FAARM: Frequent association action rules mining using FP-Tree

Djellel Eddine Difallah
eXascale Infolab
University of Fribourg
Switzerland
djelleleddine.difallah@unifr.ch

Ryan G. Benton, Vijay Raghavan
Center for Advanced Computer Studies
University of Louisiana at
Lafayette, LA
{rbenton,vijay}@cacs.louisiana.edu

Tom Johnsten
Computer & Information Sciences
Univ. of South Alabama
Mobile, AL
tjohnsten@usouthal.edu

***Abstract*—Action rules mining aims to provide recommendations to analysts seeking to achieve a specific change. An action rule is constructed as a series of changes, or actions, which can be made to some of the flexible characteristics of the information system that ultimately triggers a change in the targeted attribute. The existing action rules discovery methods consider the input decision system as their search domain and are limited to expensive and ambiguous strategies. In this paper, we define and propose the notion of action table as the ideal search domain for actions, and then propose a strategy based on the FP-Tree structure to achieve high performance in rules extraction.**

Keywords- action rules; recommendation; association mining; action table; FP-Tree.

I. INTRODUCTION

In association rule mining, the rules extracted from an information system are handed to the domain experts who need to filter what information is interesting or trivial. Action rules were introduced in [3] as a new class of rule discovery that provides hints on possible actions a business should take to achieve a desired target.

Mining action rules is defined as the process of identifying patterns in a decision system capturing the possible changes to certain object attributes that may lead to a change in the decision value [3]. Generally, action rule mining operates on a decision system [13] with objects having three classes of attributes: stable, flexible and decision. The stable attributes are attributes that cannot be changed or, in some approaches, require a prohibitive high cost to change them [4]. Examples of stable attributes are the date of birth or weather conditions. Conversely, flexible attributes are attributes on which the analysts have a certain degree of freedom in manipulating such as color, sale's percentage, and so forth. The decision is the attribute that the analyst would like to see changed; an example would be the profit of a company.

Existent action rules discovery methods use a decision table as their primary search domain; the employed strategies are limited to candidate generation-and-test. In our approach, the discovery of action rules is based on a domain of actions that we create from the decision system, called the action table. The main contribution of this paper is to reformulate the action rules mining problem into the association-mining problem framework using the action table as the new search domain. A particularly suited approach is to use an FP-Tree structure to store the action table and the FP-Growth algorithm to extract association action rules.

The rest paper is organized as follows. Section 2 briefly surveys previous works on action rules mining. Section 3 reviews the key concepts and definitions. Sections 4 and 5 present the action table and our strategy through a detailed example. Finally we experimentally compare the performance of our solution with existent algorithms.

II. OVERVIEW AND RELATED WORK

Previous works on action rules assume a generalization of an information system S as introduced in [1], where S = (X, A, V):

- X: a nonempty, finite set of objects.
- A: a nonempty, finite set of attributes.
- $V = \{V_a : a \in A\}$ all the attributes values.

Additionally, $a : X \rightarrow V_a$ is a function for any $a \in A$, that returns the value of the attribute of a given object. The attributes are divided into different categories: a stable set A_{st}, flexible set A_{fl} and a decision set D of attributes, where $A = A_{st} \cup A_{fl} \cup D$

For example, Table I. represents a decision table with eight objects: *a* is stable, *b* and *c* are flexible, *d* is the decision attribute.

TABLE I. EXAMPLE OF A DECISION TABLE

	a	***b***	***c***	***d***
x1	a1	b1	c1	l
x2	a2	b1	c1	l
x3	a2	b2	c1	h
x4	a2	b2	c2	h
x5	a2	b1	c1	l
x6	a2	b2	c1	h
x7	a2	b1	c2	h
x8	a1	b2	c2	l

In order to extract action rules, [3, 7, 8, 9, 10] were based on an existing set of classification rules. Certain pairs of these rules were combined to reclassify objects to a targeted state. One problem that arose, as argued in [4], is **"**some meaningful action rules should be missed in these classification-based techniques and thus existing algorithms cannot specify when and how the correct and complete underlying action rules are discovered**"**. Instead, they were the first to propose an inductive approach for mining directly from the decision system. The approach is formulated as a search problem based on a support-confidence-cost framework and an Apriori-like algorithm

978-0-7695-4409-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDMW.2011.82

[4]. Further work to extract action rules directly from the decision system followed.

