: Predicting Plan Failures Using Sequence Mining *

MOHAMMED J. ZAKI, NEAL LESH & MITSUNORI OGIHARA
[1]*Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA (E-mail: zaki@cs.rpi.edu);* [2]*Mitsubishi Electric Research Laboratory, 201 Broadway, 8th Floor, Cambridge, MA 02139, USA (E-mail: lesh@merl.com);* [3]*Department of Computer Science, University of Rochester, Rochester, NY 14627, USA (E-mail: ogihara@cs.rochester.edu)*

Abstract. This paper presents the PLANMINE sequence mining algorithm to extract patterns of events that predict failures in databases of plan executions. New techniques were needed because previous data mining algorithms were overwhelmed by the staggering number of very frequent, but entirely unpredictive patterns that exist in the plan database. This paper combines several techniques for pruning out unpredictive and redundant patterns which reduce the size of the returned rule set by more than three orders of magnitude. PLANMINE has also been fully integrated into two real-world planning systems. We experimentally evaluate the rules discovered by PLANMINE, and show that they are extremely useful for understanding and improving plans, as well as for building monitors that raise alarms before failures happen.

Keywords: plan monitoring, predicting failures, sequence mining

1. Introduction

Knowledge Discovery and Data Mining (KDD) refers to the process of discovering new, useful and understandable knowledge in databases. KDD techniques have been used successfully in a number of domains such as molecular biology, marketing, fraud detection, etc. Typically data mining has the two high level goals of *prediction* and *description* (Fayyad et al., 1996). In prediction, we are interested in building a model that will predict unknown or future values of attributes of interest, based on known values of some attributes in the database. In KDD applications, the description of the data in human-understandable terms is equally if not more important than prediction. The typical data mining tasks include classification, clustering, deviation detection, and association and sequence discovery. See Fayyad et al. (1996) for an excellent overview of the different aspects of KDD.

* Supported by NSF grants CCR-9705594, CCR-9701911, CCR-9725021 and INT-9726724; and U.S. Air Force/Rome Labs contract F30602-95-1-0025.

In this paper, we present the PLANMINE sequence discovery algorithm for mining information from plan execution traces. Analyzing execution traces is appropriate for planning domains that contain uncertainty, such as incomplete knowledge of the world or actions with probabilistic effects. Assessing plans in probabilistic domains is particularly difficult. For example, in (Kushmerick et al., 1995) four algorithms for probabilistic plan assessment are presented, all of which are exponential in the length of the plan. When analyzing plans directly is impractical, execution traces can be a rich, but largely untapped, source of useful information about a plan. We apply sequence data mining to extract causes of plan failures, and feed the discovered patterns back into the planner to improve future plans. We also use the mined rules for building monitors that signal an alarm before a failure is likely to happen.

PLANMINE has been integrated into two applications in planning: the TRIPS collaborative planning system (Ferguson and James, 1998), and the IMPROVE algorithm for improving large, probabilistic plans (Lesh et al., 1998). TRIPS is an integrated system in which a person collaborates with a computer to develop a high quality plan to evacuate people from a small island. During the process of building the plan, the system simulates the plan repeatedly based on a probabilistic model of the domain, including predicted weather patterns and their effect on vehicle performance. The system returns an estimate of the plan's success. Additionally, TRIPS invokes PLANMINE on the execution traces produced by simulation, in order to analyze *why* the plan failed when it did. This information can be used to improve the plan. PLANMINE has also been integrated into an algorithm for *automatically* modifying a given plan so that it has a higher probability of achieving its goal. IMPROVE runs PLANMINE on the execution traces of the given plan to pinpoint defects in the plan that most often lead to plan failure. It then applies qualitative reasoning and plan adaptation algorithms to modify the plan to correct the defects detected by PLANMINE.

This paper describes PLANMINE, the data mining component of the above two applications. We show that one cannot simply apply previous sequence discovery algorithms (Srikant and Agrawal, 1996b; Zaki, 1998) for mining execution traces. Due to the complicated structure and redundancy in the data, simple application of the known algorithms generates an enormous number of highly frequent, but unpredictive rules. We use the following novel methodology for pruning the space of discovered sequences. We label each plan as "good" or "bad" depending on whether it achieved its goal or it failed to do so. Our goal is to find "interesting" sequences that have a high confidence of predicting plan failure. We developed a three-step pruning strategy for selecting only the most predictive rules:

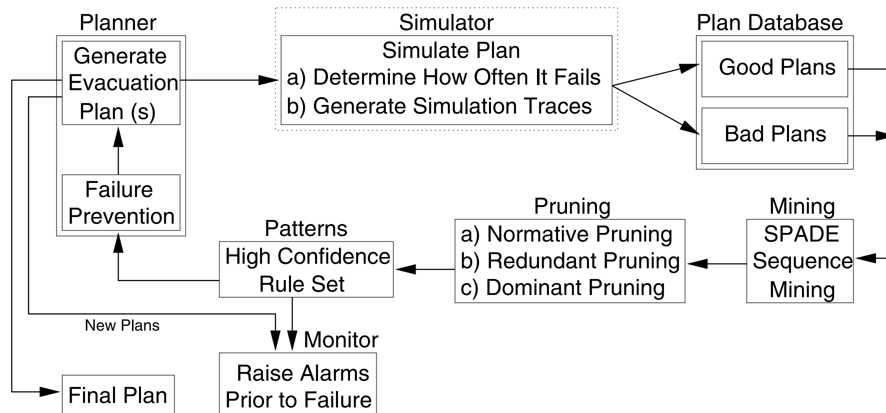

Figure 1. PLANMINE architecture.

1. *Pruning Normative Patterns*: We eliminate all *normative* rules that are consistent with background knowledge that corresponds to the normal operation of a (good) plan, i.e., we eliminate those patterns that not only occur in bad plans, but also occur in the good plans quite often, since these patterns are not likely to be predictive of bad events.
2. *Pruning Redundant Patterns*: We eliminate all *redundant* patterns that have the same frequency as at least one of their proper subsequences, i.e., we eliminate those patterns q that are obtained by augmenting an existing pattern p, while q has the same *frequency* as p. The intuition is that p is as predictive as q.
3. *Pruning Dominated Patterns*: We eliminate all *dominated* sequences that are less predictive than any of their proper subsequences, i.e., we eliminate those patterns q that are obtained by augmenting an existing pattern p, where p is shorter or more general than q, and has a higher *confidence* of predicting failure than q.

These three steps are carried out *automatically* by mining the good and bad plans separately and comparing the discovered rules from the unsuccessful plans against those from the successful plans. The complete architecture of PLANMINE is shown in Figure 1. There are two main goals: (1) to improve an existing plan, and (2) to generate a plan monitor for raising alarms. In the first case, the planner generates a plan and simulates it multiple times. It then produces a database of good and bad plans in simulation. This information is fed into the mining engine, which discovers high frequency patterns in the bad plans. We next apply our pruning techniques to generate a final set of rules that are highly predictive of plan failure. This mined information is used for fixing the plan to prevent failures, and the loop is executed multiple times till no further improvement is obtained. The planner then generates

Parameter	Values
Action	Move, Load, Unload
Outcome	Success, Tardy, Late, Overheat, Blowout, Flat, Crash
Route	Delta-Exodus, Exodus-Barnacle-Exodus, etc.
From	Abyss, Barnacle, Calypso, Delta, Exodus
To	Abyss, Barnacle, Calypso, Delta, Exodus
AtLocation	Abyss, Barnacle, Calypso, Delta, Exodus
Cargo	Person-X, Person-Y, People1, People2, etc.
VehicleType	Helicopter, Truck
Vehicle	Helicopter1, Truck1, Truck2, etc.
Weather	Good, Fair, Rough, Hazardous

Figure 2. Plan database parameters.

the final plan. For the second goal, the planner generates multiple plans, and creates a database of good and bad plans (there is no simulation step). The high confidence patterns are mined as before, and the information is used to generate a plan monitor that raises alarms prior to failures in new plans.

To experimentally validate our approach, we show that IMPROVE does not work well if the PLANMINE component is replaced by less sophisticated methods for choosing which parts of the plan to repair. We also show that the output of PLANMINE can be used to build execution monitors which predict failures in a plan before they occur. We were able to produce monitors with 100% precision, that signal 90% of all the failures that occur.

