Apriori-with-Constraint for Flexible Association Rule Discovery

Kittisak Kerdprasop, Phaichayon Kongchai, Nittaya Kerdprasop
Data Engineering Research Unit,
School of Computer Engineering,
Suranaree University of Technology, Thailand
kerdpras@sut.ac.th, zaguraba_ii@hotmail.com, nittaya@sut.ac.th

***Abstract*—Association rule discovery, or association mining, is one of the major data mining tasks that has gained much interest from researchers and general users. The knowledge obtained from association mining can be used to benefit business in many aspects such as recommend new products, design catalogs, manage sales promotion, and so on. But data processing for association rule discovery has expensive computing time because the relationships induced from data can be tremendously many more than those induced from other data mining tasks such as classification. As a consequence, most association mining software generally create so many rules from the association mining process and some of these rules are not beneficial to any users. To solve this useless rule mining problem, we propose to incorporate Apriori algorithm with constraint function for users to specify subset of association rules containing only interesting items. Besides specific items, users can also identify length of the association rules. Our two Apriori-with-constraint algorithms, called Association rule discovery with Constraints In Frequent itemset mining (ACIF) and Association rule discovery with Constraints After Frequent itemset mining (ACAF), are experimentally proven to be able to reduce processing time and also pruning a great number of useless rules.**

Keywords-association rules; frequent itemset mining; data mining; association analysis;constraint logic programming.

I. INTRODUCTION

Data mining aims at extracting hidden knowledge from data [8]. Knowledge is known to be a valuable asset to most organizations as a substantial source to enhance understanding of relationships among data instances and support better decision making to increase organizational competency. Automatic knowledge acquisition can be achieved through the availability of the knowledge induction component. The induced knowledge can facilitate various knowledge-related activities ranging from expert decision support, data exploration and explanation, estimation of future trends, and prediction of future outcomes based on present data. The methodology of knowledge induction is known as knowledge discovery in databases, or data mining.

Data mining methods are broadly defined depending on the specific research objective and involve different classes of mining tasks including regression, classification, clustering, identifying meaningful associations between data attributes. The later mining task refers to association mining, or market basket analysis [9] in the retail business domain, which is the main focus of our research.

Association mining is a popular method for discovering relations between features or variables in large databases [11], [12], [15], [20] and then presenting the discovered results as a set of if-then rules, such as {milk, bread} => {butter} to indicate that if a customer buys milk and bread, he or she is more likely to buy butter as well. Association rule generation process is composed of two major phases: frequent itemset mining and rule generation. Frequent itemset mining is to find all items or features that are frequently occurred together [13], [22]. It is an important phase of association mining because it is a difficult task to search all possible itemsets.

We thus pay attention to the design and implementation of frequent itemset discovery by applying the constraint concepts in this step. We propose two algorithms: Association rule discovery with Constraints In Frequent itemset mining (ACIF) and Association rule discovery with Constraints After Frequent itemset mining (ACAF). The first algorithm considers constraints during the frequent itemset mining phase, whereas the later one applies constraints after the frequent itemset mining steps. Constraints in our proposed method include items of interest, items to be excluded from the mining results, and desired rule length of the final association mining. Our implementation is based on the ECLiPSe constraint system (www.eclipseclp.org).

This paper is the extension of our previous work [14] on association rule discovery with constraint logic programming that was the proposal of extending Apriori [1] with more domain-specific constraints. The work presented here explains in more details the idea of incorporating domain-specific constraints both during and after the frequent itemset mining stage (ACIF and ACAF algorithms, respectively). Applicability of the proposed idea and its implementation have also been demonstrated in this paper.

The organization of this paper is as follows. The problem of association rule discovery and main concepts of logic and constraint logic programming are reviewed in Section 2. Then the design of association mining with constraint algorithms is explained in Section 3. The implementation of the two algorithms, ACIF and ACAF, together with their experimental results are presented in Section 4. Finally, Section 5 concludes the paper with discussion of our future research direction.

