Postprocessing Decision Trees to Extract Actionable Knowledge

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

Charles X. Ling and Tielin Chen
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
{ling, tchen}@csd.uwo.ca

ABSTRACT

Most data mining algorithms and tools stop at discovered customer models, producing distribution information on customer profiles. Such techniques, when applied to industrial problems such as customer relationship management (CRM), are useful in pointing out customers who are likely attritors and customers who are loyal, but they require human experts to postprocess the mined information manually. Most of the postprocessing techniques have been limited to producing visualization tools and interestingness ranking, but they do not directly suggest *actions* that would lead to an increase the objective function such as profit. In this paper, we present a novel algorithm that suggest actions to change customers from an undesired status (such as attritors) to a desired one (such as loyal) while maximizing objective function: the expected net profit. We develop these algorithms under resource constraints that are abound in reality. The contribution of the work is in taking the output from an existing mature technique (decision trees, for example), and producing novel, actionable knowledge through automatic postprocessing.

Keywords

Customer Relationship Management, action mining, profit-oriented data mining

1. INTRODUCTION

Extensive research in data mining has been done on discovering distributional knowledge about the underlying data. Models such as the Bayesian models, decision trees, support vector machines and association rules have been applied to various industrial applications such as customer relationship management (CRM). Despite such phenomenal success, most of these techniques stop short of the final objective of data mining, relying on such postprocessing techniques as visualization and interestingness ranking. While these techniques are essential to move the data mining result to the eventual application, they nevertheless require a great deal of human manual work by experts. Often, in industrial practice, one needs to walk an extra mile to extract the real "nuggets" of knowledge, the actions, in order to maximize the final objective functions.

In this paper, we present a novel postprocessing technique to mine actionable knowledge from decision trees. To illustrate our techniques, we focus on the application in CRM. Like most data mining algorithms today, a common problem in current applications of data mining in intelligent CRM is that people tend to focus on, and be satisfied with, building up the models and interpreting them, but not to use them to get profit explicitly. More specifically, most data mining algorithms (predictive or supervised learning algorithms) only aim at constructing customer profiles, which predict the characteristics of customers of certain classes. For example, what kind of customers (described by their attributes such as age, income, etc.) are likely attritors (who will go to competitors), and what kind are loyal customers? This knowledge is useful but it does not directly benefit the enterprise. To improve customer relationship, the enterprise must know what *actions* to take to change customers from an undesired status (such as attritors) to a desired one (such as loyal customers). To our best knowledge, no data mining algorithms have been made widely available to accomplish this important task in intelligent CRM.

Unlike distributional knowledge, in order to extract actionable knowledge from the output of other data mining algorithms, one must take into account resource constraints. Actions, such as direct mailing and sales promotion, cost money to the enterprise. At the same time, enterprises are increasingly constrained by cost cutting. There is thus a strong limitation on the number of customer segments that

the company can take on, or in the number of actions the company can exploit. To make a decision, one must take into account the cost as well as the benefit of actions to the enterprise. However, for each customer, there may be a large number of possible actions or action sets that can be applied to the customer. Which of the actions to take depends not only on the particular customers' situation, but also on other customers who might benefit from the same action.

In this paper, we present novel algorithms for postprocessing decision trees to maximize the profit-based objective function. We do this under resource constraints. More specifically, we take any decision tree as input, and mine the best k actions to be chosen in order to maximize the expected net profit of all the customers. We define two versions of the problem, show that finding the optimal solution for the problem is NP-complete, and design greedy heuristic algorithm to solve it efficiently. We compare the performance of the exhaustive search algorithm with a greedy heuristic algorithm, and show that the greedy algorithm is efficient while achieving results with quality very close to the optimal ones.

The rest of the paper is organized in the following order. We first present our base algorithm for finding unrestricted actions in Section 2. We then formulate two versions of the resource-limited action mining problems, and show that finding the optimal solution for the problems is NP-complete in Sections 4 and 5. We also show that our greedy algorithms are efficient while achieving results very close to the optimal ones obtained by the exhaustive search (which is exponential in time complexity). Conclusions and future work occupy Section 6.

2. ACTION MINING IN DECISION TREES

In our previous work [5, 12] we described a new data mining system that utilizes decision trees to discover actionable solutions for the status change problem in CRM. The algorithm is implemented in a data mining system called "Proactive Solution", a data mining software for intelligent CRM. In this section, we review essential components of Proactive Solution, as our new algorithms are based on them.

2.1 Proactive Solution

The data set that Proactive Solution takes consists of descriptive attributes and a class attribute. For simplicity, we consider a discrete-value problem, in which the class values are discrete values. Some of the values under the class attributes are more desirable than others. For example, in the banking application, the loyal status of a customer "stay" is more desirable than "not stay".

