Mining Plans for Customer-Class Transformation

Qiang Yang
Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Kowloon, Hong Kong
qyang@cs.ust.hk

Hong Cheng
Department of Computer Science
University of Illinois, Urbana-Champaign
Illinois 61801 USA
hcheng3@uiuc.edu

Abstract

We consider the problem of mining high-utility plans from historical plan databases that can be used to transform customers from one class to other, more desirable classes. Traditional data mining algorithms are focused on finding frequent sequences. But high frequency may not imply low costs and high benefits. Traditional Markov Decision Process (MDP) algorithms are designed to address this issue by bringing in the concept of utility, but these algorithms are also known to be expensive to execute. In this paper, we present a novel algorithm AUPlan which automatically generates sequential plans with high utility by combining data mining and AI planning. These high-utility plans could be used to convert groups of customers from less desirable states to more desirable ones. Our algorithm adapts the Apriori algorithm by considering the concepts of plans and utilities. We show through empirical studies that planning using our integrated algorithm produces high-utility plans efficiently.

1. Introduction

In business marketing, corporations and institutions are interested in executing a sequence of marketing actions to change the class of a group of customers from an undesirable class to a more desirable one. Effective class-transformation requires careful planning. For example, a financial institution may derive marketing strategies for turning their reluctant customers into active ones and a telecommunications company may plan actions to stop their valuable customers from leaving. These marketing plans are aimed at converting groups of customers from an undesirable class to a desirable one. In universities, web-based distance-learning systems may provide students with future study plans. For example, an online learning system may devise a study plan for a student with an aim to transform the student from knowledge poor to knowledge rich.

In the past, most planning activities for such class-transformation tasks have been done by hand. However, we observe that the historical plan-execution traces in a database provide valuable knowledge which could be utilized for automatic planning. These plan traces, called *plan databases*, form sequence databases consisting of sequences of actions and states, where an action has a cost value and a plan or state has a real-valued utility measure. High-utility plans could be discovered from the plan databases and used as guidelines for future plan design. Extremely low-utility plans could also be useful, as they could be used in plan-failure analysis for understanding and improving failed plans. In this paper, we consider how to find all high-utility plans from a plan database that are able to transform customers from an undesirable class to a more desirable one.

Consider a marketing campaign planning example. Suppose that a banking company is interested in marketing to a group of 100,000 customers in the financial market to promote a special loan signup. We start with a plan database with historical campaign information in Table 1. Suppose we are interested in building a three-step plan to market to new customers. There are many candidate plans in the table to consider in order to transform as many customers as possible from non-signup status to a signup one. The signup status corresponds to a positive class that we would like to move the customers to, and the non-signup status corresponds to the initial state of our customers. Our plan will choose not only low-cost actions, but also highly successful actions from the past experience. For example, a candidate plan might be:
Step 1: Offer to reduce interest rate;
Step 2: Send flyer;
Step 3: Follow up with a home phone call.
This plan may be considered a high-utility one if it converted 85% of the initially reluctant customers to more willing ones while incurring only a cost of no more than $10.00 per customer.

This example introduced a number of interesting as-

Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)
0-7695-1978-4/03 $ 17.00 © 2003 IEEE

Table 1. An example of loan signup plan database.

Plan No.	Action No.	State before action			Action Taken
		Salary	...	Signup	
1	1	50K	...	N	Send Mail
1	2	50K	...	N	Send Gift
...	...	...	...	...	...

pects for the planning problem. First, not all people in the group of 100,000 customers should be considered as candidates for the conversion. Some people should not be considered as part of marketing campaign because they are too costly or nearly impossible to convert. Second, the group-marketing problem is to use the same plan for different customers in the intended customer group, instead of a different action plan for each different customer. This makes the group-marketing problem different from the direct-marketing problem that some authors have considered in the data mining literature [8, 6]. For group or batch marketing, we are interested in finding a plan containing no conditional branches. We must build an N-step plan ahead of time, and evaluate the plan according to cross-validation from the historical records. Third, for the customers in the group to be marketed to, there are potentially many possible actions that we can use. Each action comes with an inherent cost associated with it. Fourth, it is difficult to formulate this problem as a classical planning problem, because the preconditions and effects of actions are only implicit in the database, rather than given ahead of time by "experts" in a crisp logical formulation. Finally, applying sequence mining to this database alone will not solve the problem. Sequence mining has focused on finding sequences by frequency. While in some situations highly frequent plans are useful, they do not in general give high-utility plans.

