KESO: Minimizing Database Interaction

Arno Siebes, Martin L. Kersten
CWI,
Kruislaan 413
1098 SJ Amsterdam The Netherlands
arno, mk@cwi.nl

Abstract

A[1] mature data mining system has to interact with standard DBMSs. A crucial factor in the performance of such a data mining system lies in this interaction. The KESO project aims at the development of such a tool and its interaction with the database is restricted to two-way table queries; a special kind of aggregate query.

This restriction gives rise to ample possibilities to optimize the computation of such two-way tables, e.g., by using parallelisation or by temporary storage of intermediate results.

However, the size of these two-way tables puts a large communication overhead on the database interaction of KESO. In this paper we propose to compute (certain) aggregates in the database. This approach lowers the size of the query results considerably while keeping the possibilities for optimization used in the current version.

Keywords: Data Mining Systems, Database Interaction, Aggregates, Data Cubes

Introduction

Data mining is a fast growing research area, c.f., (Fayyad & Uthurusamy 1994; 1995; Simoudis *et al.* 1996; Fayyad *et al.* 1996). There are algorithms for many applications. A mature data mining system should support a wide variety of such algorithms and applications. Moreover, it should work "on top" of standard (relational) DBMSs. The ESPRIT-IV project KESO aims to develop such a tool.

To achieve this goal, KESO is based upon an *inductive query language*. Central to this language is the notion of "data mining as a search task". More in particular, the "search space" is characterised by a *description language* and a *quality function* and the search process is specified by an *abstract search algorithm* and a collection of *operators*. Conceptually, posing a query in the KESO system is done by instantiating these abstract concepts with concrete implementations.

[1]Copyright ©1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Crucial for the performance of a data mining system is its interaction with the database. Due to the modular view KESO takes towards data mining algorithms, this interaction is localised in the quality function. More in particular, the interaction is restricted to *two-way table* queries, a special type of aggregate queries (see Section 2). This allows for different, orthogonal, schemes to optimise the interaction between KESO and the DBMS; i.e., to optimize the computation of the two-way tables.

The downside of the two-way tables is however that the communication overhead between KESO and the DBMS is considerable. In this paper we propose a way to reduce this overhead by computing aggregates in the database. This proposal leaves the option to use the optimization schemes mentioned above open.

The organisation of this paper is as follows. Section 2 presents a brief (and simplified) overview of the concepts underlying the KESO system; see e.g., (Siebes 1996) for more details. In Section 3, the three database-interaction optimization schemes are introduced and discussed. In Section 4 we show how to generalise the two-way tables to minimise the communication overhead without comprimising the optimisation schemes. Section 5 summarises the main results of this paper and it presents a discussion on related research.

Without loss of generality ((Abiteboul, Hull, & Vianu 1994)) assume that the database consists of one *Universal* relation DB. DB has schema $\mathcal{A} = \{A_1, \ldots, A_n\}$, with associated domains D_i. A database db is a multi-set of tuples over $\mathcal{D} = D_1 \times \cdots \times D_n$. In other words, tuples may occur zero or more times in the database.

We use $[\ldots]$ to denote multi-sets and $\{\ldots\}$ to denote sets. The projection of a tuple on the attributes X is denoted by π_X.

Inductive Querying in KESO

Data mining can be seen as the induction of a model from the database. The description language defines the set of models KESO considers during the execution of a task; each description is a possible (partial) model.

The basic description language for KESO is that of *selections*. A selection for DB is a description of the form:

$$A_1 \in V_1 \wedge \cdots \wedge A_n \in V_n$$

in which $V_i \subseteq D_i$. Such a selection description ϕ *describes* all tuples in the database that satisfy the *selection condition* ϕ. For example, $age \in [18,24] \wedge sex \in \{male\}$ describes the young adult males in a database.

To restrict the selections, *hierarchies* H_i can be defined for each attribute A_i. Without loss of generality, we can assume that such hierarchies have been defined for all attributes. Thus $A_1 \in V_1 \wedge \cdots \wedge A_n \in V_n$ is a selection description for DB iff $V_i \in H_i$.

Clearly, the selection description language depends both on the database schema DB and on the hierarchies defined. However, since the role of these two is merely symbolic in this paper, *the* description language of selection descriptions is denoted by simply Φ.

