A Knowledge Mining Component for Computational Health Informatics

Nittaya Kerdprasop and Kittisak Kerdprasop

***Abstract*—**Computational health informatics is an emerging field of research focusing on the devise of novel computational techniques to facilitate healthcare and a variety of medical applications. Healthcare organizations at the present day regularly generate huge amount of data in electronic form stored in databases. These data are valuable resource for automatic discovering of useful knowledge, known as knowledge mining or data mining, to gain insight knowledge and to support patient-care decisions. During the past decades there has been an increasing interest in devising database and learning technologies to automatically induce knowledge from clinical and health data using imperative and object-oriented programming styles. In this paper, we propose a different scheme using declarative programming, implemented with functional and logic-based languages, in which we argue to be more appropriate for the knowledge intensive tasks. Easy knowledge transfer to the knowledge base content is demonstrated in this paper to confirm the appropriateness of high level declarative scheme. Our system includes three major knowledge mining tasks: data classification, association analysis, and clustering. We demonstrate knowledge deployment aspects through the trigger creation and the automatic generation of knowledge base for the medical decision support system. These knowledge deployment examples illustrate advantages of the high level logic and functional scheme.

***Keywords*—** Computational health informatics, Healthcare decision support, Knowledge mining, Declarative programming, Prolog and Erlang languages.

I. Introduction

KNOWLEDGE mining is the process of deriving new and useful knowledge from vast volumes of data and other background information. Derived knowledge can possibly be patterns of data represented in summarized form, relationships among data represented as rules, or representatives of data subgroups represented by majority values. Knowledge is a valuable asset to most organizations as a substantial source to enhance understanding of data relationships and support better decisions to increase organizational competency.

Manuscript received March 10, 2012: Revised version received June 12, 2012. This work was supported by grants from the National Research Council of Thailand (NRCT) and Suranaree University of Technology through the funding of Data Engineering Research Unit.

N. Kerdprasop is an associate professor and the director of Data Engineering Research Unit, School of Computer Engineering, Suranaree University of Technology, 111 University Avenue, Muang District, Nakhon Ratchasima 30000, Thailand (phone: +66-44-224-432; fax: +66-44-224-602; e-mail: nittaya@sut.ac.th).

K. Kerdprasop is with the School of Computer Engineering and Data Engineering Research Unit, Suranaree University of Technology, Nakhon Ratchasima, Thailand (e-mail: KittisakThailand@gmail.com).

The term *knowledge assets* refer to any organizational intangible possessions related to knowledge such as know-how, expertise, intellectual property. In clinical companies and computerized healthcare applications, knowledge assets include order sets, drug-drug interaction rules, guidelines for practitioners, and clinical protocols [19]. Knowledge assets can be stored in data repositories either in implicit or explicit form. Explicit knowledge can be managed through the existing tools available in the current database technology. Implicit knowledge, on the contrary, is harder to achieve and retrieve.

Implicit knowledge acquisition can be achieved through the availability of the knowledge-mining system. *Knowledge mining* is the discovery of hidden knowledge stored possibly in various forms and places in large data repositories. Automatic knowledge acquisition can be achieved through the availability of the knowledge discovery system. The discovered knowledge facilitates expert decision support, data exploration and explanation, estimation of future trends, and prediction of future outcomes based on present data.

In health and medical domains, knowledge has been discovered in different forms such as association, classification rules, clustering, trend or temporal pattern analysis [41]. The discovered knowledge facilitates expert decision support, diagnosis and prediction.

In this paper we present the design and implementation of a knowledge-mining system, called SUT-Miner, to support a high-level decision in medical domains. The system is also applicable to any domain that requires a knowledge-based decision support. A rapid prototyping of the proposed system is provided in a declarative style using second-order Horn clauses [30]. The intuitive idea of our design is that for such a complicated knowledge-based system program coding should be done declaratively at a high level to alleviate the burden of programmers [44]. The advantages of declarative style are thus the decrease in program development time and the increases in expressiveness of knowledge representation and efficiency of knowledge utilization.

A declarative approach to the development of knowledge discovery system using logic-based language has long been an interesting research topic among data mining and machine learning researchers. For the classification task, FOIL [37] and PROGOL [29] were two examples of successful logic-based systems. Tree-based concept induction [23], [37] and rule induction [29] are major approaches normally adopted for the classification task. Association mining task was mostly based

on the well-known APRIORI [2] algorithm. WARMR [12] system upgraded APRIORI algorithm to discover frequent patterns. Its extension [13] was developed to discover frequent Datalog patterns and relational association rules. Data clustering based on logic programming and first-order logic was mainly an extension of k-means [28] clustering algorithm. K-prototypes [22] extended k-means clustering to work on first-order representation. KBG [5], [6], TIC [3], COLA-2 [14], and RDBC [21] were all first-order clustering systems.

All logic-based knowledge discovery systems proposed in the literature considered a single task. Our work, on the contrary, is a proposal of a knowledge discovery system designed as an integrated environment storing a repertoire of tools for discovering various kinds of knowledge. We also demonstrate the speedup of k-means clustering on large data with biased sampling technique. The biased clustering is implemented with functional programming using Erlang language. Both logic-based and functional styles of programming are considered declarative method, in which coding is concise and effective.

The outline of this paper is as follows. Section 2 reviews related work on computational health informatics that apply the knowledge mining techniques. Section 3 briefly discusses the basic of logic programming concepts including the notion of higher-order predicates and some preliminaries on three main knowledge mining tasks, i.e. classification, association mining, and clustering. Section 4 presents the design and implementation of our SUT-Miner system. Section 5 demonstrates knowledge deployment examples. Section 6 concludes the paper and discusses future research directions.

II. Related Work

In recent years we have witnessed increasing number of applications on knowledge mining from biomedicine, clinical and health data [4], [15], [20], [26], [45]. Roddick and colleagues [38] discussed the two categories of mining techniques applied over medical data: explanatory and exploratory. Explanatory mining refers to techniques that are used for the purpose of confirmation or making decisions. Exploratory mining is data investigation normally done at an early stage of data analysis in which an exact mining objective has not yet been set.

Explanatory mining in healthcare data has been extensively studied in the past decades employing various techniques. Bojarczuk et al [7] applied genetic programming method to discover classification rules from medical data sets. Thongkam et al [43] studied breast cancer survivability using AdaBoost algorithms. Ghazavi and Liao [16] proposed the idea of fuzzy modeling on selected features medical data. Huang et al [18] introduced a system to apply mining techniques to discover rules from health examination data. Then they employed a case-based reasoning to support the chronic disease diagnosis and treatments. The work of Zhuang et al [46] also combined mining with case-based reasoning, but applied a different mining method. Biomedical discovery support systems are recently proposed by a number of researchers [8], [10]. Some work [40], [45] extended medical databases to the level of data warehouses.

