Tree-based construction of low-cost action rules

Angelina A. Tzacheva[1] and Li-Shiang Tsay[2]

[1] University of South Carolina Upstate, Informatics Dept.
Spartanburg, SC 29303; ATzacheva@uscupstate.edu
[2] Hampton University, Computer Science Dept., Hampton, VA 23668;
ltsay@uncc.edu

Summary. A rule as actionable, if a user can do an action to his/her advantage based on that rule. Actionability can be expressed in terms of attributes that are present in a database. Action rules are constructed from certain pairs of classification rules previously extracted from the same decision table. Each such classification rule defines a preferable decision class. Attributes in a decision table are divided into two groups: stable and flexible. Flexible attributes provide a tool for making hints to a user to what changes within some values of flexible attributes are needed to reclassify a group of objects, supporting the action rule, from one decision class to another, more desirable, one. Changes of values of some flexible attributes can be more expensive than changes of other values. The notion of a cost can be introduced and assigned by an expert to each such a change.

Action rules involve flexible attributes and stable attributes listed in both classification rules from which they are constructed. The values of stable attributes listed in them are used to create action forest.

We propose a new approach, which combines the action forest algorithm of extracting action rules, and a heuristic strategy to lowest cost reclassification of objects. It presents an enhancement to both methods.

1 Introduction

Since knowledge discovery systems extract large amounts of knowledge, an essential problem in the field is measuring the interestingness of discovered patterns. Such measures of interestingness are divided into objective measures, which are domain independent, and the subjective measures - those that depend on the class of users who examine the pattern and their belief about the domain. The subjective measures include unexpectedness, novelty and actionability [16].

Silberschatz and Tuzlin [15] define actionability in terms of unexpect

Ras and Wieczorkowska [11] assume that actionability can be expressed in terms of attributes that are present in the database. They introduce a new

class of rules, called action rules that are constructed from certain pairs of association rules extracted from that database. A conceptually similar definition of an action rule was proposed independently by Geffner and Wainer [4]. Action rules have been investigated further in Ras and Gupta [13], Ras and Tsay [12, 16], and Tzacheva and Ras [17].

In order to construct action rules, it is required that attributes in a database are divided into two groups: stable and flexible. Flexible attributes are used in a decision rule as a tool for making hints to a user what changes within some of their values are needed to reclassify a group of objects from one decision class into another one. Previous strategies for generating action rules are the system DEAR [12], which generates action rules from certain pairs of association rules, and the system DEAR2 [16], which is based on a tree structure - Action Forest, that partitions the set of rules, having the same decision value, into equivalence classes each labeled by values of stable attributes (two rules belong to the same equivalence class, if values of their stable attributes are not conflicting each other). Now, instead of comparing all pairs of rules, only pairs of rules belonging to some of these equivalence classes are compared to construct action rules. This strategy significantly reduces the number of steps needed to generate action rules in comparison to DEAR system.

The action rules discovery strategy described above, would allow for flexible attributes to provide a tool for making hints to a user to what changes within some values of flexible attributes are needed to re-classify a group of objects, supporting the action rule, from one decision class to another, more desirable, one. However, changes of values of some flexible attributes can be more expensive, or more difficult to implement, than changes of other values which may be trivial. Therefore, the notion of a cost can be introduced and assigned by an expert to each such a change.

Tzacheva and Ras [17] introduce the notion of a cost and feasibility of an action rule. The average cost needed to change the attribute value from one class to another, more desirable one is a value between $(0, +\infty)$, where if the change is not feasible in practice the cost will be close to $+\infty$, whereas if the change is quite trivial to accomplish the value will be close to 0. Tzacheva and Ras [17] also propose a heuristic strategy for constructing feasible action rules which have high confidence and the lowest cost - search graph for lowest cost reclassification of objects. The search graph is a directed graph G which is dynamically built by applying action rules to its nodes. A node n in G show an alternative way to achieve the same reclassification with a cost that is lower than the cost assigned to all nodes which are preceding n in G. The leaves of the graph would show the most actionable, and therefore most interesting knowledge which has the lowest cost, and meets the feasibility and confidence requirements (thresholds) of the user.

In this paper we propose a new approach, which combines the action forest algorithm of extracting action rules, and a heuristic strategy to lowest cost reclassification of objects to present an enhancement to both methods.

2 Action-Forest algorithm and Experiment

In this section we present a new algorithm called Action-Forest algorithm for discovering E-action rules. All previously discovered classification rules are partitioned into decision classes. Namely, we place all rules defining the same decision value into the same decision class. For each decision value, a tree is built with non-leaf nodes being labeled by stable attributes and outgoing edges by values of these attributes. If a node is a non-leaf node, then the set of rules associated with that node is partitioned along the branches and each child node gets its corresponding subset of rules. Each leaf represents a set of rules which do not contradict on stable attributes and also define the same decision value. The path from the root to that leaf gives the description of objects supported by these rules. To construct extended action rules we compare pairs of rules each belonging to a different leaf node. Additionally, we assume here that both paths leading to the corresponding leaf nodes do not have contradictory descriptions.

