Planning for Marketing Campaigns

Qiang Yang and Hong Cheng

Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Kowloon, Hong Kong, China
(qyang, csch)@cs.ust.hk

Abstract

In business marketing, corporations and institutions are interested in executing a sequence of marketing actions to affect a group of customers. 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 this paper, we formulate this group marketing-plan generation problem as a planning problem. We design a novel search algorithm to find a cost-effective and highly probable plan for switching a group of customers from their initial states to some more desirable final states. We explore the tradeoff among time, space and quality of computation in this planning framework. We demonstrate the effectiveness of the methods through empirical results.

Introduction

Marketing campaign planning in business marketing can be considered as a process of planning in which the objective is to convert groups of customers from one class to another, more profitable class. In business marketing literature (Dibb et. al 1996), planning for marketing campaigns corresponds to developing action plans by taking into account the customer segmentation, marketing objectives and budgetary constraints into consideration. In the marketing practice today, it is a common practice to design action plans by a human experts through focus group studies (Bank Marketing Association 1989). The planning process itself is both long and laborious.

Marketing planning can be divided into two types. Direct marketing or one-to-one marketing plans are aimed at generating plans to target individual customers. This is an expensive process that is only applied to valuable customers. Direct marketing usually assumes that the next action to be performed can be decided based on observation of a customer's current state. In this paper we consider a second kind of marketing plans called segmentation-marketing plans. These plans are aimed at marketing to multiple groups of customers rather than a single customer, where a sequence of actions is executed on a segment of chosen customers until they are completed.

Essentially, the planning process can be considered as building a statistical model based on past data and using the model to formulate a plan to be executed on a group of customers. These marketing actions can hardly be formulated as traditional classical planning representations. Often marketing plans are expected to be effective on only a subset of the customers in the target group, and the planning is currently done by hand through various marketing studies such as focus groups. A marketing plan thus generated will be applied to a chosen subset of customers to be effective. For example, a cell-phone company may decide to reduce the monthly fee for a subgroup of its customers who are both highly valuable and likely to leave the company for its competitors.

To illustrate, we consider the following scenario. Suppose that a 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 customer-loan database with historical customer information on past loan-marketing results in Table 1. Suppose that we are interested in building a 3-step plan to market to the selected group of customers in the new customer list. There are many candidate plans to consider in order to move 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-signing up 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: Send mails;

Step 2: Call home #;

Step 3: Offer low interest rate

This example also introduced a number of interesting aspects 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. To identify a group of applicable customers, certain data mining

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

algorithms for customer segmentation can be applied. 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 different from the direct-marketing problem that some authors have considered in the data mining literature (Domingos and Richardson 2001; Pednault et. al 2002; Ling and Li 1998; Yang and Cheng 2002). For group marketing, we don't have the luxury of observing intermediate states during a plan execution in order to decide what to do next. Instead, 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 provide. Each action comes with an inherent cost associated with it. In addition, an action is not guaranteed to produce its intended result. For example, it may be more costly to call a customer at his home than to send mail. However, sending a mail to a customer may have less effect than calling a customer at home. Neither mailing nor calling can guarantee that all customers contacted will be converted to signup status afterwards. Additionally, calling a customer may have an adverse effect of annoying the customer more than necessary. Finally, 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.

We formulate the above problem as a probabilistic planning problem, where the key issue is to look for good plans for converting customer groups. Our approach is to first identify a state space and assign the potential customers to initial states. We classify the customer states into groups belonging to desirable or undesirable classes. Our objective becomes one to convert customers from undesirable class to desirable one. We propose an algorithm called *MPlan* as a solution to this planning problem using the historical database as an AND-OR tree search problem. The resulting plan will provide a basis for the final marketing plans. Our aim is to choose high-utility actions to be included in our plan where the notion of utility is introduced to increase the probability of success while reducing costs.

This research deviates from the traditional planning applications and formulation of classical planning in several aspects. Compared to classical planning, in group marketing we cannot guarantee with certainty the result of actions; each action may result in an initial group to be split into several subgroups, each landing in a potential state following a probability distribution. This distribution can be learned from historical plan traces obtained before. A second difference from classical planning is that there is no easy way to formulate the actions in terms of relations and logic formulas needed fro preconditions and effects. The only observable fact before and after an action's execution is from the historical databases, which records the customer status in various attributes. By applying a statistical classifier to the attributes, we could learn a customer's potential standing in terms of desirable versus undesirable classes.

