In Search for Best Meta-Actions to Boost Businesses Revenue

Jieyan Kuang[1] and Zbigniew W. Raś[1,2]

[1] University of North Carolina, Dept. of Computer Science,
Charlotte, NC, 28223, USA
[2] Warsaw Univ. of Technology, Inst. of Computer Science, 00-665 Warsaw, Poland
e-mail: jkuang1@uncc.edu, ras@uncc.edu

Abstract. The ultimate goal of our research is to build recommender system driven by action rules and meta-actions for providing proper suggestions to improve revenue of a group of clients (companies) involved with similar businesses. Collected data present answers from 25,000 customers concerning their personal information, general information about the service and customers' feedback to the service. This paper proposes a strategy to classify and organize meta-actions in such a way that they can be applied most efficiently to achieve desired goal. Meta-actions are the triggers that need to be executed for activating action rules. In previous work, the method of mining meta-actions from customers' reviews in text format has been proposed and implemented. Performed experiments have proven its high effectiveness. However, it turns out that the discovered action rules need more than one meta-action to be triggered. The way and the order of executing triggers causes new problems due to the commonness, differential benefit and applicability among sets of meta-actions. Since the applicability of meta-actions should be judged by professionals in the field, our concentration is put on designing a strategy to hierarchically sort and arrange so called meta-nodes (used to represent action rules and their triggers in a tree structure) as well as to compute the effect of each meta-node. Furthermore, users will have more concrete options to consider by following the path in trees built from these meta-nodes.

1 Introduction

The competition between companies dealing with similar businesses is always intensive and full of gunpowder, and all of them are trying their best to attract and keep as many customers as possible. To evaluate and improve the performance of a company's growth engine, NPS (Net Promoter System) has been designed and becomes one of the most important measurements nowadays. Generally speaking, NPS system divides customers into three groups: *Promoter, Passive* and *Detractor*, which represent customers' satisfaction, loyalty and likelihood to recommend the company to their friends in a descending order [9].

In our project, we intend to build a hierarchical recommender system driven by action rules and meta-actions for providing proper suggestions to improve clients' NPS rating. There are 34 clients (companies) in our dataset and they deal with similar businesses all over US and south Canada. To collect the research data, over 25,000 customers have been randomly selected to answer a questionnaire that asks customers'

personal information, general information about the service and customers' feedback to the service. In terms of giving feedback about the service by customers, they can assign numeric scores ranging from 0 to 10 to express their satisfaction (higher the score is, more satisfied the customer is), and they are also welcomed to give more details which are recorded in text format. Based on the numeric values, customers are categorized into three NPS statuses: 9-10 is promoter, 7-8 is passive and 0-6 is detractor. Furthermore, the NPS rating of a client can be calculated as the percentage difference between customers that are promoter and customers that are detractor.

In previous work, a hierarchical dendrogram has been generated by applying agglomerative clustering algorithm to the semantic distance matrix covering 34 clients [3], which demonstrates the similarity among clients concerning the knowledge hidden in datasets and is the foundation of our project. With the dendrogram and the fact that clients can learn from each other by exchanging their hidden knowledge, a strategy called HAMIS (Hierarchically Agglomerative Method for Improving NPS) has been designed to maximally extend individual dataset for each client by merging it with datasets of clients who are semantically similar, have better NPS rating and better classification results [4]. Then action rules, that show what minimum changes are needed for each client to move its ratings to the Promoter's group, are generated from their datasets enlarged by HAMIS. Meta-actions are the ultimate triggers that will be executed by clients to make action rules effective. Strategy for generating meta-actions is proposed and experiments have been conducted to prove its effectiveness and efficiency.

Clearly, each action rule can be triggered by different groups of meta-actions. In the ideal scenario, all discovered actions rules should be triggered. So the problem of finding the smallest sets of meta-actions triggering all action rules seems important to be solved. However each meta-action has a number of parameters assign to it like its cost or benefit. We may sort all meta-actions in a certain way and then apply them one by one to the discovered action rules only if the increment in NPS by triggering these rules is above some threshold value. Saying another words, some meta-actions may not be executed at all. The same some action rules will not get triggered because the benefit of doing that will be too small.