The overall process of Proactive Solution can be briefly described in the following four steps:

1. Import customer data with data collection, data cleaning, data pre-processing, and so on.
2. Build customer profile(s) using an improved decision-tree learning algorithm [11] from the training data. In this case, a decision tree is built from the training data to predict if a customer is in the desired status or not. One improvement in the decision tree building is to use the area under the curve (AUC) of the ROC curve [7, 10] to evaluate probability estimation (instead of the accuracy). Another improvement is to use Laplace Correction to avoid extreme probability values (see [5] for more details).
3. Search for optimal actions for each customer (see Section 2.2 for details). This is the key and novel component of our data mining system Proactive Solution.
4. Produce reports for domain experts to review the actions and selectively deploy the actions.

In the next subsection, we will discuss the `leaf-node search` algorithm used in step 3 (search for optimal actions) in detail.

2.2 Search for Un-restricted Optimal Actions

The `leaf-node search` algorithm (see below) searches for optimal actions to transfer each leaf node to another leaf node with a higher probability of being in a more desirable class. After a customer profile is built, the resulting decision tree can be used to classify, and more importantly, provide probability of customers in the desired status such as being loyal or high-spending. When a customer, who can be either an training example used to build the decision tree or an unseen testing example, falls into a particular leaf node with a certain probability of being in the desired status, the algorithm tries to "move" the customer into other leaves with higher probabilities of being in the desired status. The probability gain can then be converted into an expected gross profit.

However, moving a customer from one leaf to another means some attribute values of the customer must be changed. This change, in which an attribute A's value is transformed from v_1 to v_2, corresponds to an action. These actions incur costs. The cost of all changeable attributes are defined in a cost matrix (see Section 2.3) by a domain expert. The `leaf-node search` algorithm searches all leaves in the tree so that for every leaf node, a best destination leaf node is found to move the customer to. The collection of moves are required to maximize the net profit, which equals the gross profit minus the cost of the corresponding actions.

Based on a domain-specific cost matrix (Section 2.3) for actions, we define the net profit of an action to be as follows.

$$P_N = P_E \times P_{gain} - \sum_i COST_i \quad (1)$$

where P_N denotes the net profit, P_E denotes the total profit of the customer in the desired status, P_{gain} denotes the probability gain, and $COST_i$ denotes the cost of each action involved.

The `leaf-node search` algorithm for searching the best actions can thus be described as follows:

Algorithm `leaf-node search`

1. For each customer c, do
2. Let L be the leaf node in which c falls into;
3. Let S be a leaf node for c the maximum net profit P_N;
4. Output (L, S, P_N);

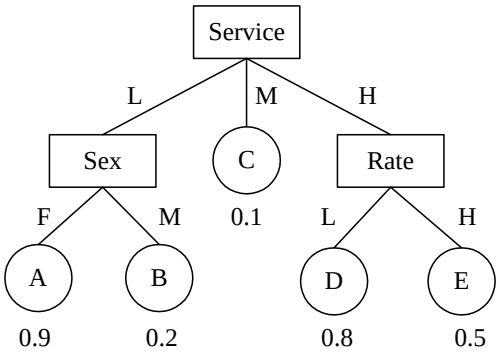

Figure 1: An example of customer profile

To illustrate, consider an example shown in Figure 1, which represents an overly simplified, hypothetical decision tree as the customer profile of loyal customers built from a bank. The tree has five leaf nodes (A, B, C, D, and E), each with a probability of customers' being loyal. The probability of attritors is simply 1 minus this probability.

Let, a customer, Jack, be given with Service (service level) being L (low), Sex being M (male), and Rate (mortgage rate) being L. The customer is classified by the decision tree. Clearly, Jack falls into the leaf B, which predicts that Jack will have only 20% chance of being loyal (or Jack will have 80% chance to churn in the future). The algorithm will now search through all other leaves (A, C, D, E) in the decision tree to see if Jack can be "replaced" into a best leaf with the highest net profit.

1. Consider leaf A. It does have a higher probability of being loyal (90%), but the cost of action would be very high (Jack should be changed to female), so the net profit is a negative infinity.
2. Consider leaf C. It has a lower probability of being loyal, so the net profit must be negative, and we can safely skip it.
3. Consider leaf D. There is a probability gain of 60% (80% − 20%) if Jack falls into D. The action needed is to change Service from L (low) to H (high). Assume that the cost of such a change is \$200 (given by the bank). If the bank can make a total profit of \$1000 from Jack when he is 100% loyal, then this probability gain (60%) is converted into \$600 (1000×0.6) of the expected gross profit. Therefore, the net profit would be \$400 (600 − 200).
4. Consider leaf E. The probability gain is 30% (50% − 20%), which transfers to \$300 of the expected gross profit. Assume that the cost of the actions (change Service from L to H and change Rate from L to H) is \$250, then the net profit of moving Jack from B to E is \$50 (300 − 250).

Clearly, the node with the maximal net profit for Jack is D, with suggested action of changing Service from L to H.

