# Modeling Relational Data as Graphs for Mining*

Subhesh Pradhan Sharma Chakravarthy Aditya Telang

University of Texas at Arlington
IT Laboratory and CSE Department
sharma@cse.uta.edu, aditya.telang@mavs.uta.edu

## Abstract

The focus of this paper is to develop algorithms and a framework for modeling transactional data stored in relational database into graphs for mining. Most of the real-world transactions (e.g., money withdrawal, travel, phone calls) are recorded as individual transactions which needs to be transformed into a graph based on structural relationships embedded in them. We present a graph representation that not only preserves all information embedded in a database, but also removes ambiguity and information redundancy. We present a suite of space- and time-efficient algorithms for modeling graphs from relational data. Extensive experimental analysis shows the scalability of our approaches. From a pragmatic viewpoint, our framework separates database-specific aspects from modeling aspects to make it applicable for all database systems. Real-world data has been used for generating graphs and mining them for various patterns.

## 1 Introduction

In social networks, activities such as monetary transactions, communications between individual or parties, travel information of individuals etc. are captured as tuples in a relational database. Relational representation lacks (or loses) the structural relationships embedded in these transactions. For example, the same person making a withdrawal, calling another person, and traveling to a destination and meeting with a person are typically recorded as 4 tuples in different relations. If it is possible to infer these structural relationships from these instances, novel mining techniques can be applied to identify same or similar patterns. In this work, a large real-world relational database (provided by an agency) was used for this purpose. The challenge was to model the structural relationships using only the tuples and information embedded in those tuples such as keys and foreign keys. The resulting graphs were mined using a graph mining system (Subdue [1] in our case) to infer interesting patterns and anomalies.

After transforming relational data into graphs, finding previously unknown but interesting patterns or finding *similar* (that are not identical but differ slightly from one another) patterns from these relations is accomplished through graph mining. The challenge is to deduce a pattern of activities for a group of people involved in frequent communication and monetary transactions. The difficulty for these applications is that although there is a structural relationship between the transactions captured (a calling b, and a moving from point x to y), they are not explicit in the stored representation. However, if we can extract and model the structural relationships from the data set and generate the structure, graph mining techniques can be readily employed. Graph-based data mining represents a collection of techniques for mining a data set with structural relationships represented in the form of a graph. Subdue [1, 2, 3] is one of the early graph mining algorithms that detects the best substructure using the minimum description length principle. Subdue can also mine for interesting concepts, detect anomalies, and similarities between structures. Once the data is converted into graphs, any graph mining system including FSG [4], gSPAN [5], and others [6, 7] can be effectively used.

In a relational database, some of the relationships between instances (of tuples) are available through foreign key values. Hence, underlying structural relationship of data is not explicit in a relational database. However, the structural relationships embedded in the data set are essential for inferring interesting patterns using graph mining. So, in order to discover the interesting pattern from relational database using graph mining, the underlying structural pattern has to be identified. In other words, transformation of a data set from relational instances into graph instances is essential for graph-based mining of the data set.

**Contributions:**

1. The primary focus of this paper is to develop a framework for modeling the underlying structure embedded in the data set of a relational database. Once

*This work was supported, in part, by the Air Force Grant 26-0601-11 and the NSF Grant IIS 0534611.

*15th International Conference on Management of Data COMAD 2009, Mysore, India, December 9–12, 2009*

the data is modeled as graphs, they can be used as the input for graph-based data mining (Subdue [1, 2, 3] in our case). Without this modeling, it is not possible to apply relevant graph mining techniques for a large class of data sets that are stored and managed by an RDBMS. Applying conventional mining techniques (e.g., association rules) is not likely to produce the same kind of end results or inferences as these approaches do not take into consideration structural aspects of the data.

