To the reader: I have added blank space to separate the section for each article so that it's easier to read, but it put the document over the page limit. I'm leaving them in so that it's easier to read.

1. Alexander Hinneburg, Dirk Habich, and Wolfgang Lehner. Combi-Operator – Database Support for Data Mining Applications. Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

The Problem

The problem that the authors address is that comparing all the possible groups of small numbers of attributes with each other from a database that has a large total number of attributes results in a very large number of possible combinations. This creates a large amount of data to deal with, which can be problematic depending on the limitations of the system being used (main memory can only hold a portion of the total data that represents all the combinations of attribute comparisons).

Limitations

The main disadvantages of previous approaches stated by the authors are:

i) The naïve approach of a single SQL statement for each required histogram would require reading the same data multiple times from the secondary storage, which is very slow.

ii) The GROUPING SET operator can be used to request all the histograms with a single SQL statement, but because the list of all the attributes is so long, it exceeds the capabilities of parsers.

Proposed Strategy

The authors' idea is to perform all of the data-intensive tasks directly within the database and then transfer the aggregated data. This will reduce the amount of data being transferred.

Their strategy is to propose a new "COMBI" ("grouping combination") operator, that fits into the OLAP "GROUP BY" extensions that already exist. This operator avoids having to list out all of the attribute combinations that will be used in the aggregation of low-dimensional projections. This means that the size of the query itself is smaller, which helps in dealing with the limitations of the parser.

The first algorithm ("main memory implementation") focuses on minimizing the number of scans of the database. It does this by selecting multiple different subsets of the of the grouping combinations of attributes that are being compared in histograms. These subsets are determined by the size restriction of main memory. They are partitioning the grouping combinations into sets that use the maximum

amount of main memory available on each pass as the database is scanned multiple times (Section 4.2). They have created both an array-based and hash map-based implementation.

The second algorithm proposed by the authors is a sort-based implementation. The main benefit of this approach is that it does not require as large of a main memory capacity compared to the "main memory implementation". It also gives output as soon as possible, which is helpful for pipelined processes.

In the first phase of the algorithm, all of the grouping combinations are considered for each tuple in the raw data; if memory is already set aside for the combination, the value of its' histogram is updated. Otherwise, memory is allocated for the new information.

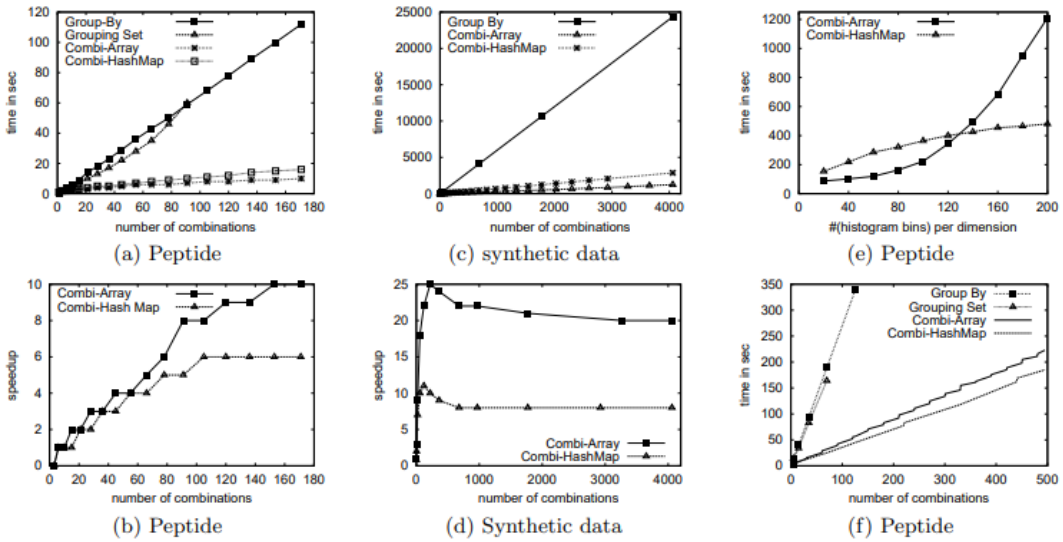
In the second phase of the sorting implementation, all the combinations in main memory are checked to see if they can be outputted, i.e. written to secondary storage. This is useful for eliminating combinations that are held in main memory earlier in the process, in order to free more space for other combinations. It is noteworthy that the combinations involving invariant grouping columns (attributes) can be immediately outputted.