Exploratory, as oppose to explanatory, is rarely applied to medical domains. Among the rare cases, Nguyen and Kawasaki [32] introduced knowledge visualization in the study of hepatitis patients. Palaniappan and Ling [35] applied the functionality of OLAP tools to improve visualization.

It can be seen from the literature that most medical knowledge discovery systems have applied only some mining techniques such as classification rules mining, association mining to discover hidden knowledge. We, on the contrary, design a knowledge-mining system aiming at providing a suite of tools to facilitate users and healthcare practitioners on discovering different kinds of knowledge from their data and background knowledge repositories. The deployment of induced knowledge has also been demonstrated through the creation of database triggers and knowledge base contents.

Triggers are a major concept of active databases, which extend traditional database systems with the mechanism to respond automatically to some specific events. Upon the occurrence of the specified event, the rule condition is evaluated. If the condition is satisfied, then some actions are performed. Although triggers are important database feature on consistency monitoring, their deployment is still limited due to the fact that creating complex trigger rules is not an easy task [9], [25]. Tools and environments to aid users and database programmers are certainly needed.

The employment of triggers to achieve active behavior is quite rare in medical domain. Most of the proposed methods are for detecting static events such as the discovery of relationships that suggest risks of adverse events in patient records [34], [42], detection of dependency patterns of process sequences for curing brain stroke patients [27], the generation of rules to annotated protein data in medical database [24], or the exploration of environmental health data [3].

The work presented in this paper differs from those appeared in the literature in that we propose to employ knowledge discovery techniques to semi-automatically create trigger rules. The utilization of our proposed method is to increase consistency in medical database. Any database modification events violating constraints will be alerted and undone. The system designed by Agrawal and Johnson [1] is also to support medical database but in a different aspect; they concentrate on security and privacy preservation of patients and other sensitive health data. Another example of induced knowledge deployment to automatically generate knowledge base in the medical decision support system is also presented in this paper.

III. Preliminaries

A. Declarative Programming with Prolog

Prolog is a programming language that are based on the concept of mathematical first-order logic. Each statement in Prolog program is a clause, which is a disjunction of literals

(atomic symbols or their negations) such as $p \vee q$ and $\neg p \vee r$. A statement is in clausal form if it is a conjunction of clauses such as $(p \vee q) \wedge (\neg p \vee r)$. Logic programming is a subset of first-order logic in which clauses are restricted to Horn clauses. A Horn clause, named after the logician Alfred Horn [33], is a clause that contains at most one positive literal such as $\neg p \vee \neg q \vee r$. Horn clauses are widely used in logic programming because their satisfiability property can be solved by resolution algorithm (an inference method for checking whether the formula can be evaluated to true).

A Horn clause with no positive literal, such as $\neg p \vee \neg q$, which is equivalent to $\neg(p \wedge q)$, is called *query* in Prolog and can be interpreted as '*:- p, q'* in which its value (true/false) to be proven by resolution method. A clause that contains exactly one positive literal such as *r* is called a *fact* representing a true statement, written in clausal form as '*r :-'* in which the condition part is empty and that means *r* is unconditionally true. Therefore, facts are used to represent data. A Horn clause that contains one positive literal and one or more negative literals such as $\neg p \vee \neg q \vee r$ is called a *definite clause* and such clause can equivalently written as $(p \wedge q) \rightarrow r$ which in turn can be represented as a Prolog *rule* as *r :- p, q*.

The symbol '*:-'* is intended to mean '$\leftarrow$', which is implication in first-order logic (it stands for 'if'), and the symbol ',' represents the operator $\wedge$ (or 'AND'). In Prolog, rules are used to define procedures and a Prolog program is normally composed of facts and rules. Running a Prolog program is nothing more than posing queries to obtain true/false answers. The advantages of using logic programming are the flexible form of query posing and the additional information regarding variable instantiation obtained from the Prolog system once the query is evaluated to be true.

The symbols *p, q, r* are called *predicates* in first-order logic programming and they can be quantified over variables such as *r(X) :- p(X,Y), q(Y)*. This clause has the same meaning as $\forall X\,(p(X,Y) \wedge q(Y) \rightarrow r(X))$.The scope of variables is within a clause (delimit the end of clause with a period). Horn clauses are thus the fundamental concept of logic programming.

Higher-order predicate is a predicate in a clause that can quantify over other predicate symbols [30], [31]. As an example, besides the rule *r(X):- p(X,Y), q(Y)*, if we are also given the following five Horn clauses (or facts): *p(1, 2). p(1, 3). p(5, 4). q(2). q(4)*. By asking the query: *?- r(X)*, we will get the response as '*true*' and also the first instantiation information as *X=1*. If we want to know all instantiations that make *r(X)* to be true, we may ask the query: *?- findall(X, r(X), Answer).*We will get the response: *Answer = [1,5]*, which is a set of all answers obtained from the predicate *r(X)* according to the given facts. The predicate symbol *findall* quantifies over the variables *X*, *Answer*, and the predicate *r*. The predicate *findall* is thus called a higher-order predicate

B. Tree-based Classification

Decision tree induction [36] is a popular method for inducing knowledge from data. Popularity is due to the fact that mining result in a form of decision tree is interpretability, which is more concern among practitioners than a sophisticated method but lack of understandability. A decision tree is a hierarchical structure with each node contains decision attribute and node branches corresponding to different attribute values of the decision node. The goal of building decision tree is to partition data with mixing classes down the tree until the leaf nodes contain pure class.

In order to build a decision tree, we need to choose the best attribute that contributes the most towards partitioning data to the purity groups. The metric to measure attribute's ability to partition data into pure class is *Info*, which is the number of bits required to encode a data mixture. To choose the best attribute, we have to calculate information gain, which is the yield we obtained from choosing that attribute. The information gain calculates yield on data set before splitting and after choosing attribute with two or more splits. The gain value of each candidate attribute is calculated. Then choose the maximum one to be the decision node. The process of data partitioning continues until the data subset has the same class label.

C. Association Mining

Association mining is the discovery of relationships or correlations between items in a database. Let $I = \{i_1, i_2, i_3, ..., i_m\}$ be a set of *m* items and $DB = \{ C_1, C_2, C_3, ..., C_n\}$ be a database of *n* cases or observations and each case contains items in *I*. A *pattern* is a set of items that occur in a case. The number of items in a pattern is called the length of the pattern. To search for all valid patterns of length 1 up to *m* in large database is computational expensive. For a set *I* of *m* different items, the search space for all distinct patterns can be as huge as 2^m-1. To reduce the size of the search space, the *support* measurement has been introduced [1]. The function *support*(*P*) of a pattern *P* is defined as a number of cases in DB containing *P*. Thus, $support(P) = |\{T \mid T \in \text{DB},\ P \subseteq T\}|$. A pattern *P* is called *frequent pattern* if the support value of *P* is not less than a predefined minimum support threshold *minS*. It is the *minS* constraints that help reducing the computational complexity of frequent pattern generation. The *minS* metric has an anti-monotone property and is applied as a basis for reducing search space of mining frequent patterns in algorithm Apriori [2].