2. A suite of algorithms have been developed to efficiently generate graphs from the data stored in a relational database using primary key and foreign key information. Using the framework described in this paper, relations of any database can be transformed into a graph. The graph representation used in our approach is a modified from conceptual graph representation [8]. The graph representation used requires less space (less number of edges and vertices) than the conceptual graph and preserves all relationship information.

3. A real-world social network data has been used for analyzing the performance and scalability of the proposed approach.

**Roadmap:** Section 2 presents related work. Section 3 discusses the various graph representation of relational data and analyzes these representations. Section 4 describes algorithms that transforms and models relational data into graphs for mining. Section 5 discusses the architecture of DB2Graph suite of algorithms. Section 6 provides experimental analysis of DB2Graph and Section 7 concludes the paper and outlines future work.

## 2 Related Work

***BANKS:*** BANKS [9] provides keyword-based search on relational databases, together with data and schema browsing. It enables users to extract information in a simple manner without any knowledge of the schema or any need for writing complex queries by modeling tuples as nodes in a graph, connected by links induced by foreign key and other relationships. Answers to a query are modeled as rooted trees connecting tuples that match individual keywords in the query. Even though it uses foreign and primary key constraints for graph representation of relations, the graph representation is not suitable for data mining.

***RDB2Graph:*** RDB2Graph [10] provides an approach for the transformation of a relational data into a graph. It infers conceptual graph [8] from the relational data and generates the instances of the graphs from the populated instances of the relations. The graph generated from RDB2Graph is used as input to Subdue for graph-based mining. Although RDB2Graph transforms data from an Oracle database to a graph, the approach is not general. It does not address the problem of conversion of a relational database having composite primary keys and foreign keys. In addition, he approach is so tightly coupled with Oracle cannot be used with other DBMSs. Further more, the algorithm does not optimize limited resources such as memory space and processing speed.

A number of reversing engineering approaches and tools [11, 12, 13] have been developed for the RDBMS. However, most of them deal with the extraction of the EER (extended entity relationship) model from the relational schema. Their main purpose is to obtain (or reverse engineer) the design to analyze it in various ways. These approaches do not address the generation of graphs from the instances for the purpose of mining.

## 3 Graph Representations for a Relational Database

Relations in a relational database can be represented as a graph in a number of ways. A graph representation used for Relational Database representation should preserve all the relational information represented by the database. In this section we will analyze different graph representations for relational databases using two small relations: Employee (SSN is primary key and DepNo references Department) and Department (DepNum is primary key) as shown in Figure 1.

***Conceptual Graph:*** Representing relations as conceptual graph [8] as shown in Figure 2 retains all relational information in the database; but, with some ambiguity. Even though the foreign key attribute node shows relationship between two tables, it does not capture the attributes on which the relations are related. Also, the significance of edges is not conveyed properly.

***Entity-to-Attribute (EA) Graph:*** An Entity-to-Attribute Graph is an alternative representation. In this, attribute names are represented as edge labels instead of nodes and common attributes between relations are shown as directed edges from each relation to the common attribute. The relational schema and data set are represented as shown in Fig 4.

***Relation of Relations Graph (RR-graph):*** Relation of Relations Graph (RR-graph) provides reference information (which relation references which relation) along with the attributes. RR-Graph is a directed graph where nodes represents relations (tables) and a directed edge from node R1 to node R2 implies that relation R1 is referenced by relation R2 and the label associated with the directed edge represents the attribute. That is, R2 has a foreign key to the primary key of R1.

