Relational Patterns

Maytal, Saar Tsechansky

IS-98-21

Relational Patterns

Maytal, Saar Tsechansky

September 1998

Working Paper Series
Stern #IS-98-21

Introduction

Hospitals maintain large administrative databases, which contain interesting demographic and medical information such as age, sex, diagnoses and medical procedures. While the data are being used for administrative operations, such as generating payment reports and hospital activity reports, it is particularly desirable to make use of the data for other value-added purposes. More particularly, assuming the data collected are reliable, administrative data may be used in the process of auditing the quality of the medical care, tracing possible explanations for delivery of high- or low-quality of medical care, highlighting processes embedded in the organization, or indicating of evolving previously unknown patterns embedded among hospitalization cases. However, in most cases these data are merely utilized for the routine administrative purposes for which they were designed in the first place.

Soaring medical costs [] and harsh competition among health care providers bring hospitals' managements to increase efforts in search for higher quality of care and lower their expenses. The management of a major medical center in Israel approached the authors with these concerns and expressed their frustration from the fact that although the hospital is collecting termendous amount of data each year, apart from statistical reports the hospitals' databases are hardly utilized to address these concerns and to shed light on the practices within the hospital. In particular, hospital management expressed its interest to discover patterns relating to the quality of medical care embedded in the hospital database such as patterns concerned with coarses of treatments that indicate on delivery of low/high quality of medical care. Also, may be embbedded in the databse are patterns that indicate an uncareful (wasteful) use of hospital resources (e.g., unnecessary test/procedures) such as patterns of coarses of treatments that show low/high quality of medical care.

The area of Knowledge Discovery in Databases (KDD) was defined in [FPS96] as the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns from data. As such it is directly applicable to respond to the concerns above by offering new methods and tools to automatically extract knowledge and analyze useful patterns from databases. In particulart, one extensively studied issue in the field is the problem of association rules [AIS93] [AMSTV95] [CNF96] [MTV94]

[SON95]. Association rules seemed to be particularly appealing for our problem since they are based on the discovery of sets of attributes and their values. These patterns may be useful as they introduce collections of attributes that co-occur in the database. In addition, this type of patterns are thought to require relatively limited interpretation. This is especially important when the products of a KDD system are not considered the ultimate result but rather a possible input to other related processes (e.g., quality improvement, decision support systems, etc.). In this study we aimed at employing association rules methods to extract patterns from a relational database of hospitalizations discharge abstracts. In the coarse of our work, we discovered that when extracting itemsets [], which is the core phase in the algorithm for association rules[], some information embedded in the relational database is lost and/or distorted. We thus suggested to extract *relational patterns* that capture not only coocurrences of attributes and their values, but also the relationships between those attributes as they appear in the database schema. In this paper we present the concept of *relational patterns* and our approach to extract relational patterns from a relational database. We also describe the experiences we had and conclusions we drew from implementing our approach on a hospital discharge abstract database. We discuss what dinstiguishes this applications from the bulk of association rules applications reported in the literature, the difficulties we encountered as a result and how we confronted them.

The contributions of this paper are first a new approach to extract *relational patterns* from a relational database with multiple tables, that maintain structural relationships embedded in the relational database schema. Second, this paper presents a case study of an application to extract *relational patterns* from a hospital database, and describes the impact of some domain related issues on the application.

The organization of this paper is as follows: we start in section1 with a definition of association rules and sequential patterns both relevant for the approach we present in this paper. We then overview existing algorithms to extract association rules and sequential patterns from a single table. In section 2 we describe adjustments we made in the algorithm to extract sequential patterns to now extract sibling patterns from a transformed database. We describe the database schema for the problem and entity representation in section 3. In section 4 we discuss the issues posed by relational database with multiple tables, and describe relational patterns. We present our approach and algorithm in section 5. In section 6 we discuss the difficulties arose when we applied the presented approach on a hospital discharge abstract database, we discuss them and how we confronted the. We also suggest other approaches as possible resolutions.

1. Association Rules & Sequential Patterns

The approach presented here enables to extend the methods presented in [AMS95] [SA9?] to extract itemsets [AIS93], and association rules from a single table, to now extract patterns from multiple relational tables' schema. While the methods presented in [AMS95] [SA9?] may be are valuable and efficient, 70% of the databases in the world are relational and contain multiple tables. By analyzing each table separately valuable knowledge embedded among the relationships between entities from different tables may not be discovered. The approach presented here, makes use of algorithms designed for single tables [AMS95] [AS9?], and it is thus necessary to describe some preliminaries on these algorithms as well.

In this section we bring a definition of association rules [AMS95] and describe the concept of itemsets [AIS93]. As will become clear shortly the primary problem is the extraction of frequent/large itemsets, from which it is rather straightforward to generate association rules.

Let I = $\{i_1, i_2, ..., i_m\}$ be a set of attributes (also called items) [AIS93], where an attribute can take any value from a discrete set of mutually exclusive values. A conjunction of conditions of the form "attribute = value" is called an *itemset* [AIS93]. An association rule is an implication of the form A$\rightarrow$B, where A and B are mutually exclusive itemsets. Example for an association rule from a hospital database is {Department = "ER", Age = "60-65"} $\rightarrow$ {Length_of_Stay = "5-7"}. The interpretation for this rule is that a patient aged between 60 to 65 and who registered to the emergency room, is likely to stay in the hospital for 5 to 7 days. Let $D = \{t_1, t_2, ..., t_N\}$ be a relation consisting of N transactions (i.e., records). A transaction *t* is said to satisfy an itemset if it contains this itemset.

An association rule holds on a database *D* if its *strength* and *statistical significance* are greater than some predefined thresholds [AIS93]. The strength of a rule is called *support*, denoted by *s* and is the percentage of transactions in the database which contain the itenset A $\cup$ B. All itemsets which satisfy the support threshold, called *minimum support* are called *large* itemsets. The statistical significance of a rule is called *confidence*, denoted by *c* and is the percentage of transactions which contain A that also contain B [AIS93]. All rules have to exhibit confidence level that is above the threshold called *minimum confidence*. Algorithms suggested to efficiently extract all association rules that hold in a single relation [AMS95] [AIS93] include two phases. In the first phase all large itemsets are generated, where the following observation is key in the process: All subsets of a large itemsets must also be large. Thus, candidate itemsets of size *k* are generated by using large itesets of size *k*-1. All candidates consisting of sub-itemsets that are not large are discarded. The support of all candidate itemsets is then computed. In the second phase, rules are generated from the set of large itemsets. The confidence factor is computed by $Confidence\,(A \rightarrow B) = \frac{Support(A \cup B)}{Support(A)}$. Since both itemsets A $\cup$ B and A are large, the task of confidence computing and rules generation is quite straightforward. Therefore, the problem of generating all association rules is reduced to the problem of generating all large itemstes (from which rules are created). The approach presented by this algorithm addresses mining large itemsets or patterns within a single relation described in [AMS95] as intra-tuple patterns. Stated differently, these patterns exist between attributes within a single tuple.