Notice that in the above example, the actions suggested for a customer-status change imply only correlations rather than causality between customer features and status. Similar to other data mining systems, the actions should be reviewed by domain experts before deployment. This is the Step 4 discussed at the beginning of this Section.

2.3 Cost matrix

In our solution, attribute-value changes will incur costs. These costs can only be determined by domain knowledge and domain experts. For each attribute used in the decision tree, a cost matrix is used to represent such costs. In many applications, the values of many attributes such as sex, address, number of children cannot be changed with any reasonable amount of money. Those attributes are called "hard attributes". For hard attributes, users must assign a very large number to every entry in the cost matrix.

If, on the other hand, some value changes are possible with reasonable costs, then those attributes such as the Service level, interest rate, promotion packages are called "soft attributes". Note that the cost matrix needs not to be symmetric. One can assign \$200 as the cost of changing service level from low to high, but infinity (a very large number) as the cost from high to low, if the bank does not want to "degrade" service levels of customers as an action.

One might ask why hard attributes should be included in the tree building process in the first place, since they can prevent customers from being moved to other leaves. This is because that many hard attributes are important in accurate probability estimation of the leaves. When the probability estimation is inaccurate, the reliability of the prediction would be low, or the error margin of the prediction would be high.

2.4 Limitations of Proactive Solution

A drawback of Proactive Solution is that it does not take resources into account. It assumes that the number of actions for each customer discovered by is unrestricted, although it is bounded by the total number of attributes in the tree. If the tree uses a total of 30 attributes, `leaf-node search` may suggest up to all 30 actions for the customers. In many situations, however, an enterprise has only limited resources in implementing the actions, as in the case of a marketing campaign. In such situations, an enterprise may wish to implement at most k actions in a particular marketing campaign. For example, a bank conducting a direct marketing campaign might choose to alter only five customer attributes, although the total number of customer attributes might be in the hundreds. In this situation, it is a critical question which k actions are the best, and how to find a subsets of those k actions for each customer, such that the total net profit for all customers is maximal.

Our subsequent sections answer the bounded resources problem. We consider two versions of the problem. The first version of the computational problem restricts the total number of attributes. It assumes that the company is interested in launching the campaign by alternating no more than k attributes for some user defined k. A customer may receive a

subset of these k attribute changes as actions. We call this problem the *bounded attribute problem.* A second version of the computational problem is when a company restricts the number of action sets to k, and each customer may receive one of the k action sets. Each action set may contain an (unlimited) number of actions. This corresponds to an action-oriented customer segmentation (clustering) of a campaign in which customers are partitioned into k subgroups, and each subgroup receives an unique action set from the k action sets. We call this second version of the problem the *bounded segmentation problem.*

Note that in the bounded segmentation problem, the total number of actions in the k action sets is usually greater than k, while in the bounded attribute problem, the total number of unique action sets (subsets of the k actions) is also normally greater than k. Thus, these two versions of the bounded resources problem do *not* subsume each other. However, in both versions, we expect the net profit from all cases in the testing set is maximized.

3. DATASET

We first briefly describe the dataset used in evaluating our new algorithms for the bounded attribute problem and the the bounded segmentation problem in this section. The dataset is from an insurance company in Canada. It consists of over 25,000 records for customers who have the status of "stay" or "leave" the insurance company. We will refer to them as positive and negative respectively. The dataset is described by over 60 attributes, many of which are not hard attributes. About 20 attributes are soft attributes with reasonable costs for value changes.

Since the data distribution in the training set is highly unbalanced, we first perform data sampling with the ratio of positive and negative examples is about 1 to 1 [6], in order to prevent the decision tree from predicting all the customers to be negative. In this setting, we have built a decision tree (See Step 2 in Section 2.1) with 153 leaf nodes. 87 of them are considered as negative leaf nodes because their probability of being positive is less than 50%, while the other 66 as positive leaf nodes. Furthermore, we have constructed a cost matrix for each attribute contained in the dataset according to their semantics in the real domain.

The testing set consists of about 300 examples. We use the built decision tree to classify them into corresponding leaf nodes.

We are interested in designing computational tools to convert a group of customers to more desirable status with bounded resources. Our new action-oriented algorithms for the bounded attribute problem and the bounded segmentation problem take the decision tree and the testing cases as input. We will discuss these two problems and algorithms that solve them in the next two sections.

4. BOUNDED ATTRIBUTE SET PROBLEM

Our first variation of the bounded resources problem is by restricting the total number of actions (attributes) to be k. For a user defined k value, we are interested in finding which k attributes (among m soft attributes) are the best to use for the maximal net profit on testing cases.

4.1 Problem Definition

The bounded-attribute set (BAS) problem is defined as follows.

Given

1. a decision tree built from training examples
2. a pre-specified constant k ($k \leq m$)
3. a set of testing examples

Find k attributes from a set of total m soft attributes such that when some of these k attributes are chosen as actions to apply to the testing examples, the total net profit is maximized.