In this paper, we formulate the above problem as a combination of probabilistic planning and classical planning, where the key issue is to look for low-cost and high-profit sequences of actions for converting customer groups. Our main contribution is a novel algorithm that combines association-rule mining and AI planning. This paper is organized as follows. Section 2 discusses related work. Section 3 gives the problem formulation. Section 4 presents the *AUPlan* algorithm. Section 5 presents the empirical results. Section 6 concludes the paper.

2. Related Work

Research on plan mining is related with both planning and data mining. This section reviews these two areas. First we present the related research work in planning using MDP models. Then we review the work of frequent sequence mining and the previous plan mining work in the data mining area.

2.1. Planning by Reinforcement Learning

Reinforcement learning refers to a class of problems and associated techniques in which the learner is to learn how to make sequential decisions based on delayed reinforcement so as to maximize cumulative rewards [4, 11, 14]. In a standard reinforcement-learning model, an agent is connected to its environment via perception and action. On each step of interaction the agent receives as input, i, some indication of the current state, s, of the environment; the agent then chooses an action, a, to generate as output. The action changes the state of the environment, and the value of this state transition is communicated to the agent through a scalar reinforcement signal, r. The agent's behavior, B, should choose actions that tend to increase the long-term sum of values of the reinforcement signal. Formally, the model consists of

- a discrete set of environment states, S;
- a discrete set of agent actions, A;
- and a set of scalar reinforcement signals.

The agent's task is to find a policy π, mapping states to actions, that maximizes a measure of utility.

A reinforcement-learning task that satisfies the Markov property is called a Markov decision process (MDP). Markov property is formally defined as: the environment's response at time $t+1$ depends only on the state and action representations at time t.

Dynamic programming techniques serve as the foundation and inspiration for the learning algorithms to determine the optimal policy given the correct model. The optimal value of a state is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy. Using π as a complete decision policy, it is written

$$V^*(s) = \max_{\pi} E(\sum_{t=0}^{\infty} \gamma^t r_t) \quad (1)$$

This optimal value function is unique and can be defined as the solution to the simultaneous equations

$$V^*(s) = \max_{a}(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')), \forall s \in S \quad (2)$$

It asserts that the value of a state s is the expected instantaneous reward plus the expected discounted value of the

next state, using the best available action. Given the optimal value function, we can specify the optimal policy as

$$\pi^*(s) = arg\max_a(R(s,a)+\gamma \sum_{s' \in S} T(s,a,s')V^*(s')) \quad (3)$$

Planning by post-processing of RL. Sun and Sessions [13] proposed an algorithm that addressed planning problem using reinforcement learning. This algorithm can be applied to batch marketing planning problem, but it suffers both computational inefficiency and non-optimality problems. In contrast to existing reinforcement learning algorithms that generate only reactive plans, a two-stage bottom-up process is devised, in which first reinforcement learning/dynamic programming is applied to acquire a reactive plan and then explicit plans are extracted from the reactive plan. Plan extraction is based on a beam search that performs temporal projection in a restricted fashion, guided by the value functions resulting from reinforcement learning and dynamic programming. For convenience, we call this algorithm $QPlan$ in this paper.

