PLANMINE: Sequence Mining for Plan Failures *

Mohammed J. Zaki, Neal Lesh, and Mitsunori Ogihara
Computer Science Department, University of Rochester, Rochester NY 14627
{zaki,lesh,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.

Introduction

In this paper, we present the PLANMINE sequence discovery algorithm for mining information from plan execution traces. PLANMINE has been integrated into two applications in planning: the TRIPS collaborative planning system (Ferguson 98), and the IMPROVE algorithm for improving large, probabilistic plans (Lesh 98).

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.

*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.

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 96; Zaki 97) 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 developed a three-step pruning strategy for selecting only the most predictive rules. First, we eliminate *normative* rules that are consistent with background knowledge that corresponds to the normal operation of a (successful) plan. Second, we eliminate those *redundant* patterns that have the same frequency as at least one of their proper subsequences. Finally, we keep only *dominating* sequences that are more predictive than all of their proper subsequences. 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 part 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. A more detailed version of this paper appears in (Zaki 98).

Discovery of Plan Failures

We cast the problem of mining for causes of plan failures as the problem of finding *sequential patterns* (Agrawal 95). An *itemset* is an unordered collection of items, all of which are assumed to occur at the same time. A *sequence* is an ordered list of itemsets. A sequence α is denoted as $(\alpha_1 \mapsto \cdots \mapsto \alpha_n)$, where each sequence element α_j is an itemset. we say 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 $AB \mapsto E \mapsto ACD$. If α is obtained by removing a single item from β, we write it as $\alpha \prec_1 \beta$. 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 as a subsequence. Given a user-specified threshold called the *minimum support* (*min_sup*), we say that a sequence is *frequent* if $fr(\alpha, \mathcal{D}) \geq min\_sup$. The *confidence* of a sequence rule $\alpha \Rightarrow \beta$, given as $\frac{fr(\alpha \mapsto \beta, \mathcal{D})}{fr(\alpha, \mathcal{D})}$, is the conditional probability

of β given that α occurs.

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 a sequence of events. Each event is an itemset composed of different items such as the action name and outcome, and a set of parameters specifying the weather condition, vehicle type, origin and destination city, cargo type, etc. 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, cause the plan to fail. However, a sequence of non-severe outcomes may also be the cause of a failure. Given a database of plans, the problem of discovering causes of plan failures can be formulated as finding high confidence rules of the form $\alpha \Rightarrow$ *Failure*, where α is frequent. For example, a rule might be *(Move Flat Truck-1)* $\mapsto$ *(Move Overheat Truck-1)* $\Rightarrow$ *Failure* indicating that the plan is likely to fail if *Truck-1* gets a *Flat* in one *Move* action, and then *Overheat* in a subsequent one.

Sequential Pattern Discovery Algorithm We use the SPADE (Zaki 97) algorithm for efficient discovery of frequent sequences. SPADE uses the observation that the subsequence relation $\preceq$ induces a lattice which is *downward closed* on the support, i.e., if β is frequent, then all subsequences $\alpha \preceq \beta$ are also frequent. SPADE decomposes the original lattice into smaller sub-lattices, so that each sublattice can be processed entirely in main-memory using a breadth-first or depth-first search for frequent sequences. Starting with the frequent single items, during each step the frequent sequences of the previous level are extended by one more item. Before computing the support of a new sequence, a pruning step ensures that all its subsequences are also frequent, greatly reducing the search space.

Mining Frequent 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 used an SGI machine with a 100MHz MIPS processor and 256MB main memory, running IRIX 6.2.

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 good and bad plans for frequent sequences. 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. Note that none of these rules 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.

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 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.

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

Table 1: Discovered Patterns and Running Times

Table 1 shows the running times and the total number of frequent sequences discovered. 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.

Extracting Interesting Rules

A discovered pattern may be uninteresting due to various reasons (Klemettinen 94). 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.

Pruning Normative Patterns Background knowledge plays an important role in data mining (Fayyad 96). 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 the domain. In our case normative knowledge is present in the database of good plans, $\mathcal{D}_g$. 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 1 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).