The problem is also different from the traditional MDP approach (Sutton and Barto 1998) to solving the probabilistic planning problem. In MDP, it is assumed that at all time during a plan's execution, the intermediate states can be known either completely or partially. The problem is that of finding a policy in which to direct an agent's action no matter where the agent is observed to land. The MDP formulation is more suitable for direct marketing (Ling and Li 1998), which is geared towards finding a plan for each individual such that an action is chosen based on the agent's observed resulting state. However, in group marketing, it is often the case that we have no such opportunity to obtain the intermediate states for a group of customers in the middle of plan execution. Instead, we have to find a single plan for a group of similar customers and to execute this plan to completion. Thus, this aspect where a sequence of actions is built and executed is more akin to classical planning.

The problem is also different from the probabilistic planning framework of (Draper et. al 1994), which considered modeling each action in a probabilistic version of the pre-conditions and post-conditions. The problem there is still to consider how to build a plan from a single initial state to a single goal state. In contrast, in our problem the actions' logical representations are not available; all that we can observe are action labels and their association with states. In addition, we consider customer groups that are scattered into multiple initial states. The goal state (Keeney and Raiffa 1976) is not clearly definable in our case either, because the positive class in general defines the potential goal state sets.

In data mining area, a related area is cost-sensitive learning and decision making (Domingos 1999; Elkan, 2001) in the machine learning community. However, significant differences remain. Cost-sensitive methods try to minimize the cost of a single decision. However, in many applications, sequences of decisions in the form of plans are needed.

Marketing-Planning Problem Formulation

We now consider how to formulate the marketing problem more formally as a planning problem. 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 KDDCUP-98 data (Blake and Merz 1998), 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 1 is an example of *Customer Table.*

Table 1. An example customer-loan database. The last attribute is the class attribute.

Customer	Salary	Cars	Mortgage	Loan Signup?
John	80K	3	None	Y
Mary	40K	1	300K	Y
…	…	…	…	…
Steve	40K	1	None	N

Table 2. An example *Marketing-log* database.

S#	A#	S#	A#	S#	A#	S#
S1	A1	S2	A2	S3	A3	S4
S0	A0	S1	A1	S2	A4	S5
S2	A2	S3	A4	S4	A4	S7
…	…	…	…	…	…	…
…	…	…	…	…	…	…
S0	A0	S3	A4	S4	A4	S7

A second source of input is the previous marketing-record database. This is a database that describes how the previous marketing 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 marketing-record database consists of state-action sequences, one for each participating customer. This sequence database will serve as the training data for our planner. For convenience, this historical marketing database table is referred to as the *Marketing-log Table*. Table 2 is an example of *Marketing-log Table.*

Given the *Customer* table and the *Marketing-log* table, 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. In this paper, we will not delve into this issue, because it involves the issues of data cleaning and data mining (Han and Chamber 1998).

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. Classification algorithms such as decision tree or Naïve Bayes are possible choices as long as the classification error rate is low enough.

Next, the state-action sequences in the *Marketing-log* table 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;

• $p_c(s)$ is a probability function that returns the probability that state s is in a desirable class. We call this classifier the state-classification function;

• $p(s_k \mid s_i, a_j)$ returns the transition probability that, after executing an action a_j in state s_i, one ends up in state s_k.

Once the customer records are converted to states and the state transition through actions are learned from the *Marketing-log* table, the state space can be formulated as an AND-OR graph. In this graph, there are two types of nodes. A state node represents a state. From each state node, an action links the state node to an outcome node, which represents the outcome of performing the action from the state. An outcome node then splits into multiple state nodes according to the probability distribution given by the $p(s_k \mid s_i, a_j)$ function. This graph essentially is an AND-OR graph, where each state is an OR node, with the actions that can be performed on the node forming the OR-branches. Each outcome node is an AND node, where the different arcs connecting the outcome node to the state nodes are the AND edges. A figure illustrating the scenario is shown in Figure 1.

Given a set of customers for whom the marketing plan is designed, we use a customer-state mapping function to convert the customer records to a set of initial states which these customers belong to initially in the state space. Note that because of the potentially large number of customers involved, there could be a set of initial states corresponding to the customers, instead of a single initial state as in classical planning. These initial states provide an initial segmentation of the customers.

In this setting, we can give a definition of the marketing-plan planning problem. Given a set of initial customers, 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 while incurring minimal costs. The plan must satisfy some constraints, in *one* of the following forms:

• *length constraint*: the number of actions must be at most N;

• *probability constraint*: the expected probability of being in a desirable class of all terminal states a plan leads to must be at least *Success_Threshold*.