The rest of the paper is organized as follows. In Section 2 we describe the plan database, and precisely formulate the data mining task. Section 3 presents the algorithm used for sequence discovery. In Section 4 we describe our automatic methodology for incorporating background or normative knowledge about the data for extracting the predictive sequences in a plan database. An experimental evaluation of our approach is presented in Section 5. We look at related work in Section 6, and present our conclusions and directions for future work in Section 7.

2. Discovery of Plan Failures: Sequence Mining

The input to PLANMINE consists of a database of plans for evacuating people from one city to another. Each plan is tagged *Failure* or *Success* depending on whether or not it achieved its goal. Each plan has a unique identifier, and

PLAN DATABASE

PlanId	Time	EventId	Action	Outcome	Route	From	To	AtLocation	Cargo	Vehicle	VehicleId	Weather
1	10	78	Move	Success	Delta-Exodus	Delta	Exodus			Helicopter	Heli1	Good
1	20	84	Load	Success				Exodus	People7		Heli1	
1	30	85	Move	Flat	Exodus-Barnacle-Abyss	Exodus	Barnacle			Helicopter	Heli1	Fair
1	40	101	Unload	Crash				Barnacle	People7	Helicopter	Heli1	Hazardous
2	10	7	Move	Flat	Delta-Calypso-Delta	Delta	Calypso			Truck	Truck1	Good
2	20	10	Move	Breakdown	Delta-Calypso-Delta	Calypso	Delta			Truck	Truck1	Good

Figure 3. Example plan database.

a sequence of actions or events. Each event is composed of several different fields or items including the event time, the unique event identifier, the action name, the outcome of the event, and a set of additional parameters specifying the weather condition, vehicle type, origin and destination city, cargo type, etc. An example of the different parameter values is shown in Figure 2, and some example plans are shown in Figure 3.

While routing people from one city to another using different vehicles, the plan will occasionally run into trouble. The outcome of the event specifies the type of error that occurred, if any. Only a few of the errors, such as a helicopter crashing or a truck breaking down, are *severe*, and cause the plan to fail. However, a sequence of non-severe outcomes may also be the cause of a failure. For example, a rule might be *(Load People-7 Truck-1)* $\mapsto$ *(Move Flat Truck-1)* $\mapsto$ *(Move Late Truck-1)* $\mapsto$ *(Load People-7 Heli-1)* $\mapsto$ *(Move Crash Heli-1 RoughWeather)* $\Rightarrow$ *Failure*, indicating that the plan is likely to fail if *Truck-1* gets *Late* due to a *Flat*. This causes the *Helicopter-1* to crash, a severe outcome, since the weather gets *Rough* with time.

We now cast the problem of mining for causes of plan failures as the problem of finding *sequential patterns* (Agrawal and Srikant, 1995). Let $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$ be a set of m distinct attributes, also called *items*. Each distinct parameter and value pair is an item. For example, in Figure 2, *Action = Move*, *Action = Load*, etc., are all distinct items. An *itemset* is an unordered collection of items, all of which are assumed to occur at the same time. Without loss of generality, we assume that the items are mapped to integers, and that items of an itemset are sorted in increasing order. An itemset i is denoted as $(i_1 i_2 \cdots i_k)$, where i_j is an item.

A *sequence* is an ordered list of itemsets. A sequence α is denoted as $(\alpha_1 \mapsto \alpha_2 \mapsto \cdots \mapsto \alpha_n)$, where each sequence element α_j is an itemset. An item can occur only once in an itemset, but it can occur multiple times in different itemsets of a sequence. A sequence with k items ($k = \sum_j |\alpha_j|$) is called a *k-sequence*. We say that α is a *subsequence* of β, denoted as $\alpha \preceq \beta$, if there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_j \subseteq b_{i_j}$ for all a_j. For example, $B \mapsto AC$ is a subsequence of $AB \mapsto E \mapsto ACD$. We say that

α is a proper subsequence of β, denoted $\alpha \prec \beta$, if $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$. If α is obtained by removing a single item from β, we write it as $\alpha \prec_1 \beta$. For example, $(B \mapsto AC) \prec_1 (BE \mapsto AC)$.

We now cast our plans in the sequence mining framework. An *event* $\mathcal{E}$ is an itemset, and has a unique identifier. For example, in Figure 3, the second row (event) of the first plan corresponds to the itemset *(84, Load, Success, Exodus, People7, Heli1)*. A *plan* or *plan-sequence* $\mathcal{S}$ has a unique identifier and is associated with a sequence of events $\mathcal{E}_1 \mapsto \mathcal{E}_2 \mapsto \cdots \mapsto \mathcal{E}_n$. Without loss of generality, we assume that no plan has more than one event with the same time-stamp, and that the events are sorted by the event-time. The input plan database, denoted $\mathcal{D}$, consists of a number of such plan-sequences.

Support. A plan $\mathcal{S}$ is said to *contain* a sequence α, if $\alpha \preceq \mathcal{S}$, i.e., if α is a subsequence of the plan-sequence $\mathcal{S}$. The *support* or *frequency* of a sequence α, denoted $fr(\alpha, \mathcal{D})$ is the fraction of plans in the database $\mathcal{D}$ that contain α, i.e.,

$$fr(\alpha, \mathcal{D}) = \frac{|\{\alpha \preceq \mathcal{S} \in \mathcal{D}\}|}{|\mathcal{D}|}.$$

According to this definition a sequence α is counted only once per plan even though it may occur multiple times in that plan. It is easy to modify this definition to count a sequence multiple times per plan, if the semantics of the problem require it. Given a user-specified threshold called the *minimum support* (denoted *min_sup*), we say that a sequence is *frequent* if $fr(\alpha, \mathcal{D}) \geq min_sup$.

Confidence. Let α and β be two sequences. The *confidence* of a sequence rule $\alpha \Rightarrow \beta$ is the conditional probability that sequence β occurs, given that α occurs in a plan, given as

$$Conf(\alpha, \mathcal{D}) = \frac{fr(\alpha \mapsto \beta, \mathcal{D})}{fr(\alpha, \mathcal{D})}.$$

Given a user-specified threshold called the *minimum confidence* (denoted *min_conf*), we say that a sequence is *strong* if $Conf(\alpha, \mathcal{D}) \geq min_conf$.

Discovery task. Given a database $\mathcal{D}$ of good and bad plans, tagged as *Success* and *Failure*, respectively, the problem of discovering causes of plan failures can be formulated as finding all strong rules of the form $\alpha \Rightarrow$ *Failure*, where α is a frequent sequence. This task can be broken into two main steps:

1. Find all frequent sequences. This step is computationally and I/O intensive, since there can be potentially an exponential number of frequent sequences.

2. Generate all strong rules. Since we are interested in predicting failures, we only consider rules of the form $\alpha \Rightarrow$ *Failure*, even though our formulation allows rules with consequents having multiple items. The rule generation step has relatively low computational cost.

We use the SPADE algorithm (Zaki, 1998) to efficiently enumerate all the frequent sequences. Generally, a very large number of frequent patterns are discovered in the first step, and consequently a large number of strong rules are generated in the second step. If one thinks of the frequent sequence discovery step as the *quantitative* step due to its high computational cost, then the rule generation step is the *qualitative* one, where the quality of the discovered rules is important and not the quantity. The main focus of this paper is on how to apply effective pruning techniques to reduce the final set of discovered rules, retaining only the rule that are most predictive of failure, and on how to do this automatically.

3. Sequential Pattern Discovery Algorithm

We now briefly describe the SPADE (Zaki, 1998) algorithm that we used for efficient discovery of sequential patterns. SPADE is disk-based and is designed to work with very large datasets.

SPADE uses the observation that the subsequence relation $\preceq$ defines a partial order on the set of sequences, also called a *specialization relation* (Gunopulos et al., 1997). If $\alpha \preceq \beta$, we say that α is more general than β, or β is more specific than α. The second observation used is that the relation $\preceq$ is a *monotone specialization relation* with respect to the frequency $fr(\alpha, \mathcal{D})$, i.e., if β is a frequent sequence, then all subsequences $\alpha \preceq \beta$ are also frequent. The algorithm systematically searches the sequence lattice spanned by the subsequence relation in a breadth-first (*level-wise*) or depth-first manner, from the most general to the maximally specific (frequent) sequences. For example, let the set of frequent items $\mathcal{F}_1 = \{A, B, C\}$, and let the maximal frequent sequences be $(ABC \mapsto A)$, and $(B \mapsto A \mapsto C)$, then Figure 4 shows the lattice of frequent sequences induced by the maximal elements (note that a sequence is maximal if it is not a subsequence of any other sequence).