The main contribution in this paper lays on designing a strategy to classify and organize meta-actions hierarchically for providing clients with an efficient way of picking right choices out of an the entire mass. Traditional influence matrix is transformed into a more straightforward structure named *advanced matrix*. Meta-node, introduced in this paper, is used to store a set of action rules and their triggers. So each meta-node is seen as a possible option to be recommended by our system to a client. When processing meta-actions from the most common one to the least one (number of action rules it is associated with), a new meta-node is built or updated if a new action rule can be triggered. Some meta-nodes are connected and form a "tree". Lower a meta-node is located in a tree, greater is its effect. Our strategy will result in a forest of trees. Details concerning the process of generating these trees is illustrated thoroughly in this paper.

2 Action Rules

The concept of an action rule was firstly proposed by Ras and Wieczorkowska in [6] and investigated further in [2], [5], [1] and [8]. Action rules indicate possibly the smallest sets of changes that should be made to achieve the desirable effect under certain possibilities. An action rule usually consists of three types of features: stable attribute, flexible attribute and decision attribute. Stable attributes refer to these features of which values can not be changed, while flexible attributes denote the features of which values can be changed. Decision attribute is a flexible attribute and the most distinguished one because it contains the decisional values that we aim to transition from or to.

Before explaining the definition of action rules, let's first recall the definition of an information system (dataset). By an information system, we mean a triple $S = (X, A, V)$, where:

1. X is a nonempty, finite set of objects
2. A is a nonempty, finite set of attributes, i.e.
   $a : U \longrightarrow V_a$ is a function (can be partial function) for any $a \in A$, where V_a is called the domain of a
3. $V = \bigcup\{V_a : a \in A\}$.

Based on the partition of attributes in an action rule, we assume that $A = A_{St} \cup A_{Fl}$, where A_{St} and A_{Fl} denote $stable$ attributes and $flexible$ attributes respectively. Additionally, $A_{Fl} = A_{fl} \cup A_d$, where A_d and A_{fl} represent $decision$ attribute and flexible attributes other than decision attribute respectively. Information system S is a decision system if decision attribute defined.

An action rule is composed of *atomic action sets*. By an *atomic action set* we mean a singleton set containing an expression $(a, a_1 \rightarrow a_2)$ called atomic action, where a is an attribute and $a_1, a_2 \in V_a$. If $a_1 = a_2$, then a is called stable on a_1. Instead of $(a, a_1 \rightarrow a_1)$, we usually write (a, a_1) for any $a_1 \in V_a$. By *Action Sets* we mean a smallest collection of sets such that:

1. If t is an atomic action set, then t is an action set.
2. If t_1, t_2 are action sets, then $t_1 \cup t_2$ is a candidate action set.
3. If t is a candidate action set and for any two atomic actions $(a, a_1 \rightarrow a_2)$, $(b, b_1 \rightarrow b_2)$ contained in t we have $a \neq b$, then t is an action set. Here b is another attribute ($b \in A$), and $b_1, b_2 \in V_b$.

By the domain of an action set t, denoted by $Dom(t)$, we mean the set of all attribute names listed in t. By an *action rule* we mean any expression $r = [t_1 \Rightarrow t_2]$, where t_1 and t_2 are action sets. Additionally, we assume that $Dom(t_2) \cup Dom(t_1) \subseteq A$ and $Dom(t_2) \cap Dom(t_1) = \emptyset$. The domain of action rule r is defined as $Dom(t_1) \cup Dom(t_2)$.