 ISBN: 978-1-61208-269-1

II. PRELIMINARIES

A. Frequent Patterns and Association Mining

Frequent patterns are common occurrences such as sets of features or items that appear in data frequently. Frequent pattern mining focuses on the discovery of relationships or correlations between items in a dataset. In frequent itemset mining, we are interested in analyzing connections among items. A collection of zero or more items is called an itemset. The discovery of interesting relationships hidden in large datasets is the objective of frequent pattern mining. The uncovered relationships can be represented in the form of association rules. An association rule is an inference of the form $X \rightarrow Y$, where X and Y are non-empty disjoint itemsets. An itemset is called a *frequent itemset* if its support, i.e., the number of transactions that contain a particular itemset, is greater than or equal to user-specified support threshold (called *MinSup*). It is the *MinSup* constraint that helps reducing the computational complexity of frequent itemset generation. If the itemset is infrequently occurred, all supersets of this itemset are also infrequent and they can be pruned to reduce the search space.

This pruning strategy is called an anti-monotone property and has been applied as a basis for searching frequent patterns in the well-known algorithm Apriori [1], [2]. The algorithms find all frequent itemsets by generating supersets of previously found frequent itemsets. This generate-and-test method is computational expensive. Han et al [10] proposed a different divide-and-conquer approach based on the prefix-tree structure that consumes less memory space. Toivonen [21] employed sampling techniques to deal with frequent pattern mining from large databases. Zaki et al [23] tackled the problem by means of parallel computation.

Some researchers [4], [7], [17], [18], [19] consider the issue of search space reduction through the concept of constraints. Our research is in the same direction as De Raedt et al [7]. We consider the problem of mining frequent patterns within a setting of constraint logic programming using the ECLiPSe constraint system [3]. Constraints can play an important role in improving the performance of mining algorithm. The problem of constraint-based pattern mining can therefore be formulated as the discovery of all patterns in a given dataset that satisfy the specified constraints.

B. Constraint Logic Programming

Constraint logic programming, or CLP, is a declarative programming style that combines the features of logic programming [16] and constraint propagation to solve combinatorial and optimization problems. Common structure of a constraint program is consisted of the part to define variables and constraints on variables and the part to search for a valid value on each variable. This is the style of constraint-and-search. The structure of constraint logic program is as follows:

```
solve(Variables) :-
        setup_constraints(Variables),
        search(Variables).
```

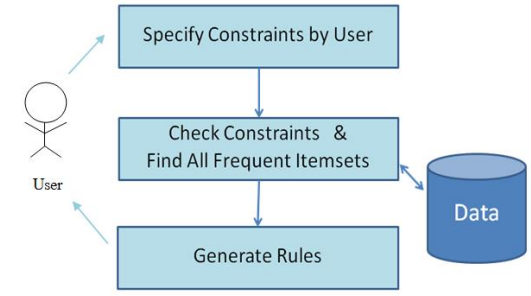
A constraint logic program to solve inequality A > B, in which both variables are integers in the range 1..5 can be shown as follows:

```
:- lib(ic).                 % include library
solve(R) :-
        R = [A,B],          % define variables
        R :: 1..5,          % define constraint
        A #> B,             % constraint over variables
        alldifferent([B]),
        labeling(R).
```

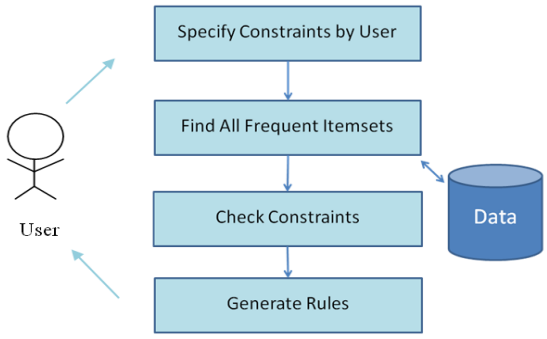
The predicate "labeling" is responsible for searching all possible values of A and B that comply with the constraints A > B and A ≠ B.