To illustrate, consider an example in Figure 2. Assume that for leaf nodes L_1 to L_4, the probability values of being in the desired class are 0.9, 0.2, 0.8, 0.5, respectively. Now consider the task of transforming all leaf nodes from a lower probability value node to a higher one, such that the net benefit of such transformation is maximized. To illustrate this point, consider a test data set such that there is exactly one member that falls in each leaf node in this decision tree.

In order to calculate the net profit, we assume all leaf nodes to have a profit of one. We also assume that the cost of transferring a customer is equal to the number of attribute value changes multiplied by 0.1. Thus, to change from L_2 to L_1, we need to modify the value of the attribute `Status`, with a profit gain of $(0.9-0.2)\times 1-0.1=0.6$. This happens to be one of the optimal solutions for this test set and decision tree for $k=1$. That is, other single attribute change (Rate or Service) would not bring a net profit larger than 0.6.

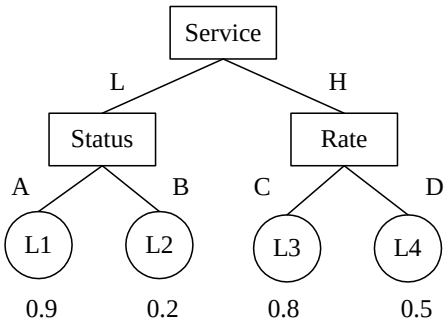

Figure 2: An example decision tree

How difficult is it to find an optimal solution for any value of k? The following theorem shows that the Bounded Attribute Set (BAS) problem is exponential to solve in the worst case.

THEOREM 1. *The BAS problem is NP-Complete.*

Proof Sketch: A special case of the BAS problem is equivalent to the Maximum SAT problem [9]. This problem is to consider every attribute as a potential literal, and every leaf node as a clause. The problem then becomes a conjunctive normal form. ■

4.2 Algorithms for BAS

For the problem with small sizes, we can enumerate all attribute sets of size k. This gives rise to an exhaustive search algorithm.

Algorithm Optimal BAS

1. From m soft attributes, choose k of them

2. Run `leaf-node search` (see Section 2.2) with those k attributes having constant cost matrix values; all other attributes have infinity cost values

3. Repeat **Step 2** for all k attributes (repeat m^k times), choose the k attributes that give rise the maximum net profit.

This algorithm runs in $O(m^k)$. If k is a variable, it is exponential in time complexity. This algorithm enumerates all attributes, and evaluate the total benefit that they bring for given test examples. For example, in our example in Figure 2, for $k = 2$ the list of attributes to consider are:

$$\{Service, Rate\}, \{Service, Status\}, \{Status, Rate\}$$

When evaluating $\{Service, Rate\}$, the cost of $Status$ is set to infinity. Thus, L_2 and L_4 should transfer to $L3$ to maximize their net profit, and the benefit brought by this set of attributes is 0.6.

By sacrificing the optimality requirement, we can solve the problem more efficiently using a greedy algorithm. Let *AttributeSet* be a set of attributes of size less than k. We denote *Cost(ASet, DTree, Test)* to be the optimal cost of transforming all members of test set *Test* in the decision tree *DTree* by setting all attributes other than those in *ASet* to infinity. This cost function can be evaluated in time N^2 where N is the number of leaf nodes in the decision tree.

Algorithm Greedy-BAS

1. $ASet \leftarrow \emptyset$,

2. for $l = 1$ to k

2.1 select $A_l \in AttibuteSet$ that maximizes $Cost(ASet \cup A_l, DTree, Test)$

2.2 $ASet \leftarrow ASet \cup A_l$

2.3 $AttributeSet \leftarrow AttributeSet - A_l$

end for

3. return $ASet$

To illustrate, consider the decision tree in Figure 2. In the first iteration of the loop in Step 2, suppose that the member which belongs to L_2 will fall into L_3 after `Service` change is applied. Similarly assume that L_4 will fall into L_1. The attribute found will be `Service`, since it will enable the maximal net profit of 1.0 by switching L_2 to L_3 and L_4 to L_1.

This greedy algorithm searches locally optimal attributes one at a time, and add those actions into the attribute set. Clearly, the algorithm only runs `leaf-node search` $k * m$ times, which is much lower when compared with the exhaustive search, which runs `leaf-node search` m^k times. The greedy algorithm may not be optimal, though, as locally optimal single action may not be globally optimal for a set of k best actions for the maximal profit. However, as many SAT problems, the probability of "hitting" a globally optimal solution is quite high. This is in particular demonstrated in the empirical evaluation section.