Now we give an example assuming that a, b and d are stable attribute, flexible attribute and decision attribute respectively in S. Expressions (a, a_2), $(b, b_1 \rightarrow b_2)$, $(d, d_1 \rightarrow d_2)$ are examples of atomic actions. Expression (a, a_2) means that the value a_2 of attribute a remains unchanged. Expression $(b, b_1 \rightarrow b_2)$ means that the value

of attribute b is changed from b_1 to b_2. Expression $r = [\{(a, a_2), (b, b_1 \rightarrow b_2)\} \Rightarrow \{(d, d_1 \rightarrow d_2)\}]$ is an example of an action rule saying that if value a_2 of a remains unchanged and value of b will change from b_1 to b_2, then the value of d will be expected to transition from d_1 to d_2.

In early research, action rules can be constructed from two classification rules [7]. Taking the example of action rule r shown above for instance, r can be seen as the composition of two association rules r_1 and r_2, where $r_1 = [\{(a, a_2), (b, b_1)\} \rightarrow (d, d_1)]$ and $r_2 = [\{(a, a_2), (b, b_2)\} \rightarrow (d, d_2)]$. In [11], the definition of support and confidence of action rule r has been proposed. However, there is a slight difference in defining the support in our case. The definition of support and confidence of r, used in this paper, is given below:

- $sup(r) = sup(r_1)$
- $conf(r) = conf(r_1) \cdot conf(r_2)$

The confidence of an action rule is still the multiplication of the confidences of two involved association rules. But the support of action rule becomes the support of association rule r_1. In our example, it is the number of objects x ($x \in X$) which values of attributes a, b and d are equal to a_2, b_1 and d_1 respectively. By defining support of action rules this way, the number of objects x in S that potentially can be affected by applying this action can be tracked and used to evaluate its performance.

3 Meta-Actions

As an action rule can be seen as a set of atomic actions that need to be made happen for achieving the expected result, meta-actions are the actual solutions that should be executed to trigger the corresponding atomic actions, Table 1 below shows an example of influence matrix which describes the relationships between the meta-actions and atomic actions influenced by them.

Table 1. Meta-actions Influence Matrix for S

	a	b	d
$\{M_1, M_2, M_3\}$		$b_1 \rightarrow b_2$	$d_1 \rightarrow d_2$
$\{M_1, M_3, M_4\}$	a_2	$b_2 \rightarrow b_3$	
$\{M_5\}$	a_1	$b_2 \rightarrow b_1$	$d_2 \rightarrow d_1$
$\{M_2, M_4\}$		$b_2 \rightarrow b_3$	$d_1 \rightarrow d_2$
$\{M_1, M_5, M_6\}$		$b_1 \rightarrow b_3$	$d_1 \rightarrow d_2$

In Table 1, a, b and d are same attributes in decision system S as mentioned in previous section, and $\{M_1, M_2, M_3, M_4, M_5, M_6\}$ is a set of meta-actions which hypothetically trigger action rules generated from S. Each cell in a row shows an atomic action that can be invoked by the set of meta-actions listed in the first column of that row. For instance, the first row shows that the atomic actions $(b_1 \rightarrow b_2)$ and $(d_1 \rightarrow d_2)$

can be activated by executing meta-actions M_1, M_2 and M_3 together. Unlike the traditional influence matrix in [11] and [10] which involves only one single meta-action in each row, here the transaction of atomic actions in our domain relies on one or more meta-actions.

Clearly, an action rule can be triggered with the set of meta-actions listed in one single row of an influence matrix as long as all of its atomic actions are listed in that row. Otherwise, selecting a proper set of meta-actions combined from multiple rows becomes necessary. If we would like to activate action rule r given in previous section, it is quite obvious that the combination of meta-actions listed in the first and second row of Table 1 could trigger r, as meta-actions $\{M_1, M_2, M_3, M_4\}$ cover all required atomic actions (a, a_2), $(b, b_1 \rightarrow b_2)$ and $(d, d_1 \rightarrow d_2)$ in *r*. On the other hand, one set of meta-actions could possibly trigger multiple action rules, like the meta-action set $\{M_1, M_2, M_3, M_4\}$ triggers not only r as mentioned but also action rule $[\{(a, a_2), (b, b_2 \rightarrow b_3)\} \Rightarrow \{(d, d_1 \rightarrow d_2)\}]$ by following the second and forth row in Table 1, if such action rule exists in S.