D. Data Clustering with K-Means

Clustering refers to the iterative process of automatic grouping of data based on their similarity. There exist a large number of clustering techniques, but the most classical and popular one is the k-means algorithm [28]. Given a data set containing *n* objects, k-means partitions these objects into *k* groups. Each group is represented by the centroid, or central point, of the cluster. Once cluster means or representatives are selected, data objects are assigned to the nearest centers. The

algorithm iteratively selects new better representatives and reassigns data objects until the stable condition has been reached. The stable condition can be observed from cluster assigning that each data object does not change its cluster.

IV. The Design and Implementation

A. System Design for Knowledge Mining

The process of knowledge mining is complex and iterative in its nature. We thus design the system (Figure 1) to be composed of two phases: knowledge induction and knowledge inferring. Knowledge induction is the back-end of the system responsible for acquiring and discovering new and useful knowledge. Usefulness is to be validated at the final step by human experts. Discovered knowledge is stored in the knowledge base to be applied to solve new cases or create new knowledge in the knowledge inferring phase, which is the front-end of the proposed system. The SUT-Miner system obtains input from heterogeneous data sources. Therefore, redundancy, incompleteness, noise can be expected from the input data. The Pre-DM component has been designed to clean, transform the format, and select only relevant data. The DM component is for performing various mining tasks including classification, association, and clustering.

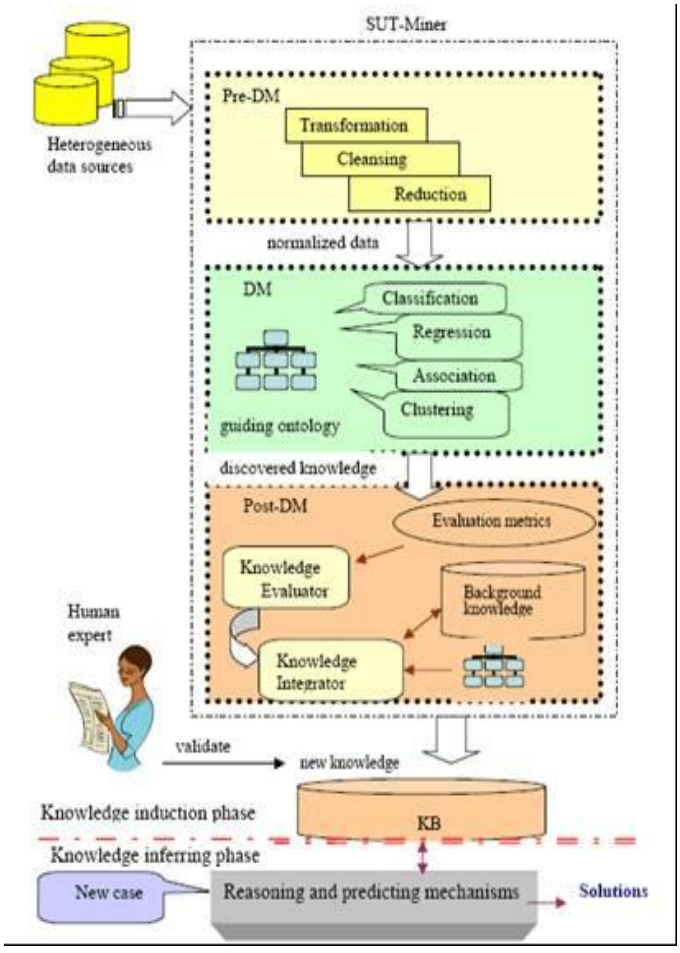


Fig. 1 The design of SUT-Miner for health informatics

The Post-DM component composed of two main features: knowledge evaluator and knowledge integrator. These features perform functionality aiming at a feasible knowledge deployment. Knowledge evaluator involves evaluation, based on corresponding measurement metrics, of the mining results. Knowledge integrator examines the induced patterns to remove redundant knowledge. Ontology has also been applied at this step to provide essential semantics regarding the domain problems.

B. Implementing a Tree-based Mining Engine

We present in Figure 2 the coding of data classification module based on decision tree induction (ID3) algorithm [36]. Prolog code is based on the syntax of SWI Prolog (www.swi-prolog.org).

The *tree_classifier*, which is the main module in Figure 2, calls the *init* procedure and starts creating edges and nodes of the decision tree via the predicates *getNode* and *create_edge*, respectively. The ID3 algorithm is implemented in a module *create_edge_one_level*.

```
tree_classifier(Min):-
        init(AllAttr,EdgeList),
        getnode(N),
        create_edge_onelevel(N,AllAttr,EdgeList),
        addKnowledge,
        selectRule(Min,Res),
        maplist(writeln,Res).

create_edge_onelevel(_,_,[]):-!.
create_edge_onelevel(_,[],_):-!.
create_edge_onelevel(N,AllAttr,EdgeList):-
        create_nodes(N,AllAttr,EdgeList).

create_nodes(N,AllAttr,[H1-H2/PB-NB|T]):-
       getnode(N1),
        assert(edge(N,H1=H2,N1)),
        assert(node(N1,PB-NB)),
        append(PB,NB,AllInst),
        ( (PB\==[], NB\==[])->
            (cand_node(AllAttr,AllInst,AllSplit),
             min_cand(AllSplite,[V,MinAttr,Split]),
             delete(AllAttr,MinAttr,Attr2),
             create_edge_onelevel( N1,Attr2,Split))
         ;  true ),
       create_nodes(N,AllAttr,T).

create_nodes(_,_,[]):-!.
create_nodes(_,[],_):-!.

addKnowledge :-
        findall([A],pathFromRootToLeaf(A,_),Res),
        retractall(_>>_>>_),
        maplist(apply(assert),Res).

selectRule(V,Res):-
        findall(N>>X>>Class,(X>>Class>>N,N>=V),Res1),
        sort(Res1,Res2),
        reverse(Res2,Res).
```