III. APRIORI-WITH-CONSTRAINT ALGORITHMS

Apriori algorithm [1], [2] is a general method that can efficiently generate all association rules satisfying minimum support and minimum confidence constraints. We argue that besides these basic constraints, users should specify their item of interest constraint over the method. We thus design two algorithms to support the extension of Apriori that takes into account user-specified constraints. The first algorithm, called Association rule discovery with Constraints In Frequent itemset mining (ACIF), applies constraints during the phase of frequent itemset mining. The second algorithm is called Association rule discovery with Constraints After Frequent itemset mining (ACAF) in which constraints are considered after all frequent itemsets are generated. The design frameworks of both algorithm are presented in Fig. 1. Detailed steps of ACIF and ACAF algorithms are shown in Fig. 2 and Fig. 3, respectively.



(a) The design of ACIF



(b) The design of ACAF

Figure 1. Design frameworks of Apriori-with-constraint

Algorithm ACIF
//Input : Database D,
Constraints: Length, Subset, NotSubset,
RHS_items, Mincon, Minsup.
//Output : Rules.

(1) L_1 = find_ frequent_1itemset(D)
(2) for (k =2; $L_{k-1} \neq \emptyset$; k++) {
(3) C_k = apriori_gen(L_{k-1}, Minsup)
(4) for each transaction t ∈ D { // scan D for counts
(5) C_1 = subset(C_k, t)
(6) for each candidate c ∈ C_1 { c.count++ }
(7) C_2 = checkcondition(Length, Subset, NotSubset, C_1)
(8) L_k={c ∈ C_2 | c.count ≥ Minsup}
(9) }
(10) }
(11) FreqSet = $\cup_k L_k$
(12) For each l ∈ FreqSet // l is frequent-itemset.
(13) k = | l | // size of frequent itemset
(14) m = $|H_m|$ // size of right hand side Items
(15) For each $h_{m+1} \in H_{m+1}$ {
(16) If h_{m+1} = RHS_items {
(17) conf = support(f_k) / support($f_k - h_{m+1}$)
(18) If conf >= Mincon {
(19) Rules = rule($f_k - h_{m+1}$) ⇒ h_{m+1}
(20) } Else
(21) delete h_{m+1} from H_{m+1}
(22) }
(23) }
(24) return Rules

Figure 2. Pseudocode of ACIF algorithm

Algorithm ACAF

//Input : Length, Subset, NotSubset, Minimum_support,

// Min_conf (Minimum confidence), RHS (right hand side Items).

//Output : Rules.

(1) L = find_all_frequent_itemset(Minimum_support)
(2) For each l ∈ L { //l = frequent_itemset
(3) If l.length > 1 {
(4) L_k = checkcondition(Length, Subset, NotSubset, L)
(5) }
(6) }
(7) Rules = generate_rule(L_k, Min_conf, RHS)
(8) return Rules

Figure 3. Pseudocode of ACAF algorithm

IV. Implementation and Experimental Results

A. CLP Implementation

Program and data in the constraint logic programming style are both in the same format, that is, a Horn clause. In our implementation, declaration of items and transactional database are in the list structure within the predicate "data" (as shown in Fig. 4). To run the program, user calls the predicate "association" and the running result will appear as shown in Fig. 5.

```
data([
[beer],[cannedmeat],[cannedveg],[confectionery],[dairy],[fish],
[freshmeat],[frozenmeal],[fruitveg],[softdrink],[wine]]
-
    [
        [beer,cannedmeat,confectionery,wine],
        [softdrink,fruitveg,wine,confectionery],
        [dairy,cannedmeat,fish],
        [dairy,freshmeat],
        [wine,softdrink,fruitveg,confectionery],
        [frozenmeal,confectionery,fish],
        [confectionery,wine],
        [fruitveg,frozenmeal,dairy,freshmeat],
        [softdrink,cannedmeat,dairy,cannedveg],
        [cannedveg,beer],
        [softdrink,cannedmeat,beer,cannedveg],
        [softdrink,dairy,beer]
    ]
).
```