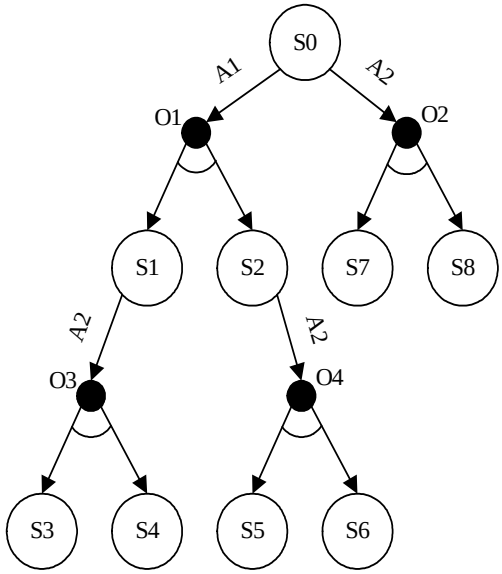

Figure1. An example of AND-OR graph.

Not all customers in the given set of customers are convertible to the desirable class. In this case, we also want to identify a subset of customers who can be converted within the constraint.

The marketing plan generation problem can be considered in several forms. A variation of the problem is to find a uniform plan for all different customer segments, regardless of which initial states they start from, so that as many customers as possible are converted to the desirable class under the length and probability constraint. This formulation corresponds to the need for corporations to market to an entire group of customers with the same actions for consistency and cost-cutting. In this paper, we focus on the first problem where we can have different plans for different segments of customers.

Marketing-Campaign Planning Algorithm

Algorithm Overview

A major difficulty in solving the marketing-planning problem stems from the fact that there are potentially many states and many connections between states. This potentially large space can be reduced significantly by observing that the states and their connections are not all equal; some states and action sequences in this state-space are more significant than others because they are more frequently "traveled" by traces in the *Marketing-log* table. This observation allows us to use an approach in which we exploit planning by abstraction.

In particular, significant state-action sequences in the state space can be discovered through a frequent string-mining algorithm. We start by defining a minimum-support threshold for finding the frequent state-action sequences. Support represents the number of occurrences of a state-action sequence from the *Marketing-log* table. More formally, let $count(seq)$ be the number of times sequence "seq" appears in the database for all customers. Then the support for sequence "seq" is defined as

$$\sup(seq) = count(seq), \qquad (1)$$

Then, a string-mining algorithm based on moving windows will mine the *Marketing-log* table database to produce state-action subsequences whose support is no less than a user-defined minimum-support value. For connection purpose, we only retained substrings both beginning and ending with states, in the form of $< s_i, a_i, s_{i+1}, a_{i+1}, ..., s_n >$.

Once the frequent sequences are found, we piece together the segments of paths corresponding to the sequences to build an abstract AND-OR graph in which we will search for plans. If $< s_0, a_1, s_2 >$ and $< s_2, a_3, s_4 >$ are two segments found by the string-mining algorithm, then $< s_0, a_1, s_2, a_3, s_4 >$ is a new path in the AND-OR graph. Since each component of the AND-OR graph is guaranteed to be frequent, the AND-OR graph is a highly concise and representative state space. Suppose that we wish to find a marketing plan starting from a state s_0, we consider all action sequences in the AND-OR graph that start from s_0 satisfying the length or probability constraint.

We used a function $f(s, p) = g(p) + h(s, p)$ to estimate how "good" a plan is. Let s be an initial state and p be a plan. Let $g(p)$ be a function that sums up the cost of each action in the plan. Let $h(s, p)$ be a heuristic function estimating how promising the plan is for transferring customers initially belonging to state s. $h(s, p)$ is a kind of utility estimation of the plan. This function can be determined by users in different specific applications. In our work, we estimated $h(s, p)$ in the following manner. We start from an initial state and follow a plan that leads to several terminal states s_i, s_{i+1}, s_{i+2}, …, s_{i+j}. For each of these terminal states, we estimate the state-classification probability $P(+ | s_i)$. Each state has a probability of $1 - P(+ | s_i)$ to belong to a negative class. The state requires at least one further action to proceed to transfer the $1 - P(+ | s_i)$ who remain negative, the cost of which is at least the minimum of the costs of all actions. We compute heuristic estimation for terminal states where the plan leads. For an intermediate state leading to several states, an expected estimation is calculated from the heuristic estimation of its successive states weighted by the transition probability $p(s_k \mid s_i, a_j)$. The process starts from terminal states and propagates back to the root, until reaching the initial state. Finally, we obtain the estimation of $h(s, p)$ for the initial state s under the plan p.

Based on the above heuristic estimation methods, we can perform a best-first search in the space of plans until the termination condition is met. The termination conditions are determined by the probability or the length constraints in the problem domain.

Search for Plans using *MPlan*

In the AND-OR graph, we carry out a procedure *MPlan Search* to perform a best-first search for plans. We maintain a priority queue Q by starting with a single-action plan. Plans are sorted in the priority queue in terms of the evaluation function $f(s, p)$.