In [5], the proposed algorithm, called Action Rules Discovery (ARD), builds rules for a given target decision using an iterative marking strategy. It considers the change in an attribute value as an atomic-action-term of length one, and then an action-term is a composition of atomic-action-terms. ARD starts by generating all atomic-action-terms for a given set of attribute values and assigning a mark (unmarked, positive, negative) based on support and confidence measures. The unmarked terms are placed into the candidate list. Next, it generates all possible action-terms of length two by combining terms in the candidate list and the atomic terms. The process continues iteratively, creating terms of greater length, until the candidates' list is empty. The action-terms marked as positive are used to construct the action rules.

In [6], authors presented an association type of action rules and used an Apriori like strategy to find frequent action sets to induce action rules. Like ARD, the algorithm AAR (Association Action Rule) considers atomic action sets being the fine granule used to construct longer rules (similar to items and item sets in association mining). The Apriori algorithm is directly used with few modifications; the main changes are mostly driven (a) by modifications to the definition of support and confidence and (b) by the calculation of the measures directly from the input decision system.

Although these approaches have different definitions for objective measures like support and confidence, they use the same idea of atomic-action set, action set and *Standard Interpretation*.

III. PRINCIPLES OF ACTION RULES

In this section, we provide a quick overview of the key concepts in action rules literature. We particularly focus on [6] as it is the most recent work on the subject and is the most related to this work in the sense that both try to map association rules mining into action rules mining.

A. Atomic action set

Defined as the expression (a, a1→a2) where a is an attribute in A and a1, a2 are values of a. If the attribute is stable or did not change its value then the atomic action set is expressed as (a, a1). The domain of an atomic action set is its attribute.

$$Dom((a, a1 \rightarrow a2))=a$$

Example: Consider *Rate* a flexible attribute with values V_{rate}={30%, 10%, 50%}. The atomic action set (*Rate*, 30%→10%) means changing the value of Rate from 30% to 10%.

B. Action sets

Constructed as the conjunction of atomic action sets with the composition operator (·). If *t1, t2* are two atomic action sets with different attributes, then $t=t1 \cdot t2$ is an action set. The domain of the action set t is the set of attributes from all its atomic action sets, here:

$$Dom(t)=Dom(t1) \cup Dom(t2)$$

Example: Consider *Age* as a stable attribute with values V_{age}={25, 50, 70} and Credit a flexible attribute with values V_{credit}={good, bad}. An action set could be the composition [(*Age*, 50) · (*Rate*, 30%→10%) · (*Credit*, bad→good)] which could be read as follows: for customers of *Age* 50, change the *Rate* from 30% to 10% and the C*redit* from good to bad.

C. The standard interpretation (noted Ns)

The introduction of the *Standard Interpretation* is the basis of measures like support and confidence. In association mining, the support of an itemset is simply the count of objects. For action rules, we need to consider two sets. The first set is all the objects with attributes value equal to the initial state of the action; the second set, respectively, is all the objects having attributes values equal to the values of the final state of the action.

Example: The *Standard interpretation* of the action set Ns[(*Age*, 50) · (*Rate*, 30%→10%) · (*Credit*, bad→good)]= [A1,A2]

Where:

A1= { x∈X: age(x) = 50 ∧ rate(x) = 30% ∧ credit(x) = bad}.

A2= { x ∈ X: age(x) = 50 ∧ rate(x) = 10% ∧ credit(x)=good}.