Fig. 2 A tree-based mining engine in Prolog

C. Implementing Association Mining Engine

We implement the association mining module based on the algorithm APRIORI [2]. The implementation (Figure 3) shows only the first pass of algorithm; that is, the generation of frequent itemsets. The second pass, which is the generation of association rules from frequent itemsets, can be easily extended from the given code.

```
association_mining :-
        input(Data), min_support(V),
        makeC1(C), makeL(C,L), apriori_loop(L,1).

apriori_loop(L,N) :- length(L) is 1,!.
apriori_loop(L,N) :- N1 is N+1,
        makeC(N1,L,C), makeL(C, Res),
        apriori_loop(Res, N1).

makeC1(Ans) :- input(D),
        allComb(1, ItemSet, Ans2),
        maplist(countSS(D), Ans2, Ans).

makeC(N,ItemSet,Ans) :- input(D),
        allComb(2,ItemSet, Ans1),
        maplist(flatten, Ans1, Ans2),
        maplist(list_to_ord_set, Ans2, Ans3) ,
        list_to_set(Ans3,Ans4),
        include(len(N), Ans4, Ans5),
        maplist(countSS(D), Ans5, Ans).
                %scan database to find: List+N

makeL(C,Res) :- include(filter, C, Ans),
        maplist(head, Ans, Res).

filter(_+N) :- input(A), length(A, I),
            min_support(V),
            N>=(V/100)*I.

head(H+_,H).
% arbitrary subset of the set containing given
%    number of elements
comb(0, _, []).
comb(N, [X|T], [X|Comb]) :- N>0, N1 is N-1,
comb(N1,T,Comb).
comb(N,[_|T],Comb) :- N>0, comb(N,T,Comb).

allComb(N,I,Ans) :- setof( L, comb(N, I, L), Ans).

countSubset(A,[],0).
countSubset(A,[B|X],N) :- not(subset(A,B)),
countSubset(A,X,N).
countSubset(A,[B|X],N) :- subset(A,B),
        countSubset(A,X,N1),
        N is N1+1.
countSS(SL, S, S+N) :- countSubset(S, SL, N).
```

Fig. 3 Association mining engine in Prolog

Main predicate of this module is *association_mining*. Upon invocation, this predicate obtains input data from the predicate *input(Data)*, and get the minimum support value through the predicate *min_support(V)*. Then the main predicate starts by making candidate and large itemsets of length one, two, three, and so on (through the predicates *makeC1*, *makeL*, and *apriori_loop*, respectively). All highlighted terms are higher-order predicates: *maplist*, *include*, and *setof*.

The predicate *maplist* takes three arguments. This predicate applies its first argument, which is also a predicate, to each element of a list appeared in the second argument. The result is a list in the third argument. Predicate *include* takes another predicate as its first argument and adds the result obtained from the first argument to the list in second argument. The result appears as a list in the third argument. The predicate *setoff* collects each answer as a list in its third argument.

D. Implementing Clustering Engine

Figure 4 demonstrates the implementation of k-means clustering [17], [28]. The main predicate is *clustering* in which the number of clusters (*k*) has to be specified and data are to be included.

```
clustering(K) :- makeInitCluster(K, AllClust),
       assignPoint(AllClust,Data,Start,AllPoint),
       OldClust=AllClust,
      repeatCompute(K,AllPoint,OldClust).
makeInitCluster(K, AllClust):- initClust(K, 1, AllClust).
initClust(K,L0,[]) :- L0>K , ! .
initClust(K,L0,[L0*L|T]) :- instance(L0,_,L),
            L1 is L0+1, initClust(K, L1, T).
assignPoint(_, U, M, []) :- M>U, !.
assignPoint(AllClust, U, M, [M-V-A|T]) :-
        maplist(freq(M),AllClust,Res),cmax(Res, A*V),
        M1 is M+1,assignPoint(AllClust, U, M1, T).
freq(X,N*Y,N*F) :-   instance(X, _, L1),
        intersection(L1, Y, I), length(I, F).
cmax(L,A*V) :- maplist(cvalue, L, L2),
        max_list(L2, V), member(A*V, L), !.
cvalue(_*V, V).
reComputeCenter(K, S, AllPoint, []) :- S>K, !.
reComputeCenter(K, S, AllPoint, [S*NewC|T]) :-
        findall(P, member(P-_-S, AllPoint), Z),
        allPointAtAllAttr(Z, NewC), S1 is S+1,
        reComputeCenter(K, S1, AllPoint, T).
allPointAtAllAttr(AllP, NewClusters) :-
        findall(AttName, (attribute(AttName,_),
                AttName\==class), AttNameL),
        maplist(allPoint(AllP), AttNameL, NewClusters).
allPoint(AllP, Att, A) :-
        findall(Att=V, (instance(X, _, K),
        member(X, AllP), member(Att=V, K)) , Z),
        maxFreq(Z, A*V).
repeatCompute(K, AllPoint, OldClust) :-
       reComputeCenter(K,Start,AllPoint,NewClus),
      ( OldClust==NewClus ->
        writeln('-No-cluster-changes***EndProcess*');
        ( assignPoint(NewClus,Data,Start,AllPoint2),
        repeatCompute(K, AllPoint2, NewClus))).
```

Fig. 4 Data clustering engine in Prolog

The predicate *makeInitCluster* creates initial *k* clusters with randomized *k* centroids, then assign each data to the closest centroid through the predicate *assignPoint*. Note that the symbol '*', such as those appear in the predicate *cmax(Res, A*V)* and *freq(X, N*Y, N*F)*, refers to the data format to represent *Attribute*Value*; it does not mean multiplication. In Prolog, numerical computation will occur in a clause with the predicate *'is'*, such as *S1 is S + 1* in the *reComputeCenter* procedure.

The iteration step, *repeatCompute* predicate, re-computes the new *k* centroids and then re-assign each data point to the new closest centroid. Iteration stops when all data do not change their clusters. The source code presented in Figure 4 works with categorical data. For numerical or data with mixing types, the distance measurement has to be modified.

E. Speedup Clustering with Biased Sampling Technique

A standard k-means algorithm groups data by firstly assigning all data points to the closest clusters, then determining the new cluster mean of each cluster based on the average value of its members. The algorithm repeats these two steps until it converges. Performance of the algorithm depends on the numbers of data points, clusters, and iterations.

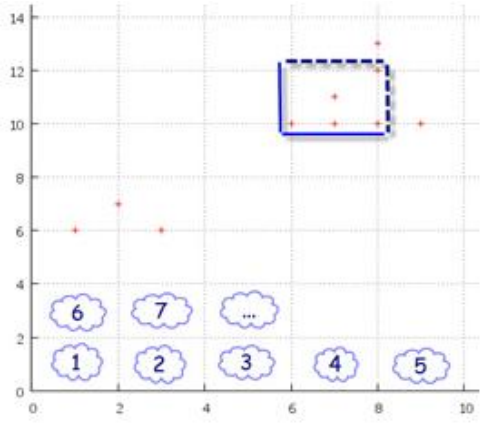
To speed up the clustering process, we investigate the density-biased sampling technique. Instead of simply randomly selecting data for clustering, we evaluate data density using a sliding window model and selectively draw samples with rejection criteria from the dense area. The idea has been schematically shown in Figure 5. In our framework, the windows are of fixed size and each window has length *W* along each dimension. That is, the window has size $W \times W$ in two dimensional (2D) data, but the size increases to $W \times W \times W$ in three dimensional (3D) data, and so forth. Therefore, to cluster high dimensional data we assume the feature selection technique such as principal component analysis or support vector machine has been applied to the data set prior to the density estimation step.