In each iteration of the algorithm, we select the plan with minimum value of $f(s, p)$ from the queue. We then estimate how promising the plan is. That is, we compute the expected state-classification probability $E(+|s_0)$ from back to front in a similar way as with $h(s, p)$ calculation, starting with the $P(+|s_i)$ of all terminal states the plan leads to and propagating back to front, weighted by the transition probability $p(s_k \mid s_i, a_j)$. We compute $E(+|s_j)$, the expected value of the state-classification probability of all terminal states. If this expected value exceeds a predefined threshold *Success_Threshold*, i.e. the probability constraint, we consider the plan to be good enough and the search process terminates. Otherwise, one more action is attached to this plan and the new plans are inserted into the priority queue. $E(+|s_i)$ is the expected state-classification probability estimating how "effective" a plan is at transferring customers from state s_i. Its calculation can be defined in the following recursive way:

$E(+|s_i) = \sum p(s_k \mid s_i, a_j) * E(+|s_k)$; if s_i is a non-terminal state; or

$E(+|s_i) = P(+|s_i)$ if s_i is a terminal state. (2)

We define ***Success_Threshold*** as a lower bound on $E(+|s_i)$. We conduct the above search procedure for all initial states, finding one plan for each. It is possible that in some AND-OR graphs, we cannot find a plan whose $E(+|s_i)$ exceeds the *Success_Threshold*, either because the AND-OR graph is not good enough or because the *Success_Threshold* is too high. To address this, we define a parameter *Max_Step* which defines the maximum length of a plan, i.e. the length constraint. We will discard a candidate plan which is longer than the *Max_Step* and $E(+|s_i)$ value less than the *Success_Threshold*. Table 3 is the pseudo code of the *MPlan Search* algorithm.

Consider an example of *MPlan Search* algorithm using the AND-OR graph in Figure 1. Suppose that we are looking for a plan for customers starting at state s_0. Suppose we have a finite set of actions and the minimum cost among these actions is denoted by MinC.

Step 1. We inserted two single-step plans – <A1> and <A2> into Q with the evaluation function as follows:

Table 3. The *MPlan Search* Algorithm

```
1. Insert all possible one-action plans into Q.
2. While (Q not empty) {
3.    Get a plan with minimum value of f(s, p)
        from Q.
4.    Calculate E(+|s) of this plan.
5.    If ( E(+|s) >= Success_Threshold)
         Return Plan;
6.    If (length(Plan) > Max_Step)
         Discard Plan;
7.    Else
         7.1 Expand plan by appending an action.
         7.2 Calculate f(s, p) for the new plans
       and insert into Q.
8  }   end while
9  Return "plan not found";
```

f(S0, A1)=Cost(A1) + P(S1|S0, A1) * (1-P(+|S1)) * *MinC* + P(S2|S0, A1) * (1-P(+|S2)) * *MinC*

f(S0, A2)=Cost(A2) + P(S7|S0, A2) * (1-P(+|S7)) * *MinC* + P(S8|S0, A2) * (1-P(+|S8)) * *MinC*

Step 2. Suppose <A1> is the plan with minimum $f(s, p)$ value in Q. Therefore, <A1> is deleted from Q and examined to see whether it is a qualified plan.

$E(+|s_0)$=P(S1|S0, A1) * P(+|S1) + P(S2|S0, A1) * P(+|S2)

If $E(+|s_0)$ is less than *Success_Threshold*, then <A1> is not a "good" plan. Thus, actions A1 and A2 are appended to the end of plan <A1> to form two new plans <A1A1> and <A1A2>. These two plans are then inserted into Q.

Because there is no path <A1A1> in the AND-OR graph, we discard the candidate <A1A1> from Q. The $f(s, p)$ value of <A1A2> is:

f(S0, A1A2)=Cost(A1A2) + P(S1|S0, A1) * (P(S3|S1, A2) * (1-P(+|S3)) * *MinC* + P(S4|S1, A2) * (1-P(+|S4)) * *MinC*) + P(S2|S0, A1) * (P(S5|S2, A2) * (1-P(+|S5)) * *MinC* + P(S6|S2, A2) * (1-P(+|S6)) * *MinC*))

Step 3. Now we have plans <A2>, <A1A2> in Q. Suppose <A1A2> is the plan with minimum $f(s, p)$. Therefore, <A1A2> is deleted from Q to see whether it is a promising plan.

$E(+|s_0)$= P(S1|S0, A1) * (P(S3|S1, A2) * P(+|S3) + P(S4|S1, A2) * P(+|S4)) + P(S2|S0, A1) * (P(S5|S2, A2) * P(+|S5) + P(S6|S2, A2) * P(+|S6))

If $E(+|s_0)$ >= *Success_Threshold*, then <A1A2> is a "good" plan with minimum cost from Q.