Agrawal and Srikant proposed another algorithm to extract sequential patterns [AS9?] from a single relation. In order to understand the concept of sequential patterns we shortly describe the concepts and the proposed algorithm.

Customer Id	Day of Purchase	Items Purchased
1	**January 12, 1998**	**A, B, C**
1	**January 25, 1998**	**D, E**
2	February 28, 1998	F, G
2	February 10, 1998	H
3	**May 15, 1998**	**B, C**
3	**June 15, 1998**	**E**
4	March 11, 1998	G
4	April 28, 1998	I
4	May 21, 1998	H

Figure 1: Supermarket purchase database

In [AS9?] a sequential pattern is an ordered list of itemsets. A sequence $\{s_1, s_2, \ldots, s_n\}$ is said to be contained in another sequence $\{t_1, t_2, \ldots, t_m\}$ $m \geq n$, if there exist integers $j_1 < j_2 < \ldots < j_n$, such that $s_1 \subseteq t_{j_1}$, $s_2 \subseteq t_{j_2}$, ..., $s_n \subseteq t_{j_n}$. Consider the single table database in Figure 1: each transaction can be viewed as an itemset, and all the transactions pertaining to a single customer can be viewed as a sequence or an ordered list of itemsets. Fig. 2 shows a sequence version of the table in Fig. 1.

Customer Id	Items Purchased
1	$\langle (A,B,C),(D,E) \rangle$
2	$\langle (F,G),(H) \rangle$
3	$\langle (B,C),(E) \rangle$
4	$\langle (G),(I),(H) \rangle$

Figure 2: a sequence version a supermarket table

The "strength" of a sequence is determined by a predefined threshold called "*minimum support*". For the database in Figure 1, for example, this threshold would be the percentage of customers which contain a certain sequence. Also, a sequence S is said to be *large* if it is contained in at least *minimum support* of the customers [AS9?]. Assume that the *minimum support* threshold for the database in Figure 1 is 50%, then the sequence $\langle \{B,C\}\{E\} \rangle$ is *large* since it is contained in the sequences of 50 % of the customers (for customers with Id 1, and 3). The algorithm is composed of 4 phases. To explain the algorithm consider the table shown in Figure 1, where each customer can have only one transaction in a single date. In the first phase the database is sorted such that all transactions pertaining to a single customer appear consecutively sorted by date of transaction. In the second phase, the Apriori algorithm [AMS95] is applied to extract large itemsets. However, itemsets are counted per customer and not per transaction as in [AMS95]. All large itemsets also comprise all large sequences of length 1 (called 1-

sequences). Each large 1-sequence is mapped to unique integer. Fig. 3a shows all large itemsets extracted from the database in Fig. 1. In the third phase the relation is transformed into a transformed database D_T, where each transaction in replaced by the set of large itemsets contained in that transaction. A sequence is represented by an ordered list of sets of itemsets. Fig 3b shows the transformed database D_T.

Large itemset	Mapped to
(B)	1
(C)	2
(E)	3
(G)	4
(H)	5
(B,C)	6

3a: All extracted large itemsets

Customer Id	Items Purchased
1	$\langle\{(1),(2),(6))\{(3)\}\rangle$
2	$\langle\{(4)\}\{(5))\rangle$
3	$\langle\{(1),(2),(6))\{(3)\}\rangle$
4	$\langle\{(4)\}\{(5))\rangle$

3b: The transformed database D_T

Figure 3: Construction of the transformed database D_T

In the forth phase, also called the sequence phase large sequences of size ≥ 2 (i.e., n-sequences, $n \geq 2$) are extracted, where the key observation is that large sequences must be comprised of large itemsets. The transformed database D_T constitutes the fundamental structure on which the operations to extract n-sequences $n \geq 2$ are performed.

2. Database Schema for the Problem

Before describing the database schema it is helpful to consider the discharge abstract database example shown in figure 4 below. Relation R1 contains one tuple for each admission to the hospital, and its key is patient Id. R2 contains details pertaining to the departments in which a patient stayed during the related hospitalization. One patient may stay in one or more departments.

R1: Hospitalization

Patient ID	Age Group	Sex	Cost ($)
100	60-65	M	7K-9K
200	5-7	F	5K-7K
300	60-65	F	7K-9K
400	10-12	M	2K-5K

R2: Departmental Admissions

Patient ID	Department	LOS*
100	Internal	2-3D
100	ER	1D
200	Pediatric	2-3D
300	Surgery	3-5D
300	ER	5-7H
400	ER	1-2H
400	Pediatric	1D
400	Orthopedic	5-7D

* Length Of Stay : D for days, H for hours

R3: Medical Procedures

Patient ID	Department	Procedures & Tests
100	ER	Blood Count
100	ER	ECG
200	Pediatric	X-Ray
200	Pediatric	Blood Count
300	ER	Blood Count
300	ER	ECG
400	ER	Blood Count
400	Pediatric	X-Ray
400	Pediatric	Fixation

Figure 4: An example for hospitalization discharge abstracts database

For example, the patient with Id 100 stayed both in the emergency room and the Internal department. The key for relation R2 is composed of patient Id, and department. Relation R3 contains information on procedures performed for a certain patient while staying in a certain department. The key for that relation is patient Id, department and procedure. Also, It is possible that no procedure is performed for a patient while in a certain department.

Relational databases can be represented as a DAGs, in which each relation (i.e. table) is a node and every relationship is an edge in the graph. Any two relations with a one to one relationship can be represented as a single node, whereas, a one to many relationship between relations R_i (one) and R_j (many), $i \neq j$: $R_i \xrightarrow{1:n} R_j$ may be represented as a directed edge connecting the two corresponding nodes from R_i to R_j. A *path* is referred here to a sequence of connected nodes in the graph. We represent a path with a series of relations with consecutive increasing indices $R_i, R_{i+1}, \ldots, R_n$, where between any two consecutive relations R_j, R_{j+1} the following relationship is defined $R_j \xrightarrow{1:n} R_{j+1}$. In this paper we consider a DAG with a single path, where all tuples from related nodes can constitute a comprehensive collection of attributes which may comprise a single entity (see section 4.1). For example, the database depicted in Fig. 4 may be represented as a DAG with the following single path: $R_1 \xrightarrow{1:n} R_2 \xrightarrow{1:n} R_3$. In addition, the related collection of tuples in a tree can constitute a comprehensive single entity for our purpose.