Sun and Sessions' $QPlan$ algorithm post-processes the reinforcement learning result to produce a sequence of actions, but it suffers two drawbacks. First, the computational cost of learning the optimal Q values is expensive in the first stage. Thus, the first stage has become a bottleneck for the entire planner. Second, the optimal policy obtained in the first stage cannot guarantee an optimal solution as a result of applying and searching plans using this policy. In the post-processing step, whenever the planner picks up an action at a time step t, it chooses the same action that can achieve the highest expected value functions for multiple states, not different actions optimal for each individual state. However, the assumption of value functions is that the optimal value of a state could only be obtained if it takes the action in the current state and follow the optimal policy forever after. Only when this assumption is satisfied, the optimality can be achieved. Therefore, a non-conditional plan (i.e., a sequence of actions) may not correspond to the optimal policy at all.

2.2. Planning by Frequent Sequence Mining

Mining sequential patterns has been studied extensively in data mining literature. These algorithms fall into several categories, including Apriori-based algorithms [1, 12, 7], lattice-based algorithms SPADE [16] and projection-based pattern-growth methods such as FreeSpan [2] and PrefixSpan [9]. These methods are aimed at finding highly frequent sequences efficiently.

Applying sequence mining algorithms to finding characteristics on highly successful plans, Han et al [3] presented a method for generating frequent plans using a *divide-and-conquer* strategy. The method exploits multi-dimensional generalization of sequences of actions and extracts the inherent hierarchical structure and sequential patterns of plans at different levels of abstraction. These patterns are used in turn to subsequently narrow down the search for more specific patterns.

Zaki et al.[15] developed a technique for plan mining which discovers sequence-patterns indicating high incidence of plan failure. The PLANMINE sequence-mining algorithm extracts patterns of events that predict failures in databases of plan executions. It combines several techniques for pruning out un-predictive and redundant patterns which reduce the size of the returned rule set by more than three orders of magnitude.

3. Planning Problem Formulation

We now formulate the plan mining problem. To explain concepts easier, we consider such a problem in the marketing planning domain. We first consider how to build a state space from a given set of customer records.

As in any machine learning and data mining schemes, the input customer records consist of a set of attributes for each customer, along with a class attribute that describes the customer status. A customer's attribute may be his age, income, gender, credit with the bank, and so on. The class attribute may be "applied", which is a Boolean indicating whether the customer has applied and is approved for loan. As with any real customer databases, the number of attributes may be extremely large; for the KDD CUP 98 data, there are a total of 481 attributes to describe each customer. Of the many attributes, some may be removed when constructing a state. For convenience, we refer to this database table as the *Customer* table. Table 2 shows an example of *Customer* table.

Table 2. An example of *Customer* table

Customer	Interest Rate	Flyer	Salary	Signup
John	5%	Y	110K	Y
Mary	4%	N	30K	Y
...	...	...	...	...
Steve	8%	N	80K	N

A second source of input is the historical plan database. This is a database that describes how the previous actions have changed each customer's attributes as a result of the actions' execution. For example, after a customer receives a promotional mail, the customer's response to the marketing action is obtained and recorded. As a result of the mailing, the action count for the customer in this marketing campaign is incremented by one, and the customer may have decided to respond by filling out a general information form

and mailing it back to the bank. The status of the customer at any instant of time is referred to as a state, and state may change as a result of executing an action. Thus, the historical plan database consists of state-action sequences, one for each participating customer. This sequence database will serve as the training data for our planner. Table 3 is an example of plan database.

Table 3. An example of plan database

State0	Action0	State1	Action1	State2
S_0	A_0	S_1	A_1	S_5
S_0	A_0	S_1	A_2	S_5
S_0	A_0	S_1	A_2	S_6
S_0	A_0	S_1	A_2	S_7
S_0	A_0	S_2	A_1	S_6
S_0	A_0	S_2	A_1	S_8
S_0	A_1	S_3		
S_0	A_1	S_4		

Given the *Customer* table and the plan database, our first task is to formulate the problem as a planning problem. In particular, we wish to find a method to map the customer records in the customer table into states using a statistical classifier. This task in itself is not trivial because it maps a large attribute space into a more concise space. The problem is more complicated when there are missing values in the database. This involves the issues of data cleaning.