Step 4. Return the plan <A1A2>. Stop.

Analysis

The Marketing-Plan algorithm has two major components in terms of time complexity – one is state space abstraction by string mining algorithm; the other is *Mplan*, the best-first algorithm.

In the string-mining algorithm, we find frequent strings which satisfy the predefined minimum support threshold. Suppose that there are *N* sequences in the *Marketing-log* table. The average length of the sequences is *K*. We scan the sequences with a finite window of size *W*. We need to find all the frequent strings with length less than or equal to *W*. For each sequence with an average length *K*, time complexity is $O(K(W+1)-W(W+1)/2)$. If $W << K$, then it is $O(K)$. For totally *N* sequences, it is $O(NK)$. If *W* is comparable to *K*, then it is $O(NK^2)$.

In the *MPlan Search* Algorithm, the number of iterations is bounded by the parameter *Max_Step*. Suppose the number of different actions is *A*. In the worst case when the algorithm exits with "No plan found", the number of iterations is $O(A^{Max_Step})$. However, in an average case the plan should complete faster than the worst case. The time complexity for a single iteration is determined by the size of the state space. In general, it takes more time to calculate $f(s,p)$ and $E(+|s_i)$ in a complex state space than in a simple one because the planner has more states and paths to explore.

Note that although our proposed state space abstraction method using string mining does not reduce the number of iterations of *MPlan* algorithm, it saves a lot of time in exploring in the state space in each iteration because many statistically trivial paths are discarded before the search process.

Experimental Setup

Although we are able to obtain *Customer* data, it has been difficult to obtain *Marketing-log* data from real world. To test our ideas, we used a simulator to generate the customer-log data according to some customer distributions we can specify. The *Customer* data are used for training a classifier for state-classification function $p_c(s)$. The *Marketing-log* data are used in two ways: (1) Frequent state-action subsequences are mined from the *Marketing-log* data to construct a highly concise and representative AND-OR graph; (2) Models of transition probability $p(s_k \mid s_i, a_j)$ are estimated from the statistics of the *Marketing-log* data.

Data set

We used the IBM Synthetic Generator (http://www.almaden.ibm.com/cs/quest/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 *Marketing-log* 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 (Quinlan 1993) on the *Customer* dataset. The classifier will be used later to decide on the class of a state.

We generated the *Marketing-log* data using another simulator. Each of the 70,000 negative 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 also defined four types of actions, each of which has a cost and impacts on attribute transitions. We can illustrate the definition of an action's impact on attribute transitions through an example.

$$M_{13} = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

In this matrix, M_{13} is a matrix representing the impact of action A_1 on Attribute three. The matrix is *n by n* if attribute i has n different values. Suppose attribute 3 has three distinct values 0, 1, 2. The first row in the matrix means if attribute 3 takes value 0, after action A_1, Attribute three will take on the first value with 80% probability, value 1 with 15% probability and value 2 with 5% probability. After an action is taken, attributes for a customer will change probabilistically according to the definition of impact of actions.

The *Marketing-log* data generation algorithm is shown in Table 4. In this procedure, we define *terminal** as follows: once a customer changes from the negative class to the positive class, the state for the first time he is classified as positive is a terminal state. If a customer received actions for a predefined number of times, say, 20 times and still remained negative, the 20th state is the terminal state. Using this method, we generated 70,000 traces for the 70,000 failed records. Figure 2 illustrates the distribution of different-length traces.

Table 4. The *Marketing-log* data generation algorithm.

Input: A set of initially failed states S_i and a set of actions A_i with state-transition matrices.

Output: Sequences of trace data preserving temporal order, in the form of $< s_i, a_i, s_{i+1}, a_{i+1}, ..., s_n >$.

Algorithm:

```
For each initially failed state s
    while s is not a terminal* state
        randomly select an action a;
        generate the next state s' according to
        the impact of action a on attributes;
    end while
end for
```

In Figure 2, the horizontal axis represents the number of actions in a trace. The vertical axis represents the number of "n-actions" traces. For example, the first dot represents that there are about 18,000 traces that has only one action before success. We can see that long traces with more than 6 actions are very rare.