2.1 Entity Representation.

We now describe the representation of an entity to which the extracted patterns relate to. Figure 5 shows a tree representation which include all the tuples in the database pertaining to a single patient (or "entity"), as well as the relations between them as determined by the database schema. Each level in the tree corresponds to a node in the DAG representation of the database (i.e., a relation). We refer to level i to all tuples from relation R_i. Each node in Fig. 5 corresponds to a single tuple. The notation t_i^k represents a tuple t_i from relation R_k. Each directed edge from t_i^k to t_j^{k+1} connects related tuples from relations R_k and R_{k+1} between which the relation: $R_k \xrightarrow{1:n} R_{k+1}$ is defined. The tuple t_i^k is a parent/mother tuple of the tuple t_j^{k+1}, and the tuple t_j^k is called a child/descendant tuple. Descendant is a more general term used for a tuple that is not necessarily a direct child of another. For example, in Fig. 5 the tuple t_1^3 since not an immediate descendant of t_1^1 is a *descendant* of t_1^1 rather than its *child.*

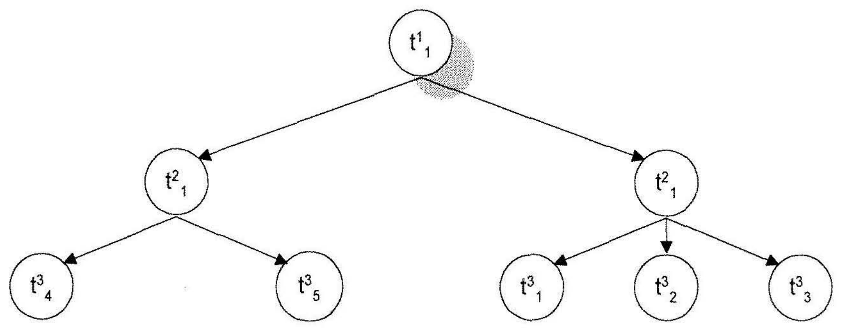

Figure 5: Tree representation of a single "entity"

Tree representations of all hospitalization cases in the databases are shown in Figure 9.

3. Sibling Patterns – Sequential Pattern Revisited

Sequential patterns are actually inter-tuple patterns [AS9?]. The concept of sequence is derived from the order among a group of tuples according to some time related attribute (in the example in Fig. 1 the order is derived by the time of purchase). In this paper, however, we refer to inter-tuple patterns as any sequence (unordered) of large itemsets, with a mutual parent tuple.

Now consider for example relation R_2 in Fig. 4, where there are no time-related attributes and thus no sequential patterns are defined. However, tuples in R_2 may be grouped by their mutual parent in relation R_1 positioned one level upward in the database's tree representation. These collections of tuples are unordered and are called: *Sibling* Collections. As shown in Fig. 6, relation R_2, for example, includes 4 sibling collections: The first collection includes children of the parent tuple with Id 100 in relation R_1. Sibling collections number 2,3, and 4 include children of tuples in R_1 corresponding to patients with Id 200, 300 and 400 respectively.

Patient ID	Department	LOS*

100	Internal ER	2-3D 1D
200	Pediatric	2-3D
300	Surgery ER	3-5D 5-7H
400	ER Pediatric Orthopedic	1-2H 1D 5-7D

Figure 6: Division of R_2 into sibling collections

A sequence in [AS9?] is defined as an ordered list of sets of large itemsets, for our purpose we define a sibling pattern/collection as an unordered list of sets, therefore, a new definition is required to determine whether a sibling collection contains a sibling pattern.

A sibling pattern p is said to be contained in a sibling collection b if we can map each set in p to an exclusive set in b, such that the set in p is a subset of the set to which it is mapped in b. Stating this differently we say that a sibling pattern p is contained in a sibling collection b, if there exist a function $f : p \to b$, such that

1. $\forall x \in p,\ f(x) \in b$ and $x \subseteq f(x)$
2. $\forall x_1, x_2 \in p,\ x_1 \neq x_2 \to f(x_1) \neq f(x_2)$

For example, the sibling pattern $\langle\{(Dept.=ER),(LOS=1D)\},\{(Dept.=Internal)\}\rangle$ is composed of two sets, one includes the attributes: (Dept.= ER) and (LOS=1D), and the other includes the attribute (Dept.= Internal). This pattern is contained in a sibling collections in R_2 corresponding to patient with Id 100. In addition, we use the term sibling pattern rather than the term sequential pattern used in [AS9?]. Similarly, the term n-sibling corresponds to the notion of n-sequence in [AS9?], and it relates to a sibling pattern with n elements (i.e., a sibling pattern of length n). For example the pattern $\langle\{(Dept.=ER),(LOS=1D)\},\{(Dept.=Internal)\}\rangle$ is a 2-sibling pattern: it includes two sets - each corresponds to a different tuple in the same sibling collection.

The change we introduced here with respect to the algorithm for sequential patterns [AS9?], increases substantially the theoretical complexity to determining whether a sibling collection contains a sibling pattern. In the worst case where we have to determine whether an m-sibling patterns is contained in an n-sibling collection we have $\frac{n!}{m!}$

comparisons to perform. However, in reality, in the database we used, most tables had an average length of sibling collections of ~ 2 . Thus, in practice the complexity did not have a significant impact on the application running time. We also believe that in most databases the picture is approximately the similar to this one.

3. Issues posed by multiple relational tables' schema

While the methods presented in [AMS95] [SA9?] are valuable and efficient, 70% of the databases in the world are relational and contain multiple tables. By analyzing each table separately valuable knowledge embedded among the relationships between entities from different tables may not be discovered

The algorithms described in section 1 were designed for a single table. However, a multiple table database requires more information to be represented in a pattern, to capture the relations between tables – information that may be essential to allow the pattern convey a complete picture, and thus should not be lost when extracting patterns from the database.

Assume for example that for the database shown in figure 4 we pool all the attributes pertaining to each patient and construct a single table. We then apply an existing algorithm to extract large itemsets. Consider now the following set of attributes: {(Age = 60-65), (Dept. = ICU), (Dept. = ER), (Procedure = ECG), (length of stay = 1D)}. Since both *Length of stay* and *Procedures* are associated with a certain department, the pattern above does not capture the information of where did the patient stayed for one day (i.e., in the ICU, the ER or in another department of which attributes are not included in the pattern?). Similarly, the pattern does not provide any clue on where the ECG procedure was performed. This information is embedded in the relations between the tables. Thus, it is important to capture not only frequent sets of attributes but also the structure between the tuples from which these attributes are extracted.

5. Relational Patterns

To address this problem we suggest to mine *relational* patterns.

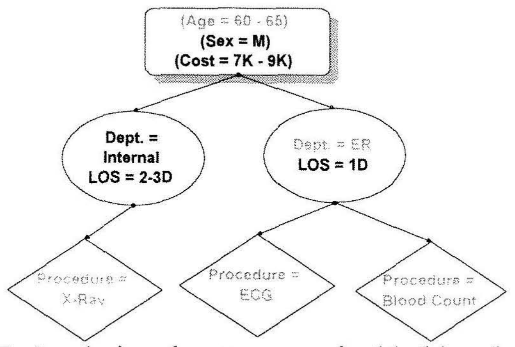

Figure 7: A relational patterns embedded in a hospitalization case

Consider the hospitalization case shown in Fig. 7, which relates to the database in Fig. 4. Also assume that the attributes in gray are frequent in the database in that structure, i.e., Patients between the age of 60 and 65, stay in the emergency room, where the go through a blood count and an ECG test, in addition they stay in another department (none of the attributed pertaining to their stay in department are part of the pattern), where they go through an X-Ray. The relational pattern also captures the relations between the tuples from which the attributes in that pattern were extracted, so that we know, for example, where the various procedures were performed. Figure 8 shows this pattern .

