Well-organized association Pattern Mining using Multi-relational Data Cubes

K.V.N.R.Sai Krishna[1], CH. Krishnamohan[2]
[1]Department of Computer Science, S.V.R.M,College,Nagaram India
[2]Department of Computer science, P.B.Siddhartha College of Arts and Science, Vijayawada India

[1]kvnrsaikrishna@gmail.com
[2]km.mohan3@gmail.com

Abstract

A large class of data mining applications involves data sets that pertain to multiple entities and relationship. This has led to the suggestion of multi-relational data mining (MRDM) that aims to incorporate and exploit the heterogeneous and semantically rich relationships that exist among entity types. Specially, given a database consisting of multiple tables linked through foreign key joins, a target table (that typically represents a certain real-world entity type) and, optionally, a target attribute (e.g. a class label attribute), MRDM aims to discover patterns and models spanning all the tables and links that either describe or predict the target entity or attribute. In this paper, we study one class of MRDM task, namely multi-relational association rule mining. For this task, we distinguish two types of target tables, namely entity *and* relationship *target tables. For the case of entity target tables, we cast the computation of frequent patterns as multi-relational iceberg cube com-putation and propose an ef cient algorithm for it. Then, we study the application and peculiar requirements of tar-get tables that are relationship tables. This study reveals a new mining task, dubbed* linkage mining, *where the mere instances of relationships are the objects of mining. We then show how our multi-relational iceberg computation algorithm is extended to do linkage mining. In the end, we present performance studies of our algorithms.*

Key words *MRDM*, WARMR, Iceberg Cube Trie, MICube trie, DAG

1 Introduction

A number of analysis and data mining tasks (in a wide variety of applications including intelligence analysis, social network analysis, inter- organizational link analysis, web data mining, genomics) are based as much on the links among heterogeneous entities and events as the properties of individual entities. Relational databases facilitate such tasks by enabling the storage of data as multiple linked tables which together represent the conceptual entities and relationships. Nonetheless, techniques, like those developed in the Inductive Logic Programming led(ILP) [12], that aim to mine patterns involving relationships on realistically large databases are hampered by scalability problems.

As a result, most widely used data mining algorithms work only on at tables. In order to apply these algorithms, one is forced to convert multiple tables into one huge joined universal relation. A number of studies have show that this has several disadvantages. First, joining the relations into a huge ”universal relation” blows up the data both horizontally and vertically. This restricts the applicability of many mining algorithms that are already resource intensive [16, 7]. Second, not all real-world data can be converted into a at joined table without losing important semantic information carried by the join links. This is particularly true for domains where reasoning about the structure of objects and relations between them is inherently required [18]. Third, it increases data redundancy (duplication) that may introduce statistical skew [8].

In this paper, we focus on one class of MRDM task, namely multi-relational association rule (pattern) mining. The goal here basically is to discover association rules re-ferring to a particular real-world entity (i.e. the target ta-ble [1]) but spanning multiple tables in an arbitrary relational database. Association rules [1] are patterns involving two or more items co-occurring in a certain event (e.g. transac-tion). The rules in our case are conjunctions of atomic pred

where is a categorical or numerical attribute and is a value from a domain of values (intervals in case of numeric attributes). Each pattern (conjunction) has an associated *support* and *con dence*. Support of a pattern is the number of tuples in the database that satisfy it while con - dence is the probability of satisfying a certain subset of the conjunction given the whole conjunction.

For prolog databases and data log queries, Deshaspe and Toivonen have developed an ILP extension of the Apriori algorithm called WARMR [4, 17] that discovers association rules over a limited set of conjunctive queries. How-ever, even for the limited class of rules it discovers, this algorithm has been found to be very inefficient. Another category of work [13, 10] has considered relational association rule mining on star-schema where the joins are of length one. However, the latter body of work is not applicable to discover the general class of multi-relational patterns at which this paper aims.