***Differences Between Conceptual and Entity-to-Attribute Graphs:*** The number of vertices and edges in a graph determines the space required for the representation of a graph. Since the graph representation of a relational database will be used as input to graph mining algorithms, both processing time and memory requirement for graph mining is dependent on the number of edges and vertices needed for the graph

Employee

| SSN | Name | DepNo | Age |
|---|---|---|---|
| 1000 | Bill | 302 | 24 |
| 1001 | John | 303 | 23 |

Department

| DepName | DepNum |
|---|---|
| EE | 302 |
| CSE | 303 |

Figure 1: Sample Tables from a Relational Database

representation of the relational database. This is critical especially when the database size is large. Hence, we compare the number of edges and vertices required for Graph representation of a relation as a Conceptual Graph and as an Entity-to-Attribute Graph. Let n be the number of tuples and m be the number of attributes in a relation. In the Conceptual Graph, for each tuple of the table we need one node for the table name, m nodes for attribute names, m nodes for the attribute values and 2m edges to show relation between each node. Thus, for a conceptual graph representing a relation, *Number of nodes = 2*m*n+n*, and *Number of edges = 2*m*n*. In an Entity-to-Attribute Graph, for each tuple of the table we need one node for table name, m nodes for attribute values and m edges representing attribute names. Thus, for an EA graph representing a relation, *Number of nodes = m*n+n*, and *Number of edges = m*n*.

In comparison to Conceptual Graph, Entity-to-Attribute Graph requires approximately half the number of vertices and edges to represent a relation (50% reduction). Hence, Graph mining algorithms will process Entity-to-Attribute Graph much faster than the Conceptual Graph.

***Graph Representation for Subdue:***

A relational database can be represented as a graph in various ways. But, important criteria for the graph representation is that it should capture the structure present in the data set, should be concise, and yield meaningful results when processed by graph mining algorithms such as Subdue and others. Any graph representation unable to attain the criteria was not considered for graph representation. For example, in BANKS, each tuple from all tables are represented as individual node. Then nodes will have edges (forward and backward) with table name as labels whenever two tuples are connected. This approach was not considered for graph representation because it cannot be processed by Subdue or other widely used graph mining systems. Also, considering an entire tuple as a node does not allow the detection of interesting patterns with respect to individual attributes of a relation. Even though Subdue can process Entity-to-Attribute graph, a few modifications on the representation was applied to maximize the effectiveness of Subdue. This representation captures both the connectivity between the relations as well as the value of the attribute without increasing the number of edges or nodes. After modifications, the graph representation approach is denoted as Entity-to-Entity (EE) Graph. In this graph representation, as shown in Fig 3, foreign key of a relation is represented as a directed edge from the referencing relation to the relation referenced. DB2Graph algorithm presented in this paper is capable of transforming any relational database to a forest of graphs according to either Entity-To-Attribute or Entity-to-Entity graph representation.

# 4 Algorithms for Relations-to-Graphs

In this section, we present several algorithms for the transformation of a relational database into a collection of graphs. These algorithms have been implemented and compared for space and time efficiency. The space requirement is dictated by the space required to maintain intermediate data (we call it lookup space) while performing the transformation. Since we are focusing on large databases we have used database relations to store the intermediate data as it gave us benefits of scalability and portability.

## 4.1 Naive Algorithm

Naive algorithm (1) is the approach implemented by RDB2Graph [10].

**Algorithm 1** Naive Algorithm

```
1: for each relation (table) R (chosen arbitrarily) from the database do
2:     for each tuple t in the relation do
3:         Create a node for relation name (will be known as table name node)
4:         Create nodes for all the attribute values in the tuple except for foreign key attributes
5:         Connect it with table name node (directed edge from table name node to attribute value node) with edge label as attribute name
6:     end for
7: end for
8: for each relation (table) R (chosen arbitrarily) from the database do
9:     Let the relation referenced by R be S
10:    for each tuple t in the relation R do
11:        Connect the table name node of t with primary key value node of the tuple t' of S that has same value as foreign key value of t. (directed graph from table name node of t to primary key value node of t').
12:    end for
13: end for
```

***Space requirement:*** The primary cost of the transformation process comes from the amount of intermediate data that needs to be maintained affecting both efficiency and scalability. As shown in Algorithm 1, it transforms data from a relational database to a forest of graphs in two steps. First step transforms all relations into graphs (done in steps 1-7). In this step, each tuple of a relation is represented as a connected subgraph. To perform this step we need schema information such as relation name, attribute names, primary and foreign key attributes of a relation. Second step connects the subgraphs on the basis of primary and foreign key attributes of the relations (done in step