Once the window grid had been generated, a single scan over the entire data set is required to count the data density on each window. The counting and synopsis producing are illustrated via example as shown in Figure 5. In the example, a data set contains 10 two-dimensional data points and the window size is 2×2 along the [*x*, *y*] axes. For ease of illustration, each window in the window grid has been assigned a number from 1 to 35. Figure 5(a) shows the point-inclusion boundary of window 29 in which its area includes points in the range [6..7.99, 10..11.99] of the [*x*, *y*] axes. A thick line represents the inclusion boundary, whereas the dashed line is the exclusion edge.

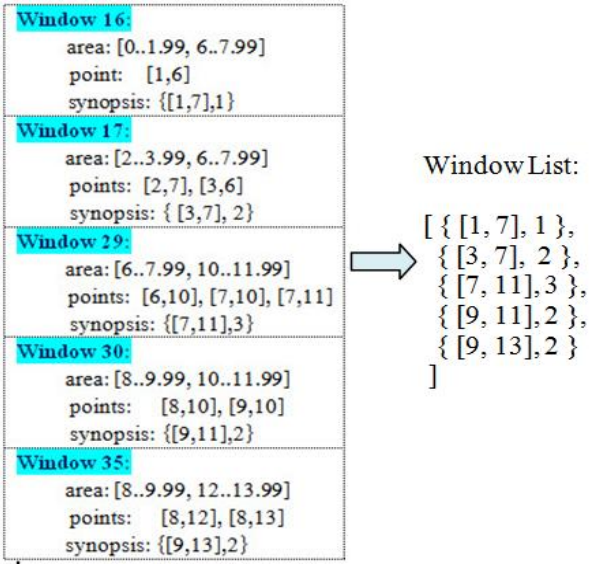
Figure 5(b) summarizes information regarding the window boundary, data points in the window, and window synopsis. The density estimation process finally produces a window list as shown in Figure 5(b). Each element of this list is a window synopsis, which in turn is the central point of the window together with the number of points in that window.

If we identify the required minimum density in each window to be at least 2, then there are only four windows eligible for the subsequent sampling phase. The step prior to the sampling phase is the generation of data points from the eligible windows, as shown in Figure 5(c).

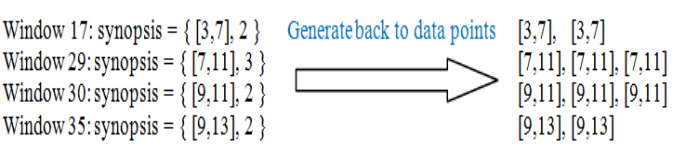
The proposed algorithm has been implemented with a functional programming paradigm using the Erlang language. The declarative style of Erlang facilitates a rapid prototyping and a concise coding, as shown in Figure 6. Our experimental results of drawing samples of high density data confirm that the shift of cluster means is minimal, whereas the decrease in memory usage is significant.



(a) the window grid on a 2D data set



(b) window synopses and a window list



(c) data points generated from the window synopsis with density at least 2

Fig. 5 Illustration of biased sampling technique

```
callWindowSliding(Dimension, AllPoints)->
      Stream=myZip(Dimension),
      eachWindow(Dimension, AllPoints, Stream).

   % prepare a window of size WxW and WxWxW
     % for 2D and 3D data sets

myZip(2)-> L=myGenL(0),
           [[X1,X2]||X1<-L,X2<-L];

myZip(3)-> L=myGenL(0),
           [[X1,X2,X3]||X1<-L,X2<-L,X3<-L];

myGenL(50)->[];        % axis scale is in the range 0 to 50
myGenL(Now)->[Now|myGenL(Now+WindowStepSize)].
     % e.g., WindowStepSize=2
     % count data points in each window

eachWindow(_,_,[])->[];
eachWindow(Del, AllPoints,[Now|StreamT])->
        Lc = count(Now, Del, AllPoints),
        Sum= lists:sum(Lc),
        [{Now,Sum}| eachWindow(Del,AllPoints,
                               StreamT)].

% --- Generate back from sliding window synopses to points
genWin2P([])->[];
genWin2P([{P,Den}|T])-> if Den>=1 ->
     dup(center(P),Den)++genWin2P(T);
                  true -> genWin2P(T) end.

%--- Hash-based density-biased reservoir sampling
specificDensBin(Bin,L,Den) ->
     take(Bin,specificDens(L,Den)).
specificDens([],_)->[];
specificDens([{P,D}|T],Den)-> if D>=Den ->
     [{P,D}|specificDens(T,Den)];
                  true -> specificDens(T,Den) end.

%--- Simple Random with density bias
simpleRandom(_,_,0)->[];
simpleRandom(WindowL,Dens,Bin)->
      Nth=random:uniform(length(WindowL)),
      {P,D}=lists:nth(Nth,WindowL),
      if D>=Dens ->
      [{P,D}|simpleRandom(WindowL,Dens,Bin-1)] ;
        true-> simpleRandom(WindowL,Dens,Bin) end.

%--- Rejection sampling with density bias
rejectionRandom(_,_,_,0)->[];
rejectionRandom(Para,WindowL,Dens,Bin)->
      Nth=random:uniform(length(WindowL)),
      Par=random:uniform(),
      {P,D}=lists:nth(Nth,WindowL),
      if (0.5-Para)=<Par,Par=<(0.5+Para),D>=Dens ->

      [{P,D}|rejectionRandom(Para,WindowL,Dens,Bin-
      1)] ;
        true->
      rejectionRandom(Para,WindowL,Dens,Bin) end.
```

Fig. 6 An Erlang module to perform density estimation with a sliding window technique, then perform density-biased sampling

V. Knowledge Deployment Examples

A. Trigger Rule Generation

We have tested our implementation with the diabetes dataset taken from the UCI database (http://www.ics.uci.edu/~mlearn/MLRepository.html). This medical data set is a collection of 768 observations on female patients investigating whether the patient shows signs of diabetes (class=1) or not (class=0) according to World Health Organization criteria. Each patient's record contains eight attributes: number of times pregnant, plasma glucose concentration (a two hours in an oral glucose tolerance test), diastolic blood pressure, triceps skin fold thickness (mm.), 2-hour serum insulin, body mass index, diabetes pedigree function, and age. The best five accurate association rules (annotated with the number of cases supporting the induced association) are shown as follows:

1. IF triceps-thickness='(0-9.9]' AND diabetes-pedigree-fn='(0-0.3122]'
   THEN 2Hr-serum-insulin='(0-84.6]'
   (support= 128 cases)
2. IF triceps-thickness='(0-9.9]' AND diabetes-pedigree-fn='(0-0.3122]' AND class=0
   THEN 2Hr-serum-insulin='(0-84.6]'
   (support= 83 cases)
3. IF diastolic-pressure='(73.2-85.4]' AND triceps-thickness='(0-9.9]'
   THEN 2Hr-serum-insulin='(0-84.6]'
   (support= 75 cases)
4. IF times-pregnant='(3-5]' AND triceps-thickness='(0-9.9]'
   THEN 2Hr-serum-insulin='(0-84.6]'
   (support= 52 cases)
5. IF diastolic-pressure='(73.2-85.4]' AND triceps-thickness='(0-9.9]' AND class=0
   THEN 2Hr-serum-insulin='(0-84.6]'
   (support = 48 cases)

To illustrate deployment of induced knowledge, we design a framework (Figure 7) to add active behavior to the medical database through the induced trigger rules. There are three major components in our model: mining, trigger generation and conflict resolution components. Mining component induces knowledge from the database contents and presents as rules: association and classification rules. The data repository contains both base data and trigger rules. Trigger generation component is responsible for converting induced classification/association rules into trigger format then stores generated triggers in the repository. In case of trigger rule application and rule conflict occurs, conflict resolution component will handle the situation.

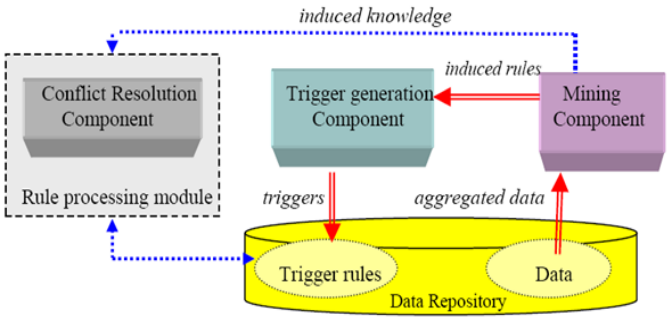


Fig. 7 Knowledge deployment by incorporating induced knowledge as a set of trigger rules in active medical databases

From the first induced rule: *IF triceps-thickness='(0-9.9]' AND diabetes-pedigree-fn='(0-0.3122]' THEN 2Hr-serum-insulin='(0-84.6]',* the database trigger can be created as follows:

```
CREATE TRIGGER rule_1 ON diabetes FOR UPDATE,
INSERT
 AS IF (SELECT COUNT(*) FROM diabetes
   WHERE (triceps-thickness = '(0-9.9]')
        and (diabetes-pedigree-fn= '(0-0.3122]' )
        and (2Hr-serum-insulin <> "(0-84.6]') ) > 0
 BEGIN
    RAISERROR ('soft constraint violation, please verify');
    END
```

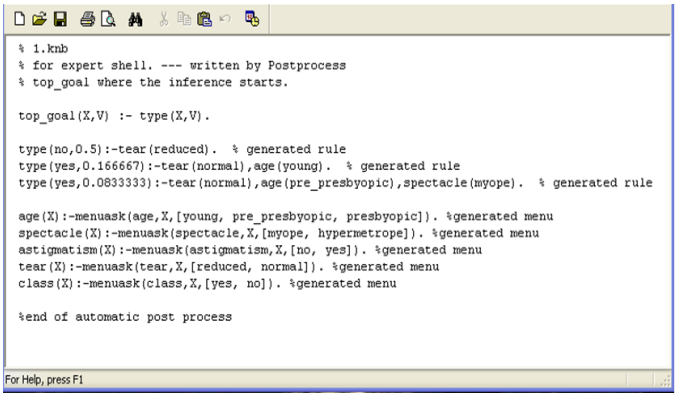
This trigger will raise a warning message upon any database updates that violates the rule "any female patient with triceps thickness in the range 0-9.9 mm and diabetes pedigree function in the range 0-0.3122, 2-hour serum insulin has to be in the rage 0-84.6". Any attempt to insert violating data will fire this trigger to draw attention from database administrator. Such trigger rule is thus deployed as a tool to support database integrity checking.

B. Automatic Knowledge Base Creation

Another example of induced knowledge deployment is the automatic generation of knowledge base in the medical decision support system. Given the contact lens recommendation data as in Figure 8, the induced classification rules can be generated and then transformed into expert rules as in Figure 9. These rules are to be consulted by the knowledge inference engine of the decision support system.

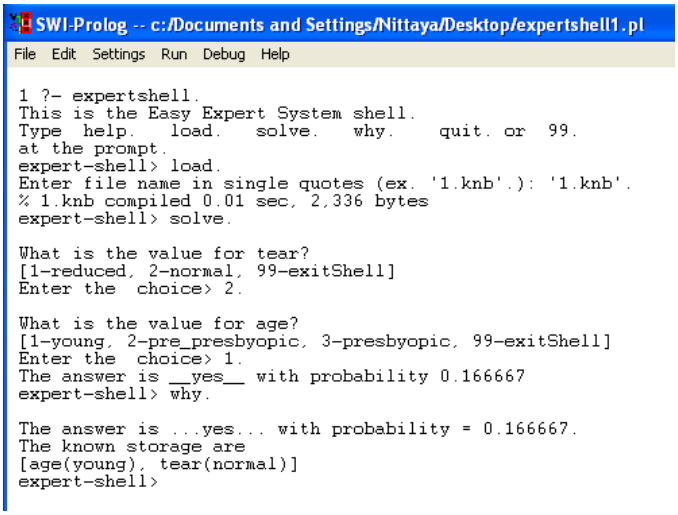
```
%% Data lens

% attributes: names and their possible values
%
attribute(age, [young, pre_presbyopic, presbyopic]).
attribute(spectacle, [myope, hypermetrope]).
attribute(astigmatism, [no, yes]).
attribute(tear, [reduced, normal]).
attribute(class, [ yes, no]).

% data
instance(1, class=no, [age=young, spectacle=myope, astigmatism=no, tear=reduced]).
instance(2, class=yes, [age=young, spectacle=myope, astigmatism=no, tear=normal]).
instance(3, class=no, [age=young, spectacle=myope, astigmatism=yes, tear=reduced]).
instance(4, class=yes, [age=young, spectacle=myope, astigmatism=no, tear=normal]).
instance(5, class=no, [age=young, spectacle=hypermetrope, astigmatism=no, tear=reduced]).
instance(6, class=yes, [age=young, spectacle=hypermetrope, astigmatism=no, tear=normal]).
instance(7, class=no, [age=young, spectacle=hypermetrope, astigmatism=yes, tear=reduced]).
```