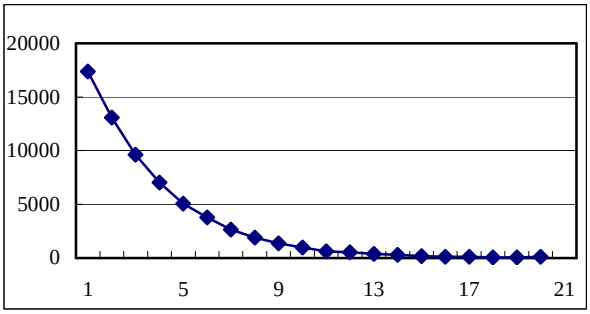

Figure2. Distribution of number of traces as a function of plan length.

Test Criteria

We evaluated the quality of the plans via simulation. Again, we used the IBM Synthetic Generator to generate 100,000 customer records that correspond to the failed class. Our goal is to find marketing plans to convert them to a successful class. This testing process corresponds to the testing phase for a trained model in machine learning.

In this simulation, if there is a plan suitable for converting a customer record, a sequence of actions is carried out on that record. The plan will then change the customer record probabilistically according to impact of actions on attributes. At the end, the classifier is used to decide whether the changed record has turned into a successful one.

We define a number of quantities to measure the success of the test results. Let $\{s_1, s_2, ...s_n\}$ be a set of terminal states in a current plan from an initial state. We wish to estimate the success probability of the plan. We define a quantity -- the expected success probability $E(+|s_0)$ recursively as equation (2).

As mentioned before, a user-defined ***Success_Threshold*** is used as a lower bound on $E(+|s_0)$; we consider a plan is found successfully for an initial state only when $E(+|s_0)$ is no less than the *Success_Threshold.*

We define the ***Max_Step*** *as* a length constraint, whereby plans longer than this limit will not be examined.

Let *N* be the number of customers who are to be converted through the marketing plans. Those are the customers who belong to the negative class initially. Let PlanSet be the set of plans that are found by the *MPLan* planner for these *N* customers. Then the ***Transition Rate*** is defined as the proportion of people who are transformed to the successful class after the application of the plan. Let M be the number of customers among the N people who belong to the successful class after the plans are applied. Then

$$Transition \quad Rate = \frac{M}{N}, \qquad (3)$$

Finally, let *L* be the number of initial segments which corresponds to *L* initial states. Let *K* be the number of initial states among the L where a marketing plan is successfully found within the stated limits. Then the ***Plan Rate*** is defined as:

$$PlanRate = \frac{K}{L}, \qquad (4)$$

We also measure the CPU time used to search for the plan. This is denoted as *Time*.

Evaluation Results

Figure 3 (a) illustrates the *Transition Rate*, *Plan Rate* of a plan as a function of *Success_Threshold*. *Success_Threshold* corresponds to a threshold on the expected probability that the terminal states are considered belonging to the positive class. This parameter determines how easy it is to find a successful plan. When *Success_Threshold* is low, many states are considered positive, and thus plans can be easily and quickly found for most of the initial states in the graph. Thus we can observe from Figures 3 (a) and (b) that searching *Time* is low and *Plan Rate* is high with low *Success_Threshold*. However, because *Success_Threshold* is low, the plans found don't guarantee high probability of success. So *Transition Rate* is also low at first. As *Success_Threshold* increases, so does the *Transition Rate and searching Time*. When *Success_Threshold* is too high, no plan can be found for some initial states. Therefore, both of the *Plan Rate* and *Transition Rate decrease*. The *Time* is much higher because the searching process doesn't terminate until all the plans expanded longer than *Max_Step*.

Figures 4 and 5 illustrate the *Transition Rate*, *Plan Rate* and searching *Time* of a plan as a function of minimum support *MinSupport*. *MinSupport* =N means a sequence has to appear at least N times in *Marketing-log* data to be frequent. An AND-OR graph with minimum support *MinSupport*=1 corresponds to a state space constructed without undergoing frequent string mining procedure. Generally speaking, *MinSupport* determines the frequent state-action sequences mined from *Marketing-log* data and thus the size of the AND-OR graph – the search space for a plan. When *MinSupport* is low, the search space is very large and complex; searching will then take a lot of time to complete. As *MinSupport* gets larger, the search space becomes more and more compressed, and search time will be shorter. When *MinSupport* becomes extremely high, the search space may lose many important states and transitions, making plan searching harder again.

However, the search efficiency also depends on other parameters. *Success_Threshold* is another important factor. When *Success_Threshold* is low, approximately the same *Plan Rate* and *Transition Rate* can be found no matter how large the state space is, as shown in Figure 4. When *Success_Threshold* is higher, plan searching becomes more difficult. In a large search space, searching for a solution plan takes a lot more time. In an extremely concise state space, no plans can be found because many states and

transitions are discarded by state space abstraction, as shown in Figure 5.