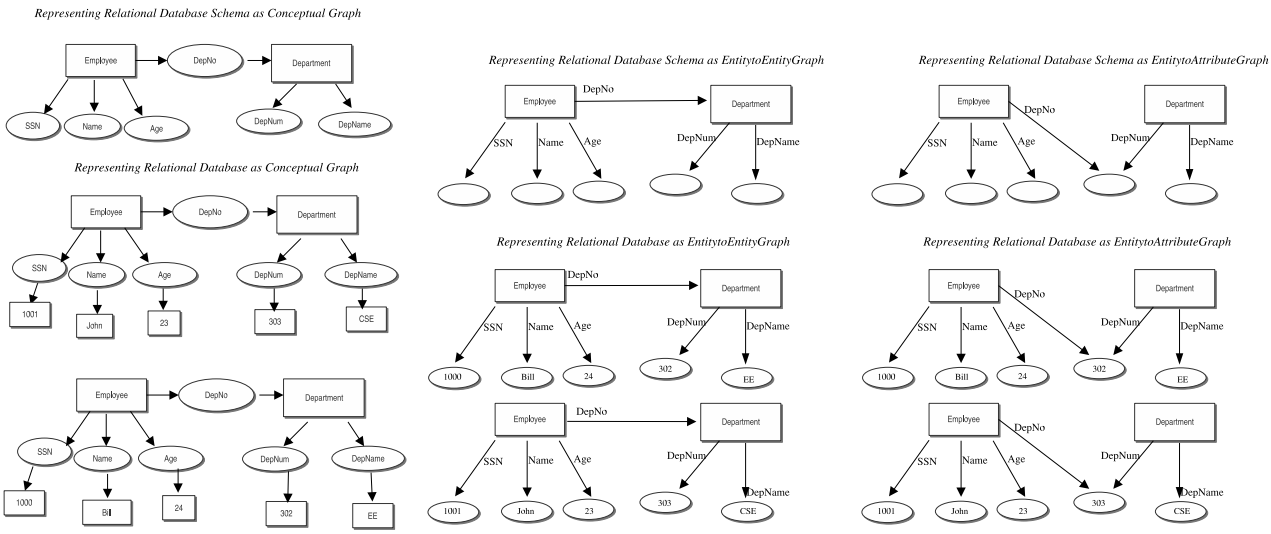


Figure 2: RDBMS as Conceptual Graph

Figure 3: RDBMS as Entity-To-Entity Graph

Figure 4: RDBMS as an Entity-to-Attribute Graph

11). To accomplish this step, we have to store primary key vertices (nodes) and foreign key values of every tuple of each relation in the database (or in memory) for lookup. In this algorithm, relations are chosen for processing arbitrarily and hence it does not keep track of the relations that have been processed and relations that are yet to be processed during the transformation. Due to the absence of this information, there is no way of determining the utility of data stored during the transformation. Hence, it requires a large amount of lookup space for completing the transformation.

### 4.2 DB2Graph Algorithm

The naive algorithm transforms the relational database in two steps and in order to complete the second step it has to maintain attribute values of all relations regardless of their utility. If we can interleave the two steps and maintain the record of the connectivity as needed, we may be able to identify the relations whose attribute values are not required and save lookup space. We can further reduce the intermediate space requirements by processing relations in some order.

***Minimizing Space Requirement for Lookup:*** In this algorithm, we interleave the two steps of the Naive Algorithm so that while transforming a relation into a graph, we also connect the subgraphs to the subgraphs of the relations that have been processed earlier. And if we maintain the information of the relations that has been processed earlier, we can delete information that will not be needed in the future. Hence, significantly less space can be used for the transformation.