Suppose a set of meta-actions $M = \{M_1, M_2, ..., M_n : n > 0\}$ can trigger a set of action rules $\{r_1, r_2, ..., r_m : m > 0\}$ that covers certain objects in S. The coverage or support of M is the summation of support of all covered action rules as shown below, which is the total number of objects in the initial state that are going to be affected by M. The confidence of M is calculated by averaging the confidence of all covered action rules. Furthermore, the effect of applying M is defined as the product of its support and confidence ($sup(M) \cdot conf(M)$). It represents the number of objects in the system that are going to be improved by applying M which also is used for calculating the increment of NPS rating caused by it. Therefore, greater is the effect of M, larger increment of NPS rating will be produced.

- $sup(M) = \sum_{i=1}^{m} sup(r_i)$
- $conf(M) = \frac{\sum_{i=1}^{m} sup(r_i) \cdot conf(r_i)}{\sum_{i=1}^{m} sup(r_i)}$

4 Process of the Strategy

4.1 Background

The importance of a well organized and informative result is mentioned briefly earlier. Now, the reasons for pursuing that are explained thoroughly below:

1. The commonness between sets of meta-actions tells about an enhancement. Although actions rules are triggered by different sets of meta-actions, there are some meta-actions which exist in different sets. The commonness between two sets of meta-actions is defined as the percentage of the common meta-actions both sets occupy in the smaller set. If the commonness between two sets equals to 1, then the smaller set belongs to the larger one. In this case, with the larger set of meta-actions applied, besides the action rules covered by it, the action rules covered by the smaller set are also invoked. So the effect of executing the larger set of meta-actions is expected to be enhanced with the effect of smaller one added.

2. The effect of sets of meta-actions tells about preference. Generally speaking, in contrast to the smaller set of meta-actions, it is understandable that the larger set to which a smaller one belongs is more preferable to clients due to its greater effect. On the other hand, for disjoint sets of meta-actions, the selection preference is no longer built on the commonness, but to some extent directly on their effect. Another words, the ones with more significant effect are certainly more preferable to clients.
3. The applicability of certain meta-actions tells about rejection. In our domain, an action rule will not be effective unless all required meta-actions have been taken. So, clients are encouraged to perform all given meta-actions to achieve the expected improvement. But in real life, we can foresee that some of them will be rejected by clients as it is undeniable that some meta-actions are more acceptable and easier to be executed than other ones because of their cost or other issues. For example, if clients are given a set of meta-actions in which lowering the price is mentioned, clients will not like it, even though they are advised by the system to do it for pleasing the customers who complain about the cost. So meta-actions with poor applicability will lead to the rejections from clients.

Considering these three reasons together, the decision of choosing meta-actions to apply does not solely depend on us any more, as our system can calculate the effect of sets of meta-actions, but we are not the experts in estimating the difficulty or cost of adopting certain meta-actions in the field. The decision must be done by clients and technicians who have professional knowledge about this practice. Also, it should be mentioned that clients probably would ignore a set of meta-actions if its effect and cost are remarkably unbalanced and there is an alternative. So the final decision should be determined by our clients who weight the effect and cost of applying some actions with their experience and expertise, which proves the necessity of the strategy proposed in this paper.

This strategy aims to create actionable paths that are well classified from groups of meta-actions and lead to certain increment of NPS rating. These actionable paths can be seen as a forest of trees where each tree is made of one or more meta-nodes that are hierarchically structured regarding their relation. A meta-node is a choice for client to consider and it represents a set of action rules and their triggers. So, the effect of a meta-node is the effect of the group of meta-actions in it. In a tree (or path), the connection between two nodes is a parent-child relationship and it indicates that the commonness between the groups of meta-actions contained in them equals to 1. So, the node with smaller set of meta-actions is defined as the parent of the node with larger set and it is put on an upper level. The parent-child relationship is a one to many mapping, as one node has at most one parent, but a parent node can have more than one child.

The details concerning the strategy will be explained in following parts, which is the main focus of this paper.