D. The support of an action set

Assume t an action set with standard interpretation Ns(t)=[*A1*, *A2*] . The support *Supp* of t is considered in AAR as[6] :

$$Supp(t)=\min\{card(A1), card(A2)\}$$

In ARD as [5]:

$$Supp(t)=card(A1)$$

Hence, for AAR, for the two states, the support is concerned only with the state having the lowest occurrences. For the ARD, it was only in terms of number of occurrences of the initial state. It should be noted that these definitions lead to very different results. With respect to ARD, a rationale for the definition was not provided; however, the definition allows for A2 to be 0 and/or below minimum support. AAR, however, ensures that both *A1* and *A2* are guaranteed to satisfy the minimum support threshold.

E. Action rule

An action rules r is expressed as r=[t1 ⇒t2], where t1 and t2 are two action sets. Typically t2 is the action comprising only the decision attribute.

Example: [(*Age*, 50) · (*Rate*, 30%→10%) · (*Credit*, bad→good)]⇒[(*Profit*, low→high)]

The support is calculated similarly to action sets by considering the t1· t2 as an action set itself.

F. The confidence measure:

The confidence of an action rule r=[t1⇒t2], Considering Ns(t1)=[A1, A2] and Ns(t2)=[Z1, Z2], with A1 and A2 not empty:

$$conf(r) = \frac{card(A1 \cap Z1)}{card(A1)} \times \frac{card(A2 \cap Z2)}{card(A2)} \quad (1)$$

In AAR, generating action rules is similar to association rule mining where frequent item sets are first extracted. The algorithm, which is based on Apriori, generates actions sets with support that exceeds a specified threshold value: minimum support (minSup); any action set that meets this criterion is a frequent action set. An action rule is constructed and tested as following:

- If *t* is a frequent action set and *t1 is* a subset of *t* then:

$$r = [\, t - t1 \Rightarrow t1 \,]$$

- If Conf(r) ≥ *minConf*, where *minConf* is the minimum confidence specified, r is a valid rule.

Like Apriori, the AAR method can generate a large number of rules. The process does not constrain what the decision attribute may be; in fact, it does not even require a decision attribute to be specified. As a side benefit, unlike ARD, the user does not have to supply what the targeted decision should be. For example, ARD would require the user to state (Rate, 30%→10%) is the decision of interest.

On the other hand, AAR suffers from two problems. First, like Apriori, the number of rules may be overwhelming to the user. Second, if the user is interested in a particular change, like (Rate, 30%→10%), there is no guarantee that the AAR method will generate the required rules. For instance, if minimum support is 8, and (Rate, 10%) has a support of 5, then no rules containing (Rate, 10%) would be generated.

As an aside, a justification of the defined confidence measure was provided for the ARD method. In [5], the authors indicated that the definition of confidence should be considered as an optimistic confidence. It requires that the card(A1) ≠0, card(A2) ≠0, card(A1 ∩ Z1) ≠ 0 and card(A2 ∩ Z2) ≠ 0. In effect, this definition was required due to ARD's definition of support. Without it, action rules could be generated that for cases in which A2 ∩ Z2 never occurred.

Interestingly, for AAR, a similar claim was made. In [6], the definition was declared optimistic, because card(A1) ≠0 and card(A2) ≠0. However, since AAR uses minSup, as long as minSup is greater than 0, it is guaranteed that card(A1) ≠0 and card(A2) ≠0.

IV. The Action table

As mentioned previously, the AAR and ARD methods operate within the search space represented by the decision table; we refer to this as the information space. As a result, the creation (and tracking) of the various flexible *atomic action sets* is intertwined with the action set generation. Hence, one would need to calculate the support of (*Rate*, 30%) and (*Rate*, 10%) first in order to obtain the support of (*Rate*, 30%→10%). This operation needs to be performed for all *k* iterations. The first iteration generates frequent action sets composed of 1 element, the second iteration generated action sets composed of 2 elements and so forth. For instance, assume you have stable attribute *Gender* with values {m,f}. Then, to calculate the action set [(*Gender*, f)·(*Rate*,10%→30%)], one would first need to calculate the support of [(*Gender*, f), *Rate*(10%)] and [(*Gender*, f), (*Rate*, 30%)].

However, this could be avoided if, instead of working within the information space, we were able to work within an 'action space', where the possible flexible action mappings are already known and represented. In this paper, this is done via the introduction of the action table.