Consider RR-Graph in Figure 5. In the figure, number of tuples (rows) for each vertex (relations) is shown above the vertex where 1K is equivalent to 1000 tuples. For simplicity, we assume that the primary key (PK) of every relations consist of only one attribute (i.e., primary key is not a composite key), the size of the all primary keys and foreign keys (FK) are same and the space required to store the corresponding vertex number of the primary key value or foreign key value is negligible. In Figure 5 the edge labels (referencing attributes) are omitted as we are only calculating the space requirement. For this graph, for the naive algorithm, the lookup space requirement will be the sum of the space required for storing all primary and foreign keys for *each* relation. That is, for the **primary keys**, the *lookup space used = (4K+4K+3K+2K+1K)*PK size = 14K*PK size*. Similarly, for the **foreign keys**, *4K*3+4K*2)*FK size = 20K*FK size is the lookup space used*. Therefore, a total of *34K*PK* size is the lookup space needed.

However if we use the sequence R2, R3, R4, R1 and R5 for performing the transformation, we do not have to maintain the lookups for relation R1 and R5. This sequence allows us to create nodes that will be references by R1 and R5. Furthermore, after processing R1 we can delete the lookup space of R4. The lookup space requirement *after processing R2,R3,R4* would be *(1K+2K+3K) = 6K*primary key size*. Likewise, *after processing R1 after deleting lookup space for R4*, the lookup space needed would be *(2K+3K) = 5K*primary key size*. As this example illustrates, by judiciously choosing the order of the relations processed, one can reduce the space requirement significantly. For the above example, the maximum memory requirement will be *6K*size of primary key* which is almost **6 times less** than the brute force approach which uses *34K*size of primary key*.

***Determining the Sequence of Relations for Transformation:*** As exemplified by the example, determining the sequence of relations that will result

in minimum lookup space requirement is important. Thus, i) *selecting Relations with smaller PK and FK is useful*, and ii) *selecting Relations with smaller cardinality (total tuple size) is useful.*

As observed above, both the size of PK and FK as well as the cardinality play a key role in reducing the space requirement. Apart from the above, the sequence that allows to delete unwanted intermediate information is the dependency of the relations. This dependency is captured in the RR-Graph in terms of the directed edges. Hence, the number of edges incident on the node (in-degree) and number of edges going out from a node (out-degree) also effects the space requirement for lookup. Also, once the primary key values of a relation in a lookup is stored, it can be referenced by any number of relations.

The indegree of a node in an RR graph denotes the number of relations that should be processed earlier in order to make use of the information already available in the lookup space. The outdegree, on the other hand, denotes the number of attributes that need to be stored for use by adjacent relations. The maximum outdegree with respect to any adjacent node is used as that denotes the number of attributes that need to be stored. Based on the above, the space required to maintain the lookup can be denoted as the weight of a node can be calculated as: *Weight of a node=(MAX{outdegree with any adjacent node} + indegree)* total tuple size.* So, a better approach to reduce the lookup space is to use the order of relations based on the weight assigned to the relations in the RR-Graph.

*Lemma: Nodes in the RR graph should be processed in the minimum weight order. If nodes have the same weight, choose one arbitrarily.*

The above equation not only takes the cardinality (or total tuple size) of each relation but also the amount of information that need to be retained for processing the rest of the relations. The weight accurately estimates the space requirement by taking into account *both* cardinality as well as the number and size of attributes that need to be stored.

Given the RR graph and meta data for each relation, the weight of each node can be calculated *statically once* and used for transforming a relational database into graphs for mining. However, the RR graph changes as each relation is processed and the static order may not remain the same. Hence dynamic re-computation (or update) of the weights after processing each relation can further optimize the space usage.