Notice that in Figure 5(a), *Transition Rate* reaches a maximum for a state space when *MinSupport=100*. This means that with appropriate frequent string mining, plans found in the resulted state space won't lose in terms of effectiveness compared with original state space without string mining; the feature of string mining even provides better performance. It also saves time in search space construction and plan searching process, as shown in Figure 5(b).

Figures 6(a) (b) illustrates the *Transition Rate*, *Plan Rate* and searching *Time* of a plan as a function of *Max_Step*. As we can see from the figure, increasing the length of *Max_Step* has little effect on *Transition Rate* and *Plan Rate*. This is due to the fact that once a group of customers are converted to a positive class boundary; any further actions will not improve their chance of being positive. However, *Time* increases greatly because searching process often continues until all possible plans within the length of *Max_Step* are explored.

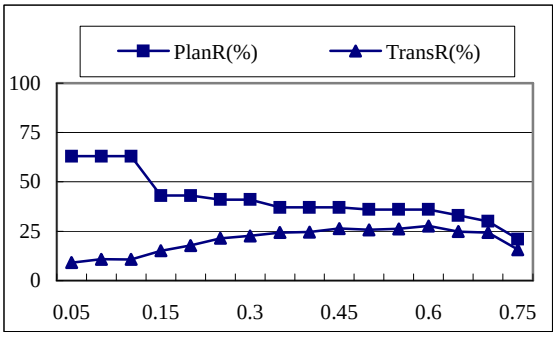

Figure 3(a). Plan Rate, Transition Rate vs. Success_Threshold. MinSupport = 100, Max_Step =5.

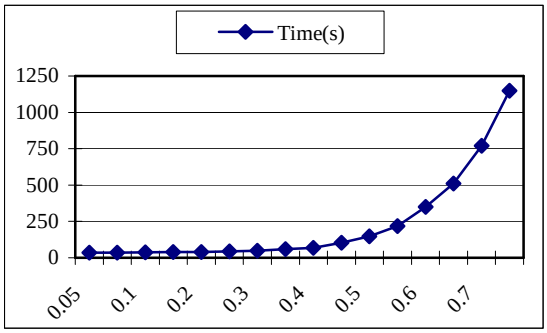

Figure 3(b). CPU Time vs. Success_Threshold. MinSupport = 100, Max_Step =5.

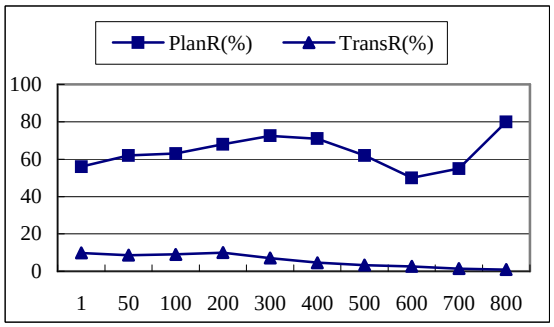

Figure 4(a). Plan Rate, Transition Rate vs. MinSupport. Success_Threshold=0.05. Max_Step =5.

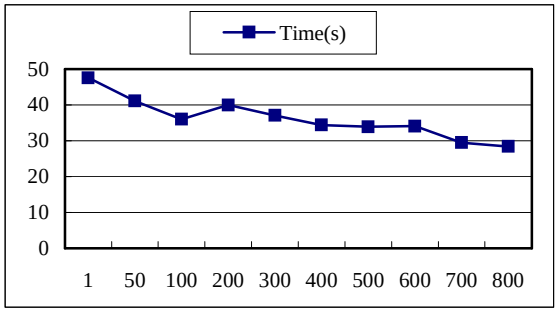

Figure 4(b). CPU Time vs. MinSupport. Success_Threshold=0.05. Max_Step =5.

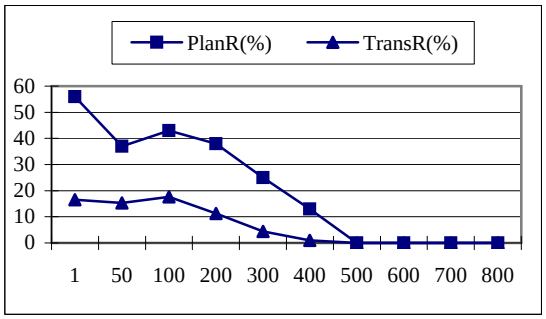

Figure 5(a). Plan Rate, Transition Rate vs. MinSupport. Success_Threshold=0.20. Max_Step =5

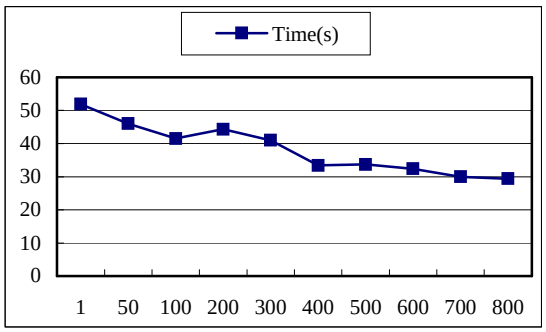

Figure 5(b). CPU Time vs. MinSupport. Success_Threshold=0.20. Max_Step =5.