4.3 Empirical Evaluation

We have implemented both the exhaustive search and the greedy search on the dataset from the insurance company (Section 3). The results can be summarized in Table 1. It is clear that the greedy algorithm is much faster than the exhaustive algorithm. It is also interesting to observe that the action set with a larger k is a superset of the action set with a smaller k. This seems to indicate that the best attributes found in the action set with a small k are also optimal in the action set with a large k. This implies that the locally optimal actions are often also global optimal, which can be verified from the fact that the total net profit (and the action set) from exhaustive and greedy algorithms are the same, at least for small values of k when the exhaustive algorithm can finish in a feasible amount of time.

This interesting phenomenon (also see Section 5.3) deserves some further explanation. Many, if not all, of the results obtained by the greedy algorithm are the same as those by the optimal algorithm. At first, this may be surprising because the problem is NP-complete. However, since the optimization problem is in fact a SAT related problem, we may understand the situation better by drawing from the experiences of research done in constraint satisfaction and satisfiability areas. For example, [8] and [2] have demonstrated that for the N-Queens problems, even when N is approaching one million, there are often easy solutions using a min-conflict (greedy) solution. In the work of [1], a theoretical study of the hard versus easy satisfiability problems are analyzed, where the researchers found out that for many problem formulations under certain conditions, even the NP complete problem can find easy greedy solutions. In this view, the problems solved by our system fall into a very interesting "easy" problem region in data mining application domain.

5. BOUNDED SEGMENTATION PROBLEM

Our second version of the bounded resources problem on action mining is to merge the leaf nodes from m to k, where for each group G_i, $1 \leq i \leq k$, we wish to apply the same set of actions on it. This corresponds to an action-oriented customer segmentation (clustering) of a campaign in which customers are partitioned into k subgroups, and each subgroup receives an unique action set from the k action sets. For example, we may have an insurance company who is interested in assigning three account managers to take charge of the next marketing campaign, one for each customer group respectively. In this case, $k = 3$.

As an example, consider our decision tree again in Figure 2. Suppose that we wish to find one single customer segment ($k = 1$). One such group is $\{L_2, L_4\}$, with a selected action

	Greedy BAS			Optimal BAS		
	Time (ms)	Profit	Actions	Time (ms)	Profit	Actions
$k = 2$	745	5900	$\{a_{10}, a_{15}\}$	3003	5900	$\{a_{10}, a_{15}\}$
$k = 3$	1062	8310	$\{a_{10}, a_{15}, a_{18}\}$	14922	8310	$\{a_{10}, a_{15}, a_{18}\}$
$k = 4$	4127	14700	$\{a_8, a_{10}, a_{15}, a_{18}\}$	84286	14700	$\{a_8, a_{10}, a_{15}, a_{18}\}$

Table 1: Comparison of greedy and optimal (exhaustive) BAS algorithms on the insurance dataset.

set $\{Service \leftarrow H, Rate \leftarrow C\}$ which can be applied on it. Assume L_2 and L_4 only contain one example. If we consider transferring this group to leaf node L_3, L_2 has a profit gain of $(0.8 - 0.6) \times 1 - 0.2 = 0.2$ and L_4 has a profit gain of $(0.8 - 0.3) \times 1 - 0.1 = 0.4$. Thus the net benefit for this group is $0.2 + 0.4 = 0.6$.

5.1 Problem Definition

Formally, the bounded segmentation problem (BSP) is defined as follows:

Given

1. a decision tree built from training examples with a set of source leaf nodes $L_s = \{L_1, L_2, \ldots, L_m\}$, and a set of destination leaf nodes $L_d = \{L_1, L_2, \ldots, L_n\}$. Each destination leaf node corresponds to a potential action set $A_i \in A$, $1 \leq i \leq n$, which can be applied to move a source leaf node to it.
2. a pre-specified constant k ($k \leq m$) for the number of customer marketing segments.
3. a set of testing examples.

Find

1. k groups of the source leaf nodes in the decision tree $G_1, G_2, \cdots, G_k$, each of which consists of one or more source leaf nodes.
2. for each group G_i, the corresponding action set A_i, $1 \leq i \leq k$, can be applied on it such that the expected net profit for k action sets is maximized.

For any k groups of source leaf nodes G_i, the expected net profit of applying the corresponding action set A_i, $1 \leq i \leq k$, is defined as:

$$\sum_{i=1}^{k} Profit(A_i(G_i)) \tag{2}$$

where $Profit(A_i(G_i))$ indicates the total net profit gain by applying the action set A_i to all the examples contained in group G_i.

As an example, consider again our decision tree in Figure 2. Assume each leaf node in the group $\{L_2, L_4\}$ contains 2 examples, if we apply the action set $\{Service \leftarrow H, Rate \leftarrow C\}$ to transfer this group to the destination leaf node L_3, the expected net profit is calculated as $2 \times 0.2 + 2 \times 0.4 = 1.2$.

To clarify the formulation of the problem, let us consider a profit matrix M formed by listing all source leaf nodes L_i and all potential action sets A_j, where $M[i, j] = P_{ij}$, $(P_{ij} \geq 0)$, $1 \leq i \leq m$, $1 \leq j \leq n$. P_{ij} denotes the profit gain by applying A_j to L_i. If $P_{ij} \neq 0$, that is, applying A_j to transfer L_i to the corresponding destination leaf node can bring net profit, in this case, we say the source leaf node L_i can be *covered* by the action set A_j, otherwise.