## 5 Implementation of DB2Graph

Different relational database platforms (e.g. Oracle, DB2, SQLServer, MySQL) use different metadata

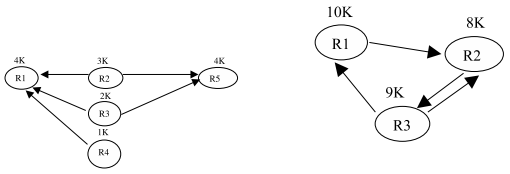


Figure 5: RR-Graph

Figure 6: Illustrative RR-Graph

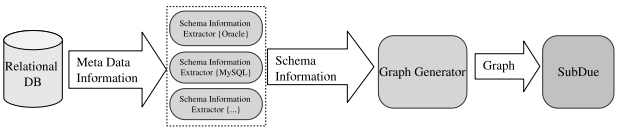


Figure 7: Architecture of DB2Graph

representations. So, it is not possible to use generalized query or module to extract the schema information such as the list of tables, list of attributes, foreign key constraints and primary key constraints. On the other hand, most of the relational databases follow SQL92 standard for data manipulation and data definition. We exploit this by using a graph generator module that is independent of a DBMS and writing separate schema extractor module for each DBMS. So, DB2Graph is architected to minimize system development time for adding a new relational database management system (DBMS) as shown in the Figure 7.

***Schema Information Extractor:*** This module extracts the schema information such as the list of relations (tables), list of primary key and foreign key attributes associated with each relation, size of those attributes, number of tuples in each relation. After extraction, it will store this information in system data structures, which will act as input to the Graph generator module. As the schema information is dependent on the database, we need to develop platform dependent schema information extractors. As shown in the Figure 7 we have currently developed schema information extractor for Oracle and MySQL.

***Graph Generator:*** Graph Generator gets schema information from the schema information extractor and transforms the relations in the database to a graph. Since all the information will be provided to this component in the form of system data structures, this component will be independent of the database being transformed. This module converts the database into graph using SQL92 standard queries. We have developed several graph representations and algorithms for each that transforms a populated database into corresponding graphs. For additional details of implementation, please refer to [14].

***DB2Graph Algorithm:*** DB2Graph algorithm maintains a weight table and RR-Graphs of relations to select the relation with minimum weight and for dynamic update of weight. Weight table is also used for evaluating the utility of lookup for the relations and the lookups which do not have utility are deleted af-

| Details | DB 1 | DB 2 | DB 3 |
|---|---|---|---|
| # of tuples | 14800 | 19381 | 41754 |
| # of attributes | 120 | 120 | 120 |
| Time for Naive (mins) | 21.76 | 30.48 | 78.41 |
| Time for DB2Graph (mins) | 12.7 | 17.15 | 75.6 |
| Max lookup space – Naive | 4002KB | 5050KB | 9152KB |
| Max lookup space – DB2Graph | 186KB | 216KB | 808.7KB |

Table 1: Lookup space & Execution time

ter processing a relation. DB2Graph algorithm transforms the relational database into a graph according to the Entity-to-Attribute Graph representation. DB2Graph algorithm can handle composite foreign keys and composite primary keys. It can also handle special case of the same attribute being part of primary key and foreign key. Since, DB2Graph algorithm uses dynamic update of weights and determines the sequence of relations to be transformed (i.e., selecting minimum weight), based on the discussions in the previous sections, DB2Graph requires minimal lookup space as compared to the Naive algorithm.

## 6 Experimental Evaluation

This section analysis of the Naive and the DB2Graph algorithms with respect to lookup space requirement and processing time. Time taken and space used by lookups for DB2Graph algorithm and SubDueDB2Graph algorithm were similar. So, only experimental results of Naive and DB2Graph algorithms are shown here. Experiments were conducted on databases varying in size. Database relations were used to store the intermediate data (lookup space size). Total space used for lookups for each algorithm was measured after transformation of each relation in database. Also, time taken by each algorithm to perform transformation was measured. Experimental results for the naive and the DB2Graph algorithms for various databases are shown in Table 1.