Initially, we partition the set of rules discovered from an information system $S = (U, A_{St} \cup A_{Fl} \cup \{d\})$, where A_{St} is the set of stable attributes, A_{Fl} is the set of flexible attributes and, $V_d = \{d_1, d_2, ..., d_k\}$ is the set of decision values, into subsets of rules defining the same decision value. In other words, the set of rules R discovered from S is partitioned into $\{R_i\}_{i:1 \leq i \leq k}$, where $R_i = \{r \in R : d(r) = d_i\}$ for any $i = 1, 2, ..., k$. Clearly, the objects supporting any rule from R_i form subsets of $d^{-1}(\{d_i\})$.

Let us take Table 1 as an example of a decision system S. We assume that $\{a, c\}$ are stable attributes and $\{b, d\}$ are flexible. The set R of certain rules extracted from S is given below:

$$(a,0) \rightarrow (d,L),\ (c,0) \rightarrow (d,L),\ (b,2) \rightarrow (d,L),$$
$$(a,2) * (b,1) \rightarrow (d,H),\ (b,1) * (c,2) \rightarrow (d,H),$$
$$(c,1) \rightarrow (d,L),\ (b,3) \rightarrow (d,L).$$

We partition this set into two subsets $R_1 = \{[(a,0) \rightarrow (d,L)], [(c,0) \rightarrow (d,L)], [(b,2) \rightarrow (d,L)], [(c,1) \rightarrow (d,L)], [(b,3) \rightarrow (d,L)]\}$, and $R_2 = \{[(a,2) * (b,1) \rightarrow (d,H)], [(b,1) * (c,2) \rightarrow (d,H)]\}$.

Assume now that our goal is to re-classify some objects from the class $d^{-1}(\{d_i\})$ into the class $d^{-1}(\{d_j\})$. In our example, we assume that $d_i = (d,L)$ and $d_j = (d,H)$.

First, we represent the set R as a table (see Table 1). The first column of this table shows objects in S supporting the rules from R (each row represents a rule). The first 5 rows represent the set R_1 and the last two rows represent the set R_2. In general case, assumed earlier, the number of different decision classes is equal to k.

Table 1. Set of rules R with supporting objects extracted from decision system S

	a	b	c	d
$\{x_1, x_2, x_3, x_4\}$	0			L
$\{x_2, x_4\}$		2		L
$\{x_1, x_3\}$			0	L
$\{x_2, x_4\}$			1	L
$\{x_5, x_6\}$		3		L
$\{x_7, x_8\}$	2	1		H
$\{x_7, x_8\}$		1	2	H

The next step of the algorithm is to build d_i-tree and d_j-tree. First, from the initial table similar to Table 1, we select all rules (rows) defining the decision value d_i and present them as a table similar to Table 2. Next, it is seeking at each stage for a stable attribute that has the least amount of values; the set of rules is split using that attribute and then the subsets that result from the split are recursively processed. When all stable attributes are processed, we build d_i tree. Similarly, from the same table (Table 1), we also select all rules (rows) which define decision value d_j and present them as a table similar to Table 3. The same strategy is used to build d_j tree for the decision value d_j .

Table 2. Set of rules R_1 with decision value L

	a	b	c	d
$\{x_1, x_2, x_3, x_4\}$	0			L
$\{x_2, x_4\}$		2		L
$\{x_1, x_3\}$			0	L
$\{x_2, x_4\}$			1	L
$\{x_5, x_6\}$		3		L

By d_i-tree we mean a tree $T(d_i) = (N_i, E_i)$, such that:

- each interior node is labeled by a stable attribute from A_{St},
- each edge is labeled either by a question mark or by an attribute value of the attribute that labels the initial node of the edge,

Table 3. Set of rules R_j with decision value H

	a	b	c	d
$\{x_7, x_8\}$	2	1		H
$\{x_7, x_8\}$		1	2	H

- along a path, all nodes (except a leaf) are labeled with different stable attributes,
- all edges leaving a node are labeled with different attribute values (including the question mark) of the stable attribute that labels that node,
- each leaf represents a set of rules which do not contradict on stable attributes and also define decision value d_i. The path from the root to that leaf gives the description of objects supported by these rules.

Now, taking (d, L) from our example as the value d_i, we show how to construct (d, L)-tree for the set of rules represented by Table 2. The construction of (d, L)-tree starts with a table corresponding to the root of that tree (Table 2 in Fig. 1). It represents the set of rules R_1 defining L with supporting objects from S. The domain of attribute a is $\{0\}$ and the domain of attribute c is $\{0, 1\}$. Clearly, card$[V_a]$is less than card$[V_c]$, so we use stable attribute a to split that table into 2 sub-tables defined by values $\{0, ?\}$ of attribute a. The question mark means an unknown value.