The key idea is explicitly enumerating how each object in S can be converted from the undesired, or initial, state to the desired one, or final. That information can be captured as a set of action sets within a table called the action table. This creates a limitation, compared to the AAR approach, by requiring the targeted decision to be known before hand; this limitation, however, is similar to that required by the ARD approach.

For example, if we consider Table I as our decision table and the targeted decision is (d, $l \rightarrow h$), then:

- L={x ∈ X: d(x)=*l*}, i.e. all the objects with decision value *l*.
- H={x ∈ X: d(x)=*h*}, i.e all the objects with decision value *h*.

The action table will contain the necessary action sets to move every object in L to an object in H. The action table will contain exactly card(L)*card(H) action sets.

Each row in the action table reflects the necessary action sets for transforming an object from low (*l*) to high (*h*). This operation is formalized as following:

∀x∈ L, y∈ H: x→y is a possible transition describing a new action set ***t***, and ∀a∈ A:

- If a is flexible and a(x) ≠ a(y): a(***t***)=a(x)→a(y)
- If a is stable and a(x) ≠ a(y), then a(t) is contradicting and is therefore discarded.
- If a(x) = a(y), then a(***t***)= a(x)

Example: To generate the action table we start from the decision Table I and organize it into two subtables, as shown in table II, with respect to the decision attribute *d*.

TABLE II. The Decision table organized with respect to the decision attribute d.

		a	***b***	***c***	***d***
L	***x1***	a1	b1	c2	l
	x2	a2	b1	c1	l
	x5	a2	b1	c1	l
	x8	a1	b2	c2	l
H	***x3***	a2	b2	c1	h
	x4	a2	b2	c2	h
	x6	a2	b2	c1	h
	x7	a2	b1	c2	h

Then we make a cross product from L into H and construct the action sets for each transition. If there is a change in the stable attribute we simply discard it. This process results in table III.

TABLE III. THE ACTION TABLE FOR THE TARGET (D, L→H)

	a	b	c
x1→x3	-	b1→b2	c2→c1
x1→x4	-	b1→b2	c2
x1→x6	-	b1→b2	c2→c1
x1→x7	-	b1	c2
x2→x3	a2	b1→b2	c1
x2→x4	a2	b1→b2	c1→c2
x2→x6	a2	b1→b2	c1
x2→x7	a2	b1	c1→c2
x5→x3	a2	b1→b2	c1
x5→x4	a2	b1→b2	c1→c2
x5→x6	a2	b1→b2	c1
x5→x7	a2	b1	c1→c2
x8→x3	-	b2	c2→c1
x8→x4	-	b2	c2
x8→x6	-	b2	c2→c1
x8→x7	-	b2→b1	c2

Note: the atomic action set with the decision attribute d will always be (d, l→h) and therefore would be redundant in the action table.

A. Implications of the Action Table

First, unlike the original decision table, the action table now explicitly includes *change* values as part of the attribute definition. Second, the decision cardinality card({*d*, l→h}) is equal to the number of rows in the action table. Third, the action table does not allow us to recover the individual counts of Z1 or Z2; nor, for that matter, the individual counts of A1 or A2. This information is not needed, then support for t, where t=t1·t2 and t2 = {*d*, l→h}, is given by:

$$Supp(t) = card(A1 \cap Z1) \times card(A2 \cap Z2) \qquad (2)$$

This calculation can take place by performing a count count operation on the action table.

What is attractive about this definition is it is the same support measure used in the traditional Association Mining [14].

Now, if we look at the confidence measure used in traditional Association Mining [14], we see:

$$conf(r) = \frac{Supp(r)}{Supp(t1)} \qquad (3)$$

One potential problem arises if the traditional association mining confidence measure is utilized. While the support of the rule can be directly calculated from the action table, the *Supp*(t1) cannot be calculated from the action table. This is because the action table does not contain all occurrences of A1 and A2. However, the support for A1 and A2 can be easily calculated from the decision table; a simple pass through the table, performing a basic count operation is required. More formally, the support of an action set *t1* with standard interpretation Ns(*t1*)=[A1, A2]:

$$Supp(t1) = card(A1) \times card(A2) \qquad (4)$$

V. THE FAARM STRATEGY:

Using the concept of action table and the resulting changes to support and confidence, we believe that any association-mining algorithm can be used to generate action rules. Here, we demonstrate the use of the FP-Growth algorithm [2] to generate the action rules. We call the resultant approach FAARM (Frequent Association Action Rule Mining).