4.2 Advanced Matrix: Transformation of Influence Matrix

To find triggers for action rules, influence matrix mentioned earlier is a semi-product for us and it should be transformed into an advanced representation to demonstrate the triggers for each rule straightforwardly. Table 2 is an example of advanced matrix. In

the table, there are n meta-actions in total and M_i represents every single meta-action, where $0 < i < n + 1$. There are m action rules and "rule j" denotes every generated action rule, where $0 < j < m + 1$. For each row, each cell corresponding to the meta-action that is responsible for triggering rule j is filled with 1, while other cells are filled with 0.

Table 2. Advanced Matrix: Action Rules and Their Triggers (Meta-actions)

	M_1	M_2	M_3	...	M_n
rule 1	0	1	1	...	1
rule 2	1	0	1	...	1
rule 3	0	0	1	...	1
rule 4	0	1	0	...	1
...	...	...	...	...	...
rule m	0	0	1	...	1

Advanced matrix is transformed from influence matrix. Based on advanced matrix, the association between meta-actions and action rules becomes more apparent and it is easy to sort all the meta-action by their popularity, which is the basis of the following procedures.

4.3 Presentation of the Strategy

As mentioned earlier, the goal of our strategy is to build a forest of trees where each tree is composited by meta-nodes that are linked via a parent-child relationship. Algorithm 1 is designed to organize the advanced matrix in a hierarchical manner and generate a list of meta-nodes *T*, so it is obvious that an advanced matrix *M* should be given. At the beginning of Algorithm 1, *meta-list* and *rules* are generated, which are a list of meta-actions descendingly sorted by their popularity in *M* and a set of all action rules in *M* respectively. Meanwhile, a map *MetaMap* and a list *T* are initialized as well. *MetaMap* is created to store the mappings (entries) from sets of meta-actions to their sets of action rules during the entire process. Generally speaking, the content in an entry or mapping in *MetaMap* is identical to a meta-node at the end of the algorithm and each action rule can be involved with only one mapping. The list *T* will be used to store the continually generated meta-nodes and it will be our final product.

With a sorted list of meta-actions, the algorithm repeatedly runs the main part with one meta-action at a time from the most frequent to least frequent one. Given a meta-action (which is represented as *meta*) in each round, all the action rules are iterated one by one and only the ones associated with *meta* in *M* will be continued to the following procedures. For each continued action rule (which is denoted as *r*), the existence of a mapping established in *MetaMap* involving *r* differs the way of handling it and we can easily tell its existence by attempting to retrieve a mapping *E* associated with *r*. If the mapping is valid, it indicates that other triggers for *r* have been processed and stored in *E* before *meta*, so a non-empty set of meta-actions (which is denoted as *meta-action-set*) is retrieved along with its *mapped-rules* set from *E*; Otherwise, a new mapping

Algorithm 1 Hierarchically Organize Triggers Algorithm

Input: M: an advanced matrix containing action rules and their corresponding triggers.
Output: T: a list containing all generated metaNodes.

```
Generate meta-list: a list of meta-actions that are descendingly sorted by their popularity in M;
Generate rules: a set of all action rules in M;
Initialize MetaSets (meta-action-set, mapped-rules);
Initialize a list T for storing all generated metaNodes;
for meta ∈ meta-list do
  for r ∈ rules do
    if meta is one of r's triggers then
      retrieve entry E (meta-action-set, mapped-rules) for r ∈ mapped-rules from MetaSets;
      if E is valid then
        add a new entry E'(meta-action-set ∪ {meta}, {r}) to MetaSets;
        mapped-rules = mapped-rules \ {r};
        if mapped-rules is empty then
          remove entry E (meta-action-set, mapped-rules) from MetaSets;
        else
          keep entry E (meta-action-set, mapped-rules) in MetaSets;
        end if
      else
        retrieve entry E' ({meta}, mapped-rules);
        put entry E' ({meta}, mapped-rules ∪ {r}) to MetaSets;
      end if
      if meta-action-set ∪ {meta} trigger r completely then
        T = TreeEditor(meta, r, meta-action-set, T);
      end if
    end if
  end for
end for
Sort meta-nodes in T by their effect in a descending order.
return T;
```