While the description language defines the models that KESO considers, the quality function determines how well a description (model) fits (part of) the database. For the purposes of this paper we can forget issues as the syntactic compexity of a model and a quality function Q determines the quality of a description $\phi \in \Phi$ based on a *two-way table* over the *support* of ϕ. The support of ϕ is simply that part of the database that is described by ϕ. In other words, the support of ϕ, denoted by $\langle\phi\rangle$ is defined by:

$$\langle\phi\rangle_{db} = [t \in db \mid \phi(t)]$$

Two way tables have sets of source and target attributes, denoted by S and T with $S,T \subseteq \mathcal{A}$. The two-way table has as schema $S \cup T \cup \{Count\}$, where $Count$ has the natural numbers as domain. The tuples in the two-way table over the support of ϕ are those tuples τ that satisfy:

1. $\exists t \in \langle\phi\rangle : (\forall A \in S \cup T : \pi_A(t) = \pi_A(\tau))$
2. $\pi_{Count}(\tau) = |[t \in \langle\phi\rangle \mid \forall A \in S \cup T : \pi_A(t) \doteq \pi_A\tau]|$

The target attributes T are parameters that are set by the quality function; in other words, T is constant during a data mining task. The source attributes S denote a context, S is variable during the execution of the task, its value is set by the search strategy. For given parameters S, T, ϕ, and db, the two-way table is denoted by $2WT(S,T,\phi,db)$

The quality function Q further computes some aggregate function F over the two-way table, which yields the final quality.

The quality function determines which descriptions are the most interesting (i.e., which models are best), finding these very interesting description is the task of the KESO system. Sometimes the search space (i.e., the description language) is small enough to allow exhaustive search. Other times (such as for association rules) there are algorithms that find the optimal solution. Most often, however, heuristic approaches have to be taken.

All these search algorithms define (partial) enumerations of the description language. More precisely, they use a *search strategy* and *operators* on the description language.

The search strategy defines the heuristic with which the search space is enumerated. Examples are *Exhaustive Search*, *Hill Climbing*, *Beam Search*, and *Genetic Algorithms*.

The operators implement the enumeration on the search space. For example, a Hill Climbing algorithm requires that the quality of the *neighbours* of the current best description ϕ is computed. One way in which the neighbours of a description could be defined is as follows. For $\phi = A_1 \in V_1 \wedge \ldots \wedge A_n \in V_n$, the neighbours of ϕ is the set of descriptions

$$\{A_1 \in W_1 \wedge \ldots \wedge A_n \in W_n \mid \exists! i \in \{1,\ldots,n\} : W_i \neq V_i\}$$

Or this set could be further restricted by requiring that this particular W_i and V_i differ one step in the hierarchy H_i.

In KESO a data mining task on a database *db* is thus specified by a four-tuple (Ψ, Q, S, O) in which Ψ is a description language, Q a quality function, S a search strategy and O the set of operators on Ψ necessary for S.

The real power of KESO stems from the fact that we can program using tasks as primitives. For example, if Δ_1 is a data mining task that discretizes a continuous domain and Δ_2 is a task that creates a classification tree, Δ_2 can use Δ_1 whenever a continuous domain is encountered. This allows Δ_2 to "discretize on the fly" and use in each branch the discretization that is best for that branch. In fact, Δ_1 can be used by any task that requires discretization. Such "programs" of tasks are the *inductive queries* of KESO. The reader is referred to (Siebes 1996) for detailed examples of such inductive queries.

Database Support for Data Mining

The interaction between the KESO system and the DBMS is restricted to the computation of the two-way tables. These tables are easily expressed in SQL by:

SELECT Source, Target,
  COUNT(Source, Target) **AS** Count
FROM *db*
WHERE ϕ
GROUP BY Source, Target

So, one way to speed-up the data mining process would be to pre-compute these tables, that is to compute a *data cube* as defined in (Gray *et al.* 1997). However, there are two major disadvantages with this approach. The first is caused by the "on the fly" discretization in KESO. Each such discretization would add another dimension to the cube, thus requiring the computation of a new cube, of which the old cube is only a sub-cube. Computing this new cube is far more expensive than computing the two-way table.

The second disadvantage is that (if there are no continuous attributes) the cube would give the two-way tables for the complete search space. Above we already mentioned that this search space is often far too large to explore completely. In other words, computing the cube is computing far too many two-way tables and would thus take far too much time.