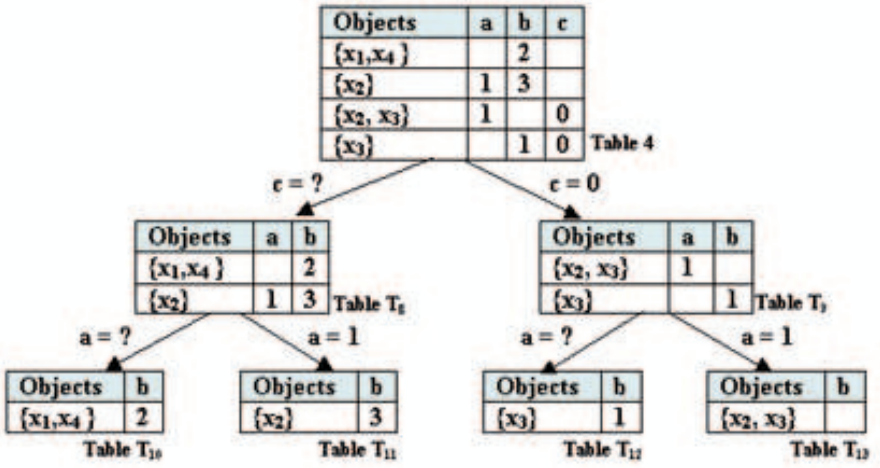

Fig. 1. (d, L)-tree

Following the path labeled by value $[a =?]$ and $[a = 0]$, we get table T_1 and table T_2, respectively. When we follow the path labeled by value $[a =?][c =?]$,

we get table T_3. Following the path labeled by value $[a =?][c = 0]$, we get table T_4. When we follow the path labeled by value $[a =?][c = 1]$, we get table T_5. Finally, by following the path having the label $[a = 0][c =?]$, we get table T_6.

Now, let us define (d, H)-tree using Table 3 as its root (see Fig. 2). Following the path labeled by value $[a =?]$ and $[c = 2]$, we get the table T_7 and the table T_8, respectively. Following the path labeled by value $[a =?][c = 2]$, we get the table T_9. When we follow the path labeled by value $[a = 2][c =?]$, we get the table T_{10}.

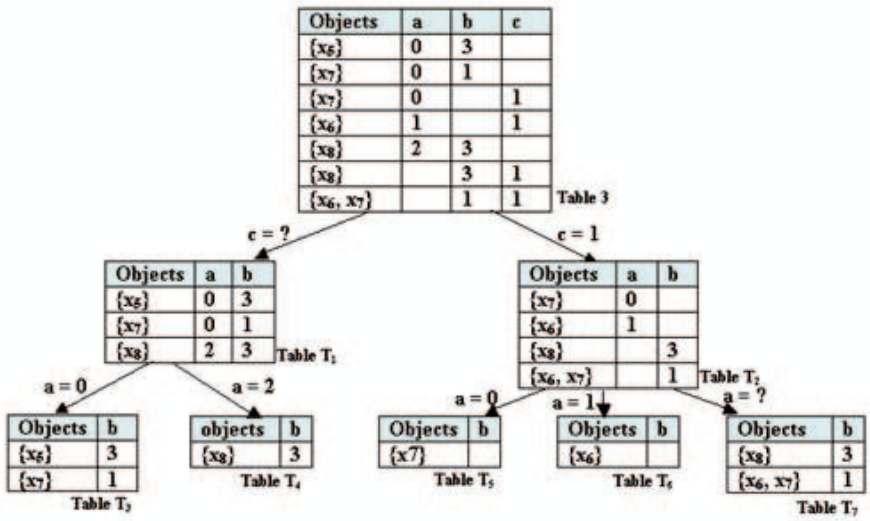

Fig. 2. (d, H)-tree

For tables T_4, T_5, T_6, the attribute b is the only flexible attribute we have and its value is arbitrary. Therefore, we cannot use these tables to construct E-action rules. Now, it can be easily checked that only pairs of rules belonging to tables $\{[T_3, T_9]$ and $[T_3, T_{10}]$ can be used for E-action rules construction.

By comparing pairs of rules belonging to tables T_3 and T_9 we generate two E-action rules:

$[[(c = 2) \wedge (b, 2 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$, and
$[[(c = 2) \wedge (b, 3 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$.

By comparing pairs of rules belonging to tables T_3 and T_{10} we generate two E-action rules:

$[[(a, 2) \wedge (b, 2 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$, and
$[[(a, 2) \wedge (b, 3 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$.

After the rules are formed, we evaluate them by checking their support and confidence. Two strong E-action rules are discovered:

$[[(a,2) \wedge (b,3 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$, sup:2, conf: 100%
$[[(c,2) \wedge (b,3 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$, sup:2 conf: 100%

3 Cost and Feasibility of Action Rules

Assume now that $DS = (\{S_i : i \in I\}, L)$ is a distributed information system (*DIS*), where $S_i = (X_i, A_i, V_i)$, $i \in I$. Let $b \in A_i$ is a flexible attribute in S_i and $b_1, b_2 \in V_i$ are its two values. By $\rho_{S_i}(b_1, b_2)$ we mean a number from $(0, +\infty]$ which describes the average cost needed to change the attribute value from b_1 to b_2 for any of the qualifying objects in S_i. Object $x \in X_i$ qualifies for the change from b_1 to b_2, if $b(x) = b_1$. If the above change is not feasible in practice, for one of the qualifying objects in S_i, then we write $\rho_{S_i}(b_1, b_2) = +\infty$. The value of $\rho_{S_i}(b_1, b_2)$ close to *zero* is interpreted that the change of values from b_1 to b_2 is quite trivial to accomplish for qualifying objects in S_i whereas any large value of $\rho_{S_i}(b_1, b_2)$ means that this change of values is practically very difficult to achieve for some of the qualifying objects in S_i.