from *meta* to *r* should be established instead. In terms of a valid mapping, it is defined as a mapping with a non-empty key (set of meta-actions) in our domain, so an empty entry will be found if a mapping is invalid. For the former situation with a valid existing mapping discovered, it is apparent that a new mapping E' to *r* has to be built due to the enlargement of its triggers, and the existing mapping *E* has to be updated accordingly to keep the distinctness of action rules. Building a new mapping in *MetaMap* by creating a new entry E' with *meta* added into the retrieved *meta-action-set* and mapped to *r* is straightforward, so is the removal of *r* from *mapped-rules* in the existing entry *E*. But there is no necessity of keeping *E* if its action rule set becomes empty after the removal, with regards to the definition of a valid mapping. Similarly, building a new entry E' with only $\{meta\}$ mapped to $\{r\}$ in *MetaMap* for the latter situation is simple as well. However, it is possible that a mapping from *meta* already exists. If this is the case, which implies another set of actions rules that can be triggered by *meta* have already been stored, then *r* should be added into the existing set of action rules mapped from *meta*, instead of creating a new mapping.

Algorithm 2 TreeEditor

Input: *meta*: a single meta-action.
r: an action rule.
meta-action-set: a set of meta-actions.
T: a tree storing the current added metaNodes and their relationship.
Output: *T*

```
retrieve metaNode N (meta-action-set ∪ {meta}, *) from T;
if N is valid then
    set N as (meta-action-set ∪ {meta}, * ∪{r});
    set effect(N) = effect(N) + sup(r) · conf(r);
    if N is someone's parent then
        update N's children's effect.
    end if
else
    set N as (meta-action-set ∪ {meta}, {r});
    set effect(N) = sup(r) · conf(r);
    add N into T;
    retrieve metaNode P (meta-action-set, *) in T;
    if P is valid then
        set P as N's parent;
        set effect(N) = effect(N) + effect(P);
    end if
end if
return T;
```

Every time a new mapping (or entry) E' is put into *MetaMap*, no matter if it is from an existing mapping or not, Algorithm 2 will be called on the condition that current action rule r is fulfilled. By r is fulfilled, we mean that the set of meta-actions in the new entry E' is all we need to trigger r according to advanced matrix M. Algorithm 2 is responsible for two aspects: constructing meta-nodes and building trees by linking meta-nodes regarding their relation, which requires relevant information including current meta-action *meta*, current action rule r, *meta-action-set* from E and of course T for storing meta-nodes. To construct a meta-node for a most recently fulfilled rule r, the first step is to check whether there is a meta-node in T which contains the same set of meta-actions as r does but fulfilling other rules. Since every meta-node can be seen as a mapping in *MetaMap*, the validity of a meta-node is evaluated in a similar way as we did for validating a mapping previously. As shown in Algorithm 2, if the meta-node N retrieved for the purpose of validation from T is non-empty, then r should be added into the action rule set along with other rules represented as "*" that have already been stored in N. Otherwise, it means that N is empty, so a new entry *(meta-action-set* $\cup \{meta\}$, $\{r\}$*)* should be assigned to N and added into T as a new meta-node. With a newborn meta-node N, the last but not least step is to set up the parent-child relationship through looking for its parent node. The parent node will not exist unless the mapping E' is extended from an existing mapping E in previous procedures in Algorithm 1, in other words, *meta-action-set* is not empty. Therefore, as long as the meta-node P retrieved by looking for a meta-node which has *meta-action-set* is valid, P is N's parent and N is

one of *P*'s children. On the other hand, it is unnecessary to set up the relationship for a newly updated meta-node because its parent must be found when it has been made.