The FP-Growth algorithm is a divide–and–conquer approach that is considered to be an order of magnitude faster than Apriori[2]. It relies on a special tree data structure called FP-Tree[2] which is obtained by ordering the transaction attributes values by their frequency, pruning those that do not meet a given minimum support, and then inserting the transaction, or action sets in our case, into a tree. The result is a condensed data structure that avoids expensive database scans and is especially tailored for dense datasets [2].

The same concepts of atomic action set, action set and *Standard Interpretation* are borrowed from previous work. However, the definitions of support and confidence used are those introduce in section IV.A.

Now, we present FAARM, the proposed process of extracting action rules from:

FAARM METHOD

1) *Specify the decision target.*
2) *Generate all the atomic action sets from A.*
3) *Calculate the frequency of each atomic set.*
4) *Build the action table.*
5) *Prune and reorder the action table.*
6) *Build the FP-Tree from the action table.*
7) *Run FP-Growth on the FP-tree.*
   a) *Extract frequent action sets.*
   b) *Build and test the action rules.*

To better explain our proposed FAARM strategy, we go through the example on the decision table described in Table I. We use *minSup*=4 and the *minConf*=80%.

1) Set the targeted decision: [*d*, l→h].

2) Generate the atomic sets: The atomic sets are all the possible transitions for every attribute.

Atomic sets={(a,a1), (a,a2), (b,b1), (b,b2), (b,b1→b2), (b,b2→b1), (c,c1), (c,c2), (c,c1→c2), (c,c2→c1)}

3) Calculate the frequency of each atomic set: In order to calculate the support of each atomic action set with regard to a decision target, it is sufficient to scan the

decision table once and count the occurrence of each attribute with respect to decision's left and right values, here *l* and *h* (Table IV). Then, using the support formula (4) and the minimum support criteria, we can calculate the exact support of all the possible atomic action sets (Table V).

TABLE IV. THE FREQUENCY OF ATTRIBUTES IN V FOR THE ACTION (d, l→h) FROM THE DECISION TABLE IN ONE PASS

	l	***h***
a1	2	0
a2	2	4
b1	3	1
b2	1	3
c1	2	2
c2	2	2

TABLE V. SUPPORT OF ALL ATOMIC ACTION SETS (MINSUP= 4)

Atomic action set	Support with regards to (d, l→h)
(a,a1)	0 (Does not meet min support)
(a,a2)	8
(b,b1)	3 (Does not meet min support)
(b,b2)	3 (Does not meet min support)
(b,b1→b2)	9
(b,b2→b1)	1 (Does not meet min support)
(c,c1)	4
(c,c2)	4
(c,c1→c2)	4
(c,c2→c1)	4

For a later use, the list of atomic action sets is ordered by descending support:

List= {(b1→b2), (a2), (c1),(c2), (c1→c2), (c2→c1)}

4) Generate the action table: (see section IV for the example).

5) Prune and reorder the action table: Once the action table is generated, we can use *List* to order action sets by descending support and remove atomic sets that does not meet the minimum support. This operation facilitates the creation of the FP-Tree.

6) Build the FP-Tree from the action table: Generating the action table and storing it in memory is expensive (time and space complexity is $O(n^2)$), where n is the number of transactions; to alleviate the space complexity, we propose to use the FP-Tree to store the table. To build the FP-Tree we insert each action set, from the ordered and pruned action table, into the tree using the atomic action sets as the nodes. Each time a node is inserted or reused we increment its local count. If the node has been inserted somewhere else in the tree we create a link to the last inserted one (Figure 1).

7a) Extract Frequent action sets: FP-Growth receives an FP-Tree as an input and does its traditional job for extracting frequent patterns given minimum support criterion.

