UNBOXING DATA MINING VIA DECOMPOSITION IN OPERATORS
Towards Macro Optimization and Distribution

Alexander Wöhrer[1], Yan Zhang[1,2], Ehtesam-ul-Haq Dar[1] and Peter Brezany[1]

[1]*Institute of Scientific Computing, Faculty of Computer Science, University of Vienna*
Nordbergstrasse 15/C/3, 1090 Vienna, Austria

[2]*School of Computer and Information Technology, Beijing Jiaotong University*
Shangyuan Residence 3, Haidian District, Beijing 100044, China

Keywords: Data mining operators, Macro optimization, Distributed data mining.

Abstract: Data mining deals with finding hidden knowledge patterns in often huge data sets. The work presented in this paper elaborates on defining data mining tasks in terms of fine-grained composable operators instead of coarse-grained black box algorithms. Data mining tasks in the knowledge discovery process typically need one relational table as input and data preprocessing and integration beforehand. The possible combination of different kind of operators (relational, data mining and data preprocessing operators) represents a novel holistic view on the knowledge discovery process. Initially, as described in this paper, for the low-level execution phase but yielding the potential for rich optimization similar to relational query optimization. We argue that such macro-optimization embracing the overall KDD process leads to improved performance instead of focusing on just a small part of it via micro-optimization.

1 INTRODUCTION

The rapidly growing wealth, complexity and diversity of data open many new opportunities in business, research, design, policy formulation and decision making. These opportunities will not be explored unless we advance the state of the art in data integration and analysis. The European project ADMIRE (Advanced Data Mining and Integration Research for Europe) (Atkinson et al., 2008) is pioneering architectures and models that will deliver a coherent, extensible and flexible framework for data mining and integration to make the best use of a wide range of distributed data resources. Research in the area of relational query processing pioneered the definition of re-usable standard sub-components dividing the overall task into smaller pieces, so called relational operators. First into logical ones and later into concrete physical ones, allowing advanced query optimization (Ioannidis, 1996) and well elaborated centralized, distributed and parallel query execution architectures (Kossmann, 2000). Many data mining algorithms have similar behavior during, at least, an important part of their execution too. This led us to consider to apply a similar operator-based approach to decompose KDD processes in order to achieve re-usability, flexibility and extensibility.

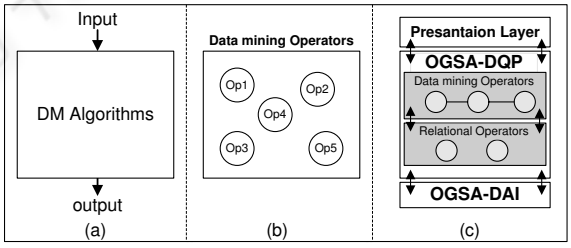

Figure 1: (a) Traditional black box DM algorithm implementation. (b) Composable operators. (c) Implementation of a KDD process in OGSA-DQP.

This requires a transition from traditional black box data mining algorithm implementation towards re-usable operator based implementations as shown in Figure 1. The current state-of-the art approach is presented in Figure 1 (a) where the functionality and intermediate steps of the used data mining algorithm is hidden. The open source data mining toolkit WEKA (Witten et al., 1999) is a well known repre-

Wöehrer A., Zhang Y., Dar E. and Brezany P. (2009).
UNBOXING DATA MINING VIA DECOMPOSITION IN OPERATORS - Towards Macro Optimization and Distribution.
In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval*, pages 243-248
DOI: 10.5220/0002333102430248

sentative following this approach. Figure 1 (b) pictures an arbitrary set of re-usable data mining operators needed to realize the functionality of one specific data mining algorithm. Figure 1 (c) presents a high-level view of how we implemented the required set of operators for a KDD process, in our case requiring sequential pattern mining, by extending the service-oriented distributed query processing engine OGSA-DQP (Alpdemir et al., 2004) with fine-grained data mining operators. As data mining tasks in the knowledge discovery process typically need one relational table as input and data preprocessing and integration beforehand, the possible combination of different kind of operators (relational, data mining and data preprocessing operators) represents a novel way of tight integration of data access, data integration and data mining.