Item

Transaction

Figure 4. Predicate *data* containing lists of items and transactions

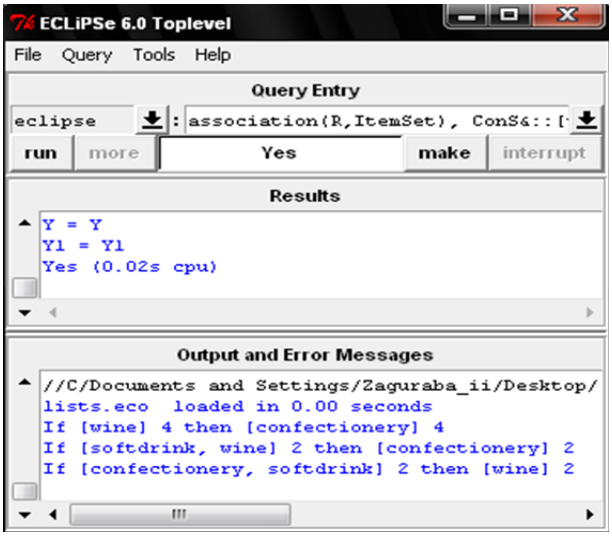


Figure 5. Running result of association rule mining

A program source code of ACAF algorithm that was implemented with the constraint logic programming language using ECLiPSe constraint system is given as follows:

```
%  User calls:  association(R,0,2,100)
:- compile("filename.txt"). % load file.
:- lib(sd).
:- lib(ic).
:- lib(ordset).
```