Methodology and Results

The authors tested the generation of histograms of combinations of attributes from their test data in two experiments. The first was designed to test the scalability of the new operator, and the second was used to test the speedup on the large dataset(Figure 4 below, "(c)" and "(d)". Their results showed that the array-based "main memory implementation" was superior to the hash map-based implementation, followed by the GROUPING SET and then naïve implementations.

The authors implemented a prototype of their COMBI operator in C++ on a DB2 V8.1 database system. They used a Linux Pentium with 512MB of RAM.

They used synthetic test data with 100 attributes with 300,000 tuples and data from a simulation of a biological molecule which had 19 attributes and 100,000 tuples.



(a) Peptide (c) synthetic data (e) Peptide

(b) Peptide (d) Synthetic data (f) Peptide

Figure 4: Experimental comparision of the array and hash map based implementations for the COMBI operator with GROUPING SETS and GROUP BY.

2. Carlos Ordonez. Horizontal Aggregations for Building Tabular Data Sets. DMKD'04 June 13, Paris, France, 2004.

The Problem

The problem that the author addresses is that the accepted practice of normalizing tables in a database causes data mining to require significantly more effort because the data must be joined (gathered from different tables) and aggregated (summarized) to make it useful for use in data mining tasks. In addition, he also makes note of the problem of non-transposed data, which is solved by many OLAP tools but might be better addressed by SQL.

Limitations

There are several areas in which improvements to data mining tasks using SQL have been studied. The other strategies are not all as simple and widely applicable as the horizontal aggregation the author proposes (Section 5). To the author's knowledge, the particular idea of horizontal aggregation he is presenting has not been studied before, so it is a novel approach.

Proposed Strategy

First, the author proposes an aggregation function that extends SQL which can be used to produce "horizontal" results, grouped by a specified attribute. In practice, this results in output that has the aggregate values in columns instead of rows and the rows contain tuples using the "grouping attribute" as the primary key. It is easier to understand what is intended by viewing the example output:

store	salesAmt							countTransactionId,dayOfWeekNo							salesAmt			total
Id	Mon	Tue	Wed	Thu	Fri	Sat	Sun	1	2	3	4	5	6	7	dairy	meat	drinks	sales
1	500	200	120	140	90	230	160	20	2	15	50	50	60	30	700	260	480	1440
2	200	100	400	100	900	100	200	8	9	5	10	40	20	40	300	500	1200	2000
3	100	100	100	200	200	200	200	5	6	4	13	44	16	50	350	350	400	1100
4	200	300	200	300	200	300	200	24	21	24	23	29	26	20	700	700	300	1700

Table 1: A tabular data set, suitable for data mining, obtained from table *transactionLine*

The example of the syntax used to produce the table:

```
SELECT
  storeId,
  sum(salesAmt BY dayofweekName),
  count(distinct transactionid BY dayofweekNo),
  sum(salesAmt BY deptIdName),
  sum(salesAmt)
FROM transactionLine
    ,DimDayOfWeek,DimDepartment,DimMonth
WHERE transactionLine.dayOfWeekNo


      =DimDayOfWeek.dayOfWeekNo
      AND
      transactionLine.deptId
      =DimDepartment.deptId
  GROUP BY storeId;
```

Similarly, the author proposes a binary attribute data preparation, i.e. 1 for the presence of a property, and 0 for the absence:

Employee Id	Gender&Marital				Salary
	M&Single	M&Married	F&Single	F&married	
1	1	0	0	0	30k
2	0	0	1	0	50k
3	0	0	0	1	40k
4	1	0	0	0	45k

Table 2: Binary codes for gender/maritalStatus from table *employee*

The example syntax:

```
SELECT
    employeeId,
    sum(1 BY gender,maritalStatus DEFAULT 0),
    sum(salary)
FROM employee
GROUP BY 1;
```

Secondly, the author proposes two strategies for evaluating the “horizontal aggregations”. The first strategy, relying only on relational operations (SPJ strategy), and essentially creates the usual “vertical” table for each column in the desired results (i.e. each aggregate property such as sum by day of the week in the examples above). The tables are then joined to form the resulting output table. The second strategy, relies on the SQL case construct (CASE strategy) and returns binary results, as in the second

example output above. Both had sub-strategies of direct aggregation to a table and indirect aggregation using a temporary table.