After the state space is obtained, we will use a second classifier to classify the states into either desirable or undesirable states based on the training data provided in the *Customer* table. In our implementation, we use decision tree as the classification algorithm.

Next, the state-action sequences in the plan database will be used for obtaining action definitions in a state space, such that each action is represented as a probabilistic mapping from a state to a set of states. To make the representation more realistic, we will also consider the cost of executing each action.

To summarize, from the two tables we can obtain the following information:

- $f_s(r_i) = s_j$ maps a customer record r_i to a state s_j. This function is known as the customer-state mapping function;
- $p(+|s)$ is a probability function that returns the probability that state s is in a desirable class. We call this classifier the state-classification function;
- $p(s_k|s_i, a_j)$ returns the transition probability that, after executing an action a_j in state s_i, one ends up in state s_k.

We now define the utility of a given plan. Recall that a plan is a sequence of actions, starting from an initial state s. A plan is divided into stages, where each stage consists of one action and a set of possible outcome states resulting from the action. In each stage the states can be different possible states as a result of the previous action, and the action in the stage must be a same, single action. Given a plan $P = a_1 a_2 \dots a_n$ and an initial state s, we define the utility $U(s, P)$ of the plan as follows. Let P' be the subplan of P after taking out the first action a_1; that is, $P = a_1 P'$. Then the utility of the plan P is defined recursively

$$U(s, P) = (\sum_{s' \in S} p(s'|s, a_1) * U(s', P')) - cost(a_1) \quad (4)$$

where s' is the next state resulting from executing a_1 in state s. If s is a leaf node, then the plan from the leaf node s is empty, then

$$U(s, \{\}) = p(+|s) * REWARD \quad (5)$$

$p(+|s)$ is the probability of leaf node s being in the desired class, *REWARD* is a predefined constant that defines the maximum possible reward for the customer to be in any state.

Using Equation 4 and 5, we can evaluate the utility of a plan in the initial state.

Given a set of initial items (records in a *Customer* table), our goal is to find a sequence of actions for each initial state that converts as many of the customers in that state from the undesirable class to the desirable one – bringing in high benefits while incurring low costs. To make the computation more efficient, we require that the plan satisfy some constraints. For example, we can impose the following constraints:

- length constraint: the length of a plan is at most N; or
- success constraint: the success probability is over a threshold σ.

4 The *AUPlan* Algorithm

Given these customer database and plan traces, the *AUPlan* algorithm, shown in Table 4, will find all high utility plans in a database using a minimum utility parameter minSU; *minSU* is a minimum threshold value on the product of utility of a plan and its support value, where support carries the same meaning as in association-rule mining [1, 12, 7]. All plans generated as output must have the product of utility and support greater than the $minSU$ value. A second input parameter to our algorithm is $maxlength$, which denotes the maximum length of plans in which we wish to find the high utility ones. Like Apriori-based association-rule mining algorithms, *AUPlan* searches the

space of action sequences in a level-wise manner. Unlike Apriori-based algoirthms, utility-based plan mining has to use the utility of plans to guide the search. However, the utility measure does not satisfy the anti-monotone property.

Table 4. *AUPlan* **algorithm**

Input: A plan database, *minSU*, *maxlength*
Output: High-utility plans.

Algorithm:
1. $C_1 = \{s_j a_i\}$, $s_j a_i$ is all possible one-step plans in the planbase.
2. $K = 1$
3. While($K <= maxlength$)
   3.1 Count support for each plan P in C_k.
   Calculate utility for each plan P in C_k.
   $L_k = \{P \in C_k | sup(P) \times U(P) \geq minSU\}$
   3.2 Generate C_{k+1} from C_k.
   3.3 $K = K + 1$.
End while
4. $L = L_1 \cup L_2 \cup ... \cup L_k$
5. Partition the frequent sequences according to their initial states.
6. For each initial state s_j
   6.1 For each plan P starting with s_j
   Calculate its utility $U(s_j, P)$.
   6.2 Select the plan with highest utility for s_j.
   $plan(s_j) = argmax_P U(s_j, P)$
7. Output plans.