 ISBN: 978-1-61208-269-1

```
association(R,LengthI,MinSup,Conf) :-
      writeln('Please specify member in [[_]] :'),read(Subset),
      writeln('Please specify member do not need in [[_]] :'),
      read(NotSubset),
      writeln('Please specify member in [[_]] :'),read(Goal),
      data(Data),Data=_-_,
       ( count(I,2,6), fromto(Data,
      S0,S1,R),param(LengthI,Subset,NotSubset,Conf,MinS
      up,Goal) do
            ( S0=A-B,
                  findCL(A-B-MinSup,R-_-
      _,LengthI,Subset,NotSubset,Conf,Goal),
                  allUnion(I,R,NewItemSet),
                  S1=NewItemSet-B ),!   ).
 findCL(ItemSet-Items-MinSup,R-Items-MinSup,LengthI,
      Subset,NotSubset,Conf,Goal) :-
      (foreach(X,ItemSet), fromto(R,S1,S0,[]),
      param(Items,MinSup) do
          (supOK(X,Items,MinSup,LenItem) ->
                  S1=[X-LenItem|S0], ! ; S1=S0)
            ),       not1Item(R,Re1),
                  findLength(LengthI,Re1,Re),
                  findSubset(Subset,Re,R1),
                  findNotSubset(NotSubset,R1,R2),
                  findRule(R2-Items-Conf,Goal).
% Check Item set length > 1
not1Item(R,Re) :- R=[H-_|_],length(H,LenItem),
         LenItem =1 -> Re = [] ; Re = R .
% findLength(0,[[a,b]-2,[a,c]-2],R).
findLength(Cons,Re,R) :-
       (foreach(I,Re), fromto(R,S1,S0,[]), param(Cons) do
        I = X-_, length(X,LenItem),LenItem > Cons ->
        S1 = [ I | S0], ! ; S1=S0   ).
% findSubset([[b],[c]],[[a,b]-2,[b,c]-2,[b,c,d]-2],R1).
findSubset([],X, X).
findSubset([Subset|Tr],Re,R1) :-
         Subset = [] -> R1 = Re ;
        (foreach(I,Re), fromto(R,S1,S0,[]), param(Subset) do
          I = X-_,
          Subset1&::Subset, Y&::X, Subset1&=Y
            -> S1 = [ I | S0], ! ; S1=S0   ),
        findSubset(Tr,R,R1).
% findNotSubset([[b],[c]],[[a,b]-2,[b,c]-2,[b,c,d]-2],R1).
findNotSubset([],X, X).
findNotSubset([NotSubset|Tr],Re,R1) :-
       NotSubset = [] -> R1 = Re ;
      (foreach(I,Re), fromto(R,S1,S0,[]), param(NotSubset) do
       I = X-_,
       NotSubset1&::NotSubset, Y&::X, \+NotSubset1&=Y
         -> S1 = [ I | S0], ! ; S1=S0   ),
       findNotSubset(Tr,R,R1).
% findGoal([[a]],[a,b],R).
findGoal([],X, X).
findGoal([Subset|Tr],Re,R1) :-
         Subset = [] -> R1 = Re ;
         Subset1&::Subset, Y&::Re, Subset1&=Y -> R1 = Re,
         findGoal(Tr,Re,R1).
supOK(X,Items,MinSup,LenItem) :-
         (foreach(I,Items), fromto(R,S1,S0,[]), param(X) do
              (my_subset(X,I) -> S1 = [ I | S0], ! ; S1=S0) ),
          length(R,LenItem),
          LenItem >= MinSup.
% findRule
findRule(ItemSet-Items-MinConf,Goal) :-
    (foreach(Set,ItemSet), param(Items,MinConf,Goal) do
          Set = X-LenItem,
          findall(Re,powerset(X,Re),PwSet),
          (foreach(ItemX,PwSet),
          param(X,Items,LenItem,MinConf,Goal) do
          ( ItemX = X ; ItemX = [] -> ! ;
          createRule(ItemX,X,Re),findGoal(Goal,Re,R1) ->
            supOK(ItemX,Items,0,LenItemX),
            conOk(LenItem-LenItemX-MinConf,ItemX,R1),! ; !
            ) ) ).
  % createRule([a],[a,b,c],Result).
  createRule([],X,X).
  createRule([H|Tr],X,Result) :-
        delete(H,X,Re),createRule(Tr,Re,Result).
  % Check Confident
  conOk(LenItem-LenItemX-MinConf,ItemX,Re) :- Re11 is
        (LenItem/LenItemX)*100,Re11 >= MinConf -> write('If
        '),
        write(ItemX),write(' '),write(LenItemX),write(' then '),
        write(Re),write(' '),writeln(LenItem) ;! .
  % my_subset compare([1],[1,2]) return T or F
  my_subset(Sub,S) :- toSet(Sub,OrdSub), toSet(S,OrdS),
                          ord_subset(OrdSub,OrdS).
  allUnion(I,ItemSet,NewItemSet) :-  combi(ItemSet,R_),
        flatten(R_,R),
        (foreach(X,R), fromto(NewItemSet_,S1,S0,[]), param(I)
        do  First-Sec=X,
          (unionN(I,First-Sec,Out) -> S1=[Out|S0], !;S1=S0)  ),
          toSet(NewItemSet_,NewItemSet).

  unionN(N,First-Sec,Out) :- toSet(First,F), toSet(Sec,S),
                              ord_union(F,S,Out),
                              length(Out,Len),Len=N.
  combi([],[]).
  combi([H|T],[HR|TR]) :- (foreach(X,T), foreach(Y,HR),
        param(H) do
        X = Set2-_,H = Set1-_ ,  Y=Set1-Set2 ),
        combi(T,TR).
  toSet(X,S) :- list_to_ord_set(X,S).
  % powerset([a,b],X).
  powerset([],[]).
  powerset([_|Rest],L) :- powerset(Rest,L).
  powerset([X|Rest],[X|L]) :- powerset(Rest,L).
% ===== End of ACAF program ==============
```

B. Experimentation with Chess data

To test correctness and effectiveness of ACIF and ACAF programs, we use the Chess dataset obtained from the website http://fimi.ua.ac.be/data/. The dataset contains 3196 records, each record has 37 attributes, or items in the context of association mining. The first and last records of Chess data can be shown as follows:

```
data([[1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14],[
  15],[16],[17],[18],[19],[20],[21],[22],[23],[24],[25],[26],[27],[
  28],[29],[30],[31],[32],[33],[34],[35],[36],[37],[38],[39],[40],[
  41],[42],[43],[44],[45],[46],[47],[48],[49],[50],[51],[52],[53],[
  54],[55],[56],[57],[58],[59],[60],[61],[62],[63],[64],[65],[66],[
  67],[68], [69],[70],[71],[72],[73],[74],[75]
]-[[1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,34,36,38,40,42,
  44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74],
  …
[2,4,5,8,9,11,13,16,17,19,21,23,26,27,30,31,35,36,38,40,42,44,
  46,48,51,52,54,56,58,61,62,64,67,68,71,73,74]  ] ).
```