Methodology and Results

The author's results indicate that the SPJ strategy is slower in all cases than the CASE strategy, and that the sub-strategy of using a temporary table was, counter-intuitively, not as efficient when tested on the real data set. For large numbers of rows, the indirect sub-strategy of the CASE strategy was still the best, but for small/medium numbers of trows, the direct sub-strategy gave better results.

The author conducted experiments on an NCR computer running the Teradata DBMS V2R5 software. The hardware had one 800MHz CPU with 256MB of RAM and 1TB of hard drive space. The SQL code generator was written in Java and used the JDBC to connect to the server.

Queries with one horizontal aggregation were conducted, and each experiment was repeated five times, to give an average time.

The author used a real dataset from the US Census which had 68 columns and 200,000 rows, and a synthetic data set intended to represent typical values from a data warehouse. The synthetic data had a uniform distribution of values for each attribute (dimension). The resulting table had 10 attributes (columns) and 1,000,000 rows.

3. Carson Kai-Sang Leung, Dale A. Brajczuk. Mining Uncertain Data for Constrained Frequent Sets. IDEAS '09, September 16–18. Cetraro, Calabria, Italy, 2009.

The Problem

There is a demand in data mining on probabilistic databases of uncertain data for frequent set mining that only searches for frequent sets that the user specifies (using constraints).

Limitations

Most previous studies find all of the frequent sets from transaction databases of precise data, not just the desired frequent sets, and not from uncertain data. This is a problem because the actual desired frequent itemsets might be a small fraction of all the frequent itemsets. Also, the actual data may not be in precise, certain terms. An example of uncertain data would be whether a patient has a certain illness, like swine flu (or COVID), or not (page 110). There exist algorithms that have constraints but use precise data (CAP, DCF, FIC) and algorithms that use uncertain data (UF-growth), but not both.

Proposed Strategy

The authors develop an uncertain constrained frequent set ("uCFS") system which reduces the cost of data mining using the properties of succinct constraints. Their uCFS system is a tree-based system which performs constrained frequent set mining on uncertain data. Their system avoids candidate generation by bringing the user's constraints into the data mining process (their "Enum*" process, as opposed to computing all frequent itemsets in their "Enum++" and "Direct*" processes), and by doing this their system avoids unnecessary computation.

Methodology and Results

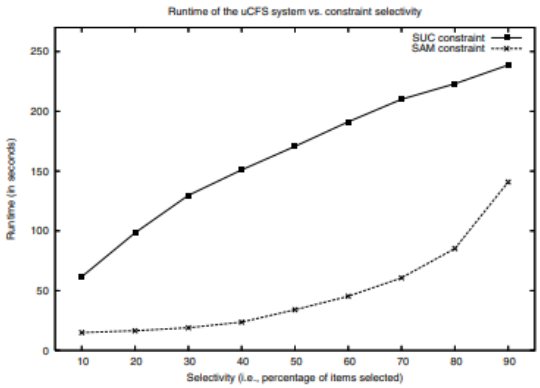
Their results show that the system was satisfactory for succinct anti-monotone constraints and succinct non-anti-monotone constraints. Additionally, constraints that are not succinct can be handled effectively because they can be converted to succinct constraints. Because the constraints are used within the data mining process itself, the cost of the data mining process is proportional to the selectivity of the constraints.