In this paper, we explore a different approach to the discovery of multi- relational association rules that is based on extending Iceberg-cube algorithms. For the case of mining association rules from a single table having multiple at-tributes, the sub-task of frequent pattern discovery on a single table is efficiently processed in databases using Iceberg-cube algorithms [2]. These algorithms basically treat the generation of the set of conjunctive predicates that meet the user specied support threshold (also called iceberg condition) as a computation of a cube consisting of cells (i.e. con-junctions) that has all the attributes as dimensions. Iceberg-cube algorithms have been shown to be efficient and can be implemented in current databases with relative ease.

An algorithm to efficiently discover multi-relational Iceberg- cubes (i.e. frequent conjunctive rules) over multiple tables. Focusing on foreign key relations in a physical database, we propose an algorithm that addresses the challenge of combining cube computation with join path traversal while retaining the good I/O and CPU performance of Iceberg-computation algorithms. In addition to discovering rules (patterns) that span the en-tire database, we demonstrate that our algorithm is also able to discover rules that involve a limited level of recursion over a certain table.

Second, motivated by the traditional distinction between entity tables and relationship tables in database conceptual design, we take a closer look at the target tables and distinguish two types of target tables: *entity tables* and *relationship target tables*. We then observe that most MRDM algorithms in general and WARMR in particular fail to discover useful rules for relationship tables. This is due to the common implicit assumption that instances of mining are some real-world entities. However, when we consider re-lationship tables as target tables, we are essentially making relationships between entities the instances to mine.

We give a formal dentition of the task of multi-relational as-association rule mining with relationships as targets (which we refer to as *relationship/linkage rules*). Then we show that our multi- relational association rule mining algorithm can effectively and correctly mine such rules. To the best of our knowledge, ours is the rest work to systematically study and address multi-relational association rules with target re-lationship tables in the context of relational databases.

1.1 Motivating Examples

Consider a database consisting of information on re-searchers and their publications. Figure 1 shows the schema of this database. On this schema, consider the following patterns:

1. Researchers from Institute name=UCI co-author with researchers from Institute name=IBM. This is a

1. Researchers from Institute=IBM tend to publish their papers in venue type= conference as compared to venue type=journal. This is a pairing rule with the tar-get table
2. Researchers who co-author a paper together commonly belong to the same institute. That is, if researchers R1 and R2 are in a co-authorship relationship, then R1.Institute.Name = R2.Institute.Name. This is a pairing rule with the target table. Notice here that the pattern involves the *same* attribute of paired entities.
4. Researchers from universities that co-author with re-searchers from companies also tend to write papers that cite papers written by researchers in companies.

5. Researchers who co-author a paper together also tend to cite one another's paper.

2 The WARMR family of algorithms

The WARMR algorithm is an ILP algorithm that adapts the Apriori algorithm to mining association rules (represented as data log queries) within a certain class of conjunctive queries. Like Aprioiri, WARMR has two phases: candidate predicate set (conjunction) generation that meets a support threshold and then generation of con dent rules. To constrain the space of candidates, WARMR species a sub-set of all conjunctive queries (using *a declarative language bias* as is typically done in ILP). This constraints specify (1) the (like a primary key of a certain entity table) for mining that need to be available in all data log predicates (i.e. tables); this essentially determines the entity to be counted, and (2) a list of allowed patterns in which atomic predicates can be combined to form data log queries. To determine when a query is a superset (i.e. generalization) of another query in order to exploit Apriori like pruning, it applies an equivalence test similar to query containment in databases (called - subsumption in ILP). This is a very expensive process. Subsequent work [14, 15] has tried to reduce the cost of this computation by removing the need for PROLOG and using a different method for equivalence testing.

Our approach differs from WARMR in the following ways: (1) WARMR is based on a prolog database consisting of a set of predicates (facts) whereas in this paper we deal with a relational database. We do not require the speciation of any pattern structure; instead we directly use the links (i.e. foreign key constraints) among tables found in the database. (2) our candidate generation is based on a much more efficient Iceberg-cube computation algorithms.