Given $\mathcal{F}_k$, the set of frequent sequences of length k, we say that two sequences belong to the same equivalence class if they share a common $k-1$ length prefix. For example, from the $\mathcal{F}_2$ shown in Figure 4, we obtain the following three equivalence classes: $[A] = \{A \mapsto A, A \mapsto C, AB, AC\}$; $[B] = \{B \mapsto A, B \mapsto C, BC\}$; and $[C] = \{C \mapsto A\}$. Each class $[\wp]$ has complete information for generating all frequent sequences with the prefix $\wp$. Each class can thus be solved independently.

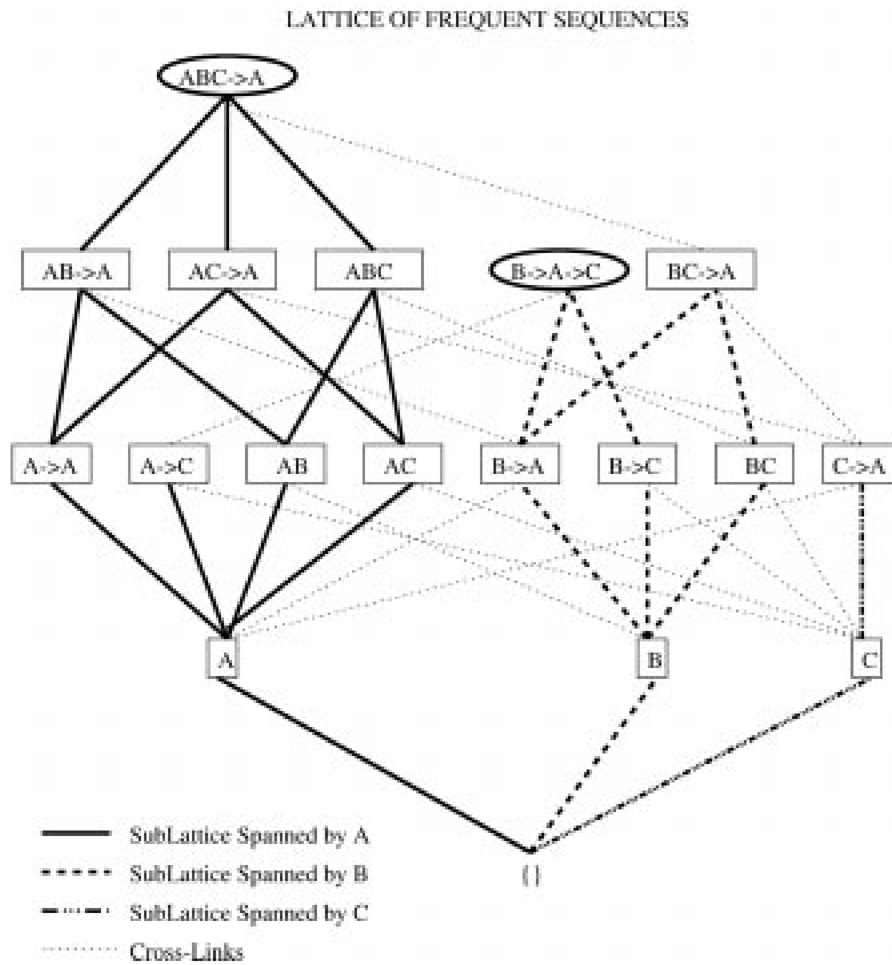

Figure 4. Lattice induced by maximal sequences.

SPADE decomposes the original problem into smaller subproblems by the recursive partitioning into equivalence classes (sub-lattices). This allows each class to be processed entirely in main-memory, and generally requires up to two complete database scans. Figure 5 shows the outline of the algorithm. SPADE systematically searches the sub-lattices in a breadth-first manner (SPADE can also use a depth-first search if main-memory is limited), i.e. it starts with the frequent single items, and during each iteration the frequent sequences of the previous level are extended by one more item. Before computing the support of a newly formed sequence, a check is made to ensure that all its subsequences are frequent. If any subsequence is found to be infrequent, then the sequence cannot possibly be frequent due to the monotone

```
SPADE (min_sup, D):
1. F_1 = { frequent items };
2. F_2 = { frequent 2-sequences };
3. for all classes C_2 ∈ F_2 do
4.    for (k = 3; C_{k-1} ≠ ∅; k = k + 1) do
5.       for all classes [ε] ∈ C_{k-1} do
6.          for all sequences α, β ∈ [ε] do
7.             if (|L(α) ∩ L(β)| ≥ min_sup) then
8.                [ℵ] = [ℵ] ∪ (α ∪ β)
9.          C_k = C_k ∪ ℵ;
```

Figure 5. The SPADE algorithm.

support property. This pruning criterion is extremely effective in reducing the search space. For applying global pruning across all equivalence classes, all the cross class links have to maintained, which corresponds to storing all frequent sequences in memory. If memory is limited, then only local pruning within a class can be applied.

For fast frequency computation, SPADE maintains, for each distinct item, an inverted list (denoted $\mathcal{L}$) of *(PlanId, EventTime)* pairs where the item occurs. For example, from our initial database in Figure 3, we obtain, $\mathcal{L}(Move) = \{(1, 10)(1, 30)(2, 10)(2, 20)\}$, and $\mathcal{L}(Flat) = \{(1, 30)(2, 10)\}$. To compute the support of a sequence from any two of its subsets, their lists are intersected in a special way. For example, to obtain $\mathcal{L}(Move\ \ Flat) = \{(1, 30)(2, 10)\}$, an equality check is made for each pair, and to obtain $\mathcal{L}(Move \mapsto Flat) = \{(1, 30)\}$, a check is made whether there exists any *EventTime* for *Flat* that follows any *EventTime* for *Move*, for pairs with the same *PlanId*.

4. Mining Strong Sequence Rules

We now describe our methodology for extracting the predictive sequences on a sample plan database. Let $\mathcal{D}_g$ and $\mathcal{D}_b$ refer to the good and bad plans, respectively. All experiments were performed on an SGI machine with a 100MHz MIPS processor and 256MB main memory, running IRIX 6.2.

4.1. *Mining the whole database* ($\mathcal{D} = \mathcal{D}_g + \mathcal{D}_b$)

We used an example database with 522 items, 1000 good plans and 51 bad plans, with an average of 274 events per good plan, 196 events per bad plan, and an average event length of 6.3 in both. We mined the entire database of

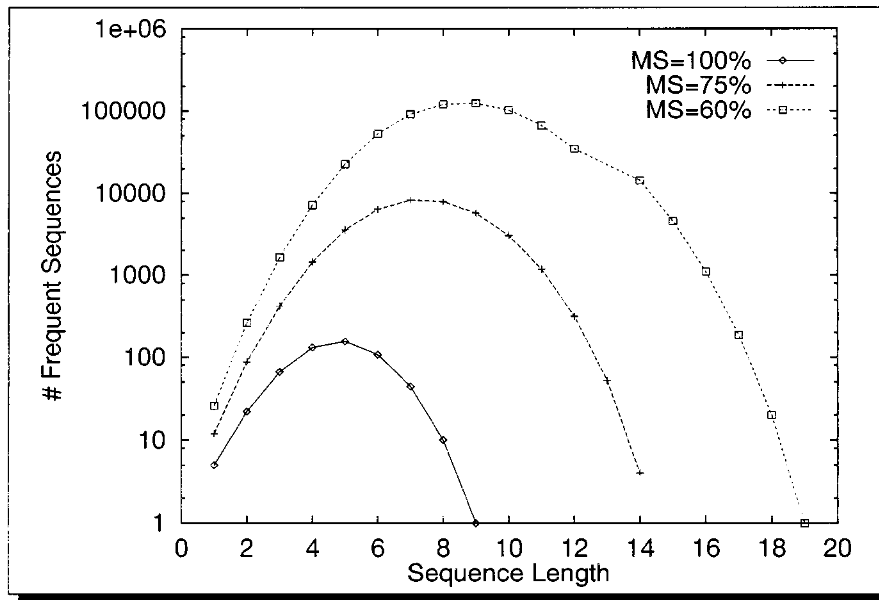

Figure 6. Sequences of different length mined at various levels of minimum support (MS).

good and bad plans for frequent sequences. Since there are about 5% bad plans, we would have to use a minimum support of at least 5% to discover patterns that have some failure condition. However, even at 100% minimum support, the algorithm proved to be intractable. For example, we would find more than a 100 length sequence of the form *Move* $\cdots \mapsto$ *Move*, all 2^{100} of whose subsequences would also be frequent, since about half of the events contain a *Move*. Such long sequences would also be discovered for other common items such as *Success*, *Truck*, etc. With this high level of item frequency, and long plan-sequences, we would generate an exponential number of frequent patterns. Mining the whole database is thus infeasible. Note also that none of these rules can have high confidence, i.e., none can be used to predict plan failure, because they occur in all the good as well as the bad plans. The problem here is that the common strategy of mining for all highly frequent rules and then eliminating all the low confidences ones will be infeasible in this highly structured database.