In our CLP implementations of association rule mining with constraints, users can specify three kinds of constraints: (1) items that must appear in the association rules, (2) items that must not appear in the rules, and (3) items that must appear in the consequence part of the rule. Constraints on items can use a conjunction (AND), a disjunction (OR), and an empty list of items to identify no constraint. These constraints have to be specified prior to the generation of association rules. We test the performance of ACIF and ACAF programs against the original Apriori (which is also implemented with the CLP paradigm for a fair comparison), and then compare the results in terms of computation time and number of association rules. Figs. 6 and 7 demonstrates the running time and number of rule comparisons, respectively. For the case of no further constraint (except the basic minimum support and confidence) are identified, the three programs can discover the same set of association rules, but ACIF and ACAF take more time to find such rules.

For the search of rules with constraints on rule length and items to appear/exclude, Apriori cannot perform such association mining because the algorithm does not support constraints on specific items. It can be noticed that ACIF may run faster than ACAF, but it is incomplete in the sense that there are some rules that should be appeared in the final result but are excluded during the phase of frequent itemset mining. We thus can conclude that ACIF and ACAF are correct, but ACAF is better than ACIF in terms of completeness.

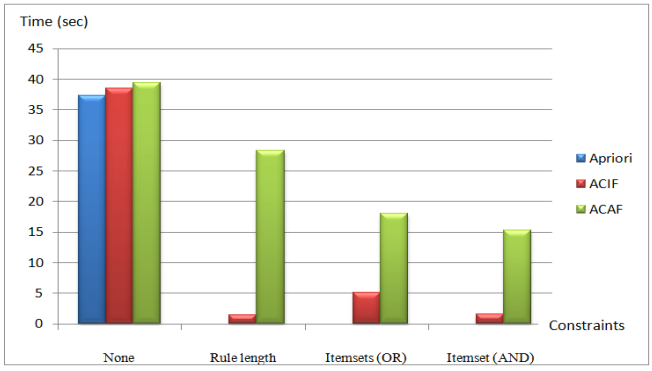


Figure 6. Running time comparison of ACIF and ACAF programs against Apriori program

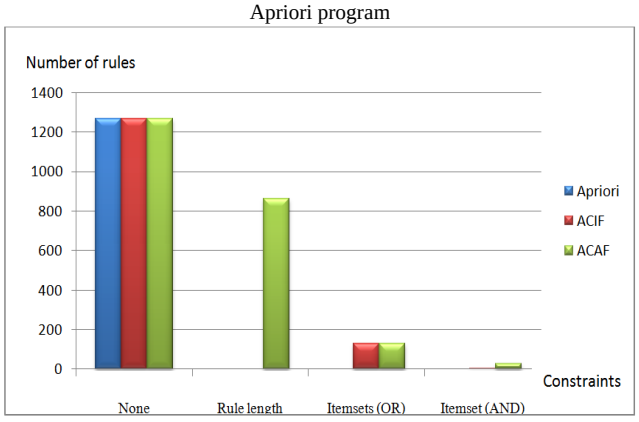


Figure 7. ACIF, ACAF, and Apriori comparison in terms of number of association rules discovered by the program

C. *Experimentation with Customer Churn data*

To test effectiveness of ACAF program over various constraints, we use the Customer Churn dataset obtained from the website http://www.sgi.com/tech/mlc/db/. The dataset contains information of 3333 customers. In the original data set, each record has 21 features (or attributes) in which the last one is the label churn/non-churn customer. The first step of our experimentation is feature selection; the selected 12 features are state, account length, area code, international plan, voice mail plan, number vmail messages, total day calls, total eve calls, total night calls, total intl calls, number customer service calls, and churn. The other nine features are removed because of their insignificance in inducing the association model.