If $\rho_{S_i}(b_1, b_2) < \rho_{S_i}(b_3, b_4)$, then we say that the change of values from b_1 to b_2 is *more feasible* than the change from b_3 to b_4.

We assume here that the values $\rho_{S_i}(b_{j1}, b_{j2})$ are provided by experts for each of the information systems S_i. They are seen as atomic expressions and will be used to introduce the formal notion of the feasibility and the cost of action rules in S_i.

So, let us assume that $r = [(b_1, v_1 \rightarrow w_1) \wedge (b_2, v_2 \rightarrow w_2) \wedge \ \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$ is a (r_1, r_2)-action rule. By the *cost* of r denoted by $cost(r)$ we mean the value $\sum\{\rho_{S_i}(v_k, w_k) : 1 \leq k \leq p\}$. We say that r is *feasible* if $cost(r) < \rho_{S_i}(k_1, k_2)$.

It means that for any feasible rule r, the cost of the conditional part of r is lower than the cost of its decision part and clearly $cost(r) < +\infty$.

Assume now that d is a decision attribute in S_i, $k_1, k_2 \in V_d$, and the user would like to re-classify customers in S_i from the group k_1 to the group k_2. To achieve that, he may look for an appropriate action rule, possibly of the lowest cost value, to get a hint which attribute values have to be changed. To be more precise, let us assume that $R_{S_i}[(d, k_1 \rightarrow k_2)]$ denotes the set of all action rules in S_i having the term $(d, k_1 \rightarrow k_2)$ on their decision site. For simplicity reason, in Section 5 of this paper, attribute d will be omitted in $(d, k_1 \rightarrow k_2)$. Now, among all action rules in $R_{S_i}[(d, k_1 \rightarrow k_2)]$ he may identify a rule which has the lowest cost value. But the rule he gets may still have the cost value much to high to be of any help to him. Let us notice that the cost of the action rule

$r = [(b_1, v_1 \rightarrow w_1) \wedge (b_2, v_2 \rightarrow w_2) \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$ might be high only because of the high cost value of one of its sub-terms in the conditional part of the rule.

Let us assume that $(b_j, v_j \rightarrow w_j)$ is that term. In such a case, we may look for an action rule in $R_{S_i}[(b_j, v_j \rightarrow w_j)]$ which has the smallest cost value.

Assume that $r_1 = [(b_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (b_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (b_{jq}, v_{jq} \rightarrow w_{jq})](y) \Rightarrow (b_j, v_j \rightarrow w_j)(y)$ is such a rule which is also feasible in S_i. Since $x, y \in X_i$, we can compose r with r_1 getting a new feasible rule which is given below:

$[(b_1, v_1 \rightarrow w_1) \wedge ... \wedge [(b_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (b_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (b_{jq}, v_{jq} \rightarrow w_{jq})] \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$.

Clearly, the cost of this new rule is lower than the cost of r. However, if its support in S_i gets too low, then such a rule has no value to the user. Otherwise, we may recursively follow this strategy trying to lower the cost needed to reclassify objects from the group k_1 into the group k_2. Each successful step will produce a new action rule which cost is lower than the cost of the current rule. Obviously, this heuristic strategy always ends.

One can argue that if the set $R_{S_i}[(d, k_1 \rightarrow k_2)]$ contains all action rules reclassifying objects from group k_1 into the group k_2 then any new action rule, obtained as the result of the above recursive strategy, should be already in that set. We agree with this statement but in practice $R_{S_i}[(d, k_1 \rightarrow k_2)]$ is only a subset of all action rules. Firstly, it is too expensive to generate all possible classification rules from an information system (available knowledge discovery packages extract only the shortest or close to the shortest rules) and secondly even if we extract such rules it still takes too much time to generate all possible action rules from them. So the applicability of the proposed recursive strategy, to search for new rules possibly of the lowest cost, is highly justified.

Again, let us assume that the user would like to reclassify some objects in S_i from the class b_1 to the class b_2 and that $\rho_{S_i}(b_1, b_2)$ is the current cost to do that. Each action rule in $R_{S_i}[(d, k_1 \rightarrow k_2)]$ gives us an alternate way to achieve the same result but under different costs. If we limit ourself to the system S_i, then clearly we can not go beyond the set $R_{S_i}[(d, k_1 \rightarrow k_2)]$. But, if we allow to extract action rules at other information systems and use them jointly with local action rules, then the number of attributes which can be involved in reclassifying objects in S_i will increase and the same we may further lower the cost of the desired reclassification.