Fig. 8 Contact lens data set as an input of the classification engine



```
% 1.knb
% for expert shell. --- written by Postprocess
% top_goal where the inference starts.

top_goal(X,V) :- type(X,V).

type(no,0.5):-tear(reduced).  % generated rule
type(yes,0.166667):-tear(normal),age(young).  % generated rule
type(yes,0.0833333):-tear(normal),age(pre_presbyopic),spectacle(myope).  % generated rule

age(X):-menuask(age,X,[young, pre_presbyopic, presbyopic]). %generated menu
spectacle(X):-menuask(spectacle,X,[myope, hypermetrope]). %generated menu
astigmatism(X):-menuask(astigmatism,X,[no, yes]). %generated menu
tear(X):-menuask(tear,X,[reduced, normal]). %generated menu
class(X):-menuask(class,X,[yes, no]). %generated menu

%end of automatic post process
```

For Help, press F1



```
SWI-Prolog -- c:/Documents and Settings/Nittaya/Desktop/expertshell1.pl
File  Edit  Settings  Run  Debug  Help

1 ?- expertshell.
This is the Easy Expert System shell.
Type  help.   load.   solve.   why.    quit. or  99.
at the prompt.
expert-shell> load.
Enter file name in single quotes (ex. '1.knb'.): '1.knb'.
% 1.knb compiled 0.01 sec, 2,336 bytes
expert-shell> solve.

What is the value for tear?
[1-reduced, 2-normal, 99-exitShell]
Enter the  choice> 2.

What is the value for age?
[1-young, 2-pre_presbyopic, 3-presbyopic, 99-exitShell]
Enter the  choice> 1.
The answer is __yes__ with probability 0.166667
expert-shell> why.

The answer is ...yes... with probability = 0.166667.
The known storage are
[age(young), tear(normal)]
expert-shell>
```

Fig. 9 Induced classification rules that are automatically transformed into knowledge base rules (top), and these rules are to be used in a decision support system (bottom)

VI. Conclusion

In this paper we have proposed the design and implementation of SUT-Miner, a declarative knowledge mining system. The system is intended to support automatic knowledge acquisition in medical and healthcare domains that require new knowledge to support better decisions as well as to enhance understandability of the stored data. The proposed knowledge discovery environment is composed of tools and methods suitable for various kinds of knowledge discovery tasks including data classification, association discovery, and data clustering.

Most of the implementation of our proposed system is based on the concept of logic programming, except the biased sampling for clustering that are implemented with functional language. Both languages support the advanced concept of higher-oder programming. In Prolog implementation, we use some higher-order predicates such as *maplist*, *findall*, *setoff*, and *include*. These predicates are higher-order in the sense that they take other predicates as their arguments. With such expressive power of higher-order predicates, program coding of the designed system is very concise as demonstrated in the paper. Program conciseness contributes directly to program verification and validation, which are important issues in

software engineering. The declarative style of programming also eases the extension of the present system towards the constraint higher-order mining [11], [39], which is our future research plan.

To illustrate knowledge deployment, we provide a framework to semi-automatically generate trigger rules from current database contents by means of association mining technique. Induced trigger rules, in addition to predefined triggers, can be viewed as supplementary constraints to help increasing database consistency. Our proposed framework is thus a preliminary design of active medical databases. Another example of utilizing the induced knowledge is demonstrated through the automatic generation of knowledge base to support medical decision. We also plan to further our implementation on the knowledge inferring part and then test the knowledge induction component on various medical data sets. The induced knowledge is also to be verified by the experts of the field.

REFERENCES

[1] R. Agrawal and C. Johnson, "Securing electronic health records without impeding the flow of information," *International Journal of Medical Informatics*, vol. 76, 2007, pp. 471-479.

[2] R. Agrawal and R. Srikant, "Fast algorithm for mining association rules," *Proceedings of the 20th VLDB*, 1994, pp. 487-499.

[3] Y. Bedard, P. Gosselin, S. Rivest, M.J. Proulx, M. Nadeau, G. Lebel, and M.F. Gagnon, "Integrating GIS components with knowledge discovery technology for environmental health decision support," *International Journal of Medical Informatics*, vol. 70, 2003, pp. 79-94.

[4] S. Begum, M.U. Ahmed, and P. Frank, "Case-based systems in health sciences – a case study in the field of stress management," *WSEAS Transactions on Systems*, vol. 8, no. 3, 2009, pp. 344-354.

[5] G. Bisson, "Conceptual clustering in a first-order logic representation," *Proceedings of the 10th European Conference on Artificial Intelligence*, 1992, pp. 458-462.

[6] H. Blockeel, L. De Raedt, and J. Ramon, "Top-down induction of clustering trees," *Proceedings of the 5th International Conference on Machine Learning (ICML-98)*, 1998, pp. 55-63.

[7] C. Bojarczuk, H. Lopez, A. Freitas, and E. Michalkiewicz, "A constrained-syntax genetic programming system for discovering classification rules: Application to medical data sets," *Artificial Intelligence in Medicine*, vol. 30, 2004, pp. 27-48.

[8] C. Bratsas, V. Koutkias, E. Kaimakamis, P. Bamidis, G. Pangalos, and N. Maglaveras, "KnowBaSICS-M: An ontology-based system for semantic management of medical problems and computerised algorithmic solutions," *Computer Methods and Programs in Biomedicine*, vol. 83, 2007, pp. 39-51.

[9] S. Ceri, R. Cochrane, and J. Widom, "Practical applications of triggers and constraints: Successes and lingering issues," *Proceedings of the 26th VLDB*, 2000, pp. 254-262.

[10] R. Correia, F. Kon, and R. Kon, "Borboleta: A mobile telehealth system for primary homecare," *Proceedings of ACM Symposium on Applied Computing*, 2008, pp. 1343-1347.

[11] L. De Raedt, T. Guns, and S. Nijssen, "Constraint programming for itemset mining", *Proceedings of KDD*, 2008, pp. 204-212.

[12] L. Dehaspe and H.Toivonen, "Discovery of frequent datalog patterns," *Data Mining and Knowledge Discovery*, vol. 3, no. 1, 1999, pp. 7-36.

[13] L. Dehaspe and H. Toivonen, "Discovery of relational association rules," In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, Springer, 2001, pp. 189-212.

[14] W. Emde, "Inductive learning of characteristic concept descriptions from small sets to classified examples," *Proceedings of the 7th European Conference on Machine Learning*, 1994, pp. 103-121.

[15] P. Gago and M.F. Santos, "Adaptive knowledge discovery for decision support in intensive care units," *WSEAS Transactions on Computers*, vol. 8, no. 7, 2009, pp. 1103-1112.

[16] S. Ghazavi and T. Liao, "Medical data mining by fuzzy modeling with selected features," *Artificial Intelligence in Medicine*, vol. 43, no. 3, 2008, pp. 195-206.