We performed a series of eight experiments with the customer churn data set. Minimum support threshold in each experiment is 50 (that means there must be at least 50 records satisfying the rule's content), whereas the minimum confidence is 80%. Additional constraints are as follows:

Exp. 1: No other constraint.
Exp. 2: The results must contain the feature churn_False.
Exp. 3: The results must contain rules that has at least three features.
Exp. 4: The results must NOT contain the feature 'churn_False'.
Exp. 5: The results must contain the feature 'churn_False' at the then-part of the rule.
Exp. 6: The results must contain either the feature 'churn_False', or 'churn_True'.
Exp. 7: The results must contain both the feature 'churn_True' and 'vMailPlan_no'.
Exp. 8: The results must contain rules that has at least three features, must contain both 'churn_False' and 'vMailPlan_no', the results must NOT contain either the feature 'vMailMessage_0', or 'intlCalls_2', and the target clause of the rules must be 'churn_False'. (Running result is shown in Fig. 8.)

Running time in each experiment and number of rules reported as the association mining results after constraining with different conditions are varied. We comparatively illustrate the experimental results in Fig. 9.

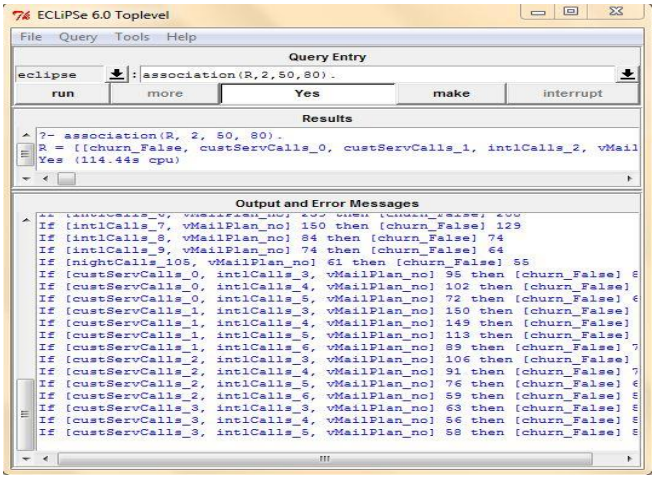


Figure 8. Running result of experiment 8

 ISBN: 978-1-61208-269-1

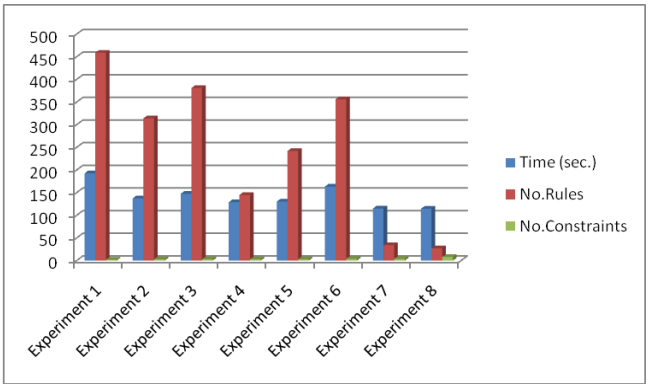


Figure 9. A comparison of computational time usage and number of rules received from varying constraints in each of the eight experiments

V. CONCLUSION AND FUTURE WORK

Association rule discovery is one major problems in the areas of data mining, statistical learning, and business intelligence. The problem concerns finding frequent patterns, or itemsets, hidden in a large database. Finding such frequent itemsets has become an important task because it reveals associations, correlations, and many other interesting relations among items in the transactional databases. We suggest that the problem of frequent itemset and association rule mining can be efficiently implemented with the incorporation of constraints.

We design the two versions of association mining with constraint algorithms called Association rule discovery with Constraints In Frequent itemset mining (ACIF) and Association rule discovery with Constraints After Frequent itemset mining (ACAF). We demonstrated that the proposed algorithms can be concisely implemented with high-level declarative language using ECLiPSe, a constraint logic language. Coding in declarative style takes less effort because pattern matching is a fundamental feature supported by most logic-based languages. Experimentation to verify effectiveness of the proposed methods has been performed and compared against the well-known Apriori method. The results confirm the correctness of the ACIF and ACAF programs and also reveal the power of constraints that have been applied over the frequent itemsets. We focus our future research on the design of constraint formulating and processing to optimize the speed and storage requirement.