7b) Generating Association Action Rules: The following frequent action sets are extracted from the FP-Tree using the FP-Growth algorithm [2]:

t1= (*a*,a2) · (*b*, b1→b2) · (*c,* c1) · (*d*, *l*→*h*)

Supp(*t1*)= 4=minSup

t2= (*a*,a2) · (*c*, c1→c2) · (*d*, l→h)

Supp(*t2*)=4=minSup

Finally, we can easily construct the association action rules and check their confidence. This operation takes exactly one scan of the Table 1 for each frequent action set:

*r1= (a,*a2*) · (b,* b1→b2*) · (c,* c1*)* ⇒ (*d*, l→h)

supp(*r1*)= 4=minSup, conf(*r1*)= 4/4 >minConf

*r2= (a,*a2*) · (c*, c1→c2*)* ⇒ *(d,* l→h*)*

supp(*r2*)=4=minSup, conf(*r2*)=4/8<minConf

Only *r1* meets the minimum confidence for a valid association action rule extracted from S.

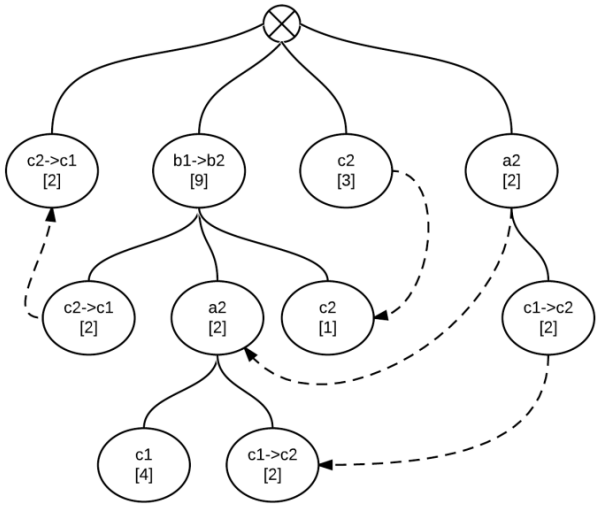

Figure 1. FP-Tree generated from the action table.

VI. EXPERIMENTS

The experimental goal is to compare the performances of our algorithm FAARM to other action rules discovery algorithms that do not use pre-existing classification; the ones selected are AAR and ARD. We also considered datasets used in previous action rules discovery literature, namely Hepatitis [9] and Nursery [12]; the datasets can be obtained from the UCI Machine Learning Repository [11]. Finally, we used the same classification (Stable, Flexible, Decision) for the attributes as used in [9, 12].

A. Description of the datasets

A brief description of each dataset is provided in this section.

1) Hepatitis dataset

This dataset contains clinical data of patients affected by the *Hepatitis* disease. It has 155 records and 19 attributes, not including the decision attribute. The attributes are decomposed into 2 stable attributes and 17 flexible attributes. Each flexible attributes is composed 9 values or less; the majority are composed of only 2. The patients are classified into: Die, Live. Our target is to find rules to change the likelihood of this classification i.e. targeted decision effect is:

[*class*, die→live]

2) Nursery dataset

This dataset is composed of the evaluation forms of applications to nursery schools. The dataset has 12960 records and 8 attributes, not including the decision attribute. The attributes are decomposed into 4 stable attributes and 4 flexible attributes. Each flexible attributes is composed of 3 values or less. Our target is to find rules to enhance the chances of having an application go from being not recommended to priority i.e. targeted decision effect is:

[*rank*, not_recom→priority]

B. Experimental Methodology

In order to achieve fair performance comparison, we have implemented a version of ARD and AAR with the following modifications:

- The same definition of support is used as in FAARM: using different definition led to different set of rules, which is expected, as using different definitions would results in different action sets to be discarded/kept.
- AAR: First, we initially prune atomic-sets of the class attribute that are different from our targeted decision. Second, we stop the Apriori iterations if none of the frequent action sets contains the targeted decision.
- Because the search space is different in the three algorithms (N in AAR and ARD, N^2 in FAARM), the final support value for an action set in AAR and ARD must be squared.