Even though plan utility itself does not satisfy the anti-monotone property, we can design an upper bound of utility to guarantee the anti-monotone property and still allow significant pruning of the search space. In addition, using the utility upper bound ensures the mining result accurate and complete. For a plan P, we denote the upper bound value to be $upperUtil(P)$. We denote the support of a plan P to be $sup(P)$. Pruning plans using the upper bound amounts to pruning a plan P if $upperUtil(P) \times sup(P) < minSU$. The application of the utility upper bound pruning is shown in Table 5.

Suppose the number of sequences in the database is $|N|$, the average length of sequences in the form of $s_i a_j s_{i+1} a_{j+1} ... s_n$ in the database is l, the mining process iterates m times. The complexity of one database scan is $O(|C_k| * |N| * \lfloor l/2 \rfloor)$. The overall time complexity for m iterations is $O(m * |C| * |N| * \lfloor l/2 \rfloor)$, where $|C|$ is the average size of $|C_k|$.

The calculation of the transition probability is realized by matrix multiplication. We define a n by n matrix Tr_k for each action a_k, where n is the number of states. $Tr_k[i][j]$ represents the probability that the transition from state s_i to state s_j occurs under the action a_k. We use a 1Xn vector V' to represent the probability distribution among the states. Initially, for a starting state s_j of a plan, $V_0[j] = 1$ and $V_0[i] = 0, \forall i \neq j$. For a plan $s_j a_{i+1} a_{i+2} ... a_{i+q}$, we calculate $V' = V_0 \times Tr_{i+1} \times Tr_{i+2} \times ... \times Tr_{i+q}$. The time complexity of vector and matrix multiplication is $O(qn^2)$. This is the time complexity of calculating utility for one plan. The time complexity for utility calculation for all plans in C_k in one iteration is $O(|C_k| * qn^2)$. The overall time complexity of utility calculation for all plans in m iteration is $O(\sum_{k=1}^{m} |C_k| * qn^2)$, or $O(m * |C| * qn^2)$, where $|C|$ is the average size of $|C_k|$.

The overall time complexity for m iterations is $O(m * |C| * (|N| * \lfloor l/2 \rfloor + qn^2))$, where $|C|$ is the average size of $|C_k|$.

Table 5. Candidate generation and pruning (Step 3.2) in *AUPlan*

```
(In Step 3.2)
if(sup(P) × U(P) ≥ minSU)
{
        insert(L_k, P);
        for(a in action_set)
        {
                P' = append a to P;
                insert(C_{k+1}, P');
        }
 }

else if(sup(P) × U(P) < minSU)
{
        for(a in action_set)
        {
                P' = append a to P;
                if (upperUtil(P') × sup(P) < minSU)
                        prune P';
                else
                        insert(C_{k+1}, P');
        }
}
```

5. Experimental Results

In order to test the hypothesis that our approach can efficiently mine good plans as compared to the optimal solutions, we run a series of simulated tests to compare *AUPlan* with planning using MDP and planning by exhaustive search.

5.1. Data Generation

We used the IBM Synthetic Generator (http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html) to generate a *Customer* dataset with two classes and nine attributes.

The positive class has 30,000 records representing successful customers and negative has 70,000 representing unsuccessful ones. Those 70,000 negative records are treated as starting points for plan database data generation. We carried out the state abstraction and mapping by feature selection, only keeping four attributes out of nine. Those four attributes were converted from continuous range to discrete values. The state space has 400 distinct states. A classifier is trained using the C4.5 decision tree algorithm [10] on the *Customer* dataset. The classifier will be used later to decide on the class of a state. However, since our focus here is not on training the classifier, and since the choice of the classifier is fairly independent from the subsequent planning algorithms, we will not delve into details here.