The rest of the paper is structured as follows. Section 2 introduces needed background while Section 5 discusses related work. Section 3 elaborates on data mining operators for sequential patterns and describes their proof of concept implementation via the OGSA-DQP framework. Section 4 discusses macro optimization of KDD process enabled by an operator based view on it. Section 6 concludes the paper and outlines future work.

2 BACKGROUND

Sequential Patterns were first introduced in (Agrawal and Srikant, 1995) and deals with the extraction of knowledge which consists of frequently occurring events or elements. Let us suppose that T = $\{I_1,I_2,I_3,..I_n\}$ be the non empty set of items or events. Then a sequence is an ordered list of items or events which can be represented as S= $\langle s_1,s_2,s_n\rangle$,where s_i is an itemset. The order of itemsets depends upon time or date. To extract the frequent sequences different constrains are being defined by the user. *Sequential pattern* is a frequent pattern which satisfies the minimum threshold frequency called *support*, defined as

$$support(S) = \frac{\text{Total occurrence of Sequence S in DB}}{\text{Total number of transactions in DB}}$$

A sequence s is a *maximal sequence* if it is not contain in any other sequence. To get the maximal frequent sequence the non-maximal sequences have to be eliminated.

Relational Operators are the building blocks of a query tree representing some SQL statement to retrieve data. To perform its tasks operators typically follow the iterator model (Graefe, 1993) which consists of *open*, *fetch-next*, and *close* methods. The open-function allocates memory for an operator's input(s) and output while fetch-next retrieves all the tuples iteratively from its child operators and process them according to the task of the operator. When all input tuples have been processed and related output was generated the close-function is called to deallocate the resources which were allocated during processing. The operators can be divided into two categories, blocking and non-blocking query operators.

- *blocking*: An operator which produces output only when it goes through all the input tuples first. An example for a blocking relational operator is the *group-by* operator.
- *non-blocking*: Such operators produce the output without reading all the available input tuples. An example for a non-blocking relational operator is the *select* operator.

OGSA-DQP (Alpdemir et al., 2004) is a service based framework capable of performing distributed query processing over OGSA-DAI (Antonioletti et al., 2005) data services and other Web services on the Grid. OGSA-DAI is a de-facto standard middleware to access databases on the Grid. The OGSA-DQP coordinator service produces a query plan from the user query. This query is translated into different query partitions which are being executed by the evaluator services.

The process of **relational query optimization** and its components is pictured in Figure 2. The sin-

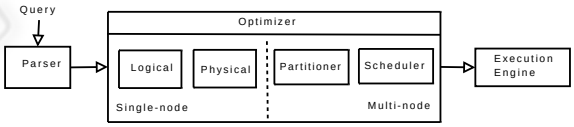

Figure 2: Components of relational query optimization and processing.

gle node optimizer produces a query plan as it would run on one processor. The parser transforms the query into an internal logical representation which gets optimized by the logical optimizer. The physical optimizer transforms the logical expression into executable physical query plans by selecting algorithms that implement each of the operations in the logical plan, e.g. hash join for a join operation. If multiple processors are available, the multi node optimizer divides the query plan into several partitions (by the partitioner) which are allocated to machine resources (by the scheduler). Optimizations are typically based on cardinality estimation, data distribution and knowledge about the runtime behavior and requirements of the operators (Ioannidis, 1996).

3 OPERATORS

The sequential pattern mining process is divided into six logical operators. One of these six operators is a common relational operator and five are specific data mining operators to find sequential patterns. The following sections elaborates on their definitions, relations and implementation.