With these modifications, the three algorithms will now produce the same rules. Thus, the purpose of this comparison is to determine which method, if any, is faster. In particular, as action set generation is more expensive than rule generation, we examined the impact of changing the support value. For this study, we use a minimum confidence threshold of 80% for all experiments.

C. Results

The results in Figure 2 show that FAARM achieves better performances on both Hepatitis and Nursery datasets with both high and low support values. With FAARM, generating the action table from the Nursery would take (4320)*(4266) operations, these values are the number of records with status=*not_recom* and status=*priority*, respectively. This operation is expensive, but once generated and encoded, the FP-Tree could be mined quickly.

AAR performs better on Hepatitis, mainly because of the lower number of candidates. ARD, on the other hand, showed the worst performance. During the experiments, we noticed that ARD was fast when finding shortest rules and slower finding longer rules. Furthermore, ARD is very poor at finding a good stopping point; in fact, it actually goes through all the iteration phases (the maximum number of iteration being the length of the transaction). The reason is that, for each generation, there are always some candidates to test, even if a positive mark isn't achievable.

An interesting aspect is the extent of FP-Tree compression over the action table. Figure 4 shows the size of the FP-Tree constructed from the action table in the Nursery experiment with varying minimum support values. While the action table size is always 250Mb, the size of the FP-Tree is 3.5Mb even for very small minimum support. The size of the FP-Tree shrinks even further for increasing minimum support values and this is due to the pruning at this level.

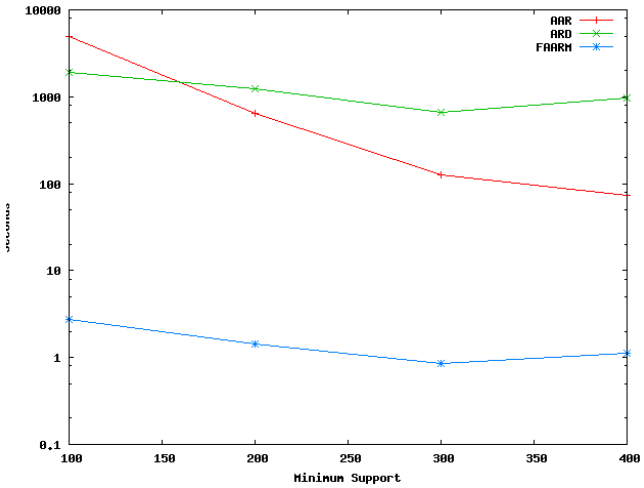

Figure 2. Speed comparison of FAARM, AAR and ARD extracting action rules on Hepatitis dataset with varying minimum support threshold

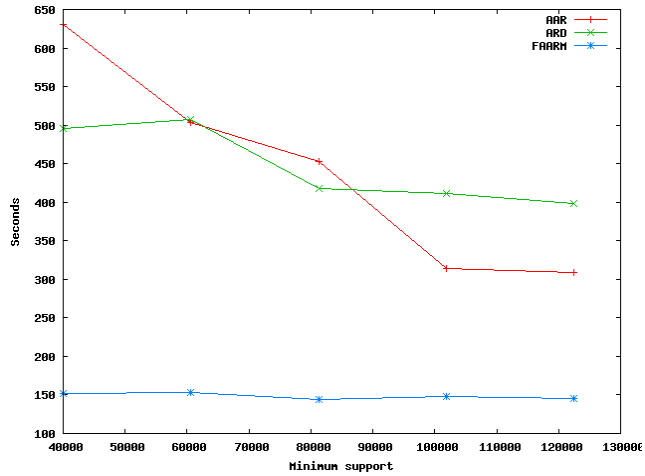

Figure 3. Speed comparison of FAARM, AAR and ARD extracting action rules on Nursery dataset with varying minimum support threshold