The authors used several different databases to test their system, and use a database generated by the IBM Almaden Research Center in their discussion, which has 1M records, 1000 possible items, and an

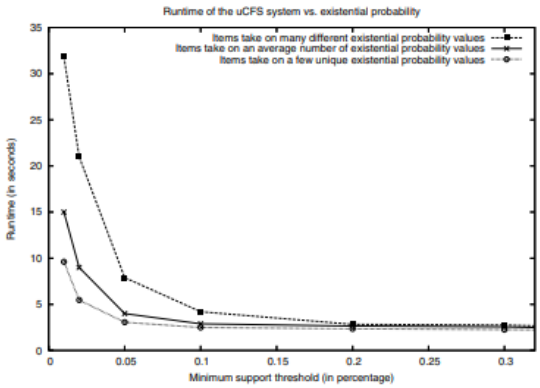
average size of 10 items per transaction. A minimum support value of 0.01% was used for the data mining algorithm (i.e. 10,000). The experiments were run on a time-shared 700MHz computer.

There were six sets of experiments, testing the effect of the selectivity of the constraint, the efficiency compared to other algorithms, the scalability (increased number of transactions—i.e. tuples—used), and the effect of changing the minimum support value. The results confirmed that the uCFS system provides significant benefits over mining all frequent itemsets, and that it returned the same itemsets as the other constrained algorithms.

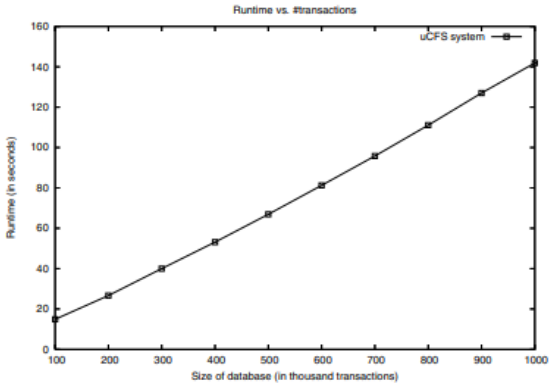
Results graphs are shown below:



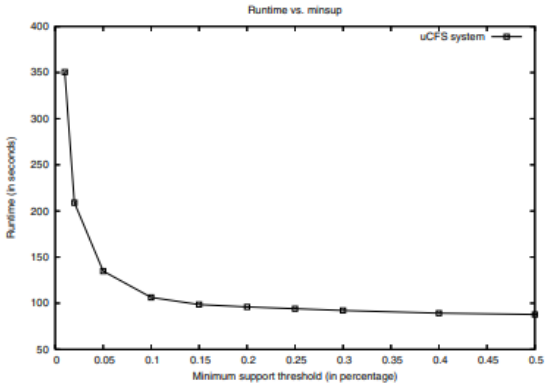
(a) Runtime vs. constraint selectivity



(b) Runtime vs. existential probability



(c) Runtime vs. TDB size



(d) Runtime vs. *minsup*

Figure 3: Experimental results: runtime of our proposed uCFS system.

4. Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, Vladislav Shkapenyuk. Mining Database Structure; Or, How to Build a Data Quality Browser. AT&T Labs-Research, ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA, 2002.

The Problem

The problem that the authors are addressing is that the databases used in data mining are frequently different enough that they must be adjusted (cleaned) in order to create a data set for data mining, and in order to do this, the structure of the databases must be understood (schema mapping). Because the databases used can be very large and complex, this presents a major obstacle to creating the data set and conducting data mining operations.

Limitations

Previous work has studied the problem of data cleaning, such as finding duplicate values in a table and assessing potential joins based on string distance. In general, previous work has focused on identifying specific relationships in tables, or between particular fields, whereas the authors wish to identify all of the relationships between fields and the structure of the whole database in question.

Proposed Strategy

The authors have developed a data quality browser, which they call “Bellman”. Besides typical tools, their browser provides tools/services focused on making it easier to understand the structure of a database being examined. They use database profiling to summarize the tablespaces, tables, and fields of the database. Bellman presents information by allowing the user to make queries of the database summaries it has created. Because it is using summaries, the results are returned quickly (seconds). The more sophisticated profiles collected by their browser are the subject of the paper.

Their contribution (page 241) in this paper is (i) methods for finding related fields using small summaries of the database, (ii) checking the effect of the size of the small summaries necessary, and (iii) new algorithms for determining the database structure.