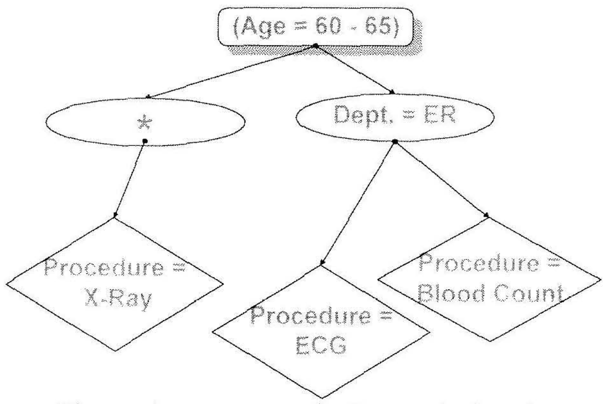

Figure 8: An example for a relational pattern

The asterisk stands to note that no attribute from the related tuple is part of the pattern, however we keep the tuple in the tree representation to maintains the original relationships between the various attributes.

6. Our Approach

Consider the tree representation of each "entity" in the database. We may extract *relational patterns* with respect to a certain relation R_i in the tree representation of the database. These patterns may include attributes from R_i and/or from any of its descendant relations.

When we extract relational patterns wrt relation R_i, the entities we related to and from which we extract these patterns are all sub-trees of hospitalization trees shown in Fig. 9, where the root of each sub-tree is a node corresponding to a tuple in R_i. For simplicity we say that a sub-tree with the root node corresponding to tuple t_j^k, is tuple t_j^k's tree .

A *relational* pattern *with respect to* relation R_i may also be represented with a tree, that is a sub-tree of all trees of tuples in R_i, and where each node includes an itemset (rather than a tuple). We say that a relational pattern T' with respect to R_i is contained in tuple t_j^k *'s* tree, denoted by *T*, if each node in T' is an itemset that is contained in the corresponding node in *T*. We also adopt the concept of *support* to be applied for relational patterns. A relational pattern with respect to relation R_i is said to be *large* if it is contained in more than minimum support of the trees of tuples in R_i. For example, Fig. 9 shows a tree representation of all the hospitalization entities in the database. Fig. 10 shows a relational pattern from the database depicted in Fig. 4, extracted with respect to relation R_2: Patients are hospitalized in the Emergency Room, where they go through a blood count. Assuming that the minimum support threshold for this database is 30%, this pattern is also *large,* since more than *minimum support* of the trees of tuples in R_2 contain that pattern. More particularly, 3 out of 8 trees contain that pettern: the itemset {(Dept. = ER) is contained in 3 tuples in R_2 that have a descendant tuples in R_3 that contains the itemset {(Procedure = Blood Count)}.

A *descendent* pattern with respect to R_i is also a relational pattern but it only includes itemsets pertaining to descendant tuples of the tuples in R_i. Thus, a descendant patterns is a sub-tree of a relational pattern with respect to R_i, that only includes nodes corresponding to descendant tuples of the tuples in R_i. For example, Fig. 11 shows a descendant pattern with respect to relation R_2, which is a sub-tree of the relational pattern shown in Fig. 10. (It is also *large* since it is contained in 4 of the 8 trees of tuples in R_2). We include the root node with an asterisk merely to capture the relationships between the attributes.