3.1 Definition

SeqTableScan η Operator. This common relational operator retrieves all the tuples from a data source. The definition of this operator is shown below:

$$T1 = \eta_{(id,date,items)}(table)$$

The operator takes as argument the input table and target columns in it. The output of this operator consists of a table with columns *id*, *date* and *items*. The operator[1] sorts the input tuples according to *id* and *date* respectively in ascending order.

Conversion ρ Operator. This operator is used to create sequence per *id*. It receives its input tuples from the *SeqTableScan* operator η and then clusters the *items* having the same *id* and order them according to *date*. The operator has the following definition:

$$T2 = \rho_{id,sequence(items\_(1,....,n))}(T1)$$

As shown by the expression this operator takes $T1$ as an input argument and the output consists of tuples with *id* and *sequence*.

PowerSet σ Operator. The third operator receives it input tuples from the *Conversion* operator ρ. The operator works according to the mathematical principle of Power Sets. This operator converts every sequence of a *id* into its corresponding power sets. The operator definition is shown below:

$$T3 = \sigma_{id,ps\_sequence \leftarrow ps(s_1,s_2,....,s_n)}(T2)$$

The output $T3$ consists of *id* and *ps_sequence*. Here *ps_sequence* will be achieved by applying the *ps* operation to every sequence s_i available in the 'sequence' column of the input tuples from $T2$.

LargeItemSet γ Operator. The fourth operator recieves its input tuples from the *PowerSet* operator σ and then searches the large itemsets within them. The large itemsets are those itemsets which satisfy the minimum support threshold, another input argument provided by the user. It compares each of sequences related to one *id* with other sequences and this comparison will give the total number of occurrences of a sequence of specific *id*. If that count complies with the user provided support then that sequence will qualify for the large itemset while the sequences which do not fulfill the minimum support threshold will be eliminated. The operator is defined as follows:

$$T4 = \gamma_{sequence,support_{\geq support_{user}}}(T3)$$

Its output tuples consist of *sequence* and *support*.

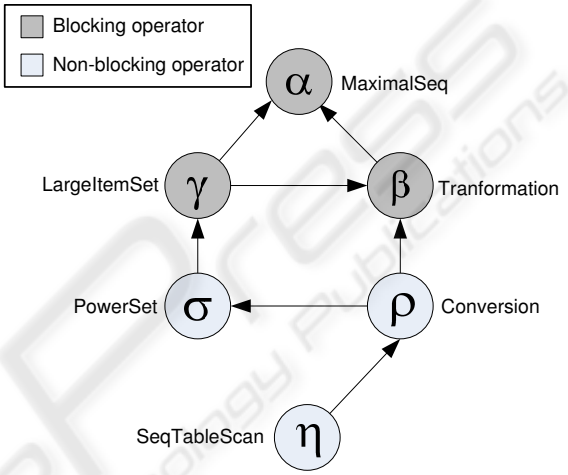

Figure 3: Operator hierarchy.

Transformation β Operator. This operator retrieves its input tuples from the *LargeItemSet* and *Conversion* operator. The definition of this operator is given below:

$$T5 = \beta_{id,transformed}(T4, T2)$$

With the help of these two inputs this operator transforms the *Customer Sequence* to corresponding *Transformed Sequences*. For this process two steps are required:

1. Take the customer sequences received from *Conversion* operator and replace them with the corresponding power sets.
2. Using data retrieved from *LargeItemSet* eliminates those sequences which do not satisfy the minimum support threshold.

The output tuples of this operator consist of *id* and *transformed* fields.

Maximal Sequence α Operator. This operator requires tuples from two input operators, namely *Tranformation* and *LargeItemSet* and finds the maximal sequences. The operator is defined as follows:

$$T6 = \alpha_{max\_seq}(T4, T5)$$

[1] combines several needed relational operators into one. This is done due the possibility to push-down the task towards the RDBMS via an appropriate SQL statement like 'select ... from ... order by'

The output of this operator are the *maximal sequential patterns*. The description of the operators, their relation and semantics result in the operator tree shown in Figure 3.