Methodology and Results

The authors have measures of resemblance between two sets or multisets, using signature and multisignature values (page 241, 243). They also use substring resemblance (“q-grams”) to evaluate the

similarity of field names, using a signature value as well as a "sketch" (a "reduced representation" of the string)

The authors use the signatures and sketches gained from the database to execute queries about the database structure more quickly. Specifically, they use the examples of finding join paths, composite fields (the same data in multiple fields, but modified, page 245), and hetereogeneous tables (tables after multiple modifications).

The authors conducted experiments using signatures and fields, to test if they could be used to find similar fields, and also the size of the signatures/sketches required to find the similar fields.

The data set they used was a snapshot of a large data networking service's database, which was confidential data, preventing discussion of specific queries. It contained 926 tables, 15,442 fields, and had 4 tablespaces (they used the three smallest tablespaces, which had 267 tables and 3356 fields). Signature and sketch profiles were created for all of the fields that had at least 20 distinct values (1,078 fields), and were created using 250 samples.

The authors conducted several quantitative experiments: estimating the size of field intersections between two fields using resemblance, estimating the size of joins of two multisets using multiset signatures, q-gram signature similarity versus the actual similarity of the strings, and q-gram sketch accuracy using distance (string metric). The results of these were positive: the signatures, sketches, and distance metrics provided good estimates overall (page 246-9), which means that the small summaries are sufficient for creating models of the structure of a large, complex database.

The authors also conducted some qualitative experiments, essentially reporting from the experience of using the data quality browser that they have built. They comment on using multiset resemblance—joinable fields rarely had the same name, but examining their properties was more helpful—and q-gram similarity—examining queries to find results by observation (e.g. noticing city names in the fields). They conclude that the methods used in their browser are a useful addition when trying to understand the structure of a database (page 250).

5. Xuequn Shang, Kai Uwe Sattler. Depth-First Frequent Itemset Mining in Relational Databases. AC'05, March 13-17, Santa Fe, New Mexico, USA, 2005.

The Problem

The problem that the authors address is that the performance of data mining algorithms implemented using SQL is usually inferior to algorithms that process the data outside of the database system. They state that the reason for this is that the non-SQL algorithms minimize the number of scans of the database, whereas the SQL-based algorithms require more scans of the database, or require far more joins, which are also more complex.

Limitations

The authors point out that other SQL-based approaches (SETM) have not performed well on large data sets (page 1117). Also, other algorithms are based on an Apriori-like approach, which the authors are attempting to improve on because it generates large candidate sets, which increase computation cost.

Proposed Strategy

The authors propose an algorithm in SQL for mining frequent itemsets, which they call "Projection Pattern Discovery" ("Propad"). Their algorithm avoids making multiple passes over the original data that is given as input, and avoids complex joins between tables. They use a divide-and-conquer strategy, projecting the transaction table into several tables related to frequent items.

The general method that they propose is to examine frequent 1-itemsets: scan for other items that are frequent with the 1-itemset, and build a new table of frequent 2-itemsets. This is repeated to build larger frequent itemset tables, and the overall operation is carried out on each frequent 1-itemset. So, the "Depth-First" description is appropriate for the algorithm, because the algorithm is recursive, and forms the largest possible table for each item before moving on. They choose the depth-first approach rather than breadth-first because it avoids added costs of creating and dropping tables (page 1115).

They use a tree representation of the items and tables, and it conveys the background idea reasonably well:

TID	$Transaction$	$FrequentItems$
1	$1, 3, 4, 6, 7, 9, 13, 16$	$1, 3, 6, 13, 16$
2	$1, 2, 3, 6, 12, 13, 15$	$1, 2, 3, 6, 13$
3	$2, 6, 8, 10, 15$	$2, 6$
4	$2, 3, 11, 16, 19$	$2, 3, 16$
5	$1, 3, 5, 6, 12, 13, 16$	$1, 3, 6, 13, 16$