Besides organizing sets of meta-nodes hierarchically as a tree, computing the expected effect of each meta-node during the process is another characteristic step in our strategy. It provides clients with concrete clues to evaluate the worth of those meta-actions. Since the effect of triggering a single action rule is defined as the product of its support and confidence, the effect of a meta-node is the summation of the effect of triggering all action rules in it. So whenever a new action rule is added into an existing meta-node, its effect must be added into the meta-node as well. Additionally, if this existing meta-node which has just been updated has any children, its children's effect need to be updated as well. And for a newborn meta-node, besides its own influence computed on the basis of our regulations, its effect is strengthened with its parent's help unless its parent does not exist.

Algorithm 1 will sort the meta-nodes in *T* by their effect in a descending order and return it after all the meta-actions in *meta-list* have been processed. An example of printed result will be shown in the next section.

5 Experiment

To test its performance, experiments are carried out with the JAVA based implementation of the proposed algorithm. Firstly, we take a small sample of data as an example to show the procedures in Algorithm 1 and the representation of final results. Table 3 is the advanced matrix in our example, and there are six action rules and four meta-actions involved in the sample. These four meta-actions are descendingly sorted by their popularity in *meta-list* as $\{M_3, M_4, M_2, M_1\}$ and they will be checked one by one in such order. Hence, with M_3 as the most frequent meta-action in the list, all the action rules associated with it will be stored in an entry in *MetaMap* at the first round, which is $(\{M_3\}, \{r_1, r_2, r_3, r_4, r_6\})$. There is no action rule that has been fulfilled, so no meta-node will be made yet. When it comes to M_4, the action rules involving it are handled in different ways. Action rules that have already been stored in *MetaMap*, like r_1, r_2, r_3 and r_6, should be moved to another mapping with $\{M_3, M_4\}$, and the previous mapping becomes $(\{M_3\}, \{r_4\})$. For action rules that are new to *MetaMap*, a new entry is built and it is $(\{M_4\}, \{r_5\})$ in this example. In terms of building meta-nodes, when iterating action rules top down, we would find that r_3 is fulfilled with $\{M_3, M_4\}$. Obviously a meta-node $(\{M_3, M_4\}, \{r_3\})$ should be built and its parent who is suppose to be $(\{M_3\}$, *) doesn't exist, so no parent is affiliated and its effect is $sup(r_3) \cdot conf(r_3)$. The same meta-actions trigger r_6 as well, so the meta-node becomes $(\{M_3, M_4\}, \{r_3, r_6\})$, and its effect is updated as $sup(r_3) \cdot conf(r_3) + sup(r_6) \cdot conf(r_6)$. We should take care of its children's effect as well, but there isn't any, so nothing needs to be done. As all the action rules are appeared in *MetaMap* now, we focus on creating new mappings from existing ones and updating the existing mappings by following the requirements. Along with the same procedures being performed to the rest meta-actions, more meta-nodes will be built, such as $(\{M_3, M_4, M_2\}, \{r_1\})$, $(\{M_2, M_4\}, \{r_4\})$ and so on. Whenever a new meta-node is made, it is necessary to look for its parent. For instance, meta-node $(\{M_3, M_4, M_2\}, \{r_1\})$ is a mapping extended from $\{M_3, M_4\}$, and meta-node

($\{M_3, M_4\}, \{r_3, r_6\}$) does exist, thus these two meta-nodes should be connected with a parent-child relationship as the former is a child of the latter. Meanwhile, the effect of the child's node is its own effect plus its parent's effect.

Table 3. Advanced Matrix for the Sample Data

	M_1	M_2	M_3	M_4
r_1	0	1	1	1
r_2	1	0	1	1
r_3	0	0	1	1
r_4	0	1	0	1
r_5	1	1	1	0
r_6	0	0	1	1