(3) Although we require a target table, we do not require the key of this table to appear in all tables as in WARMR. (3) As will show later, WARMR cannot properly handle target tables that are relationship tables

3 Iceberg-CUBE Algorithms

To apply association rule mining over a table with multiple attributes, one approach is to transform it into a transaction (market-basket) table by making each atomic predicate (Attribute-value pair) an item. Obviously this representation is inefficient since the number of items involved be-comes large. An efficient way to generate predicate sets (i.e. ”item sets”) and do the required counting on arbitrary at-tribute tables is to use a data cube computation algorithm. A data cube consists of all possible group-bys (also called *cuboids*) over all grouping attributes (each of which forms the cube’s). A type of cube called Iceberg-cube is suitable for association rule mining. An *Iceberg query* is basically an SQL query with a HAVING clause. An *Iceberg-cube* is the complete cube over all attributes (dimensions) of a table where each cell is meets an aggregation constraint specied in the HAVING clause [2].

A popular Iceberg cube computation algorithm is BUC (Bottom-Up Cubing) [2]. BUC builds the cubes from smaller number of dimension combinations to larger ones. It first selects an attribute and groups the data on that at-tribute producing partitions pertaining to the attributes distinct values [2]. It then continues recursively partitioning (and aggregating) the database to compute cuboids composed of larger number of dimensions. At each step of recursive portioning, it evaluates the iceberg condition (minimum sup-port criteria) to remove those which fail to meet this criteria from further partitioning and aggregation. This approach of pushing the iceberg condition test down the cubing process allows BUC to achieve good performance. The order in which BUC performs group-bys is shown in gure scales well to large databases since it needs to load only a partition of the database that its in memory at a time. Moreover, it caches no data across multiple passes. However, being a depths approach, BUC is not optimal in terms of pruning. For example, suppose all cells in and in a cuboids AC at step fail to meet the condition. Then, it is not necessary to partition cells in the cuboids AB on C. But also notice that for this to happen values of cells in the cuboids AC should fail the. Otherwise, it is still

necessary to process ABC although we know that some cells in AC do not meet criteria. The later observation means that the pruning optimality of a breadth- first approach (e.g. Apriori [1]) may not result in greater savings in the case of Iceberg-cube computation. This is because unlike the case of Apriori which is based on item set construction through combinations of atomic items, here computation is based on combinations of attributes each of hose distinct values is an ”item”. Due to this, we adopt the BUC approach to use as a basis of our multi-relational Iceberg cube computation algorithm.

4 Multi-relational Iceberg-cubes

In this section, we explore how to exploit the approach in BUC to develop an algorithm for iceberg-cube computation over multiple relations for use as a basis of multi-relational data mining. Two major challenges are addressed here:

- how to efficiently interleave cubing (predicate set enumeration) with join path traversal. In particular, how to retain the scalability of BUC by ensuring that only a partition of a single table is read to memory at a time.
- how to minimize the number of tuples involved in the join paths from the target table to the linked table, and then to the tables that are two joins apart from that tar-get table, and so on.

In this section, we give an algorithm that addresses these challenges.

4.1 The Multi-Relational Iceberg Cube Trie

Before we go on to the description of our algorithm, we describe the data structure that we employ to enumerate group-by combination orders and to store counts and other information sufficient for mining multi-relational association rules. Iceberg cuboids and support counts for each cell in each cuboids are stored in a pre x trees (trie) referred to as *multi-relational iceberg cube trie* or simply *MICube trie*). Each node in an MICube trie corresponds to an attribute. The target table constitutes the root node. A node at depth represents the cuboids resulting from grouping by the at-tributes in the path from itself to the root. A particular node stores all information pertaining to cells in the cuboids rep-resented by the path from the root to that node. In addition to storing counts, the MICube trie makes it easy to enumerate cuboids (predicate sets) as will be illustrated with an example below.

In order to be able to represent group-bys over attributes from multiple tables in the MICube trie, we distinguish between two types of attributes: data attributes and join at-tributes. Join attributes consist of foreign and primary keys of tables while all the other attributes are considered to be data attributes. We make the reasonable assumption that se-mantic information about entities is held in data attributes while join attributes serve to store relationship information only. However, we still perform cubing on both types of at-tributes. Cubing on data attributes results in cells that consist of pairs while for join attributes the are foreign keys. Minimum support conditions apply to group-bys of join attributes as well. Semantically, aggregation (count) for group-bys on join attributes captures the number of links that exist between the two entities involved. In the MICube trie, we employ two types of nodes, namely data nodes and join nodes, that correspond to the two types of attributes. Hence, the multi-relational cube represented by MICube trie stores patterns over entity attributes as well as patterns over relationships.