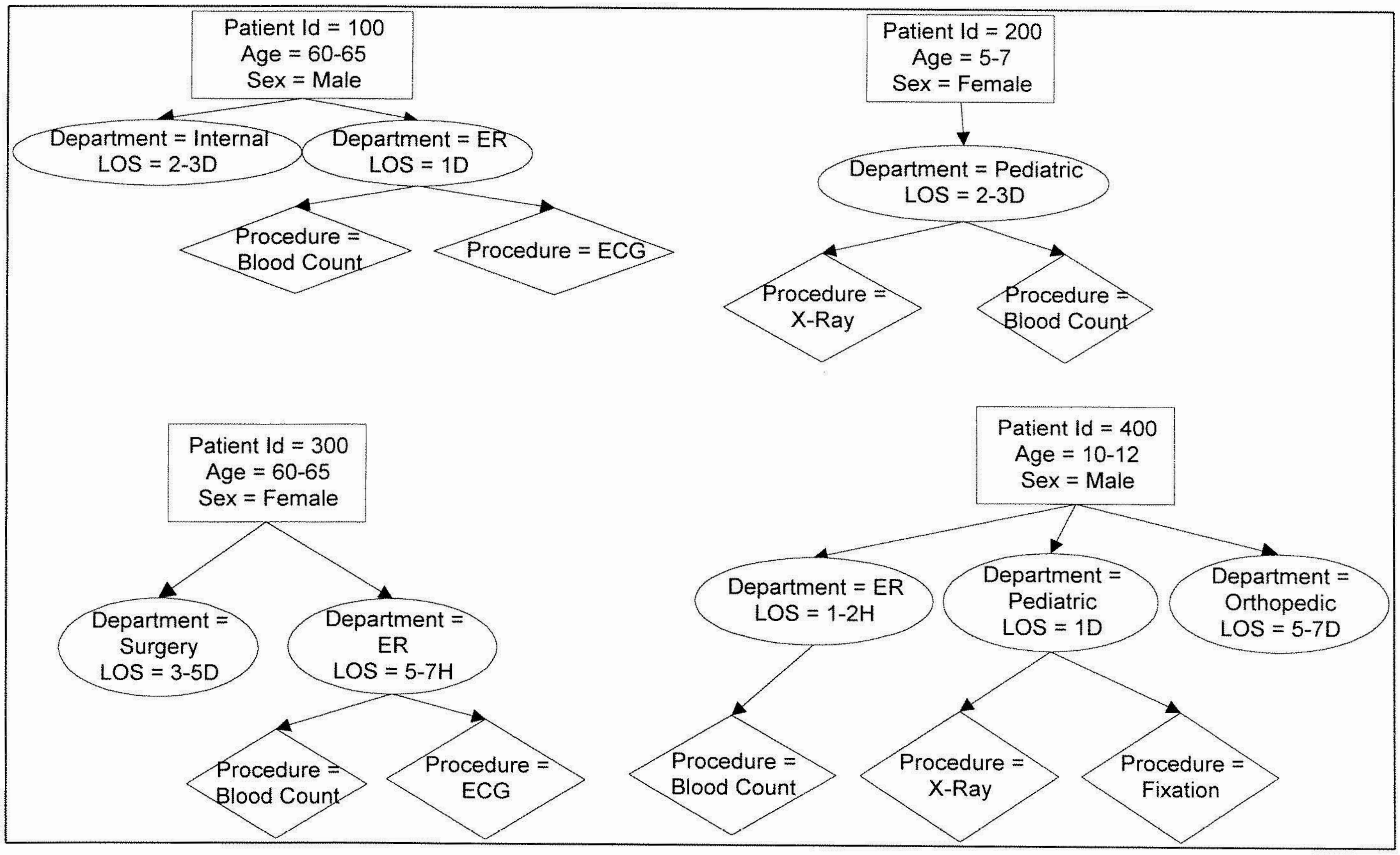

Figure 9: A tree representation of all hospitalization entities in the database

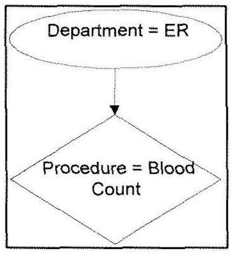

Figure 10: A relational pattern with respect to R_2

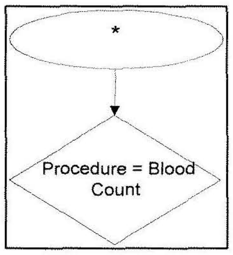

Figure 11: A descendant patterns with respect to relation R_2

6.1 The Algorithm

As descendant patterns are extracted **with respect to** a particular relation R we may extract relational patterns wrt any relation in the database. This is so since the support count for each descendant pattern is computed with respect to R, i.e., we compute the percentage of trees of tuples in relation R that contain that pattern. Due to arbitrary tree structures, and different number of tuples in each relation in the database, the count for support needs to be performed with respect to a certain relations.

Next we describe our approach to extract relational patterns with respect to an arbitrary relation R_{i-n}, where the leaf relation R_i is n levels downward in tree representation of the database.

To extract relational patterns for relation R_{i-n}, we first extract *descendant* patterns with respect to that relation. We extract descendant patterns bottom up starting from the leaf relation up to relation R_{i-n+1}. Then, we extract relational pattern for R_{i-n}. The tuples at the leaf relation R_i have no descendant tuples (i.e., their trees are of a single node), and therefore the treatment is different and simpler than for any other relation in the path.

In the following description assume we aim at extracting relational patterns with respect to relation R_{i-n}. We extract relational patterns from the leaf relation upward, and in the following description we extract patterns at an arbitrary level of relation R_j. For each level/ relation R_j we extract relational patterns that include itemsets corresponding to a tuple in that relation as well as itemsets from descendant tuples. We then construct an alternative database where each tuple in R_j is replaced with all the previously extracted relational patterns contained in its tree. These patterns are represented with unique integers. We extend the patterns to the width by utilizing the alternative database to extract sibling patterns. For the leaf level the alternative database does not include any descendant patterns. In addition, for the root relation (with respect to which we extract relational patterns) we do not extract sibling patterns (this is since there is no parent relation with respect to which sibling collections are defined).

Phase 1

At the first phase we employ the Apriori algorithm [AMS95] described in section 1 to extract large itemsets from relation R_j, however, differently from Apriori we increase the support count of an itemset per tuple in R_{i-n} that the itemset is contained in one of its descendant tuples in relation R_j. We also map each large itemset to a unique integer and store them in the set AL_j^{i-n}.

Phase 2

We apply phase 2 only if the current relation from which we extract descendant patterns is not a leaf relation. In this phase we generate relational patterns also called *Join* patterns. *Join* patterns are sub-trees with nodes corresponding to tuples from descendant relations of R_j, as well as a root node corresponding to a tuple in R_j. If R_j is a leaf relation its tuples have no descendants and thus we skip this phase.

DS_{j-1}^{i-n} is the set of all large descendant patterns wrt relation R_{i-n} extracted from all descendant relations up to relation R_{j-1} (see phase 4). Each pattern is represented in DS_{j-1}^{i-n} by a unique integer.

The set $\overline{DS}_{j-1}^{i-n}$ includes members of the form $\langle PID, \{P_{PID}\}\rangle$, where PID is the Id of a tuple t in R_j, and the set $\{P_{PID}\}$ contains all descendant patterns in DS_{j-1}^{i-n} contained in t's tree. Each member of $\overline{DS}_i^{i-n}$ corresponding to tuple t in R_j is thus $\langle t.PID, \{ds \in DS_i^{i-n} \mid ds$ is contained in t's tree$\} \rangle$.

To generate candidates for Join patterns, we join all descendant patterns in DS_{j-1}^{i-n} with all the itemsets in AL_j^{i-n} generated from R_j in phase 1. We then compute the support of each candidate. A descendant pattern generated by joining an itemset $a \in AL_j^{i-n}$ from relation R_j and a descendant pattern $ds \in DS_{j-1}^{i-n}$ is contained in tuple t's tree (where t is from in R_j), if the following are satisfied:

1. t contains a, and
2. There exist $\langle PID, \{P_{PID}\}\rangle \in \overline{DS}_{j-1}^{i-n}$, such that $PID = t.Id$, and $ds \subseteq \{P_{PID}\}$, where t.Id is tuple t's Id.

A *join* pattern extracted wrt relation R_{i-n} is also a descendant pattern, thus, it is said to be large if it is contained in more than minimum support of the trees of tuples in R_{i-n}. We then map each extracted large join pattern to a unique integer. The set $Join_j^{i-n}$ composes all large join patterns extracted at relation R_j wrt relation R_{i-n}

Phase 3:

This phase is applied for descendant relations of R_{i-n}, and not for relation R_{i-n} itself. Following the algorithm to extract sibling patterns, we construct a transformed database. In [AS9?], a transformed database D_T is constructed. Here however, we construct a different transformation. We use the notation $D_{T_j}{}^{i-n}$ for the transformed database constructed for relation R_j, while extracting relational patterns wrt relation R_{i-n}. In [AS9?], each tuple is replaced by the set of 1-sequences that are contained in that tuple.

For sibling patterns, for an arbitrary relation R_j that is not a leaf relation we use as 1-sibling descendant patterns from three groups: Large itemsets extracted at phase 1, Join patterns extracted at phase 2, and descendant patterns from DS_{j-1}^{i-n}. Each entry in $D_{T_j}{}^{i-n}$ includes all large itemsets extracted at phase 1, Join patterns extracted at phase 2, and descendant patterns from DS_{j-1}^{i-n} that are contained in the tree of the corresponding tuple in R_j (A pattern $ds \in DS_{j-1}^{i-n}$ is contained in the tree of tuple t from relation R_j, if there exist $\langle PID, \{P_{PID}\}\rangle \in \overline{DS}_{j-1}^{i-n}$, such that $PID = t.Id$, and $ds \subseteq \{P_{PID}\}$, where t.Id is tuple t's Id).