4.2. *Mining the bad plans ($\mathcal{D}_b$)*

Since we are interested in rules that predict failure, we only need to consider patterns that are frequent in the failed plans. A rule that is frequent in the successful plans cannot have a high confidence of predicting failure. To reduce

Table 1. Discovered patterns and running times.

	MS = 100%	MS = 75%	MS = 60%
#Sequences	544	38386	642597
Time	0.2s	19.8s	185.0s

the plan-sequence length and the complexity of the problem, we decided to focus only on those events that had an outcome other than a *Success*. The rationale is that the plan solves its goal if things go the way we expect, and so it is reasonable to assume that only non-successful actions contribute to failure. We thus removed all actions with a successful outcome from the database of failed plans, obtaining a smaller database of bad plans, which had an average of about 8.5 events per plan.

The number of frequent sequences of different lengths for various levels of minimum support are plotted in Figure 6, while the running times and the total number of frequent sequences is shown in Table 1. At 60% support level we found an overwhelming number of patterns. Even at 75% support, we have too many patterns (38386), most of which are quite useless when we compute their confidence relative to the entire database of plans. For example, the pattern *Move* $\mapsto$ *Truck-1* $\mapsto$ *Move* had a 100% support in the bad plans. However, it is not at all predictive of a failure, since it occurs in every plan, both good and bad. The problem here is that if we only look at bad plans, the confidence of a rule is not an effective metric for pruning uninteresting rules. In particular, every frequent sequence α will have 100% confidence, since $fr(\alpha \mapsto Failure, \mathcal{D}_b)$ is the same as $fr(\alpha, \mathcal{D}_b)$. However, all potentially useful patterns are present in the sequences mined from the bad plans. We must, therefore, extract the interesting ones from this set.

4.2.1. *Reducing discovered sequences*

We can also reduce the number of patterns generated by putting limits on the maximum number of itemsets per sequence or the maximum length of an itemset. Figure 7 plots the total number of frequent sequences discovered under length restrictions. For example, there are 38386 total sequences at 75% *min_sup* (ISET-SA). But if we restrict the maximum itemset length to 2, then there are only 14135 sequences. If we restrict the maximum number of itemsets per sequence to 3, then we discover only 8037 sequences (ISET-S3), and so on. Due to the high frequency character of our domain, it makes sense to put these restrictions, especially on the maximum length of an itemset,

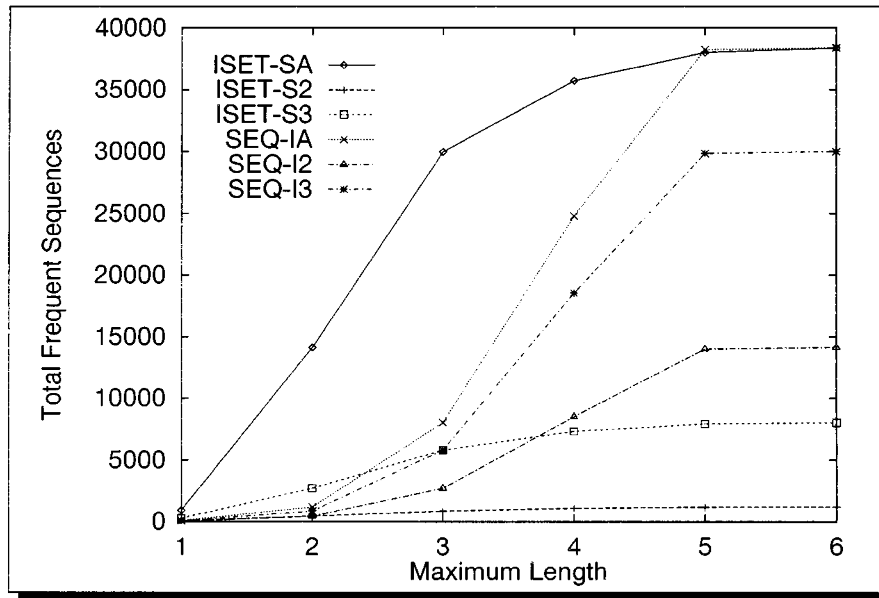

Figure 7. Number of sequences discovered with length rescrictions (75% minimum support). ISET-SA shows all sequences plotted against limits on the itemset length; ISET-S2(3) the sequences with 2(3) itemsets per sequence plotted against the itemset length; SEQ-IA all sequences against varying #itemsets/sequence; and SEQ-I2(3) the sequences with fixed itemset length of 2(3) against varying #itemsets/sequence.

especially if we want to use a low minimum support value, and discover long sequences.

4.3. *Extracting interesting rules*

A discovered pattern may be uninteresting due to various reasons (Klemettinen et al., 1994). For example, it may correspond to background knowledge, or it may be redundant, i.e., subsumed by another equally predictive but more general pattern. Below we present our pruning schemes for retaining only the most predictive patterns.

4.3.1. *Pruning normative patterns*

Background knowledge plays an important role in data mining (Fayyad et al., 1996). One type of background knowledge, which we call *normative* knowledge, corresponds to a set of patterns that are uninteresting to the user, often because they are obvious. Normative knowledge can be used to constrain or prune the search space, and thereby enhance the performance. Typically, the normative knowledge is hand-coded by an expert who knows

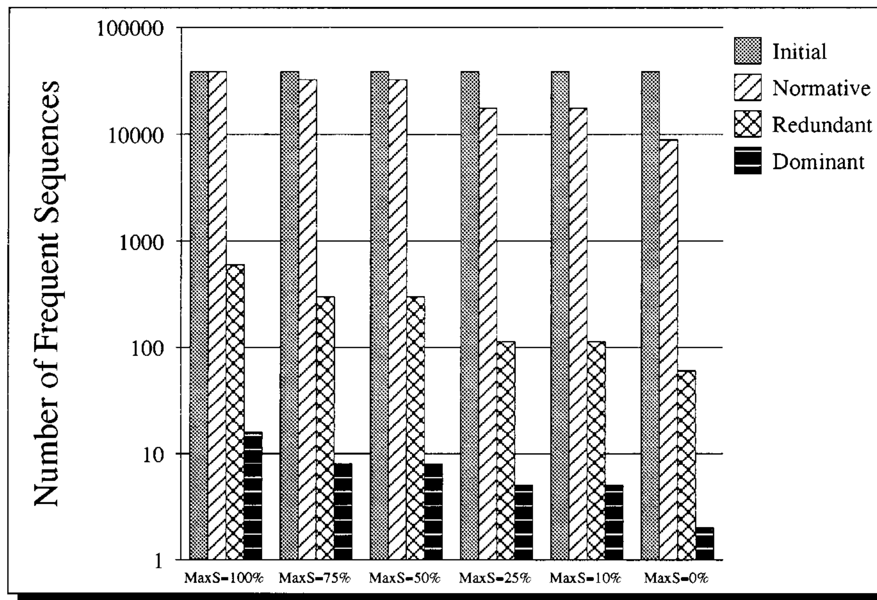

Figure 8. Effect of different pruning techniques.

the domain. In our case, normative knowledge is present in the database $\mathcal{D}_g$ of good plans. The good plans describe the normal operations, including the minor problems that may arise frequently but do not lead to plan failure. We automatically extract the normative knowledge from the database of good plans as follows: We first mine the bad plans $\mathcal{D}_b$ for frequent sequences. We also compute the support of the discovered sequences in the successful plans. We then eliminate those sequences that have a high support (greater than a user-specified *max_sup* in $\mathcal{D}_g$) in the successful plans, since such sequences represent the normal events of successful plans. This automatic technique for incorporating background knowledge is effective in pruning the uninteresting patterns. Figure 8 shows the reduction in the number of frequent sequences by excluding normative patterns. At 25% maximum support in $\mathcal{D}_g$, we get more than a factor of 2 reduction (from 38386 to 17492 rules).