As an example, part of the profit matrix corresponding to leaf node L_2 is:

	A_1(to L_1)	A_2 (To L_3)	A_3 (to L_4)
L_2	0.6	0.4	0.1

where $A_1 = \{Status = A\}$, $A_2 = \{Serice = H, Rate = C\}$ and $A_3 = \{Service = H, Rate = D\}$.

Given this profit matrix M, the problem is to select k action sets $C = \{A_1, A_2, \ldots, A_k\}$, $(C \subseteq A)$ among all of the n action sets. Let $\mathsf{cover(C)}$ denote the set of leaf nodes covered by the k action sets C. For each $L_i \in \mathsf{cover}(k)$, there exists a mapping $\mathsf{assign}(L_i) = A_s$, where $s = \arg_j max\{P_{ij}\}$, $j = 1, 2, \ldots, k$, such that the net profit of the mapping is maximized, that is, the following formula is maximized:

$$\sum_{L_i \in \mathsf{cover}(C)} max\{P_{ij}\}, j = 1, 2, \ldots, k. \tag{3}$$

In other words, for each leaf node covered by k action sets C, since it may be covered by several action sets at the same time, what we really want is to select one action set with maximal profit gain to apply on it. Therefore, the total net profit is the summation of possibly maximal profit gain for each covered leaf node.

THEOREM 2. *The BSP problem is NP-Complete.*

Proof Sketch: A special case of the BSP problem is equivalent to the maximum coverage problem with unit costs, which aims at finding k sets such that the total weight of elements covered is maximized [4], where the weight of each element is the same for all the sets. ■

5.2 Algorithms for BSP

Similar to algorithms for BAS (Section 4.2), our first solution is an exhaustive search algorithm for finding the k optimal action sets with maximal net profit.

Algorithm Optimal-BSP

1. for each $A_l \in A$, $1 \leq l \leq n$, choose any combination of k action sets, do

1.1 Group the leaf nodes into k groups

2.1 Evaluate the net benefit of the action sets on the groups

end for

2. return the k action sets with associated leaf node groups which have the maximal net benefit.

Since the optimal-BSP needs to examine every combination of k action sets, the computational complexity is $O(n^k)$, which is exponential in the value of k.

To avoid the exponential worst case complexity, we have also developed a greedy algorithm (similar to the one in Section 4.2) which can reduce the computational cost but also guarantee the quality of the solution at the same time. Consider the following generalization of the maximum coverage problem. Given a set with m leaf nodes $L_s = \{L_1, L_2, \ldots, L_m\}$, each associated with a different profit P_{ij} ($P_{ij} \geq 0$) for each action set A_j, $1 \leq j \leq n$. Each A_j can be denoted as a subset of L_s which only contains the covered leaf nodes L_i for $P_{ij} \neq 0$, $1 \leq i \leq m$. The goal is to choose k action sets so as to maximize the net profit of covered leaf nodes. We can solve this problem using a greedy algorithm below, where C is the resulting k action sets.

Algorithm Greedy-BSP

1. $C \leftarrow \emptyset$

2. for $l = 1$ to k

2.1 select $A_l \in A$ that maximizes

$$\sum_{L_i \in \mathsf{cover}(C \cup A_l)} max\{P_{ij}\}, j = 1, 2, \ldots, l$$

2.2 $C \leftarrow C \cup A_l$

end for

3. return C

This algorithm can be shown to perform close to the optimal result. In particular, we can exploit the complexity analysis of the approximate maximum coverage algorithm given in [3]. In order to prove the approximation ratio of the solution returned by Greedy-BSP to one by Optimal-BSP, We firstly need to establish the following two lemmas.

Let $Profit(Greedy)$ and $Profit(OPT)$ be the net profit returned by the Greedy-BSP algorithm and the Optimal-BSP algorithm respectively, we have

LEMMA 1. *For $l = 1, 2, \ldots, k$, we have*

$$Profit(\cup_{i=1}^{l} A_i) - Profit(\cup_{i=1}^{l-1} A_i) \geq \frac{(Profit(OPT) - Profit(\cup_{i=1}^{l-1} A_i)}{k} \quad (4)$$