Serve as the boundary of the cubing phase. By the time the recursion returns to the root, all the leaves in the cube tree will have either join nodes or a data node with the nest granularity. It is important to notice here that the attribute values in the join nodes are foreign keys.

Once cubing phase is done, the expansion phase takes over. Expansion extends the cubing to joined tables. The goals here are to perform expansion without increasing the memory requirement (as joins are carried out) and to avoid reading a data partition

more than once thereby retaining the advantage of the recursive approach of BUC. We achieve both of these goals by the following technique: pick the left most join node that has the smallest depth(the reason for this is given below). For this node, we also identify and store in a queue all other leaf join nodes

(called ”clones”) that also correspond to the same attribute by traversing the MICube trie horizontally backward from right-to-left. We then join only the rst node (by joining the foreign keys stored in this join node with the actual referenced table). Notice here that only tuples that meet the are joined.

Next, we launch the cubing phase again using the joined table and the clone queue. However, this time each data partition that is loaded to memory will be used to concurrently carry out cubing by all the clone nodes as well. This approach results in considerable I/O and CUP efficiency. Also notice that in addition to storing counts and enumeration of cubing order, our algorithm also exploits the MICube trie to effectively integrate cubing with join path traversal.

Our algorithm for multi-relational iceberg cube computation is shown in guru 5. The input to it are a data partition (initially the whole target table) and a queue of join nodes on which the partition is to be joined as expansion (which is initially empty). In line 2 the group-by order to be applied on the current partition is computed from the MICube trie. Line 4 partitions the data after sorting it based on the current grouping attribute. Then for each partition, once we test if it passes the iceberg condition (line 7) and also if the current cubing attribute is not a join attribute (line 10), we make a recursive call to). Notice that for join nodes, short of making a recursive call, we per-form all the other processing (i.e. partitioning, counting, updating the relevant node in the cube tree). Since at this point is empty, the expand function in the loop (lines 11-13) is not executed. But, as described above, when the recursion is being done after the expansion phase, this loop will be run as many times as there are clone join nodes in the MICube trie on which cubing can be performed using the current partition.

Once all the data attributes of the target table are processed (loop in lines 3- 19), the recursion returns upwards through all data nodes, at the completion of which the expansion phase is launched. We start, in line 22, by picking the join node that has the least depth and is at the eft most of the MICube trie. The

reason for picking the join node with the least depth is because it is the least aggregated one. This property of a least-depth join node guarantees that result of joining foreign keys in this node with the referenced table is sufficient to expand all the nodes in the MICube trie pertaining to a join attribute. The following lemma states this concretely:

Now we explain the second part of the join node selection criteria, namely picking the left most join node. The rationale for this also relates to the fact that cubing order is enumerated by appending all right-brothers of a MICube trie node. Expanding the left most join node allows this procedure to work correctly since all the right-brothers of that node (if any) can also be appended as the join node grows due to expansion. Notice that these two criteria for join node selection alone are sufficient to guarantee that the join nodes are traversed (expanded) correctly and efficiently without having to keep track of the order in which they were appended to the trie.

When all instances of selected join node and its clones are completely processed (i.e. cubed until all their leaves are join nodes or no more cubing is possible), we pick the next unique join node, and repeat the above process. This continues until no more cubing or expansion is possible (i.e. until all leaf nodes are data nodes with no right-brother). Notice that once a join phase starts, the cubing is encapsulated inside a join node expansion. Also notice that in the MICube algorithm, if a table (either the target table or any joined table) does not t in memory, then the algorithm will keep on partitioning it until it does.