4.3.2. *Pruning redundant patterns*

Even after pruning based on normative knowledge, we are left with many patterns (17492) which have high frequency and high confidence, i.e., they are highly predictive of failure. The problem is that the existence of one good rule implies the existence of many almost identical, and equally predictive rules. For example, suppose *(Flat Truck-1)* $\mapsto$ *(Overheat Truck-1)* is highly predictive, and that the first action of every plan is a *Move*. In this case, *Move*

```
/* Mine Bad Plans */
1. I = SPADE (min_sup, D_b)
/* Prune Normative Patterns */
2. H = {α ∈ I | fr(α, D_g) < max_sup}
/* Prune Redundant Patterns */
3. R = {α ∈ H | ∄β ≻_1 α such that fr(α, D_b) = fr(β, D_b)
        and fr(α, D_g) = fr(β, D_g)}
/* Prune Dominated Patterns */
4. F = {α ∈ R | ∄β ≻_1 α such that fr(α, D_b) ≥ fr(β, D_b)
        and fr(α, D_g) ≤ fr(β, D_g)}
```

Figure 9. The complete PLANMINE algorithm.

$\mapsto$ *(Flat Truck-1)* $\mapsto$ *(Overheat Truck-1)* will be equally predictive, and will have the same frequency. The latter sequence is thus redundant. Formally, β is *redundant* if there exists $\alpha \prec_1 \beta$, with the same support as β both in good and bad plans (recall that $\alpha \prec_1 \beta$, if α is obtained by removing a single item from β).

Given the high frequency of some actions in our domain, there is tremendous redundancy in the set of highly predictive and frequent patterns obtained after normative pruning. Therefore, we prune all redundant patterns. Figure 8 shows that by applying redundant pruning in addition to normative pruning we are able to reduce the pattern set from 17492 down to 113. This technique is thus very effective.

4.3.3. *Pruning dominated patterns*

After applying normative and redundant pruning, there still remain some patterns that are very similar. Above, we pruned rules which had equivalent support. We can also prune rules based on confidence. We say that β is *dominated* by α, if $\alpha \prec_1 \beta$, and α has lower support in good and higher support in bad plans (i.e., α has higher confidence than β). Figure 8 shows that dominant pruning, when applied along with normative and redundant pruning, reduces the rule set from 113 down to only 5 highly predictive patterns. The combined effect of the three pruning techniques is to retain only the patterns that have the highest confidence of predicting a failure, where confidence is given as:

$$Conf(\alpha) = \frac{fr(\alpha \mapsto Failure, \mathcal{D})}{fr(\alpha, \mathcal{D})} = \frac{|\alpha \preceq \mathcal{S}_b \in \mathcal{D}_b|}{|\alpha \preceq \mathcal{S} \in \mathcal{D}|} \tag{1}$$

Figure 9 shows the complete pruning algorithm. An important feature of our approach is that all steps are automatic. The lattice structure on sequences makes the redundancy and dominance easy to compute. Given the databases

$\mathcal{D}_b$ and $\mathcal{D}_g$, min_sup, and max_sup, the algorithm returns the set of the most predictive patterns.

5. Experimental Evaluation

In this section we present an experimental evaluation of PLANMINE. We show how it is used in the TRIPS (Ferguson and James, 1998) and IMPROVE (Lesh et al., 1998) applications, and how it is used to build plan monitors.

5.1. *TRIPS and IMPROVE applications*

TRIPS is a collaborative planning system in which a person and a computer develop an evacuation plan. TRIPS uses simulation and data mining to provide helpful analysis of the plan being constructed. At any point, the person can ask TRIPS to simulate the plan. The percentage of time that the plan succeeds in simulation provides an estimate of the plan's true probability of success. Techniques for quick estimation are important because past methods for assessing probabilistic plans have focused on analytic techniques which are exponential in the length of the plan (Kushmerick et al., 1995). After a plan has been simulated, the next step is to run PLANMINE on the execution traces in order to find explanations for why the plan failed when it did. The point of mining the execution traces is to determine which problems are the most significant, or at least which ones are most correlated with plan failure. We believe that this information will help focus the user's efforts on improving the plan.

It is difficult to quantify the performance of TRIPS or how much the PLANMINE component contributes to it. However, both seem to work well on our test cases. In one example, we use TRIPS to develop a plan that involves using two trucks to bring the people to the far side of a collapsed bridge near the destination city. A helicopter then shuttles the people, one at a time, to the destination city. The plan works well unless the truck with the longer route gets two or more flat tires, which delays the truck. If the truck is late, then the helicopter is also more likely to crash, since the weather worsens as time progresses. On this example, PLANMINE successfully determined that *(Move Truck1 Flat)* $\rightarrow$ *(Move Truck1 Flat)* $\Rightarrow$ *Failure*, as well as *(Move Heli1 Crash)* $\Rightarrow$ *Failure*, is a high confidence rule for predicting plan failure.

We now discuss the role of PLANMINE in IMPROVE, a fully automatic algorithm which modifies a given plan to increase its probability of goal satisfaction (Lesh et al., 1998). IMPROVE first simulates a plan many times and then calls PLANMINE to extract high confidence rules for predicting plan failure. IMPROVE then applies qualitative reasoning and plan adaptation

Table 2. Performance of IMPROVE (averaged over 70 trials).

	initial plan length	final plan length	initial success rate	final success rate	num. plans tested
IMPROVE	272.3	278.9	0.82	0.98	11.7
RANDOM	272.3	287.4	0.82	0.85	23.4
HIGH	272.6	287.0	0.82	0.83	23.0

techniques by adding actions to make the patterns that predict failure less likely to occur. For example, if PLANMINE produces the rule *(Truck1 Flat)* $\rightarrow$ *(Truck1 Overheat)* $\Rightarrow$ *Failure*, then IMPROVE will conclude that *either* preventing $Truck1$ from getting a flat or from overheating might improve the plan. In each iteration, IMPROVE constructs several plans which might be better than the original plan. If any of the plans perform better in simulation than the original plan, then IMPROVE repeats the entire process on the new best plan in simulation. This process is repeated until no suggested modification improves the plan.

Table 2 shows the performance of the IMPROVE algorithm, as reported in Lesh et al. (1998), on a large evacuation domain that contains 35 cities, 45 roads, and 100 people. The people are scattered randomly in each trial, and the goal is always to bring all the people, using two trucks and a helicopter, to one central location. The trucks can hold 25 people and the helicopter only one person, so the plans involve multiple round trips. The plans succeed unless a truck breaks down or the helicopter crashes. For each trial we generate a random set of road conditions and rules which give rise to a variety of malfunctions in the vehicles, such as a truck getting a flat tire or overheating. Some malfunctions worsen the condition of the truck and make other problems, such as the truck breaking down, more likely. The process is not completely random in that by design there usually exists some sequence of two to three malfunctions that makes a breakdown or crash very likely. Furthermore, there are always several malfunctions, such as trucks getting dents or having their windows cracked, that occur frequently and never cause other problems. We use a domain-specific greedy scheduling algorithm to generate initial plans for this domain. The initial plans contain over 250 steps.

We compared IMPROVE with two less sophisticated alternatives. The RANDOM approach modifies the plan randomly five times in each iteration, and chooses the modification that works best in simulation. The HIGH

approach replaces the PLANMINE component of IMPROVE with a technique that simply tries to prevent the malfunctions that occur most often. As shown in Table 2, IMPROVE with PLANMINE increases a plan's probability of achieving its goal, on average, by about 15%, but without PLANMINE only by, on average, about 3%.