So, let us assume the following scenario. The action rule $r = [(b_1, v_1 \rightarrow w_1) \wedge (b_2, v_2 \rightarrow w_2) \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$, extracted from the information system S_i, is not feasible because at least one of its terms, let us say $(b_j, v_j \rightarrow w_j)$ where $1 \leq j \leq p$, has too high cost $\rho_{S_i}(v_j, w_j)$ assign to it.

In this case we look for a new feasible action rule $r_1 = [(b_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (b_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (b_{jq}, v_{jq} \rightarrow w_{jq})](y) \Rightarrow (b_j, v_j \rightarrow w_j)(y)$ which concatenated with r will decrease the cost value of desired reclassification. So, the current setting looks the same to the one we already had except that this time we additionally assume that r_1 is extracted from another information system in DS. For simplicity reason, we also assume that the semantics and the granularity levels of all attributes listed in both information systems are the same.

By the concatenation of action rule r_1 with action rule r we mean a new feasible action rule $r_1 \circ r$ of the form:
$[(b_1, v_1 \rightarrow w_1) \wedge ... \wedge [(b_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (b_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (b_{jq}, v_{jq} \rightarrow w_{jq})] \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$
where x is an object in $S_i = (X_i, A_i, V_i)$.

4 Heuristic Strategy for the Lowest Cost Reclassification of Objects

Let us assume that we wish to reclassify as many objects as possible in the system S_i, which is a part of DIS, from the class described by value k_1 of the attribute d to the class k_2. The reclassification $k_1 \rightarrow k_2$ jointly with its cost $\rho_{S_i}(k_1, k_2)$ is seen as the information stored in the initial node n_0 of the search graph built from nodes generated recursively by feasible action rules taken initially from $R_{S_i}[(d, k_1 \rightarrow k_2)]$. For instance, the rule

$r = [(b_1, v_1 \rightarrow w_1) \wedge (b_2, v_2 \rightarrow w_2) \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$

applied to the node $n_0 = \{[k_1 \rightarrow k_2, \rho_{S_i}(k_1, k_2)]\}$ generates the node

$n_1 = \{[v_1 \rightarrow w_1, \rho_{S_i}(v_1, w_1)], [v_2 \rightarrow w_2, \rho_{S_i}(v_2, w_2)], ..., [v_p \rightarrow w_p, \rho_{S_i}(v_p, w_p)]\}$,

and from n_1 we can generate the node

$n_2 = \{[v_1 \rightarrow w_1, \rho_{S_i}(v_1, w_1)], [v_2 \rightarrow w_2, \rho_{S_i}(v_2, w_2)], ..., [v_{j1} \rightarrow w_{j1}, \rho_{S_i}(v_{j1}, w_{j1})], [v_{j2} \rightarrow w_{j2}, \rho_{S_i}(v_{j2}, w_{j2})], ..., [v_{jq} \rightarrow w_{jq}, \rho_{S_i}(v_{jq}, w_{jq})], ..., [v_p \rightarrow w_p, \rho_{S_i}(v_p, w_p)]\}$

assuming that the action rule

$r_1 = [(b_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (b_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (b_{jq}, v_{jq} \rightarrow w_{jq})](y) \Rightarrow (b_j, v_j \rightarrow w_j)(y)$

from $R_{S_m}[(b_j, v_j \rightarrow w_j)]$ is applied to n_1. /see [17]/

This information can be written equivalently as: $r(n_0) = n_1$, $r_1(n_1) = n_2$, $[r_1 \circ r](n_0) = n_2$. Also, we should notice here that r_1 is extracted from S_m and $Sup_{S_m}(r_1) \subseteq X_m$ whereas r is extracted from S_i and $Sup_{S_i}(r) \subseteq X_i$.

By $Sup_{S_i}(r)$ we mean the domain of action rule r (set of objects in S_i supporting r).

The search graph can be seen as a directed graph G which is dynamically built by applying action rules to its nodes. The initial node n_0 of the graph

G contains information coming from the user, associated with the system S_i, about what objects in X_i he would like to reclassify and how and what is his current cost of this reclassification. Any other node n in G shows an alternative way to achieve the same reclassification with a cost that is lower than the cost assigned to all nodes which are preceding n in G. Clearly, the confidence of action rules labeling the path from the initial node to the node n is as much important as the information about reclassification and its cost stored in node n. Information from what sites in DIS these action rules have been extracted and how similar the objects at these sites are to the objects in S_i is important as well.

Information stored at the node
$\{[v_1 \rightarrow w_1, \rho_{S_i}(v_1, w_1)], [v_2 \rightarrow w_2, \rho_{S_i}(v_2, w_2)], ..., [v_p \rightarrow w_p, \rho_{S_i}(v_p, w_p)]\}$
says that by reclassifying any object x supported by rule r from the class v_i to the class w_i, for any $i \leq p$, we also reclassify that object from the class k_1 to k_2. The confidence in the reclassification of x supported by node $\{[v_1 \rightarrow w_1, \rho_{S_i}(v_1, w_1)], [v_2 \rightarrow w_2, \rho_{S_i}(v_2, w_2)], ..., [v_p \rightarrow w_p, \rho_{S_i}(v_p, w_p)]\}$ is the same as the confidence of the rule r.