4.2 Cubing order for Neighboring Join Nodes

While the elegant strategy of combining a data node with its right-brother to exhaustively enumerate cubing orders works well for data nodes, it presents difficulty for group-by combinations of two brother (same level) join nodes. We will illustrate this by example. In guru 4, consider the join node D at the right side of the tree represented by a dotted box (as a child of node C). Basically, this group-by order is equivalent to merging two join branches from the ancestor table (in this case the target table). Hence, a

cubing based on these two nodes (i.e. two foreign keys) is equivalent to pairing nodes from the cuboids formed by two join paths. However, the semantics of such pairings needs to be de-ned 3. We will do that in section 5. At this point we simply disallow such cubing orders.

4.3. MICube Trie Node structure

The MICube trie stores the iceberg cuboids that are computed on the path from the root to each node. Recall that a path in this trie represents a cuboids. Information on the cuboids and their resulting cells is stored in each node of the tree.

In each child node, all the attribute values are stored in an array and all the counts are in another array. In particular, the counts stored at the current node correspond to every pairing of an attribute value in the current node with every value of the parent node. For example if there are attribute values in a node and attribute values in its parent node and assuming no pairing is pruning, the current node will store counts. All these counts are stored as a single array and each value in the parent node stores the starting array index for all pairings done with it.

4.4 Join Node Expansion

We can view the tree given in guru 4 as a superimposition of three trees each at different level of granularity. We have already discussed the two levels. At the lowest (cell) level, we have a set of iceberg-cuboids. Links at this level are represented by the pointers stored in each MICube node. At the medium (cuboids) level, we represent the processing order of the cubing algorithm (i.e. group-bys ordering of attributes). The third and highest-level tree is that of the tables linked by join nodes. At this level, the tree - actually a directed acyclic graph (DAG) - can be interpreted as traversing join paths starting from the target table following join keys (attributes).

As mentioned in the algorithm description, we have two basic types of expansions on join nodes, namely one that involves join (done on the least-depth left join node) and another that is performed on clone nodes. For the former one, we use simple foreign key join. The latter can be per-formed by applying intersection (e.g. hash set intersection) of the keys in the join node and the corresponding key in a partition. To facilitate this as expansion is simultaneously done on multiple join nodes, all the current leaves with a join node as ancestor also store the keys of tuples in each cell.

4.5 Generation of rules from MICube trie

Once we computed the multi-dimensional Iceberg-cubes, the cuboids and counts stored in the MICube trie are used to generate multi-relational association rules. This can easily be performed by picking each path in the MICube

5 Cubing on neighboring join nodes

Now consider the problem of interpreting cubing combinations for two join attributes of a table which we simply disallowed in the MICube algorithm We address a simplied version of the problem where a table contains only join nodes since an approach for this case will also be applicable in the general case where the join attributes occur with data attributes in a table. Tables that contain only join attributes are commonly used to represent relationships between two or more entities in the database. Such tables are called relationship tables in traditional database conceptual design [19].

Now suppose this table is our target table. One way to interpret a multi- relational association rule

based on this table is as a rule that captures the pairing of entities in the relationship. Below we give a more concrete semantics of such a pattern:

5.1 Mining pairing with MICube

A naive approach to adapt the MICube algorithm for this task is to rst apply it on the table corresponding to each entity involved in the relationship to generate the Iceberg cubes and then generate pairings of cells from each iceberg cube to generate candidate pairing rules. Those candidates that are satis ed by sif cient tuples in the target relationship table are computed. Below we give a considerably more ef cient algorithm The algorithm has double advantage: we prune the cubing work for one of the tables, and more importantly we interleave the generation of the pairing rules along with the computation of iceberg cubes. The algorithm is given be-low:

6 Performance Analysis

In this section, we present the results of our

comprehensive study to validate the efficiency and scalability of our algorithm. In particular we show that:

✳Our algorithm scales well with the increase in average data size per table, average dimensionality per table, number of tables involved, average join path length, support threshold.

✳Our MICube Trie representation is compact and that it scales well as the leaf branching factor (fan out) in-creases.

Setup. We implemented our algorithm using Java. The implementation works on tables stored in DB2 running on a Xeon 2.4GHz PC with 2GB RAM. Our program runs as a stored procedure and interacts with DB2 only to load ta-bles. However, we do not always load a whole table. Par-ticularly, as has been described earlier, during an expansion operation, only parts of a new table that join with foreign keys in the current table that also meet the Iceberg crite-ria are loaded. The join of qualifying foreign keys with the referenced table is executed by the DB2.