5.2. *Plan monitoring*

We now describe experiments to directly test PLANMINE. In each trial, we generate a training and a test set of plan executions. We run PLANMINE on the training set and then evaluate the discovered rules on the test set. We used the sequence rules to build monitors, which observe the execution of the plan, and sound an alarm when a plan failure is likely. The hypothesis behind these tests is that predicting failure accurately will be useful in avoiding errors during plan execution. We used the same evacuation domain described above. The training set had 1000 plan traces, with around 5% plan-failure rate. Only 300 of the good plans were used for background knowledge. We used a min_sup of 60% in the bad plans, and a max_sup of 20% in the good plans.

We run PLANMINE on the training data and use the discovered set of rules $\mathcal{R}$ to build a *monitor* – a function that takes as input the actions executed so far and outputs failure iff any of the rules in $\mathcal{R}$ is a subsequence of the action sequence. For example, a monitor built on the rules *(Truck-1 Flat)* $\mapsto$ *(Truck-1 Overheat)* $\Rightarrow$ *Failure* and *(Truck-2 Flat)* $\mapsto$ *(Truck-2 Flat)* $\Rightarrow$ *Failure* sounds its alarm if *Truck-1* gets a flat tire and overheats, or if *Truck-2* gets two flat tires. The *precision* of a monitor is the percentage of times the monitor signals a failure, and a failure actually occurs (i.e., the ratio of correct failure signals to the total number of failure signals). The *recall* of a monitor is the percentage of failures signaled prior to their occurrence. A monitor that always signals failure has 100% recall and p% precision where p is the rate of plan failure. To generate monitors, first we mine the database of execution traces for sequence rules. We then build a monitor by picking some threshold λ, varied in the experiments, and retain only those rules that have at least λ precision or confidence (see Equation 1) on the training data.

Figure 10a shows the evaluation of the monitors produced with PLANMINE on a test set of 500 (novel) plans. The results are the averages over 105 trials, and thus each number reflects an average of approximately 50,000 separate tests. It plots the precision, recall, and frequency of the mined rules in the test database against the precision threshold in the training data. The frequency graph shows the percent of trials for which we found at least one rule of the given precision. The figure clearly shows that our mining and pruning techniques produce excellent monitors, which have 100% precision with

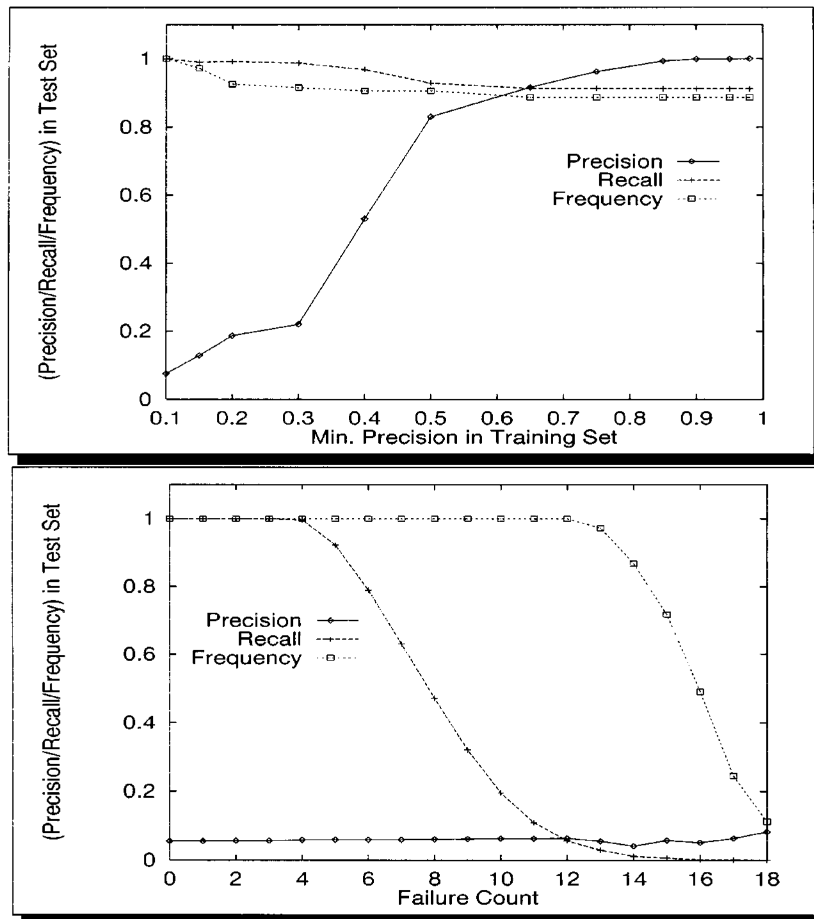

Figure 10. (a) Using PLANMINE for prediction; (b) using failure count for prediction.

recall greater than 90%. We can produce monitors with significantly higher recall, but only by reducing precision to around 50%. The desired tradeoff depends on the application. If plan failures are very costly then it might be worth sacrificing precision for recall. For comparison we also built monitors that signaled failure as soon as a fixed number of malfunctions of any kind occurred. Figure 10b shows that this approach produces poor monitors, since there was no correlation between the number of malfunctions and the chance of failure (precision).

We also investigated whether or not data mining was really necessary to obtain these results. The graphs in Figure 11 describe the performance of the system if we limit the length of the rules. For example, limiting the rules to length two corresponds to building a monitor out of the pairs of actions that

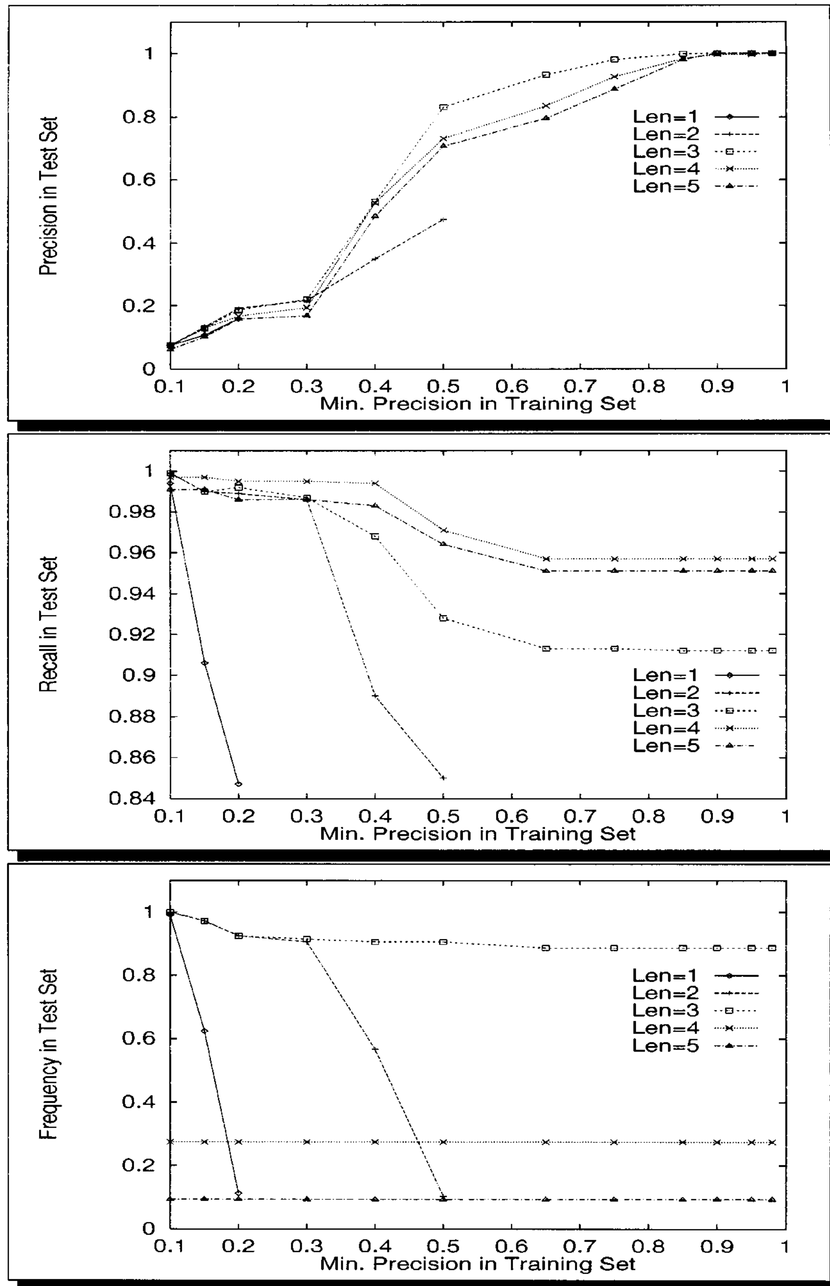

Figure 11. (a) Precision, (b) recall and (c) frequency of discovered sequences in test set.

best predict failure. The figure shows the precision, recall, and frequency of the rules of different lengths discovered in the test database plotted against the precision threshold in the training data. The frequency graph shows the percent of trials for which we found at least one rule of the given precision and the given length. For example, at 0.5 training precision, out of the 105 trials, we found a frequent rule of length 3 in more than 90% of the trials, and a rule of length 5 in 10% of the trials, and so on.