Before we give a heuristic strategy for identifying a node in G, built for a desired reclassification of objects in S_i, with a cost possibly the lowest among all the nodes reachable from the node n_0, we have to introduce additional notations.

So, assume that N is the set of nodes in our dynamically built directed graph G and n_0 is its initial node. For any node $n \in N$, by $f(n) = (Y_n, \{[v_{n,j} \rightarrow w_{n,j}, \rho_{S_i}(v_{n,j}, w_{n,j})]\}_{j \in I_n})$ we mean its domain, the reclassification steps related to objects in X_i, and their cost, all assigned by *reclassification function* f to the node n, where $Y_n \subseteq X_i$ /Graph G is built for the client site S_i/.

Let us assume that $f(n) = (Y_n, \{[v_{n,k} \rightarrow w_{n,k}, \rho_{S_i}(v_{n,k}, w_{n,k})]\}_{k \in I_n})$. We say that action rule r, extracted from S_i, is applicable to the node n if:

- $Y_n \cap Sup_{S_i}(r) \neq \emptyset$,
- $(\exists k \in I_n)[r \in R_{S_i}[v_{n,k} \rightarrow w_{n,k}]]$. /see [17] for definition of $R_{S_i}[...]$/

Similarly, we say that action rule r, extracted from S_m, is applicable to the node n if:

- $(\exists x \in Y_n)(\exists y \in Sup_{S_m}(r))[\rho(x, y) \leq \lambda]$, /$\rho(x, y)$ is the similarity relation between x, y (see [17] for its definition) and λ is a given similarity threshold/
- $(\exists k \in I_n)[r \in R_{S_m}[v_{n,k} \rightarrow w_{n,k}]]$.

It has to be noticed that reclassification of objects assigned to a node of G may refer to attributes which are not necessarily attributes listed in S_i. In this case, the user associated with S_i has to decide what is the cost of such a reclassification at his site, since such a cost may differ from site to site.

Now, let $RA(n)$ be the set of all action rules applicable to the node n. We say that the node n is completely covered by action rules from $RA(n)$ if $X_n = \bigcup\{Sup_{S_i}(r) : r \in RA(n)\}$. Otherwise, we say that n is partially covered by action rules.

What about calculating the domain Y_n of node n in the graph G constructed for the system S_i? The reclassification $(d, k_1 \rightarrow k_2)$ jointly with its cost $\rho_{S_i}(k_1, k_2)$ is stored in the initial node n_0 of the search graph G. Its domain Y_0 is defined as the set-theoretical union of domains of feasible action rules in $R_{S_i}[(d, k_1 \rightarrow k_2)]$ applied to X_i. This domain still can be extended by any object $x \in X_i$ if the following condition holds:

$(\exists m)(\exists r \in R_{S_m}[k_1 \rightarrow k_2])(\exists y \in Sup_{S_m}(r))[\rho(x, y) \leq \lambda]$.

Each rule applied to the node n_0 generates a new node in G which domain is calculated in a similar way to n_0. To be more precise, assume that n is such a node and $f(n) = (Y_n, \{[v_{n,k} \rightarrow w_{n,k}, \rho_{S_i}(v_{n,k}, w_{n,k})]\}_{k \in I_n})$. Its domain Y_n is defined as the set-theoretical union of domains of feasible action rules in $\bigcup\{R_{S_i}[v_{n,k} \rightarrow w_{n,k}] : k \in I_n\}$ applied to X_i. Similarly to n_0, this domain still can be extended by any object $x \in X_i$ if the following condition holds:

$(\exists m)(\exists k \in I_n)(\exists r \in R_{S_m}[v_{n,k} \rightarrow w_{n,k}])(\exists y \in Sup_{S_m}(r))[\rho(x, y) \leq \lambda]$.

Clearly, for all other nodes, dynamically generated in G, the definition of their domains is the same as the one above.

5 Tree-based construction of low-cost action rules

In this section we show how to combine the Action-Forest algorithm for discovering E-action rules, and the heuristic strategy for lowest cost reclassification of objects to produce low-cost action rules using a tree-based construction. By adding the Action-Forest algorithm to the heuristic strategy for lowest cost reclassification we propose a new enhanced algorithm.

Let us assume that a rule

$$r = [(b_1, v_1 \rightarrow w_1) \wedge (b_2, v_2 \rightarrow w_2) \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$$

was extracted using the Action-Forest algorithm by creating trees for the values of $d - k_1, k_2, ..., k_j$. However, the user noticed that the cost of the conditional sub-term $(b_2, v_2 \rightarrow w_2)$ is too high. In other words, The action rule r, extracted from the information system S_i, is not feasible because at least one of its terms, let's say $(b_2, v_2 \rightarrow w_2)$ has too high cost $\rho_{S_b}(v_2, w_2)$ assigned to it.