ACKNOWLEDGMENT

This research work has been funded by grants from the National Research Council of Thailand (NRCT) and Suranaree University of Technology.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases,” in Proc. ACM SIGMOD, 1993, pp. 207-216.

[2] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in Proc. VLDB, 1994, pp. 487-499.

[3] K. R. Apt and M. Wallace, Constraint Logic Programming using ECLiPSe, Cambridge University Press, 2007.

[4] S. Bistarelli and F. Bonchi, “Soft constraint based pattern mining,” Data and Knowledge Engineering, vol. 62, 2007, pp. 118-137.

[5] I. Bratko, Prolog Programming for Artificial Intelligence, 3rd ed., Pearson, 2001.

[6] A. Cegler and J. Roddick, “Association mining,” ACM Computing Surveys, vol.38, no.2, 2006.

[7] L. De Raedt, T. Guns, and S. Nijssen, “Constraint programming for itemset mining,” in Proc. KDD, 2008, pp. 204-212.

[8] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, “Knowledge discovery in databases: an overview,” AI Magazine, vol. 13, no. 3, 1992, pp. 57-70.

[9] J. Han and M. Kamber, Data Mining: Concepts and Techniques, 2nd ed., Morgan Kaufmann, 2006.

[10] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in Proc. ACM SIGMOD, 2000, pp. 1-12.

[11] J. Hu and X. Li, “Association rules mining including weak-support modes using novel measures,” WSEAS Trans. on Computers, vol. 8, no. 3, 2009, pp. 559-568.

[12] M.C. Hung, S.Q. Weng, J. Wu, and D.L. Yang, “Efficient mining of association rules using merged transactions,” WSEAS Trans. on Computers, vol. 5, no. 5, 2006, pp. 916-923.

[13] N. Kerdprasop and K. Kerdprasop, “Recognizing DNA splice sites with the frequent pattern mining technique,” Int. J. of Mathematical Models and Methods in Applied Science, vol.5, no. 1, 2011, pp. 87-94.

[14] P. Kongchai, K. Kerdprasop, and N. Kerdprasop, “Association rule discovery with constraint logic programming,” in Proc. 11th WSEAS Int. Conf. on Computational Intelligence, Man-Machine Systems and Cybernetics, 2012, pp. 135-140.

[15] R. Kuusik and G. Lind, “Algorithm MONSA for all closed sets finding: basic concepts and new pruning techniques,” WSEAS Trans. on Information Science & Applications, vol. 5, no. 5, 2008, pp. 599-611.

[16] S.-H. Nienhuys-Cheng and R.D. Wolf, Foundations of Inductive Logic Programming, Springer, 1997.

[17] J. Pei and J. Han, “Can we push more constraints into frequent pattern mining?” in Proc. ACM SIGKDD, 2000, pp. 350-354.

[18] J. Pei, J. Han, and L. Lakshmanan, “Pushing convertible constraints in frequent itemset mining,” Data Mining and Knowledge Discovery, vol. 8, 2004, pp. 227-252.

[19] R. Srikant, Q. Vu, and R. Agrawal, “Mining association rules with item constraints,” in Proc. KDD, 1997, pp. 67-73.

[20] H. Sug, “Discovery of multidimensional association rules focusing on instances in specific class,” Int. J. of mathematics and Computers in Simulation, vol. 5, no. 3, 2011, pp. 250-257.

[21] H. Toivonen, “Sampling large databases for association rules,” in Proc. VLDB, 1996, pp. 134-145.

[22] G. Yu, S. Shao, and X. Zeng, “Mining long high utility itemsets in transaction databases,” WSEAS Trans. on Information Science & Applications, vol. 5, no. 2, 2008, pp. 202-210.

[23] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, “Parallel algorithm for discovery of association rules,” Data Mining and Knowledge Discovery, vol. 1, 1997, pp. 343-374.

 ISBN: 978-1-61208-269-1