PROOF. Let the optimal solution returned by Optimal-BSP consists of k optimal action sets. Suppose Greedy-BSP has already selected $(l-1)$ action sets so far, m of which are contained in the optimal solution. Now we consider the situation where Greedy-BSP selects the next A_l action set. Because of the heuristic strategy used in the step 2.1 of the Greedy-BSP algorithm, $Profit(\cup_{i=1}^{l} A_i) - Profit(\cup_{i=1}^{l-1} A_i)$ represents the additional profit gain achieved by A_l. In addition, A_l should be a set that can achieve maximal additional profit gain. On the other hand, assume Greedy-BSP selects those $(k-m)$ optimal action sets in the optimal solution and yet have not chosen by itself, the profit gain of this batch procedure is at least $Profit(OPT) - Profit(\cup_{i=1}^{l-1} A_i)$. According to the pigeonhole principle, there must exist one single action set in the remaining $(k-m)$ optimal action sets, whose profit is at least $\frac{Profit(OPT) - Profit(\cup_{i=1}^{l-1} A_i)}{k-m}$. Since this action set is also a candidate for selecting the next A_l, we have

$$\begin{aligned} & Profit(\cup_{i=1}^{l} A_i) - Profit(\cup_{i=1}^{l-1} A_i) \\ \geq & \frac{Profit(OPT) - Profit(\cup_{i=1}^{l-1} A_i)}{k-m} \\ \geq & \frac{Profit(OPT) - Profit(\cup_{i=1}^{l-1} A_i)}{k} \end{aligned}$$

□

LEMMA 2. *For $l = 1, 2, \ldots, k$, we have*

$$Profit(\cup_{i=1}^{l} A_i) \geq [1 - (1 - \frac{1}{k})^l] Profit(OPT) \quad (5)$$

Proof Sketch: The proof can be done by induction using Lemma 1, similar to the proof of Lemma 3.13 in [3]. ■

Based on the above two established lemmas, we have the following theorem:

THEOREM 3. *The Greedy-BSP is a $(1-\frac{1}{e})$-approximation algorithm.*

$$\begin{aligned} Profit(Greedy) &\geq [1 - (1 - \frac{1}{k})^k] Profit(OPT) \\ &> (1 - \frac{1}{e}) Profit(OPT) \quad (6) \end{aligned}$$

PROOF. Theorem 3 follows directly from Lemma 2 by letting $l = k$. In addition, because $\lim_{k \to \infty} 1 - (1 - \frac{1}{k})^k = 1 - \frac{1}{e}$ and $1 - (1 - \frac{1}{k})^k$ is decreasing, it follows that $1 - (1 - \frac{1}{k})^k > 1 - \frac{1}{e}$. □

5.3 Empirical Evaluation

To compare the performance of the Greedy-BSP algorithm and the Optimal-BSP algorithm, we have again carried out experiments on the dataset from the insurance company. We apply Greedy-BSP and Optimal-BSP respectively to find the pre-specified k action sets with maximal net profit.

Figure 3 shows the net profit obtained by the two algorithms over the number of action sets k. As shown in the figure, the

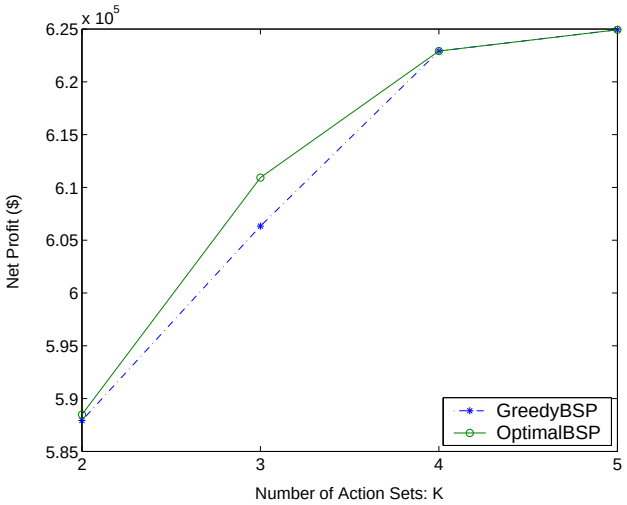

Figure 3: Net Profit vs. Number of action sets

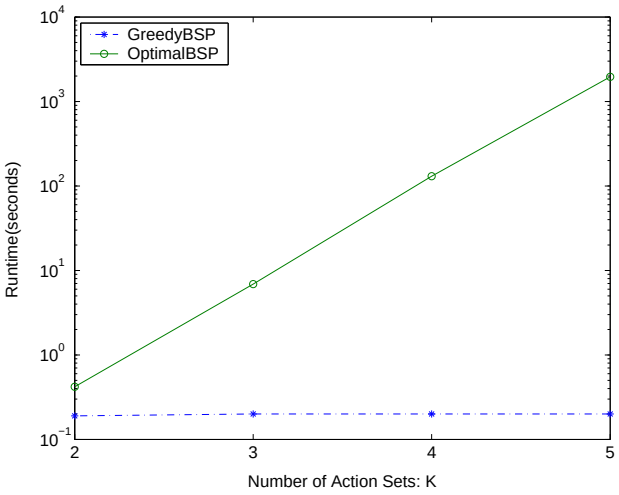

Figure 4: Runtime vs. Number of action sets

net profit increases for both Greedy-BSP and Optimal-BSP with increasing number of action sets k, similar to the results of BAS in Section 4.3. This is because if more customers are transformed to a desired status, it is more possible to obtain higher profit. In addition, an important property to note is that, for a specific k, the net profit obtained by Greedy-BSP is very close to or the same as that by Optimal-BSP, which can guarantee the quality of solution provided by Greedy-BSP.