DB2Graph algorithm has to refer to lookup relations (tables) for every primary key and foreign key for each tuple of every relation. So, processing relations with more number of foreign keys is comparatively slower than processing relations with less number of foreign keys. In Database 3, size of relations having more number of foreign keys was larger (number of tuples in relation with 5 foreign keys was 14942, with 3 foreign keys was 7540, and with 2 foreign keys was 8,008). So, difference in processing time of Naive algorithm and RDB2GraphGen algorithm was less for Database 3. However, the savings in storage is as expected.

From the experiment results, it is evident that maximum space requirement for DB2Graph Algorithm is order of magnitude less when compared to the naive algorithm. Also, Naive algorithm requires more processing time than DB2Graph algorithm in all cases. The gain depends on the connectivity of relations. Scalability for transforming databases of very large sizes is accomplished with optimal intermediate storage by the DB2Graph algorithm.

## 7 Conclusions

In this paper, we have addressed the problem of modeling structural relationships from a relational database for the purposes of graph mining. We explored alternative graph models and evaluated their space and time efficiency as well as their suitability for mining. In order to achieve portability and database independence, the architecture has been modularized. The significance of this work has been in applying the results of this paper to large real-world activities data sets for mining patterns that could not be done otherwise.

## References


[1] Cook, D.J., Holder, L.B.: Graph Based Data Mining. IEEE Intelligent Systems **15(2)** (2000) 32–41

[2] Potts, J., Holder, L.B., Cook, D.J., oble, J.C.: Learning Concepts from Intelligence Data Embedded in a Supervised Graph . FLAIRS (2005) 480–484

[3] Jonyer, I., Holder, L.B., Cook, D.J.: Graph-Based Hierarchical Conceptual Clustering in Structural Database. Seventeenth National Conference on Artificial Intelligence **17** (2000) 1078

[4] Kuramochi, M., Karypis, G.: Frequent Subgraph Discovery. In: ICDM 2001: Proc. of the 2001 IEEE International Conference on Data Mining, Washington, DC, USA, IEEE Computer Society (2001) 313–320

[5] Yan, X., Han, J.: gSpan: Graph-Based Substructure Pattern Mining. In: ICDM'02: Proc. of the 2002 IEEE Int. Conf. on Data Mining. (2002) 721–731

[6] Washio, T., Motoda, H.: State of the Art of Graph-Based Data Mining. SIGKDD Explor. Newsl. **5**(1) (2003) 59–68

[7] Inokuchi, A., Washio, T., Motoda, H.: Complete Mining of Frequent Patterns from Graphs: Mining Graph Data. Mach. Learn. **50**(3) (2003)

[8] Sowa, J.F.: Conceptual Graphs Summary. In Gerholz, L.L., Nagle, T.E., Nagle, J.A., Eklund, P.W., eds.: Conceptual Structures: Current Research and Practice, Ellis Horwood (1992) 3–51

[9] Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., S.Sudarshan: Keyword Searching and Browsing in Databases Using BANKS. ICDE '02 (2002) 431

[10] Palod, S.: Transformation of Relational Database Domain into Graphs Based Domain for Graph Based Data Mining. Master's thesis, Department of Computer Science and Engineering,University of Texas at Arlington: Arlington (2004)

[11] Ramanathan, S., Hodges, J.E.: Extraction of object-oriented structures from existing relational databases. SIGMOD Record **26**(1) (1997) 59–64

[12] Henrard, J., Hick, J.M., Thiran, P., Hainaut, J.L.: Strategies for data reengineering. In: WCRE. (2002) 211–220

[13] Hainaut, J.L., Englebert, V., Henrard, J., Hick, J.M., Roland, D.: Database reverse engineering: From requirements to care tools. Autom. Softw. Eng. **3**(1/2) (1996) 9–45

[14] Pradhan, S.: A Relational Database Approach to Frequent Subgraph (FSG) Mining. Master's thesis, The University of Texas at Arlington (2006)