For a leaf relation we only include large itemsets generated at phase 1.

A sibling collection is now represented by a (unordered) list of sets of 1-siblings. Each set of 1-siblings is represented by $\{b_1, b_2, ..., b_n\}$ where b_i is a 1-sibling. As in [AS9?], if a tuple in R_i does not contain any 1-sibling, it is dropped from the transformed database. In addition, a sibling collection that non of the tuples that compose it contain any 1-sinblng, it is also discarded from $D_{T_i}{}^{i-n}$, however, it still contributes to the count of the total number of tuples.

Since the elements that compose $D_{T_i}{}^{i-n}$ are patterns represented by unique integers as are large intemsets which compose D_T in [AS9?], we apply the approach used in the sequence phase in [AS9?] to extract ($n \geq 2$)-sequential patterns, to extract ($n \geq 2$)-sibling patterns from $D_{T_i}{}^{i-n}$. Each extracted large ($n \geq 2$)-sibling patterns is then mapped to a unique integer. The set S_j^{i-n} comprise all large ($n \geq 2$)-sibling patterns extracted at level j wrt relation R_{i-n}.

Phase 4:

In this phase we construct DS_j^{i-n} and $\overline{DS}_j^{i-n}$. The set DS_j^{i-n} contains all patterns contained in the sets DS_{j-1}^{i-n}, S_j^{i-n} and $Join_j^{i-n}$. To generate the set $\overline{DS}_j^{i-n}$, for each member $\langle PID, \{P_{PID}\}\rangle \in \overline{DS}_j^{i-n}$ corresponding to tuple t in relation R_{j+1}, the set $\{P_{PID}\}$ is composed of all descendant patterns in DS_j^{i-n} that are contained in t's tree (i.e., it contains all sets $\{P_{PID}\}$ from the elements $\langle PID, \{P_{PID}\}\rangle \in \overline{DS}_{j-1}^{i-n}$ corresponding to t's descendants in R_j, all patterns in $Join_j$ that are contained in the trees of t's descendants in R_j, and all ($n \geq 2$)-sibling patterns generated from $D_{T_j}^{i-n}$ contained in t' s tree).

The results of the mining procedure includes all the extracted relational patterns (including descendant patterns): $\bigcup_j AL_j^{i-n}$, $\bigcup_j S_j^{i-n}$, and $\bigcup_j J_j^{i-n}$ where j goes from the leafs index i to $i-n$.

It is important to differentiate between relational patterns extracted at different levels, since the same sibling pattern extracted at different levels actually represent different patterns. Take for example the database in Fig, 4, and assume we mine the database wrt relation R_1. Now consider the following descendant pattern extracted at level 1 (i.e., from relation R_3): $\langle\{(\Pr oc. = ECG)\}, \{(\Pr oc. = BloodCount\}\rangle$. As a descendant pattern extracted at the level of relation R_3, the interpretation of this pattern is that the two procedures are performed on patients while hospitalized in the same department. This is so since the sets {(Proc. = ECG)}and {(Proc. = BloodCount)} are contained in sibling tuples (i.e., with a mutual parent tuple in R_2). However the same descendant pattern extracted at level 2 from the transformed database $D_{T_2}^1$ means that each procedure is performed on the patient while hospitalized in different departments. This is so, since the patterns {(Proc. = ECG)}and {(Proc. = BloodCount)}were contained in different entries of $D_{T_2}^1$ pertaining to different tuples in R_2.

Example.

We now show an example for the algorithm described above. Assume we are interested in extracting relational patterns from the database depicted in Fig. 4 with respect to hospitalization cases (i.e., wrt relation R_1). Thus, we will compute the support for relational patterns wrt relation R_1. Also assume that the minimum support parameter s is 30%.

We start with extracting descendant patterns from the leaf relation in the database tree representation, i.e., relation R_3.

Extracting Descendant patterns from R_3 wrt relation R_1:

Phase 1: Following phase 1 we extract large itemsets from relation R_3, as shown in Figure 12 below.

AL_3^1

Large Itemsets	Mapped to
{Proc = BloodCount}	1
{Proc = ECG)	2
{Proc = X-Ray}	3

Figure 12: Large itemsets extracted from R_3

Since R_3 is a leaf relation in the tree representation of the database we skip phase 2.

Phase 3: In phase 3 we construct the transformed database $D_{T_3}^1$ shown in Fig. 13, where each entry corresponds to a tuple in R_3 and contains all the large itemsets that are contained in that tuple. $D_{T_3}^1$ is shown in Fig. 13 in its sibling collection form, where a sibling collection is represented by an unordered list of sets of integers, and where each integer represents a pattern contained in the tree of the corresponding parent tuple in relation R_3.

Since relarion R_3 is a leaf relation in the tree representation of the database, and therefore the tuples in R_3 have no descendants, we don't incorporate any other patterns in $D_{T_3}^1$ apart from large itemsets extracted from R_3 itself.

$D^1_{T_3}$

Patient ID	Dept.	Sibling Collections
100	ER	$\langle\{1\},\{2\}\rangle$
200	Pediatric	$\langle\{1\},\{3\}\rangle$
300	ER	$\langle\{1\},\{2\}\rangle$
400	ER	$\langle\{1\}\rangle$
400	Pediatric	$\langle\{3\}\rangle$

S^1_3

Sibling patterns	**Mapped to**
$\langle\{1\},\{2\}\rangle$	4

Figure 13: The transformed table $D^1_{T_3}$ and the et S^1_3 of sibling patterns extracted from it

We then use $D^1_{T_3}$ as it is used in [AS9?], and employ the sequence phase to extract ($n \geq 2$)-sibling patterns. All ($n \geq 2$)-sibling patterns extracted from $D^1_{T_3}$ are shown in Fig. 9.

Phase 4: We now construct DS^1_3 and $\overline{DS}^1_3$. The set DS^1_3 contains all previously extracted descendant patterns. Thus it includes all large itemsets, Join patterns, and ($n \geq 2$)-sibling patterns extracted up to this stage. DS^1_3 and $\overline{DS}^1_3$ are shown in Fig. 14.

$$DS^1_3 = \{1,2,3,4\}$$

$\overline{DS}^1_3$

Patient ID	Dept.	
100	ER	1,2,4
200	Pediatric	1,3
300	ER	1,2,4
400	ER	1
400	Pediatric	3

Figure 14: The set of descendant patterns DS^1_3 and the set $\overline{DS}^1_3$

We now move upward to extract descendant patterns at the level of relation R_2.

Extracting patterns at level 2:

Phase 1: All large itemsets extracted from R_2 wrt relation R_1 are shown in Fig. 15 below.