#Action Sets	Selected Action Sets	
	Greedy BSP	Optimal BSP
$k = 2$	$\{\mathbf{A_3}, A_{29}\}$	$\{\mathbf{A_1}, A_{29}\}$
$k = 3$	$\{\mathbf{A_3}, A_{20}, A_{29}\}$	$\{\mathbf{A_1}, A_{20}, A_{29}\}$
$k = 4$	$\{A_1, A_3, A_{20}, A_{29}\}$	$\{A_1, A_3, A_{20}, A_{29}\}$
$k = 5$	$\{A_1, A_3, A_{20}, A_{29}, A_{42}\}$	$\{A_1, A_3, A_{20}, A_{29}, A_{42}\}$

Table 2: Selected action sets vs. Number of action sets

Table 2 compares the k action sets selected by both Greedy-BSP and Optimal-BSP over the number of action sets k. Similar conclusions can be drawn on these action sets as the ones discovered by BAS algorithms (see Section 4.3). There are totally 66 action sets provided by the decision tree built in our experiments. As shown in the table, the action sets selected by Greedy-BSP is very close to that by Optimal-BSP for the same number of action sets k.

Notice the apparent difference between Table 1 in Section 4.3 and Table 2 here. Table 1 lists actual actions (such as a_8) while Table 4.3 lists action sets (each action set contains a number of actions; for example, A_3 contains four (4) actions on a_8, a_{15}, a_{22}, and a_{25}). Recall that the total number of actions in the k action sets is usually greater than k, while the total number of unique action sets (subsets of the k actions) is also normally greater than k. Thus, these two versions of the problems do not subsume each other, and the actions suggested by the two versions are different (though with a good overlap) on the same testing set.

Figure 4 shows the runtime of the two algorithms over the number of action sets k. Note that the runtime of y-axis uses a logscale. As expected, Greedy-BSP is much more efficient and scalable than Optimal-BSP. Similar results are observed in the BAS algorithms (Section 4.3). For Greedy-BSP, The runtime is always around 0.20 seconds irrespective of the number of action sets k. On the other hand, the runtime for Optimal-BSP increases exponentially with the increasing number of action sets k. This is because it needs to compare much more combinations for larger k in order to obtain maximal net profit.

We conclude from our experiments that, Greedy-BSP can find k action sets with maximal net profit, which is very close to those found by Optimal-BSP, at least for small values of k for which Optimal-BSP terminates in a reasonable amount of time. At the same time, Greedy-BSP can scale well with the increasing number of action sets k, which is more efficient than Optimal-BSP. We have drawn similar conclusions from the BAS experiments (Section 4.3).

6. SUMMARY

Intelligent CRM improves customer relationship from the data about customers. Unfortunately, very little work has been done in data mining on how to use actions to improve such relationship of customers. Such actions change customers from an undesired status to a desired one. This paper improve previous work further by mining the best k actions that maximize the net profit. We first formulate two versions of the problem, and show that it is NP-complete. We then describe a greedy algorithm that efficiently discovers the best k actions, and is shown that the total net profit using those k actions on all customers is very close to the optimal value obtained by the exhaustive (and exponential) algorithm. The results discussed in this paper offer effective solutions to intelligent CRM for Enterprises.

In our future work, we will evaluate the effectiveness of our algorithms in the real-world deployment of the action-oriented data mining.

7. REFERENCES

[1] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.

[2] Jun Gu. *Parallel Algorithms and Architectures for Very Fast AI Search*. PhD thesis, University of Utah, 1989.

[3] D. S. Hochbaum. Approximation algorithms for np-hard problems. page 136, Chapter 3, 1995. PWS Publishing Company.

[4] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in covering problems. In *Unpublished manuscript*, 1994.

[5] C. X. Ling, T. Chen, Qiang Yang, and Jie Cheng. Mining optimal actions for intelligent crm. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2002.

[6] C. X. Ling and C. Li. Data mining for direct marketing – specific problems and solutions. In *Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 73 – 79. 1998.

[7] C. X. Ling and H. Zhang. Toward bayesian classifiers with accurate probabilities. In *Proceedings of The Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. Springer, 2002.

[8] S Minton and Philips A B Laird P. Johnston, M. D.

[9] M.R.Garey and D.S. Johnson. *Computers and Intractability: A guide to the Theory of NPCompleteness*. 1979.

[10] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1998.

[11] J.Ross Quinlan. *C4.5 Programs for machine learning*. Morgan Kaufmann, 1993.

[12] Qiang Yang and Hong Cheng. Case mining for action recommendations. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2002.