As has been done in previous Iceberg papers, we assume that each table (not all tables involved in the computation) ts in memory[20, 2, 9, 21]. However, the results we report below include both IO time needed to load tables and CPU time.

In addition, in our implementation, we assume that all attributes have integer values. This, too, is a common as-assumption in cube computation in general [2, 21] in order to save space and facilitate sorting which is used for data partitioning. Other attribute types like string and interval values are mapped to integers in a preprocessing step. We do not include the time taken by this step in our experiments. A preliminary experimental result on efficiency of the MICube algorithm as measured by run time. This gure compares MICube's runtime with the ”Join-then-mine” approach that first creates a join of the relations to be mined (using outer joins with the target table when possible) and applies the single-table Iceberg cubing algorithm on it. In addition to the improved performance gain using MICube, the cubes generated by MICube have also effectively retained the relationship structure.

7 Related Work

The problem of multi-relational association rule mining was presented in [3, 4, 17]. This body of work de-scribed WARMR summarized in section 2. Subsequent work [14, 15] on a system called FARMR has tried to optimize WARMR by removing the need for PROLOG and

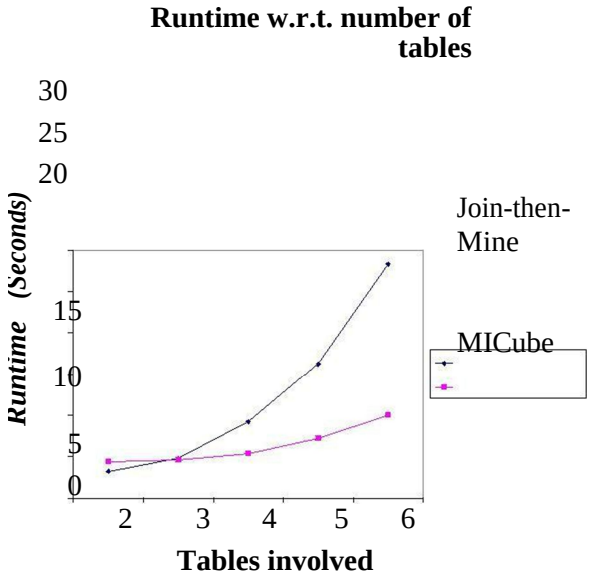

Figure 1. run time comparison of MICube with universal join approach

using a different method for rule equivalence testing. How-ever, FARMR also assumes a prolog database of facts and a WARMMODE style declarative bias.

There is also a recent body of work that focused on star schema based association rule mining [10, 13]. The problem addressed here is to mine association rules over the fact table without having to perform a join. The approach followed is to apply Apriori [1] on each table first and then merge the discovered frequent item sets to form multi-table frequent item sets. Unlike this body of work, we consider association rules over tables with a arbitrary depth of joins. Besides, use the recursive depth-first based approach of Iceberg-cube computation algorithms that allow us to efficiently generate frequent patterns as we traverse join paths.

In addition to the BUC algorithm [2], there are more optimized algorithms for Iceberg-cube computation [9, 20]. The algorithms suggested in these papers exploit shared computations of partitioning and aggregation to achieve better performance while retaining the possibility of applying Iceberg-conditions. We chose BUC as the basis of our approach because it is the most popular one and because it was simpler. However, the optimizations achieved by the latter works can also be incorporated with our algorithm to achieve even better performance.

Finally, the body of work on association rule mining over graphs [11, 22, 23]. However, this body of work considers only structural relationships and ignores the node and edge attribute. In this paper, we have tried to discover patterns that pertain to both entity attributes and their relationship structures.

8 Conclusions

In this paper our focus has been the development of techniques for frequent pattern discovery for the purpose of multi-relational association rule mining. To this end, we developed an algorithm that integrates Iceberg-cube computation methods with systematic join path traversal. In particular, our two phased algorithm performs recursive cubing and joined table expansion iteratively in order to compute all Iceberg-cubes that span multiple linked tables. We also provide a compressed data structure that enables to store the discovered Iceberg-cube along with cell-level counts. Then, for the purpose of discovering patterns over branch-ing multi-relational cuboids, we give semantics for one type of pattern and suggest a method to compute it efficiently based on our MICube algorithm.