AL_2^1

Large Itemsets	Mapped to
{(Dept.=ER)}	5
{(Dept.=Pediatric)}	6
{(LOS=1D)}	7
{(LOS=2-3D)}	8

Figure 15: Large itemsets extracted from relation R_2 wrt R_1

Phase 2: We generate Join patterns by joining members in the set DS_3^1 with those in the set AL_2^1. Figure 16 shows the set J_2^1 containing all the large Joins mapped each to a unique integer.

J_2^1

Join Patterns	Mapped to
5 and 1	5
5 and 2	6
5 and 4	7
6 and 3	8

Figure 16: The set J_2^1 of Join patterns

Phase 3: The transformed database $D_{T_2}^1$ is shown in Fig. 17. Each entry is replaced with all previously extracted large itemsets, join patterns and sibling patterns that are contained in the tree of the corresponding tuple in R_2.

$D_{T_2}^1$

100	$\langle\{(8)\}\{(1),(2),(4),(5),(9),(10),(11)\}\rangle$
200	$\langle\{(1),(3),(6),(8),(12)\}\rangle$
300	$\langle\{(1),(2),(4),(5),(9),(10),(11)\}\rangle$
400	$\langle\{(1),(5),(9)\}\{(3),(6),(12)\}\rangle$

Figure 17: The transformed database $D_{T_2}^1$

We then extract ($n \geq 2$)-sibling patterns from $D_{T_2}^1$, however, no large sibling patterns are discovered.

Phase 4: DS_2^1 and $\overline{DS}_2^1$ are constructed to support the discovery of Join patterns at the next iteration at level 1. Fig. 18 shows DS_2^1 and $\overline{DS}_2^1$.

$$DS_2^1 = \{1,2,3,4,5,6,7,8,10,11,12\}$$

$$\overline{DS}_2^1$$

100	1,2,4,5,9,10,11
200	1,3,6,8,12
300	1,2,4,5,9,10,11
400	1,3,5,6,9,12

Figure 18: The sets of descendant patterns DS_2^1 and $\overline{DS}_2^1$

Extracting patterns at level 2:

Phase 1: Finally, at level 1, in the first phase we extract all large itemsets in R_1 wrt relation R_1 shown in Fig. 19.

Phase 2: At the second phase we extract all large Join patterns as shown in Fig. 19.

AL_1

Large Itemsets	Mapped to
{(Sex=Female)}	13
{(Sex=Male)}	14
{(Age=60-65)}	15

J_1^1

Join Patterns	Mapped to
1 and 15	16
2 and 15	17
4 and 15	18
5 and 15	19
9 and 15	20
10 and 15	21
11 and 15	22

Figure 19: Set of all large itemsets and all Join patterns extracted at level 1

As we mentioned earlier, we may extract relational patterns wrt to any relation in the database, and the choice of relation with respect to which we extract patters pertains to the application and analyst requirements. For example, if the subject for the inquiry are patients' visits in the hospital's departments, our natural choice would be to extract patterns with respect to relation R_2

7. Implementation and experience

The approach presented in this paper was implemented in a prototype of which architecture is presented in Fig. 20. The data source is discharge abstract data, collected for every hospitalized patient on a routine basis. The data are extracted from the patients' medical files, comprised during the hospitalization period, whereas only selected details from the medical file are entered via remote dumb terminals to a centralized database, to comprise a digital medical record.

The *data learning and visualization module* enables the user to perform via a graphical user interface, simple statistical tasks. These features provide preliminary data exploration and visualizations capabilities, through which the user can get a feel of the data, and that may also trigger some further investigations . This module includes features such as fields distribution via graphs, mutual distributions, and SQL queries.

The *knowledge discovery module*, uses the user's specifications for the discovery task. The specifications, which are acquired via a graphical user interface, instruct the extraction of relational patterns.

Finally the *presentation module* presents the resulted analyses and inquiries, in a comprehensible manner.

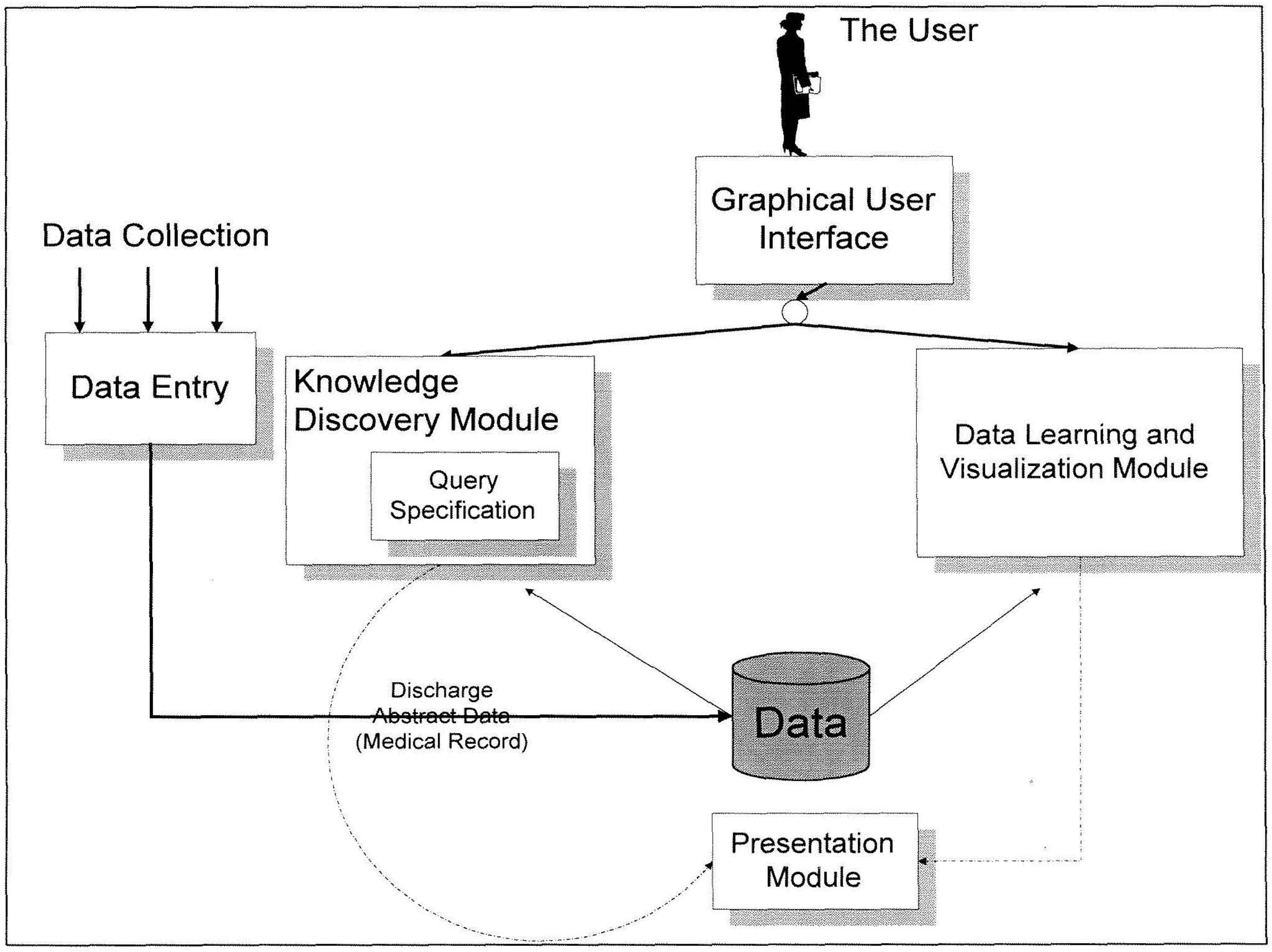
The User
Graphical User Interface
Data Collection
Data Entry
Knowledge Discovery Module
Query Specification
Data Learning and Visualization Module
Discharge ~~Abstract Data~~ (Medical Record)
Data
Presentation Module