3.2 Implementation

The OGSA-DQP evaluator - our implementation environment - follows the iterator model (Graefe, 1993). To implement operators in this model each operator has to implement three abstract methods. The code example given in Listing 1 illustrates how the operator tree shown in Figure 3 is created inside OGSA-DQP.

Listing 1: Operator usage for sequential pattern mining.

```
SeqTableScan Op1=SeqTableScan("select...from...")
Conversion   Op2=Conversion(Op1,...)
PowerSet     Op3=PowerSet(Op2,...)
LargeItemset Op4=LargeItemset(Op3,support,...)
Transform    Op5=Transform(Op2,Op4,...)
MaximalSeq   Op6=MaximalSeq(Op4,Op5,support,..)
Op6.open();
do{
    tuple = Op6.next()
    if(valid tuple) {
      store or display retrieved tuple
    }
}while(!EOF)
Op6.close();
```

To show the feasibility of our approach initial performance results of our implementation are presented based on datasets from the UCI KDD Archive (Hettich and Bay, 1999). They consist of 41,096, 77,285, 159,128, 316,969 records respectively. These datasets are about the page visits of users who visited *msnbc.com*. The performance of each data mining operator for all different datasets is being shown in Figure 4.

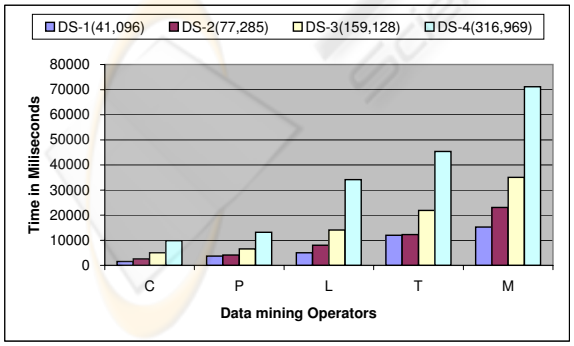

Figure 4: Operator performance with four different datasets.

4 MACRO OPTIMIZATION

An operator based logical description of a KDD process will allow optimizations considering the overall process structure, as pictured in Figure 5 below. We call this envisioned process *macro optimization*, using knowledge about data statistics, the execution environment, operator characteristics and their requirements.

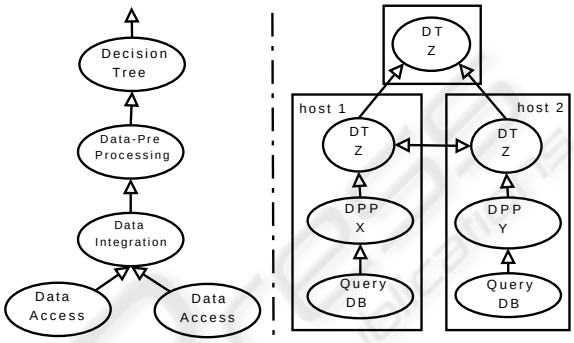

Figure 5: Logical KDD process description vs. optimized distributed physical execution plan.

In our example a decision tree on a distributed data set requiring some preprocessing has to be constructed. The logical description follows the typical subsequent phases of a KDD process. First data selection (and integration in our case), then data preparation and later on the ultimate data mining. Now we envision a similar optimization process on this integrated KDD process than for relational query optimization, pictured in Figure 2 and briefly discussed in Section 2. The logical optimizer applies similar heuristics than in relational optimization, e.g. apply data preprocessing as early as possible but at least before data mining operators instead of pushing-down conditions as far as possible the relational query tree. The physical optimizer uses data statistics, e.g. estimated cardinalities of operators, tuples size, column content, in order to choose the most appropriate implementations of the required data preprocessing and decision tree tasks. The multi-node optimizer knows the available execution environment and recognizes the initial data distribution. It replaces the intended data integration operator, e.g. UNION or JOIN, with a decision tree implementation supporting the given horizontal or vertical tuple distribution and includes required exchange (Graefe and Davison, 1993) operators. Additionally, an operator based model encourages adaptivity during execution (Gounaris et al., 2002).