There are a number of ways we plan to extend the work presented here. Primarily, we plan to study whether the techniques described in this paper can also be applied to other multi-relational data mining tasks in particular classication methods like decision trees. We also plan to study the various types of patterns (for e.g. pairing rules over same-entity type relationships like co-authorship where the two entities have shared attributes). Another issue to be explored is measures of interestingness for pairing rules other than the conventional support-con dence framework.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Min-ing association rules between sets of items in large databases. In *Proceedings of SIGMOD*, pages 207–216, 1993.

[2] Kevin Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD*, pages 359–370, 1999.

[3] L. Dehaspe, , and L. De Raedt. Mining association rules with multiple relations. In *Proceedings of the 7th International Workshop on Inductive Logic Pro-*

gramming, volume 1297 of Lecture Notes in Arti cial Intelligence. Springer-Verlag, 125-132.

[4] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Dis-covery*, 3(1):7 – 36, 1999.

[5] S. Dzeroski and N. Lavrac. *Relational data mining*. Springer, Berlin, 2001.

[6] Saso Dzeroski. Multi-relational data mining: an in-troduction. *ACM SIGKDD Explorations*, 5(1):1 – 16, 2003.

[7] Pedro Domingos Geoff Hulten and Yeuhi Abe. Min-ing massive relational databases. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 53–60, 2003.

[8] L. Getoor. Multi-relational data mining using prob-abilistic relational models: research summary. In Proceedings of the First Workshop in Multi- *relational Data Mining*, 2001.

[9] G. Dong K. Wang J. Han, J. Pei. Ef cient computation of iceberg cubes with complex measures. In *Proceed-ings of SIGMOD*, pages 1–12, 2001.

[10] V. C. Jensen and N. Soparkar. Frequent itemset count-ing across multiple tables. In *Proceedings of Paci c-Asia Conference on Knowledge Discovey and Data Mining (PAKDD)*, pages 49–61, 2000.

[11] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.

[12] . Lavrac and S. Dzeroski. *Inductive Logic Program-ming: Techniques and Applications*. Ellis Horwood, New York, 1994.

[13] Eric Ka Ka Ng, Ada Wai-Chee Fu, and Ke Wang. Mining association rules from stars. In *Proceddings of IEEE International Conference on Data Mining (ICDM)*, pages 322–329, 2002.

[14] S. Nijssen and J. Kok. Faster association rules for mul-tiple relations. In *Proc. of International Joint Con-ferences on Arti cial Intelligence (IJCAI)*, pages 891–896, 2001.

[15] S. Nijssen and J. Kok. Ef cient frequent query dis-covery in farmer. In *Proceedings of Paci c-Asia Conference on Knowledge Discovey and Data Mining (PAKDD)*, pages 350–362, 2003.

[16] L. De Raedt. Attribute-value learning versus induc-tive logic programming: the missing links (extended abstract). In *Proceedings of the 8th International Con-ference on Inductive Logic Programming*, 1998.

[17] L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. Discovery of relational association rules. In *Dzeroski, S. and Lavrac, N. editors. Relational Data Mining*. Springer-Verlag, 2001.

[18] L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. Three companions for data mining in rst order logic. In *Dzeroski, S. and Lavrac, N. editors. Rela-tional Data Mining*. Springer-Verlag, 2001.

[19] J. D. Ullman and J. Widom. *A First Course in Database Systems, 2nd ed*. Prentice Hall, New Jersey, 2002.

[20] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *Proc. 2003 Int. Conf. on Very Large Data Bases (VLDB'03)*, pages 476–487, 2003.

[21] P.M. Deshpande Y. Zhao and J. F. Naughton. An array-based algorithm for simultaneous multidimen-sional aggregates. In *Proceedings of SIGMOD*, pages 159–170, 1997.

[22] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, 2002.