Figure 20: System Architecture

We now provide a brief description of some observation in light of the use of the system. A significanat part of the difficulties that emerge are associated with the high diversity in the database. Many KDD link analysis applications [FPS96] reported in the literature were in the marketing domain (e.g., analyzing basket data) [] [] where the database contained items purchased in each transaction. In these cases the extracted patterns could be navigated toward decisions on positioning items, for example, or promotion campaigns. The point here is that it is reasonable to mine the database as a whole or maybe use some simple segments of it. When extracting relational patterns from hospital discharge abstracts based on statistical strength of "support", it was often meaningless to extract patterns from an extremely heterogeneous population of which partial list are geriatric patients, pregnant women merely arriving to the hospital to give birth, heart patients and chronic patients. Since we extract patterns according to their statistical strength it is important to perform the investigation on a more meaningful subset of the population. It was thus essential to enable the analyst to specify the population for analysis, and moreover, since we think of the mining process as iterative it was necessary for the analyst to specify different preferences based on the previous iterations. Another simple mean we used to specify the analyst preferences is choice of database fields (e.g., age, diagnoses, and length of stay).

Both the choice of population and relevant fields for the mining task supported the analyst in focusing the investigation according to one's interest. However these capabilities also helped moderating another difficulty: *Too many patterns* were extracted. An analyst extracting patterns in search for relevant knowledge practically can not consider hundreds or even thousands of pattern. This is a known problem, in particular in relation to link analysis [FPM91 [KMR94] [BMU97] [ST95] [ST96] [PT98]. Reducing the dimensionality of the database also enabled to substantially reduce the number of patterns extracted. However, this was still an obstacle. Most of the extracted patterns were known and/or of no interest to the analyst. Several interesting studies have been done on *interestingness* of patterns [ST95], [ST96] [PT98]. Some of these studies are based on incorporating domain knowledge or beliefs [PT98] which are then used to extract only patterns that are not already known and/or patterns that embed knowledge which may trigger some corrective action (i.e., actionability [MPM95] [MPM94] [AT97]]). Another difficulty the emerged concerns navigating extracted patterns into decisions. Whereas in the bulk of KDD applications, patterns required minimal or no further analysis to be navigated toward decisions, we learned that as the underlying problem is less structured and more knowledge intensive the more analysis is required to translate raw patterns into decision making.When quality management utilized the system to rip knowledge relating to quality related events it became evident that an additinal layer of analysis is required to refine the extracted pattrns into meaningful knowledge for them to act uppon.

References

[AIS93] Agrawal, R.; Imielinski, T.; and Swami, A.; 1993. Mining Association Rules Between Sets of Items in Large Databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pp. 207-216.

[AMS95] Agrawal, R.; Mannila, H.; Srikant, R.; Toivonen, H.; and Verkamo, A. I.; 1995. Fast Discovery of Association Rules. In Fayyad U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R.; eds.; *Advances in Knowledge Discovery and Data Mining*. AAAI Press.

[AS9?] Agrawal, R.; and Srikant, R.; 199?. Mining Sequential Patterns. ?????????

[AT97] Adomavicius, G.; and Tuzhilin, A.; 1997. Discovery of Actionable Patterns in Databases: The Action Hierarchy Approach. In *Proc. of the Third International Conference on Knowledge Discovery and Data Mining* (KDD 97).

[FPS96] Fayyad U. M.; Piatetsky-Shapiro, G.; and Smyth, P., 1996. From Data Mining to Knowledge Discovery: An Overview. In .; Fayyad U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R.; eds.; *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press.

[KMR94] Klemettinen, M.; Mannila, H.; Ronkainen, P.; Toivonen, H.; and Verkamo, A. I.; 1994. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proc. of the* Third *International Conference on Information and Knowledge Management,* pp. 401-407.

[PT98] Padmanabhan, B.; and Tuzhilin, A.; 1998. A Belief-Driven Method for Discovering Unexpected Patterns. Forthcoming in the *Proc. of the Forth International Conference on Knowledge Discovery and Data Mining* (KDD 98)

[SA9?] Srikant, R.; and Agrawal, R.; 199?. Mining Quantitative Association Rules in Large Relational Tables. ???????????/

[ST95] Silberschatz A.; and Tuzhilin, A.; 1995. On Subjective Measures of Interestingness in Knowlede Discovery. In *Proc. of the First International Conference on Knowledge Discovery and Data Mining*, pp. 275-281.

[ST96] Silberschatz A.; and Tuzhilin, A.; 1996. What Makes Patterns Interesting in Knowledge Discovery Systems. *IEEE Transactions on Knowledge and Data Engineering. Special Issue on Data Mining*, Vol. 5, No. 6, pp.970-974.

[CNF96] Cheung D. W.; Ng, V. T., and Fu Y. "*Effecient Mining of Association Rules in Distributed Databases*". IEEE transactions on knowledge and data engineering, vol. 8, no. 6, December 1996

[MTV94] Manila, H; Toivonen, H., and Verkamo, A. "*Effecient Algorithms for Discovering Asso ciation Rules*". Proc. AAAI Workshop Knowledge Discovery in Databases, pp. 181-192, July 1994.

[SON95] Savasere, A; Omiecinski, E., and Navathe, S. "*An Effecient Algorithm for Mining Association Rules in Large Databases*". Proc. 21th Int'l Conf. Very Large Data Bases, pp. 432-444, September 1995.

[MPM95] Matheus, C. J; Piatetsky-Shapiro, G.; and McNeill, D. "*Selecting and Reporting What is Interesting: The KEFIR Application to Health Data*". Advances in Knowledge Discovery and Data Mining, AAAI / MIT Press, 1995, pp.495-516.

[MPM94] Matheus, C. J; Piatetsky-Shapiro, G.; and McNeill, D. "*An Application of KEFIR to the Analysis of Healthcare Information*". AAAI Workshop on Knowledge Discovery in Databases, pp. 441-452, July 1994