We generated the plan database using a second simulator. Each of the 70,000 negative-class records is treated as an initially failed customer. A trace is then generated for the customer, transforming the customer through intermediate states to a final state. We defined four types of actions, each of which has a cost and impacts on attribute transitions. An action's impact on attribute transitions is defined by an *Action-Impact* matrix. For example, M_{ij} is a matrix representing the impact of Action i on Attribute j. The matrix is n by n if attribute j has n different values. The matrix element $M_{ij}[k][l]$ means: of the n different values of Attribute j, if the original value of Attribute j is the k^{th} value, after action i, there is $M_{ij}[k][l]$ probability that the value of Attribute i is changed to the l^{th} value. The plan database generation algorithm is shown in Table 6.

Given the *Customer* table and plan database, *AUPlan* will select plans in the state space. To evaluate the quality of plans found by the algorithm, we calculate the utility of each initial state s with the plan under state s. Then add up these utilities under different states. The sum of plan utility, TU, reflects the overall quality of plans returned by the algorithm over the state space.

$$TU = \sum_{s \in S} max_i U(s, P_i) \quad (6)$$

where Pi represents different plans starting from state s.

5.2. Experimental Result

We wish to test our algorithm *AUPlan* on plan databases of different sizes against the MDP-based algorithm *QPlan* and an exhaustive search benchmark. Our test data is set up using the IBM Synthetic Generator in the following manner. All plan databases have a total of 70,000 plans, but different plan databases have different sequence length limits. Plans in the plan database will not exceed the length limit. This variation in sequence length allows the different plan-mining algorithms to return plans of different utility values. Table 7 shows the features of the five plan databases. In these databases, *Switching Rate* refers to the percentage of customers in the 70,000 who initially belong to the failed class but can be converted successfully by some plans in corresponding plan database. It is expected that both *QPlan* and *AUPlan* will return solution plans with increasing utility values while incurring more computation when we move from DB1 to DB5.

Table 6. Plan database Generation Algorithm

Input: *maxlength*, A set of initial failed states S_i,
a set of actions A_i with *Action-Impact* matrices.
Output: Sequences of trace data with temporal order
$< S_i, A_j, S_{i+1}, A_{j+1}, ..., S_n >$

Algorithm:
for each initially failed state S
  while(trace_length $<$ *maxlength*)
    randomly select an action a;
    generate next state S' according to S and
    Action-Impact matrix of a;
    trace_length ++;
    if(S' is a successful state)
      break;
  end while
end for

Table 7. Features of different plan databases

plan database	Length Limit (Max # of actions)	Switching Rate (%)
Plan DB1	5	20
Plan DB2	9	40
Plan DB3	14	60
Plan DB4	29	80
Plan DB5	100	100

We compare three algorithms. The first is a most naive algorithm, which serves both as a benchmark for quality (it returns the highest possible utility) and efficiency (it performs an exhaustive search). We call this algorithm the *OptPlan* algorithm. Given an action set A and a parameter $maxlength$ for the maximum length of plans one wishes to explore, *OptPlan* tries all possible combinations of actions

in a corresponding database to find optimal plans no longer than $maxlength$ in length. This method will produce optimal solutions in terms of plan quality. The drawback of this method is its computation cost. Suppose the number of actions in A is $|A|$, the *maxlength* equals L. The number of length-1 plans is $|A|$, length-2 plans $|A|^2$, ..., length-L plans $|A|^L$. The number of plans, and therefore the computational time, grows exponentially with *maxlength*.

A second algorithm we compare to is the *QPlan* algorithm in [13]. As described in the related work section, this algorithm runs in two stages. In the first stage, we apply Q-learning first to get an optimal policy. We then extract plans guided by the Q-values through a beam search. Q-learning is carried out using "batch reinforcement learning" [8]. It tries to estimate the value function $Q(s, a)$ by the value-iteration algorithm. The major computational bottleneck of *QPlan* is thus from Q-learning.