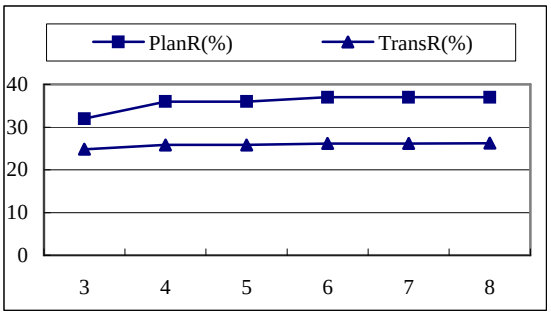

Figure 6(a). Plan Rate, Transition Rate vs. Max_Step. MinSupport=100, Success_Threshold=0.50.

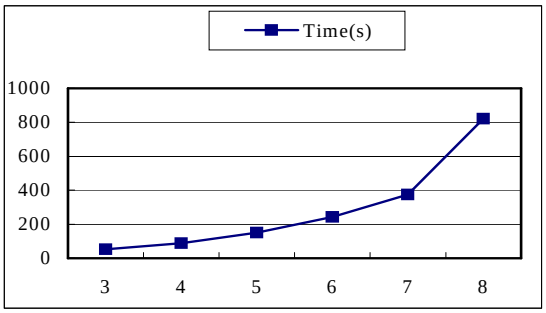

Figure 6(b). CPU Time vs. Max_Step. MinSupport=100, Success_Threshold=0.50.

Summary of the Test Results

In this test we used sequence mining as a filter before state space construction. We observe that using frequent string mining can indeed save a lot of searching time while at the same time, provide much more desirable plans in terms of their performance. Increasing the *Success_Threshold* can bring about much more promising plans. Increasing *Max_Step* does not bring any significant improvement in plan searching when this parameter is over a certain threshold.

Conclusions and Future Work

In this paper, we explored planning in the marketing planning domain, where the problem formulation is significantly different from the classical or probabilistic planning situations. Our approach combines both data mining and planning in order to build an abstraction space in which the plans are obtained. The plans are no longer transforming an agent's state from one initial state to a goal state; instead, in our situation we formulate plans that transform groups of customers from a set of initial states to positive class states. This formulation has many realistic applications in the real world, well beyond marketing planning.

In the future, we wish to consider different variations of the problem in marketing and other related domains. We wish to obtain some more realistic data from the customer relationship management domain and build a realistic system for marketing planning.

Acknowledgments

We thank Professors Charles Ling, D.Y. Yeung and Nevin Zhang for their valuable comments on this work. This work is supported by Hong Kong Research Grant Committee and SSRI grants.

References

S. Dibb, L. Simkin, and J. Bradley. 1996. *The Marketing Planning Workbook.* Routledge.

Bank Marketing Association, 1989. *Building a Financial Services Plan: Working Plans for Product and Segment Marketing.* Bank Marketing Association, Financial Sourcebooks, Naperville, Illinois.

C. L. Blake and C.J. Merz. 1998. *UCI Repository of machine learning databases* Irvine, CA: University of California, Department of Information and Computer Science. http://www.ics.uci.edu/~mlearn/MLRepository.html

P. Domingos, 1999. *MetaCost:* A general method for making classifiers cost sensitive. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155-164. ACM Press.

P. Domingos and M. Richardson, 2001. Mining the Network Value of Customers. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 57-66. San Francisco, CA: ACM Press.

D. Draper, S. Hanks, and D. Weld, 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on A.I. Planning Systems*.

C. Elkan, 2001. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973-978.

R. L. Keeney and H. Raiffa, 1976. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York.

C. X. Ling and C. Li. 1998. Data mining for direct marketing: Problems and solutions. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining* (KDD'98), pages 73-79, New York.

E. Pednault, N. Abe and B. Zadrozny. 2002. Sequential Cost-Sensitive Decision Making with Reinforcement Learning. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining* (KDD'02), pages 259-268. Edmonton, Canada.

J. R. Quinlan. 1993. C4.5: *Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

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

Q. Yang and H. Cheng, 2002. Mining Case Bases for Action Recommendation. In *Proceedings of 2002 IEEE International Conference on Data Mining* (ICDM'02). Maebashi, Japan.