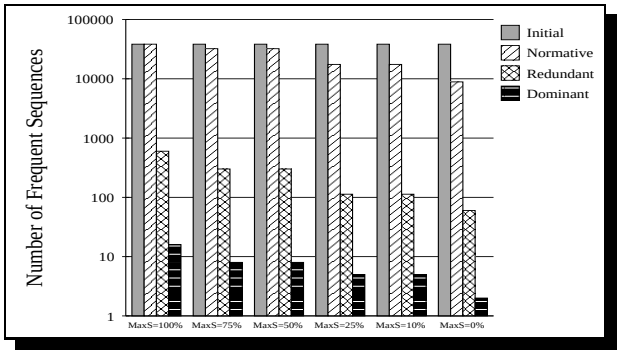

Figure 1: Effect of Different Pruning Techniques

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* $\mapsto$ *(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 1 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.

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 1 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:

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

Figure 2 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.

/* Mine Bad Plans */
1. $\mathcal{I}$ = SPADE ($min\_sup, \mathcal{D}_b$)
/* Prune Normative Patterns */
2. $\mathcal{H} = \{\alpha \in \mathcal{I} \mid fr(\alpha, \mathcal{D}_g) < max\_sup\}$
/* Prune Redundant Patterns */
3. $\mathcal{R} = \{\alpha \in \mathcal{H} \mid \nexists \beta \succ_1 \alpha$ such that $fr(\alpha, \mathcal{D}_b) = fr(\beta, \mathcal{D}_b)$ and $fr(\alpha, \mathcal{D}_g) = fr(\beta, \mathcal{D}_g)\}$
/* Prune Dominated Patterns */
4. $\mathcal{F} = \{\alpha \in \mathcal{R} \mid \nexists \beta \succ_1 \alpha$ such that $fr(\alpha, \mathcal{D}_b) \geq fr(\beta, \mathcal{D}_b)$ and $fr(\alpha, \mathcal{D}_g) \leq fr(\beta, \mathcal{D}_g)\}$

Figure 2: The Complete PLANMINE Algorithm

Experimental Evaluation

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. 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 delay 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

	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

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

increase its probability of goal satisfaction (Lesh 98). 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 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 performs better in simulation than the original plan, then IMPROVE repeats the entire process on the plan that performed best 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 98), 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. For each trial we generate a random set of road conditions, which give rise to a variety of malfunctions. Some malfunctions worsen the condition of the truck and make other problems, such as the truck breaking down more likely. 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%.

Plan Monitoring We now describe experiments to directly test PLANMINE. In each trial, we generate a training and 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 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 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. 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 3a 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. The figure clearly shows that our mining and pruning techniques produce excellent monitors, which have 100% precision with 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 3b 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 4 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 best predict failure. Figure 4 shows that the 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.

Related Work

Sequential Patterns The problem of mining sequential patterns was introduced in (Agrawal 95). The GSP algorithm (Srikant 96) improved upon the earlier work. Recently, the SPADE algorithm (Zaki 97), was shown to outperform GSP by more than a factor of 2. The *frequent episodes* (Mannila 95; 96) approach discovers the frequent patterns in long event sequences. The MSDD (Oates 97) algorithm finds sequences of length two, among blocks of events that happen at fixed intervals.

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 96), 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 96) the support

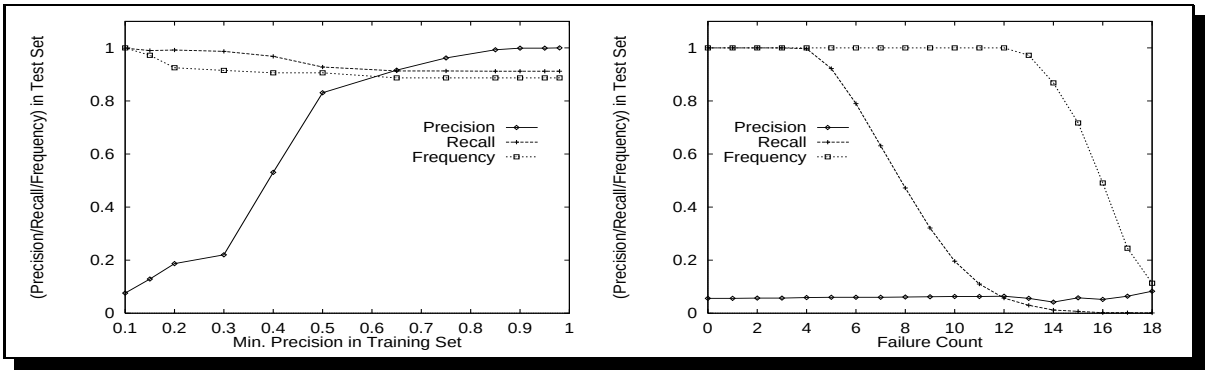

Figure 3: a) Using PLANMINE for Prediction; b) Using Failure Count for Prediction