[17] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd edition, Morgan Kaufmann, 2006.

[18] M. Huang, M. Chen, and S. Lee, "Integrating data mining with case-based reasoning for chronic diseases prognosis and diagnosis," *Expert Systems with Applications*, vol. 32, 2007, pp. 856-867.

[19] N. Hulse, G. Fiol, R. Bradshaw, L. Roemer, and R. Rocha, "Towards an on-demand peer feedback system for a clinical knowledge base: A case study with order sets," *Journal of Biomedical Informatics*, vol. 41, 2008, pp. 152-164.

[20] N. Kerdprasop and K. Kerdprasop, "Knowledge induction from medical databases with higher-order programming," *WSEAS Transactions on Information Science and Applications*, vol. 6, no. 10, 2009, pp. 1719-1728.

[21] M. Kirsten and S. Wrobel, "Relational distance-based clustering," *Proceedings of the 8th International Conference on Inductive Logic Programming*, 1998, pp. 261-270.

[22] M. Kirsten and S. Wrobel, "Extending k-means clustering to first-order representations," *Proceedings of the 10th International Conference on Inductive Logic Programming*, 2000, pp. 112-129.

[23] S. Kramer and G. Widmer, "Inducing classification and regression trees in first order logic," In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, Springer, 2001, pp. 140-159.

[24] E. Kretschmann, W. Fleischmann, and R. Apweiler, "Automatic rule generation for protein annotation with the C4.5 data mining algorithm applied on SWISS-PROT," *Bioinformatics*, vol. 17, no. 10, 2001, pp. 920-926.

[25] D. Lee, W. Mao, H. Chiu, and W. Chu, "Designing triggers with trigger-by-example," *Knowledge and Information System*, vol. 7, 2005, pp. 110-134.

[26] Y.C. Lin, "Design and implementation of an ontology-based psychiatric disorder detection system," *WSEAS Transactions on Information Science and Applications*, vol. 7, no. 1, 2010, pp. 56-69.

[27] F. Lin, S. Chou, S. Pan, and Y. Chen, "Mining time dependency patterns in clinical pathways," *International Journal of Medical Informatics*, vol. 62, no. 1, 2001, pp. 11-25.

[28] J. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1*, 1967, pp. 281-297.

[29] S. Muggleton, "Inverse entailment and Progol," *New Generation Computing*, vol. 13, 1995, pp. 245-286.

[30] G. Nadathur and D. Miller, "Higher-order Horn clauses," *Journal of ACM*, vol. 37, 1990, pp. 777-814.

[31] L. Naish, "Higher-order logic programming in Prolog," *Technical Report 96/2*, Department of Computer Science, University of Melbourne, Australia, 1996.
[32] D. Nguyen, T. Ho, and S. Kawasaki, "Knowledge visualization in hepatitis study," *Proceedings of Asia-Pacific Symposium on Information Visualization*, 2006, pp. 59-62.
[33] S.-H. Nienhuys-Cheng and R. Wolf, *Foundations of Inductive Logic Programming*, Springer, 1997.
[34] G. Noren, A. Bate, J. Hopstadius, K. Star, and I. Edwards, "Temporal pattern discovery for trends and transient effects: Its application to patient records," *Proceedings of KDD Conference*, 2008, pp. 963-971.
[35] S. Palaniappan and C. Ling, "Clinical decision support using OLAP with data mining," *International Journal of Computer Science and Network Security*, vol. 8, no. 9, 2008, pp. 290-296.
[36] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, 1986, pp. 81-106.
[37] J. Quinlan, J. and R. Cameron-Jones, "FOIL: A midterm report," *Proceedings of the 6th European Conference on Machine Learning*, 1993, pp. 3-20.
[38] J. Roddick, P. Fule, and W. Graco, "Exploratory medical knowledge discovery: experiences and issues," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 1, 2003, pp. 94-99.
[39] J. Roddick, M. Spiliopoulou, D. Lister, and A. Ceglar, "Higher order mining," *ACM SIGKDD Explorations Newsletter*, vol. 10, no. 1, 2008, pp. 5-17.
[40] T. Sahama and P. Croll, "A data warehouse architecture for clinical data warehousing," *Proceedings of the 12th Australasian Symposium on ACSW Frontiers*, 2007, pp. 227-232.
[41] A. Shillabeer and J. Roddick, "Establishing a lineage for medical knowledge discovery," *Proceedings of the 6th Australasian Conference on Data Mining and Analytics*, 2007, pp. 29-37.
[42] A. Silva, P. Cortez, M. Santos, L. Gomes, and J. Neves, "Rating organ failure via adverse events using data mining in the intensive care unit," *Artificial Intelligence in Medicine*, vol. 43, no. 3, 2008, pp. 179-193.
[43] J. Thongkam, G. Xu, Y. Zhang, and F. Huang, "Breast cancer survivability via AdaBoost algorithms," *Proceedings of the 2nd Australasian Workshop on Health Data and Knowledge Management*, 2008, pp. 55-64.
[44] K. Truemper, *Design of logic-based intelligent systems*, John Wiley & Sons, New Jersey, 2004.
[45] T.Y. Wah, N.H. Peng, and C.S. Hok, "Development of specific disease data warehouse for developing content from general guide for hypertension screening, referral and follow up," *WSEAS Transactions on Computers*, vol. 7, no. 4, 2008, pp. 190-195.
[46] Z. Zhuang, L. Churilov, and F. Burstein, "Combining data mining and case-based reasoning for intelligent decision support for pathology ordering by general practitioners," *European Journal of Operational Research*, vol. 195, no. 3, 2009, pp. 662-675.

Nittaya Kerdprasop is an associate professor and the director of Data Engineering Research Unit, School of Computer Engineering, Suranaree University of Technology, Thailand. She received her B.S. in radiation techniques from Mahidol University, Thailand, in 1985, M.S. in computer science from the Prince of Songkla University, Thailand, in 1991 and Ph.D. in computer science from Nova Southeastern University, U.S.A., in 1999. She is a member of IAENG, ACM, and IEEE Computer Society. Her research of interest includes Knowledge Discovery in Databases, Data Mining, Artificial Intelligence, Logic and Constraint Programming, Deductive and Active Databases.

Kittisak Kerdprasop is an associate professor at the School of Computer Engineering and one of the principal researchers of Data Engineering Research Unit, Suranaree University of Technology, Thailand. He received his bachelor degree in Mathematics from Srinakarinwirot University, Thailand, in 1986, master degree in computer science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in computer science from Nova Southeastern University, USA, in 1999. His current research includes Data mining, Machine Learning, Artificial Intelligence, Logic and Functional Programming, Probabilistic Databases and Knowledge Bases.