Table 1: A transaction database and $\xi = 3$

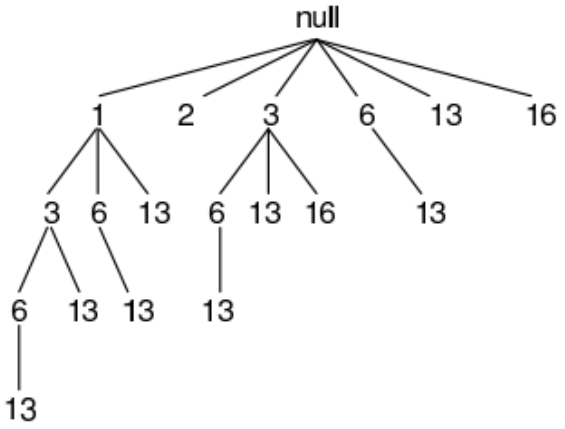


Figure 3: A frequent item set tree

set to 3. Figure 3 represents the frequent item set tree for given example. The path (null)-(1)-(3)-(6)-(13) in Figure 3 represents the item sets 1,3,6,13. The mining process can be regarded as a process of frequent item set tree construction. We can observe that:

- X and Y are frequent and X is an ancestor of Y, then all patterns between X and Y are frequent.
- To find a child pattern X_y of X, only frequent items that co-occur with X is needed.

Methodology and Results

The authors conducted many experiments on many different kinds of data sets. They compared their Propad and optimized Propad algorithms with K-Way join, an SQL-based algorithm using FP-growth, and a hybrid Propad/K-Way join algorithm. They used synthetic transaction data generated by a program described in another paper (page 1116). They used the sparsest of their four generated data sets. The database software used was DBMS IBM DB2 EEE V8, and the metric collected was the execution time of each algorithm on the data sets, with different support thresholds.

The authors observed that their Propad algorithm performs better than K-Way join (based on an Apriori-like algorithm) on all the data sets. They also observed that SQL-based algorithms (Propad, Propad optimized, SQL-based FP-growth) had better performance than the K-Way join algorithm on large data sets, or when the required minimum support was low (This causes tables of candidate frequent itemsets to increase in size, increasing the cost of joining candidate itemset tables with transaction tables).

The main reason for the superior performance of Propad over SQL-based FP-growth was that the Propad algorithm does not create the same tables, and avoids the most expensive part of the other algorithm (page 1116). Propad optimized was inferior to the regular Propad algorithm when determining whether all frequent items occurred in the same transaction, and the hybrid Propad/K-way join was efficient on sparse data sets when k was selected properly ("level 2" in these experiments).

The results graphs and nomenclature explanation are below:

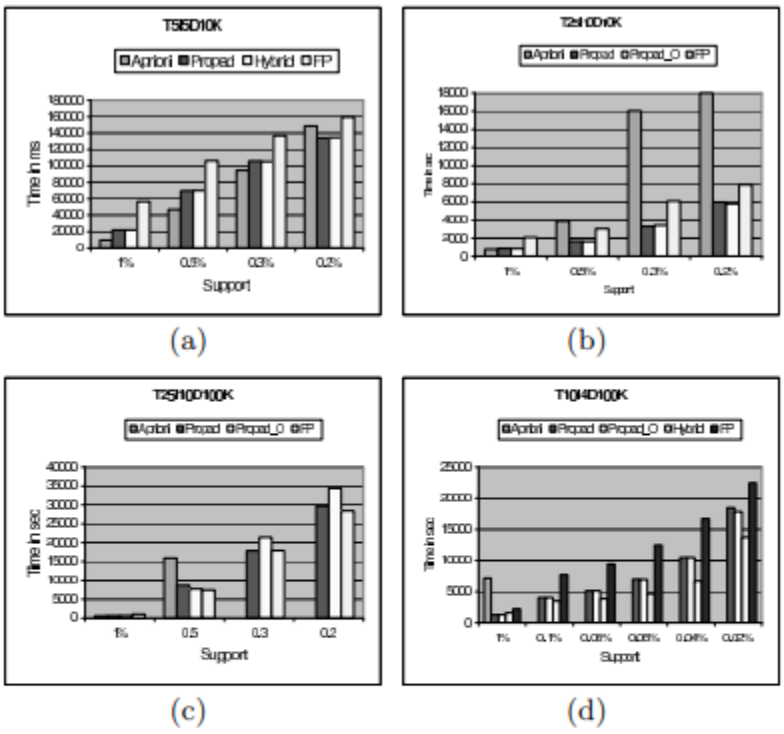


Figure 7: Comparison of five approaches. In (b), for K-Way join approach with the support value of 0.2% , in (c), with the support value of 0.3% and 0.2%, and in (d), with the support threshold that are lesser than 0.1%, the running times were so large that we had to abort the runs in many cases.