The results indicate that monitors built out of rules of length less than three are much worse than monitors built out of longer rules. In particular, the graphs show that there were very few rules of length one or two with even 50% or higher precision. Furthermore, rules of higher length always had better recall for the same level of precision. However, only 30% of our experiments produced useful rules of length four and only 10% produced rules of length five. But when these rules were produced, they were highly effective.

6. Related Work

We now describe related work in sequence mining, classification and planning.

6.1. *Sequential patterns*

The problem of mining sequential patterns was introduced in (Agrawal and Srikant, 1995). They also presented three algorithms for solving this problem. Two of the algorithms, *AprioriSome* and *DynamicSome*, generated only maximal sequential patterns. However, many applications require all frequent patterns. The *AprioriAll* algorithm found all patterns, and was shown to perform equal to or better than the other approaches. In subsequent work (Srikant and Agrawal, 1996b), the same authors proposed the GSP algorithm that outperformed *AprioriAll* by up to 20 times. GSP also introduced constraints (such as maximum gap, minimum gap and sliding windows), and item hierarchies on the discovered sequences. Recently, the SPADE algorithm (Zaki, 1998), was shown to outperform GSP by more than a factor of 2 and by more than an order of magnitude if some pre-processed information is also kept.

The problem of finding *frequent episodes* in a sequence of events was presented in Mannila et al. (1995). An episode consists of a set of events and an associated partial order over the events. Their algorithm is targeted to discover the frequent episodes in a single long event sequence, although it can be adapted to find frequent sequences across many different plan-sequences, as in our study. They further extended their framework in Mannila

and Toivonen (1996) to discover *generalized episodes*, which allows one to express arbitrary unary conditions on individual episode events, or binary conditions on event pairs. The TASA system (Hatonen et al., 1996) applied frequent episode mining to discover frequent alarms in telecommunication network databases. The MSDD (Oates et al., 1997) algorithm finds patterns in multiple streams of data. The patterns are evaluated using a metric similar to the chi-square significance test.

The high item frequency in our domain distinguishes it from previous applications of sequential patterns. For example, while extracting patterns from mail order datasets (Srikant and Agrawal, 1996b), the database items had very low support, so that support values like 1% or 0.1% were used. For discovering frequent alarm sequences in telecommunication network alarm databases (Hatonen et al., 1996) the support used was also 1% or less. Furthermore, in previous applications (Hatonen et al., 1996) only a database of alarms (i.e., only the bad events) was considered, and the normal events of the network were ignored. In our approach, since we wanted to predict plan failures, we have shown that considering only the frequent sequences in the bad plans is not sufficient (all these have 100% confidence of predicting failure). We also had to use the good plans as a source of "normative" information, which was used to prune out unpredictive rules.

Since sequential patterns are essentially associations over temporal data, they utilize some of the ideas initially proposed for the discovery of association rules (Agrawal et al., 1996; Zaki et al., 1997a). While association rules discover only intra-plan (transaction) patterns (itemsets), sequential patterns also include the inter-plan patterns (sequences). The set of all frequent sequences is thus a superset of the set of all frequent itemsets. Other relevant work includes discovery of association rules when the quantity of items bought is also considered (Srikant and Agrawal, 1996a), and when a taxonomy is imposed on the items (Han and Fu, 1995; Srikant and Agrawal, 1995). Lately, there has been an increasing interest in developing efficient parallel algorithms for these problems (Agrawal and Shafer, 1996; Zaki et al., 1997b). Parts of this paper have also appeared in Zaki et al. (1998).

6.2. *Classification*

The problem formulation for predicting failures that we presented in Section 2 can be thought of as discovering high confidence classification rules, where the class being predicted is whether a plan fails. However, there are several characteristics that render traditional classifiers ineffective in our domain. Firstly, we have a large number of attributes, many of which have missing values. Secondly, we are trying to predict rare events with very low frequency, resulting in skewed class distribution. Lastly, and most import-

antly, we are trying to predict sequence rules, while most previous work only targets non-temporal domains. The main difficulty is in finding good features to use to represent the plan traces for decision trees construction. One can find a reasonable set of features for describing an individual event (i.e., action in the plan) and then have a copy of this feature set by every time step. However, very soon we run into trouble, since our feature space of multiple event sequences can become exponential. A method to control this explosion is to bound the feature length to sequences of size 2 for example. However, this is likely to miss out on longer predictive rules. As we showed in Figure 11, it was not uncommon to find predictive sequences of length 5 in out datasets.

An approach similar to ours, but applied in a non-temporal domain, is the partial classification work in Ali et al. (1997). They try to predict rules for a specific class out of two or more classes. They isolate the examples belonging to a given class, mine frequent associations, and then compare the confidence based on the frequency for that class, and the frequency in the remaining database, which is similar to our background pruning. However, they don't do any redundant or dominant pruning and, as we mentioned above, they do not consider sequences over time. A brute-force method for mining high-confidence classification rules using frequent associations was presented in Bayardo (1997). He also describes several pruning strategies to control combinatorial explosion in the number of candidates counted. One key difference is that we are working in the sequence domain, as opposed to the association domain. The Brute algorithm (Riddle et al., 1994) also performs a depth-bounded brute-force search for predictive rules, returning the k best ones. In one of their datasets applied to Boeing parts they do consider time, but their treatment is different. The dataset consists of part type and the time it spends at a particular workstation in a semi-automated factory. They treat time as another attribute, and the discovered rules may thus have a temporal component. In our data format, each instance corresponds to a sequence of event sets. Time is not an attribute but is used to order the events, and we explicitly mine predictive sequences in this database.

6.3. *Planning*

There has been much research on analyzing planning episodes (i.e., invocations of the planner) to improve future planning performance in terms of efficiency or quality (e.g. Minton, 1990). Our work is quite different in that we are analyzing the performance of the plan, and not the planner.

In McDermott (1994), a system is described in which a planning robot analyzes simulated execution traces of its current plan for bugs, or discrepancies between what was expected and what occurred. Each bug in the

simulations is classified according to an extensive taxonomy of failure modes. This contrasts to our work in which we mine patterns of failure from databases of plans that contain many problems, some minor and some major, and the purpose of analysis is to discover important trends that distinguish plan failures from successes.

CHEF (Hammond, 1990) is a case-based planning system that also analyzes a simulated execution of a plan. CHEF simulates a plan once, and if the plan fails, applies a deep causal model to determine the cause of failure. This work assumes that the simulator has a correct model of the domain. Reece and Tate (1994) also described a method of producing execution monitors by analyzing the causal structure of the plan.

Additionally, there has been work on extending classical planners to probabilistic domains (e.g. Kushmerick et al., 1995). These methods have been applied to very small plans because the analytic techniques are exponential in the length of the plan. Furthermore, in this work, plan assessment and plan refinement are completely separate. They also mention the importance of using the results of assessing a probabilistic plan to help guide the choice of how to improve it. Currently, the probabilistic planner chooses randomly from among all the actions that *might* improve the plan. As shown in our second set of experiments, the patterns extracted by data mining can help focus a planner on what part of the plan to improve.

Haigh and Veloso (1998) apply machine learning techniques to robot path planning. Their system uses predictive features of the environment to create *situation-dependent costs* for arcs in a path map used by the planner to create routes for the robot. They use regression trees (Breiman et al., 1984) for the mining engine, to learn separate models for each arc in the path. In our domain this would correspond to learning rules for each route in the evacuation domain. However, our goal is different in that we are trying to learn long sequences of events that cause plan failure.