5 RELATED WORK

According to (Johnson et al., 2000), research in data mining can be classified broadly into two *"directions"*. The first has focused primarily on which kinds of patterns to mine, and how fast they can be computed. The second recognizes that mining would be far more effective if not considered in isolation and has focused on how mining can interact with other "components" in a more general framework of knowledge discovery. Our research is following the second direction.

Different frameworks were proposed to support the knowledge discovery in databases (KDD) process in a uniform manner. (Geist and Sattler, 2004) proposes one based on constraint database concepts as well as interestingness values of patterns. The overall KDD process is divided into three main steps - namely pre-processing, data mining and post-processing - and for each of these different operators are being defined and their implementation issues for decision trees are being discussed. The *3W model and algebra* (Johnson et al., 2000) uses regions, dimensions and hierarchies to define a uniform framework and operators. The model consists of three worlds (intensional, extensional and data world) and operators for moving in and out of the worlds and for the intensional world are defined. (Fernandez-Baizan et al., 1998) outlines the design of a RDBMS that will provide the user with traditional query capabilities as well as KDD queries based on operators. Further research on exploiting relational and inductive database technology for data mining exists, e.g. (Botta et al., 2004). In (Meo et al., 1996) a model is defined to extract association rules via an SQL-like operator named *'MINE RULE'*. (YUAN, 2003) uses data mining operators approach to find association rules using nested relations from the database. For this purpose a new query language with the name *M2MQL* is being defined and semantics of its relevant operator's algebra is being discussed.

The following two efforts support users in defining data mining processes. The *Intelligent Discovery Assistant (IDA)* (Bernstein et al., 2005) provides users with systematic enumerations of valid data mining processes and effective rankings of these valid processes by different criteria, to facilitate the choice of processes to execute. This is done via ontological operator descriptions and heuristic ranking methods. (Abe and Yamaguchi, 2004) describes constructive meta-learning by recomposing methods into learning schemes with mining (inductive learning) method repositories that come from decomposition of popular mining algorithms. It constructs the learning scheme proper to a given data set

The work described in this paper differs from the research introduced above by the following features:
open: The concepts nor the implementation is bound to a specific RDBMS.
holistic: The possible combination of different kind of operators (relational, data mining, etc.) represents a novel way of tight integration of so far sub-sequent treated phases in the knowledge discovery process for optimization as well as execution.
distributed/parallel: Distribution of input data as well as of operators and parallel execution on different machines is anticipated and supported by our implementation environment and its underlying *iterator model* already.
granularity: Our operators are typically much more fine grained in order to be re-used for data mining algorithms sharing similar behavior.
logical vs. physical: A unified view on the KDD process with different abstraction levels, e.g. logical operator tree and its actual physical implementation, allows for various optimization mechanisms.

6 CONCLUSIONS

The opportunities of the rapidly growing wealth, complexity and diversity of data will not be explored unless we advance the state of the art in data integration and analysis. Research in the area of relational query processing pioneered the definition of reusable standard sub-components dividing the overall task into smaller pieces, so called relational operators. Many data mining algorithms have similar behavior during, at least, an important part of their execution too. This led us to consider to apply a similar operator-based approach to decompose KDD processes in order to achieve re-usability, flexibility, extensibility and optimisation.

We introduced an operator based approach to extract sequential patterns by using relational and data mining operators in combination. This represent a transition from traditional black box data mining algorithm implementation towards re-usable operator based implementations. The defined operators have been implemented inside OGSA-DQP, showing the feasibility of our approach.

Data mining tasks in the knowledge discovery process typically need one relational table as input and data preprocessing and integration beforehand. The possible combination of different kind of operators (relational, data mining and data preprocessing operators) represents a novel holistic view on the knowledge discovery process. Initially for the low-

level execution phase but yielding the potential for rich macro optimization of the overall process similar to relational query optimization (logical, physical, distributed/parallel).