In such case we may look for a new action rule in $R_{S_i}[(b, v_2 \rightarrow w_2)]$ which has the smallest cost value, which concatenated with r will decrease the cost

value of desired reclassification. Therefore, we can apply Action-Forest algorithm again, this time creating trees for the values of $b_2 - v_2$ and w_2, where b_2 will be our decision attribute. Assume the following feasible action rule was generated:

$$r_1 = [(e_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (e_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (e_{jq}, v_{jq} \rightarrow w_{jq})] \Rightarrow (b_2, v_2 \rightarrow w_2)$$

We can then compose r with r_1, i.e. $r_1 \circ r$ getting a new feasible rule as follows:

$$[(b_1, v_1 \rightarrow w_1) \wedge [(e_{j1}, v_{j1} \rightarrow w_{j1}) \wedge (e_{j2}, v_{j2} \rightarrow w_{j2}) \wedge ... \wedge (e_{jq}, v_{jq} \rightarrow w_{jq})] \wedge ... \wedge (b_p, v_p \rightarrow w_p)](x) \Rightarrow (d, k_1 \rightarrow k_2)(x)$$

We may recursively follow this strategy trying to lower the cost needed to re-classify objects from the group k_1 into the group k_2. Each successful step will produce a new action rule which cost is lower than the cost of the current rule.

We recall that the search graph can be seen as a directed graph G which is dynamically built by applying action rules to its nodes. The initial node n_0 of the graph G contains information coming from the user, associated with the system S_i, about what objects in X_i he or she would like to reclassify, and what is the current cost of this reclassification. Any other node n in G shows an alternative way to achieve the same reclassification with a cost that is lower than the cost assigned to all nodes which are preceding n in G.

For instance, if we take as an example the strong E-action rule r_1 generated in Section 2 :

$[[(a, 2) \wedge (b, 3 \rightarrow 1)] \Rightarrow (d, L \rightarrow H)]$

and the user identified that this rule is not feasible because the cost of $\rho_{S_b}(3, 1)$ of sub-term $(b, 3 \rightarrow 1)$, the tree-based construction of low-cost action rules is illustrated on Fig. 3. below:

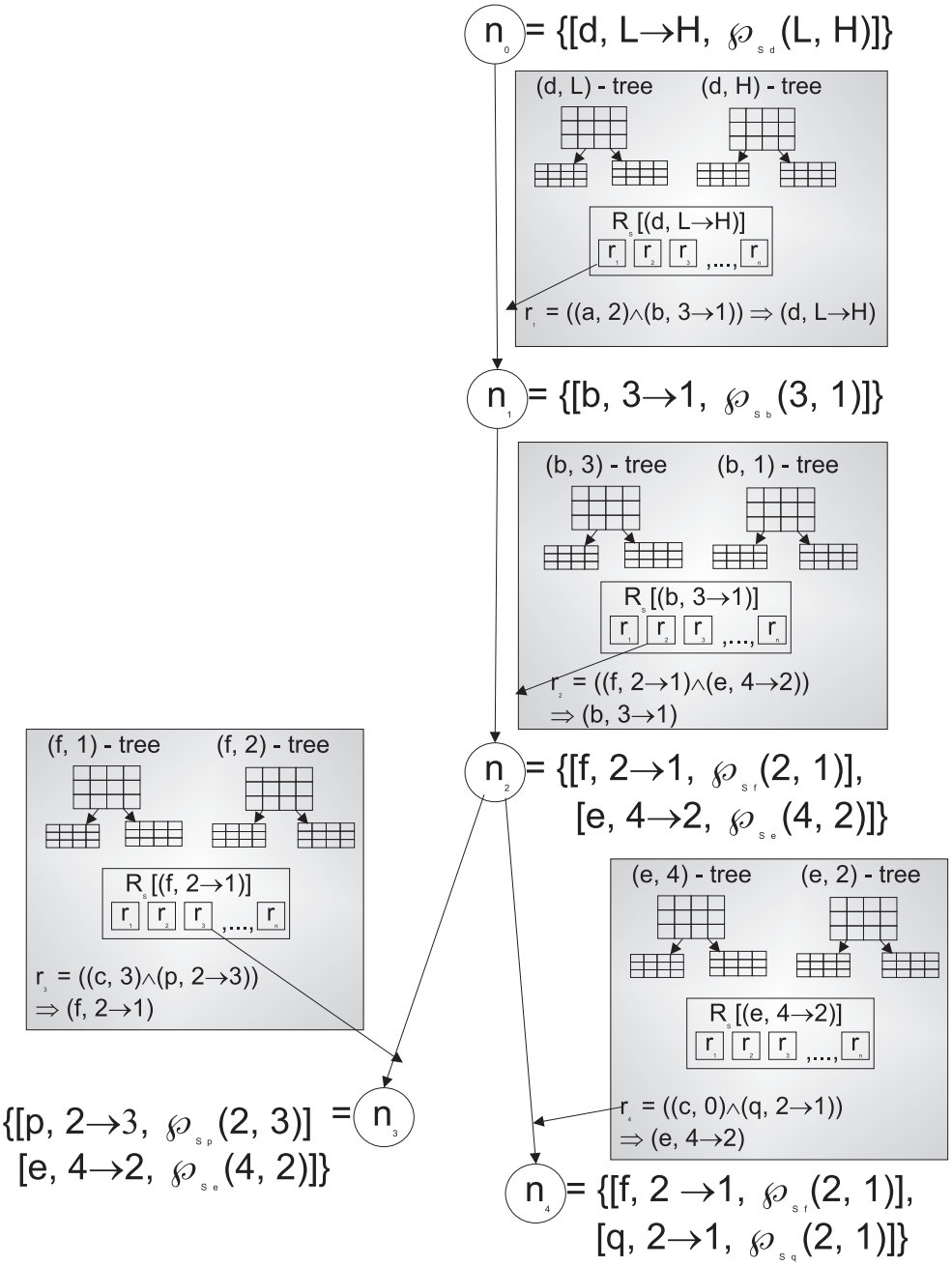

Fig. 3. Tree-based construction of low-cost action rules - Search Graph G

6 Conclusion

The proposed approach presents an elegant way to combine the Action-Forest algorithm, and the heuristic strategy for lowest cost reclassification. It allows for enhancement of both algorithms. In the first case, it associates the notion of cost to the E-action rules, thus allowing for cheapest rules to be discovered. In the second case, it allows for more efficient means of generating the base of of action rules $R_{S_i}[(d, k_1 \rightarrow k_2)]$ by using tree-based construction.

References

1. Adomavicius G, Tuzhilin A (1997) Discovery of actionable patterns in databases: the action hierarchy approach. In: Proceedings of KDD97 Conference. Newport Beach, CA. AAAI Press
2. Chmielewski M R, Grzymala-Busse J W, Peterson N W, Than S (1993) The rule induction system LERS - a version for personal computers. In: Foundations of Computing and Decision Sciences. Vol. 18, No. 3-4, Institute of Computing Science, Technical University of Poznan, Poland: 181–212
3. Dardzińska A, Raś Z W (2003) On Rule Discovery from Incomplete Information Systems. In Proceedings of ICDM'03 Workshop on Foundations and New Directions of Data Mining, (Eds: T.Y. Lin, X. Hu, S. Ohsuga, C. Liau). Melbourne, Florida, IEEE Computer Society, 31–35
4. Geffner H, Wainer J (1998) Modeling action, knowledge and control. In: ECAI 98, Proceedings of the 13th European Conference on AI, (Ed. H. Prade). John Wiley & Sons, 532–536
5. Greco S, Matarazzo B, Pappalardo N, Slowinski R (2005) Measuring expected effects of interventions based on decision rules. In: Special Issue on Knowledge Discovery, (Ed. Z.W. Raś). Journal of Experimental and Theoretical Artificial Intelligence. Taylor and Francis, Vol. 17, No. 1-2, 103–118
6. Grzymala-Busse J (1997) A new version of the rule induction system LERS. In: Fundamenta Informaticae, Vol. 31, No. 1, 27–39
7. Liu B, Hsu W, Chen S (1997) Using general impressions to analyze discovered classification rules. In: Proceedings of KDD97 Conference, Newport Beach, CA, AAAI Press
8. Pawlak Z (1991) Rough sets-theoretical aspects of reasoning about data. Kluwer, Dordrecht
9. Pawlak Z (1981) Information systems - theoretical foundations. In: Information Systems Journal, Vol. 6, 205–218
10. Polkowski L, Skowron A (1998) Rough sets in knowledge discovery. In: Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer
11. Raś Z, Wieczorkowska A (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, LNCS/LNAI, No. 1910, Springer-Verlag, 587–592
12. Raś Z W, Tsay L-S (2003) Discovering extended action-rules (System DEAR). In: Intelligent Information Systems 2003, Proceedings of the IIS'2003 Symposium, Zakopane, Poland, Advances in Soft Computing, Springer-Verlag, 293–300

13. Raś Z, Gupta S (2002) Global action rules in distributed knowledge systems. In: Fundamenta Informaticae Journal, IOS Press, Vol. 51, No. 1-2, 175–184
14. Silberschatz A, Tuzhilin A (1995) On subjective measures of interestingness in knowledge discovery. In: Proceedings of KDD́95 Conference, AAAI Press
15. Silberschatz A, Tuzhilin A (1996) What makes patterns interesting in knowledge discovery systems. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 6
16. Tsay L-S, Raś Z.W (2005) Action rules discovery: System DEAR2, method and experiments. In: Special Issue on Knowledge Discovery, (Ed. Z.W. Raś). Journal of Experimental and Theoretical Artificial Intelligence. Taylor and Francis, Vol. 17, No. 1-2, 119–128
17. Tzacheva A, Raś Z.W (2005) Action rules mining. International Journal of Intelligent Systems. Wiley, Vol. 20, No. 7, 719–736