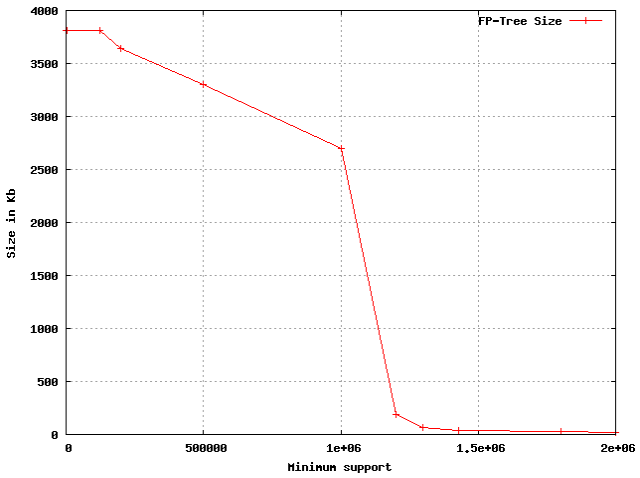

Figure 4. Compression of the action table from Nursery dataset of size 250Mb into FP-Tree with varying minimum support values

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose the action table as the ideal search domain for action rules mining. The action table transforms the complex problem of finding action rules from a plain decision table, into finding action rules from an action table. As a result, the problem of action rules mining is reformulated into association-mining.

In practice, we applied FAARM on the Hepatitis and Nursery datasets and compared the results and performances with AAR and ARD. Although the space and time complexity associated with generating the action table are $O(n^2)$, experiments show that FAARM has a better execution time on relatively small dataset, over ARD and AAR.

Generating the action table directly into the FP-Tree could mitigate the space complexity associated with action table. As a future work, we propose to look at parallel implementation of Apriori and FP-Growth to test the scalability of using the action table with large datasets.

REFERENCES

[1] Z. Pawlak, “Information systems - theoretical foundations”, Information Systems Journal, Elsevier, Vol. 6, 1981, 205-218.

[2] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation”, ACM SIGMOD International Conference on Management of Data, 2000, 1–12.

[3] Z.W. Ras and A. Wieczorkowska, "Action-Rules: How to Increase Profit of a Company", The Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases, 587-592.

[4] Z. He, X. Xu, S. Deng, R. Ma, “Mining action rules from scratch”, Expert Systems with Applications, Elsevier, Vol. 29, No. 3, 2005, 691-699.

[5] Z.W. Ras and A. Dardzinska, "Action Rules Discovery without Pre-existing Classification Rules", The Sixth International Conference on Rough Sets and Current Trends in Computing, 2008, 181-190.

[6] Z.W. Ras, A. Dardzinska, L. Tsay, and H. Wasyluk, "Association Action Rules", IEEE International Conference on Data Mining Workshops, 2008, 283-290.

[7] Qiang Yan, Jie Yin, Charles Ling, Tielin Chen,"Postprocessing Decision Trees to Extract Actionable Knowledge", IEEE International Conference on Data Mining, 2003, 685-688.

[8] Z.W. Ras and L. Tsay, "Discovering Extended Action-Rules (System DEAR)", International IIS IIPWM'03 Conference, 2003, 293-300.

[9] L. Tsay and Z.W. Ras, "Action rules discovery: system DEAR2, method and experiments", Journal of Experimental & Theoretical Artificial Intelligence, 2005, 119-128.

[10] Z.W. Ras, E. Wyrzykowska, and H. Wasyluk, "ARAS: Action Rules Discovery Based on Agglomerative Strategy", Third International Workshop on Mining Complex Data, 2007, 196-208.

[11] http://archive.ics.uci.edu/ml/datasets/

[12] S. Im and Z.W. Ras, "Action Rule Extraction from a Decision Table: ARED", International Syposium on Methodologies for Intelligent Systems, 2008, 160-168.

[13] J. S. Deogun, V. V. Raghavan, and H. Sever, “Rough set based classification methods and extended decision tables”, International Workshop on Rough Sets and Soft Computing, 1994, 302-309.

[14] R. Agrawl and R. Srikant, “Fast algorithm for mining assocation rules,” International Conference on Very Large Data Bases, 1993, 487-499.