When the algorithm comes to the end, a list of meta-nodes will be printed as Fig. 1 shows. There are five meta-nodes generated and they are listed in an descending order according to their effect, which is basically the priority that clients follow to judge them. Similar as running the program with this sample, we run it with a larger set consisting of 127 action rules and 627 meta-actions, and the program sorts the result in a format as shown in Fig 1, which lists 78 meta-nodes, their relations and their effect.

```
meta-node0= ({m3, m4, m2}, {r1}), its effect is 2+6.15=8.15
meta-node0's parent is ({m3, m4}, {r6, r3})
meta-node1= ({m3, m4, m1}, {r2}), its effect is 1.8+6.15=7.95
meta-node1's parent is ({m3, m4}, {r6, r3})
meta-node2= ({m3, m4}, {r6, r3}), its effect is 3.6+2.55=6.15
meta-node2's parent is not valid
meta-node2's children are: ({m3, m4, m2}, {r1})
meta-node2's children are: ({m3, m4, m1}, {r2})
meta-node3= ({m3, m2, m1}, {r5}), its effect is 4.35
meta-node3's parent is not valid
meta-node4= ({m4, m2}, {r4}), its effect is 1.6.
meta-node4's parent is not valid
```

Fig. 1. Experiment Result with Given Example

6 Conclusion

From the example result shown in Fig.1, it is clear that the strategy helps clients select actions of top priority by considering the predicted effect as a partial reason for the

final decision. What's more, the experiment with larger dataset proves the efficiency of organizing the meta-actions in the proposed way. It is justified by the number of generated meta-nodes which is much smaller than the number of action rules or sets of meta-actions triggering them (it is over 30% off the original choices within clients' evaluation). On the other hand, the effect of applying sets of meta-actions is calculated more accurately with the definition of parent-child relationship. Therefore, this strategy provides a more effective way for clients to choose the right set of meta-actions out of the generated suggestions, and conclusively fits our anticipation.

References

1. A. Hajja, Z.W. Ras, A. Wieczorkowska (2014) Hierarchical object-driven action rules, Journal of Intelligent Information Systems, Springer, Vol. 42, No. 2, 2014, pp. 207-232
2. Z. He, X. Xu, S. Deng, R. Ma (2005) Mining action rules from scratch, in Expert Systems with Applications, Vol. 29, No. 3, Elsevier, pp. 691-699
3. J. Kuang, A. Daniel, J. Johnston, Z.W. Ras. (2014) Hierarchically Structured Recommender System for Improving NPS of a Company. Proceedings of 9th International Conference, RSCTC 2014, Granada and Madrid, Spain, LNAI, Vol. 8526, Springer, pp. 347-357.
4. J. Kuang, Z.W. Ras, A. Daniel (2015) Hierarchical agglomerative method for improving NPS. Proceedings of the International Conference on Pattern Recognition and Machine Intelligence, LNCS, Vol. 9124, Springer, 2015, pp. 54-64
5. R. Paul, A.S. Hoque (2010) Mining irregular association rules based on action and non-action type data, in Proceedings of the Fifth International Conference on Digital Information Management (ICDIM), pp. 63-68
6. Z.W. Ras, A. Wieczorkowska (2000) Action-rules: how to increase profit of a company, in Principles of Data Mining and Knowledge Discovery, (Eds. D.A. Zighed, J. Komorowski, J. Zytkow), Proceedings of PKDD'00, Lyon, France, LNAI, No. 1910, Springer, pp. 587-592
7. Z. Ras, A. Dardzinska (2011) From Data to Classification Rules and Actions, in Proceedings of the Joint Rough Sets Symposium (JRS07), LNAI, Vol. 4482, Springer, pp. 322-329
8. J. Rauch, M. Simunek (2009) Action rules and the GUHA method: Preliminary con- siderations and results, in Proceedings of ISMIS 2009, LNAI 5722, Springer, pp. 76-87
9. F.F. Reichheld (2003) The one number you need to grow, in Harvard Business Review, December 2003, pp. 1-8
10. H. Touati, J. Kuang, A. Hajja and Z. Ras (2013) Personalized Action Rules for Side Effects Object Grouping. International Journal of Intelligence Science, Vol. 3 No. 1A, pp. 24-33
11. A. Tzacheva, Z.W. Ras (2010) Association action rules and action paths triggered by meta-actions, in Proceedings of 2010 IEEE Conference on Granular Computing, Silicon Valley, CA, IEEE Computer Society, pp. 772-776