To allow *QPlan* to terminate at any intermediate state that has high enough utility, we add one special *null* action to the Q-learning part of the *QPlan* algorithm. That is, besides the Q-values of each state/action pair, we learn one more Q-value $Q(s, \phi)$ for each state s, ϕ means "no action, stop at the current state". $Q(s, \phi)$ is the reward that would be obtained if no action is taken at state s. It is defined as:

$$Q(s, \phi) = p(+|s) \times REWARD \quad (7)$$

When the agent finds that staying at a state s will bring higher utility than taking any actions from that state, it should stop taking any actions wisely. This can be realized by comparing the Q-values of $Q(s, \phi)$ and $Q(s, a)$ for all a in the action set. If $Q(s, \phi)$ is greater than $Q(s, a)$ for all a, then the agent stops at state s. Otherwise, it picks up the action a that maximizes $Q(s, a)$.

We run all three algorithms, *QPlan*, *AUPlan* and *OptPlan*, on the different plan databases and obtained the CPU time as a function of the size of plan databases. We also compared the utility of plans returned by each algorithm. For each plan database, we converted the utilities of *QPlan* and *AUPlan* as the percentage of the utility of the *OptPlan* algorithm, which is at 100% level.

Figure 1 shows the CPU time of different algorithms versus the size of plan databases. The top line corresponds to the CPU time of *OptPlan* which is at a constant for different databases in this test, because we set the parameter $maxlength$ to be a constant 6 for each plan database. As $maxlength$ is fixed, the number of plans by exhaustive search in $OptPlan$ is also the same for different plan databases, so is the computational cost. In this case, the only difference between different plan databases is the transition probability $p(s_k|s_i, a_j)$. Note that the Y-axis is displayed in log scale. Thus the CPU time of *OptPlan* shows that exhaustive search is simply not practical at all. *AUPlan* is shown to be much more efficient than *QPlan*. This is expected, since *AUPlan*'s mining and pruning algorithms is able to scale up the planning problems.

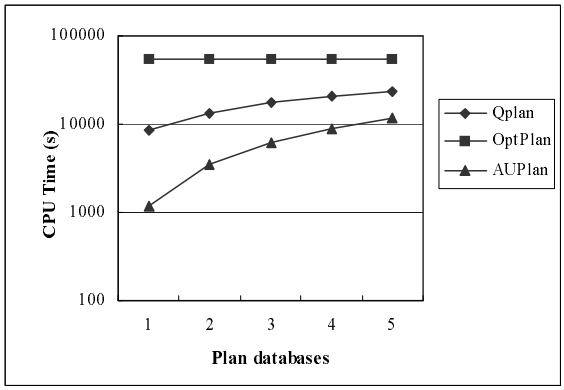

Figure 1. CPU time of Different algorithms versus different plan databases

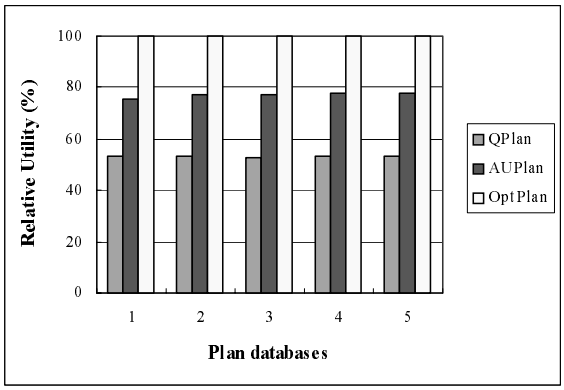

Figure 2. Relative Utility of Different algorithms vs. different plan database

We next turn to a comparison on the utility of plans found. Figure 2 shows the relative utility of different algorithms versus different plan databases. *OptPlan* has the maximal utility by exhaustive search. *AUPlan* comes next, about 80% of the optimal solution. *QPlan* have less than 60% of the optimal solution.