The nomenclature of these data sets is of the form TxxIyy-DzzzK, Where xx denotes the average number of items presen per transaction, yy denotes the average support of each item in the data set and zzzK denotes the total number of transactions in K (1000's). We report experimental results on four data sets, they are respectively T5I5D10K, T25I10D10K, T25I20D100K that are relatively dense, T10I4D100K that is very sparse. (Here we have chosen the dataset T10I4D10K, because for this dataset, the experiment runs for 9 passes and we want to see how these approaches perform when mining long pattern.)

6. Zhen Guo, Zhongfei (Mark) Zhang, Eric P. Xing, Christos Faloutsos. Enhanced Max Margin Learning on Multimodal Data Mining in a Multimedia Database. DD'07, August 12–15, San Jose, California, USA, 2007.

The Problem

The problem that the authors try to address in this paper is learning the relationships between different modalities in multimodal data in order to exploit the synergy in the data set. Stated more simply: the relationship between different data sources that are of different types (for example, images and lists of descriptor words, as used in the paper) must be learned by the (machine learning) model before those relationships can be used to make predictions or ask further questions about the data.

Limitations

The limitations of other research include the potentially exponential number of structures in structured output variables (page 341), the limited reduction in the number of constraints when using a dual problem approach, and having to compute the most violated constraint for cutting plane algorithms that aim to find a small set of active constraints.

Proposed Strategy

The authors have created a new enhanced max margin learning framework (EMML), which attempts to find a small set of active constraints, which makes it more efficient than existing max margin learning implementations in other studies (page 347). They sidestep the problem of calculating the most violated constraint by arbitrarily selecting a constraint that violates the optimality condition of the optimization problem (page 341).

Results and Methodology

The authors confirmed that their EMML implementation has a much faster convergence rate than max margin learning implementations from other sources (page 347). Also, the authors' multimodal data mining solution based on EMML was highly scalable because the query response time was independent of the size of the database. This was confirmed by experimental results as well as complexity analysis.

The authors used the Berkeley Drosophila embryo image database as their data set. The image database contained embryo images (of fruit flies) arranged into 6 folders, each containing images that

represent 2 to 4 of the 16 stages of embryonic development: there were a total of 36628 images, and 227 words in the six folders.

They evaluated retrieval-based and across-stage inferencing-based multimodal datamining. The fifth folder was used for the retrieval-based data mining (Stage 11 and 12, about 5,500 images with 64 words used to annotate the images). Two-thirds of the images were used for training, and the other third was used for evaluating the result (i.e. the algorithm produced by the machine learning algorithm after training). Similarly, the across-stage inferencing-based datamining was evaluated using multiple folders of images with annotation words (third, fourth, and fifth folders).

They also compared their EMML implementation with another multimodal data mining method ("MBRM", page 346). The result was that in two scenarios their EMML implementation significantly outperformed the MBRM implementation, and only slightly in one other scenario, but all the other scenarios were comparable (page 346).

To test scalability, they used image annotation and randomly selected three subsets of the image database (50, 100, 150 images respectively) and measured the query response time of both methods (average over 1648 queries). They had to adjust the results because EMML was implemented in MATLAB, whereas MBRM was implemented in C (Linux), so they have reported the results as a difference from the baseline established by the query response time using the 50 image set. The results showed that MBRM had a linear response time, whereas EMML had a constant response time (page 347).

In comparison with another implementation of a max margin learning method, "TCKG", the EMML method took 10 minutes to perform the learning task that took the TCKG method 14 hours (page 347). This seems to be due to the decreased number of constraints used by the EMML method: the increase in speed and the decrease in constraints were both improved by a factor of 70.