In Howe and Cohen (1995), a methodology called *dependency interpretation* is presented. This methodology uses statistical dependency detection to identify interesting (unusually frequent or infrequent) patterns in plan execution traces. They then interpret the patterns using a weak model of the planner's interaction with its environment to explain how the patterns might have been caused by the planner. One limitation is that once the patterns have been detected, the user is responsible for picking up an interesting pattern from among the set of mined patterns, and using it for interpretation. In contrast, we automatically extract the set of the highly predictive patterns. They also applied the discovered patterns for preventing plan failures. However, they detect dependencies between a precursor and the failure that immediately follows it, and they found that they were likely to miss dependencies

by not considering longer sequences. Our approach on the other hand detects long failure sequences. In their failure prevention study they only used traces of failure and recovery methods, and did not include other influences (e.g., changing weather). In contrast, we use all possible influences for discovering patterns. In Oates and Cohen (1996), they applied the MSDD algorithm to detect rules among planning operators and the context. Our work contrasts with theirs in that, in addition to detecting the frequent operators in the bad plans, we apply effective pruning techniques to automatically extract the rules most predictive of failure.

7. Conclusions

We presented PLANMINE, an automatic mining method that discovers event sequences causing failures in plans. We developed novel pruning techniques to extract the set of the most predictive rules from highly structured plan databases. Our pruning strategies reduced the size of the rule set by three orders of magnitude. The rules discovered by PLANMINE were extremely useful for understanding and improving plans, as well as for building monitors that raise alarms before failures happen, i.e., we show that not only we can analyze the simulation traces of a single plan to improve it, but we can also analyze multiple plan executions and detect common failure modes.

There are several directions in which this work can be extended. In the experiments we were limited to using a 60% minimum support. A lower value would easily generate more than a million patterns. But a high support value can miss long failure sequences. It will only find fragments of the long sequences. We would thus like to be able to lower the minimum support threshold, and we would like to do this without making the problem intractable. One reason for the combinatorial explosion of patterns with decreasing support is that we do not impose any restrictions on the event times. It might be reasonable to assume that failures are caused by events that follow closely in time, for example only within a specific window of time. Such constraints can significantly reduce the number of patterns and can enable us to mine at lower support levels. Other types of constraints include putting restrictions on the sequence lengths, minimum and maximum gaps between successive sequence elements, and so on.

In our current approach we first mine the bad plans, and then apply the pruning in a separate step by comparing the support in both the good and bad plans. A promising direction is to incorporate the pruning directly into the first step, and to mine both the databases simultaneously. This can result in significant speedups by pruning patterns as early in the computation as possible. One can perhaps use information about the planner and the kinds of

action sequences that can even be generated to improve the efficiency of this application significantly.

Using support as a percentage of the whole plan database can also be potentially limiting. For example, if the planner performs a wide variety of plans for differing goals, we would need a lower support threshold to compensate for the diversity. While incorporating constraints on discovered patterns is one solution, an alternative would be to change the denominator in the frequency formula to reflect similarity in the goals of the plans. It would also be interesting to study the long term effects of data mining, i.e., what happens if the process is repeated on new traces? How does one merge new rules with the existing ones? and so on.

References

Agrawal, R. & Shafer, J. (1996). Parallel Mining of Association Rules. *IEEE Trans. on Knowledge and Data Engg.* **8**(6): 962–969.

Agrawal, R. & Srikant, R. (1995). Mining Sequential Patterns. In *11th Intl. Conf. on Data Engg.*

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. & Verkamo, A. I. (1996). Fast Discovery of Association Rules. In Fayyad, U. and et al. (eds.) *Advances in Knowledge Discovery and Data Mining*, 307–328. AAAI Press, Menlo Park, CA.

Ali, K., Manganaris, S. & Srikant, R. (1997). Partial Classification using Association Rules. In *3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining*.

Bayardo, R.J. (1997). Brute-force Mining of High-confidence Classification Rules. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*.

Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984). *Classification and Regression Trees*. Belmont: Wadsworth.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996). From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining.* (1996).

Ferguson, G. & James, A. (1998). TRIPS: An Integrated Intelligent Problem-Solving Assistant. In *15th Nat. Conf. AI.*

Gunopulos, D., Khardon, R., Mannila, H. & Toivonen, H. (1997). Data Mining, Hypergraph Transversals, and Machine Learning. In *16th ACM Symp. Principles of Database Systems*.

Haigh, K.Z. & Veloso, M.M. (1998). Learning Situation-dependent Costs: Improving Planning from Probabilistic Robot Execution. In *Intl. Conf. Autonomous Agents*.

Hammond, K. (1990). Explaining and Repairing Plans that Fail. *J. Artificial Intelligence* **45**: 173–228.

Han, J. & Fu, Y. (1995). Discovery of Multiple-level Association Rules from Large Databases. In *21st VLDB Conf.*

Hatonen, K., Klemettinen, M., Mannila, H., Ronkainen, P. & Toivonen, H. (1996). Knowledge Discovery from Telecommunication Network Alarm Databases. In *12th Intl. Conf. Data Engineering*.

Howe, A.E. & Cohen, P.R. (1995). Understanding Planner Behavior. *J. Artificial Intelligence* **76**(1): 125–166.

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H. & Verkamo, A.I. (1994). Finding Interesting Rules from Large Sets of Discovered Association Rules. In *3rd Intl. Conf. Information and Knowledge Management*, 401–407.

Kushmerick, N., Hanks, S. & Weld, D. (1995). An Algorithm for Probabilistic Planning. *J. Artificial Intelligence* **76**: 239–286.

Lesh, N., Martin, N. & Allen, J. (1998). Improving Big Plans. In *15th Nat. Conf. AI*.

Mannila, H. & Toivonen, H. (1996). Discovering Generalized Episodes Using Minimal Occurences. In *2nd Intl. Conf. Knowledge Discovery and Data Mining*.

Mannila, H., Toivonen, H. & Verkamo, I. (1995). Discovering Frequent Episodes in Sequences. In *1st Intl. Conf. Knowledge Discovery and Data Mining*.

McDermott, D. (1994). Improving Robot Plans during Execution. In *2nd Intl. Conf. AI Planning Systems*, 7–12.

Minton, S. (1990). Quantitative Results Concerning the Utility of Explanation-based Learning. *Artificial Intelligence* **42**(2–3).

Oates, T. & Cohen, P. R. (1996). Searching for Planning Operators with Context-dependent and Probabilistic Effects. In *13th Nat. Conf. AI*.

Oates, T., Schmill, M.D., Jensen, D. & Cohen, P.R. (1997). A Family of Algorithms for Finding Temporal Structure in Data. In *6th Intl. Workshop on AI and Statistics*.

Reece, G. & Tate, A. (1994). Synthesizing Protection Monitors from Causal Structure. In *2nd Intl. Conf. AI Planning Systems*.

Riddle, P., Segal, R. & Etzioni, O. (1994). Representation Design and Brute-force Induction in a Boeing Manufacturing Domain. *Applied Aritficial Intelligence* **8**: 125–147.

Srikant, R. & Agrawal, R. (1995). Mining Generalized Association Rules. In *21st VLDB Conf.*

Srikant, R. & Agrawal, R. (1996a). Mining Quantitative Association Rules in Large Relational Tables. In *ACM SIGMOD Conf. Management of Data*.

Srikant, R. & Agrawal, R. (1996b). Mining Sequential Patterns: Generalizations and Performance Improvements. In *5th Intl. Conf. Extending Database Technology*.

Zaki, M.J., Parthasarathy, S., Ogihara, M. & Li, W. (1997a). New Algorithms for Fast Discovery of Association Rules. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*.

Zaki, M.J., Parthasarathy, S., Ogihara, M. & Li, W. (1997b). New Parallel Algorithms for Fast Discovery of Association Rules. *Data Mining and Knowledge Discovery: An International Journal* **1**(4): 343–373.

Zaki, M.J., Lesh, N. & Ogihara, M. (1998). PLANMINE: Sequence Mining for Plan Failures. In *4th Intl. Conf. Knowledge Discovery and Data Mining*.

Zaki, M.J. (1998). Efficient Enumeration of Frequent Sequences. In *7th Intl. Conf. on Information and Knowledge Management*.