Figure 3 shows the relative utility of different algorithms versus *maxlength* parameter used in the *AUPlan* algorithm. *maxlength* is the maximum number of actions to be allowed in a solution plan. In this experiment, we fixed a database (DB3) and varied the *maxlength* parameter. When the *maxlength* is three, *AUPlan* has about 85% of the optimal solution. For different values of *maxlength*, *AUPlan* clearly represents a tradeoff between the optimal solution *OptPlan* and the Q-learning based solution *QPlan*.

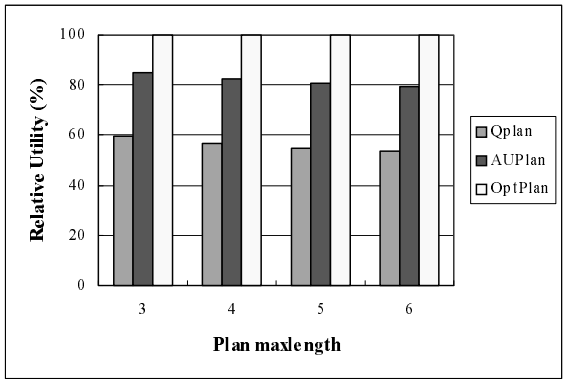

Figure 3. Relative Utility of different algorithms vs. *maxlength* **of the plans**

6 Conclusion and Future Work

We proposed a plan mining algorithm that integrates both data mining and planning to obtain high-utility plans. We add states, actions, utilities to sequence mining algorithms and returns not only the frequent sequences, but actual plans for agents to execute. In large plan databases, scalability is an important issue. Experimental results demonstrate that the integrated data mining and planning approach is more efficient and effective than planning by MDP and exhaustive search. The quality of plans by *AUPlan* is about 80% of the optimal solution in our tests.

In the future, we will consider the following directions:

- Test our algorithms on more real plan data. Our algorithms could have a broad applications, such as marketing domains, robot world, etc. If we could apply to real data, such as plans executed by robots, we could make data mining truly actionable.
- We assume the intermediate states between plan executions could be observed. If part of the data about states is not observable, our algorithms have to be modified. It is possible to formulate the problem as an approximation to POMDP [5] when the intermediate states are partially observable.

Acknowledgment The authors are supported by a Hong Kong Government RGC grant and the Hong Kong University of Science and Technology.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In P. Yu and A. Chen, editors, *Proceedings of 11th International Conference on Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, March 1995. IEEE Computer Society Press.

[2] J. Han, J. Pei, Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the 2000 International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pages 355–359, Boston, MA, August 2000.

[3] J. Han, Q. Yang, and E. Kim. Plan mining by divide-and-conquer. In *Proceedings of SIGMOD'99 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'99)*, 1999.

[4] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[5] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08, 1996.

[6] C. Ling and C. Li. Data mining for direct marketing: Problems and solutions. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 73–79, New York, 1998.

[7] F. Masseglia, F. Cathala, and P. Poncelet. The psp approach for mining sequential patterns. *Principles of Data Mining and Knowledge Discovery*, pages 176–184, 1998.

[8] E. Pednault, N. Abe, and B. Zadrozny. Sequential cost-sensitive decision making with reinforcement learning. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pages 259–268, Edmonton, Canada, 2002.

[9] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix projected pattern growth. In *Proceedings of the 2001 International Conference on Data Engineering (ICDE'01)*, pages 215–226, Heidelberg, Germany, April 2001.

[10] J. R. Quinlan. *C4.5: Programming for Machine Learning*. Morgan Kaufmann Pbulishers, San Mateo, CA, 1993.

[11] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, NJ, 1995.

[12] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In P. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Proceedings of 5th International Conference on Extending Database Technology (EDBT'96)*, volume 1057, pages 3–17, SpringerVerlag, March 1996.

[13] R. Sun and C. Sessions. Learning plans without a priori knowledge. *Adaptive Behavior*, 8(3/4):225–253, 2001.

[14] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[15] M. Zaki, N. Lesh, and M. Ogihara. Planmine: Sequence mining for plan failures. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, 1998.

[16] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning, special issue on Unsupervised Learning*, 42(1/2):31–60, Jan/Feb 2001.