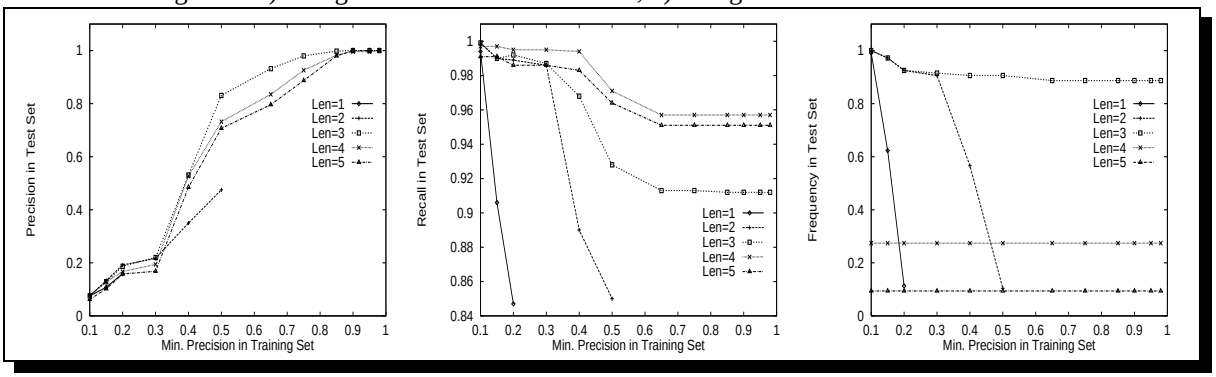

Figure 4: a) Precision, b) Recall, and c) Frequency of Discovered Sequences in Test Set

used was also 1% or less.

Planning There has been much research on analyzing planning episodes to improve future planning performance (Minton 90). Our work is quite different in that we are analyzing the performance of the plan, not the planner. (McDermott 94) describes a system in which a planning robot analyzes simulated execution traces of its current plan for bugs, or discrepancies between what was expected and what occured. We mine patterns of failure from large 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 90) 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.

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.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In *11th Intl. Conf. on Data Engg.*

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R. 1996. *Advances in KDD*. AAAI Press.

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

Hammond, K. 1990. Explaining and repairing plans that fail. *J. Artificial Intelligence* 45:173–228.

Hatonen, K.,et al. 1996. Knowledge discovery from telecommunication network alarm databases. In *Intl. Conf. Data Engg.*

Klemettinen, M., et al. 1994. Finding interesting rules from large sets of discovered association rules. In *Conf. Info. Know. Mgmt.*

Lesh, N.; Martin, N.; and Allen, J. 1998. Improving big plans. In *15th Nat. Conf. AI*.

Mannila, H., and Toivonen, H. 1996. Discovering generalized episodes using minimal occurences. In *2nd Intl. Conf. on KDD*.

Mannila, H.; Toivonen, H.; and Verkamo, I. 1995. Discovering frequent episodes in sequences. In *1st Intl. Conf. on KDD*.

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.; Schmill, M. D.; Jensen, D.; and Cohen, P. R. 1997. A family of algorithms for finding temporal structure in data. In *6th Intl. Workshop on AI and Statistics*.

Srikant, R., and Agrawal, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *5th Intl. Conf. Extending Database Technology*.

Zaki, M. J. 1997. Fast mining of sequential patterns in very large databases. Tech. Report 668, University of Rochester.

Zaki, M. J.; Lesh, N.; and Ogihara, M. 1998. PLANMINE: Sequence Mining for Plan Failures. Tech. Rep. 671, U. Rochester.