Our future work plan includes the following: migrating to the latest version of OGSA-DAI allowing complex workflow graphs, applying the operator based approach to our earlier work for distributed decision trees (Hofer and Brezany, 2004), extending the set of data mining operators to cover association rule mining, and higher level optimization components for KDD processes.

ACKNOWLEDGMENTS

This work has been supported by the ADMIRE project which is financed by the European Commission via Framework Program 7 through contract no. FP7-ICT-215024. Yan Zhang has been supported by Project No. 2006AA01A121 of the National High-Tech. R&D Program of China.

REFERENCES

Abe, H. and Yamaguchi, T. (2004). Constructive meta-learning with machine learning method repositories. In *IEA/AIE*, pages 502–511.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *ICDE*, pages 3–14.

Alpdemir, N. M., Mukherjee, A., Gounaris, A., Paton, N. W., Watson, P., Fernandes, A. A., and Fitzgerald, D. J. (2004). Ogsa-dqp: A service for distributed querying on the grid. In *Advances in Database Technology - EDBT 2004*, pages 858–861.

Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., Hong, N. P. C., Collins, B., Hardman, N., Hume, A., Knox, A., Jackson, M., Magowan, J., Paton, N., Pearson, D., Sugden, T., Watson, P., and Westhead, M. (2005). The design and implementation of grid database services in ogsa-dai. *Concurrency and Computation: Practice and Experience*, 17:357–376.

Atkinson, M., Brezany, P., Corcho, O., Han, L., van Hemert, J., Hluchy, L., Hume, A., Janciak, I., Krause, A., Snelling, D., and Wöhrer, A. (2008). Admire white paper: Motivation, strategy, overview and impact. http://www.admire-project.eu/docs/ADMIRE-WhitePaper.pdf.

Bernstein, A., Provost, F., and Hill, S. (2005). Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518.

Botta, M., Boulicaut, J.-F., Masson, C., and Meo, R. (2004). Query languages supporting descriptive rule mining: A comparative study. In *Database Support for Data Mining Applications*, pages 24–51.

Fernandez-Baizan, M., Ruiz, E. M., Pena-Sanchez, J., and Pastrana, B. (1998). integrating KDD algorithms and RDBMS code. In *Proceedings of RSCTC'98*, pages 210–213.

Geist, I. and Sattler, K.-U. (2004). Towards data mining operators in database systems: Algebra and implementation. Technical Report 124, University of Magdeburg.

Gounaris, A., Paton, N. W., Fernandes, A. A. A., and Sakellariou, R. (2002). Adaptive query processing: A survey. In *BNCOD*, pages 11–25.

Graefe, G. (1993). Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170.

Graefe, G. and Davison, D. (1993). Encapsulation of parallelism and architecture-independence in extensible database query execution. *IEEE Transactions on Software Engineering*, 19(8):749–764.

Hettich, S. and Bay, S. (1999). The UCI KDD Archive.

Hofer, J. and Brezany, P. (2004). Digidt: Distributed classifier construction in the grid data mining framework gridminer-core. In *In Proceedings of the Workshop on Data Mining and the Grid (GM-Grid 2004) held in conjunction with the 4th IEEE International Conference on Data Mining (ICDM'04)*.

Ioannidis, Y. E. (1996). Query optimization. *ACM Computing Surveys*, 28(1).

Johnson, T., Lakshmanan, L., and Ng, R. (2000). The 3w model and algebra for unified data mining. In *VLDB*, pages 21–32.

Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4).

Meo, R., Psaila, G., and Ceri, S. (1996). A new sql-like operator for mining association rules. In *VLDB*, pages 122–133.

Witten, I. H., Frank, E., Trigg, L., Hall, M., Holmes, G., and Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with Java implementations. In *Proceedings of the Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems*, pages 192–196.

YUAN, X. (2003). Data mining query language design and implementation. Master's thesis, The Chinese University of Hong Kong.