The approach taken in KESO uses a lattice structure on the two way tables, similar to the lattice in (Harinarayan, Rajaraman, & Ullman 1996). Note that $2WT(S_1, T, \phi_1, db)$ can be computed from $2WT(S_1 \cup S_2, T, \phi_1 \vee \phi_2, db)$ as follows. Project the S_2 attributes out of $2WT(S_1 \cup S_2, T, \phi_1 \vee \phi_2, db)$ and sum the counts of those tuples that become identical. Then select that part of the result that satisfies ϕ_1. This observation induces the following order on two-way tables: $2WT(S_1, T, \phi_1, db) \prec 2WT(S_2, T, \phi_2, db)$ iff $S_1 \subseteq S_2 \wedge \phi_1 \rightarrow \phi_2$. It is obvious that the set of all two-way tables forms a lattice under this order.

This lattice structure is used in two ways in KESO. Firstly, it tells us that if the search algorithm telescopes in on the database (queries smaller and smaller subsets of the database) it is worthwhile to cache intermediate two-way tables since subsequent two-way tables can be computed from these intermediate results. Since the two-way tables are in general far smaller than the database, this speeds-up query processing notably; see (Holsheimer, Kersten, & Siebes 1996) for more details and performance figures.

Secondly, KESO sends one batch of two-way queries per "generation" of the search process to the database. These batches contain two-way queries that are closely related. Using the lattice structure, we first compute the two-way table for their smallest common ancestor and derive from that the necessary two-way tables.

The important difference between our usage of the lattice structure and that in (Harinarayan, Rajaraman, & Ullman 1996), is that the authors of (Harinarayan, Rajaraman, & Ullman 1996) use the lattice to compute which views to store once and for all while we take a dynamic approach to view materialisation. We are dynamic by necessity, since we do not know which queries will be asked. Since most (heuristic) search processes are Markov processes, the best we can do is to stay on the heels of the search process.

A similar observation is that $2WT(S, T, \phi_1 \vee \phi_2, db)$ can be computed from $2WT(S, T, \phi_1, db)$ and $2WT(S, T, \phi_2, db)$ by merging the two tables and sum the counts of identical tuples.

This is important for KESO to run on top of a parallel or a distributed database. It tells us that we can compute the two-way tables on each of the database fragments in parallel and subsequently merge the results; note, this is also observed in (Gray *et al.* 1997). As of yet, we do not have experimental evidence that this speeds up the mining process, but the advantages seem obvious.

Computing Aggregates in the Database

The motivation for two-way tables as "the" query on which quality functions are based is the fact that all statistics that can be derived from the database can be derived from two-way tables. This follows from the observation that $2WT(\mathcal{A}, \emptyset, true, db)$ simply yields the *set* of tuples in the database extended with their multiplicity.

The disadvantage of using two-way tables is the communication between KESO and the underlying DBMS. Although in practice the two-way tables are far smaller than the database, the overhead is considerable. In other words, KESO would become far more efficient if the evaluation function itself would be computed in the database. In fact, it would be optimal if we could compute the evaluation function while we are constructing the two-way table. What SQL offers in this respect are aggregates beyond **COUNT** and **SUM**.

Allowing such aggregation functions in our two-way tables does offer huge potential savings in the communication between KESO and the underlying DBMS. Generalising the two-way tables to allow the computation of aggregate functions, however, is a potential threat to the optimization schemes outlined in the previous section. In other words, the savings in communication costs could be anihilated by the increase in the costs of computing the tables. Clearly, the aggregated value itself cannot function as the intermediate result that can be re-used for subsequent quality calculations. What should be re-used is the table on which this aggregate is computed. This table itself (the generalisation of the two-way table) may be computed using a different aggregation function, say G. If we want our observations of the previous section to go through, G has to be, (Gray *et al.* 1997), *distributive:*

Let $X = \{X_{i,j} \mid i \in \{1, \ldots, I\}, j \in \{1, \ldots, J\}\}$ be a two-dimensional data set. The aggregate function G is distributive if there exists an aggregate function H such that

$$G(X) = H(\{G(\{X_{i,j} \mid i \in \{1, \ldots, I\}) \mid j \in \{1, \ldots, J\}\})$$

Examples of distributive aggregate functions are **SUM** and **COUNT**.

Clearly, there is no need that F is based on only one distributive aggregate function G, it may depend on a *vector* of such functions provided there is a function M that combines these aggregated values into one aggregate value. In the spirit of (Gray *et al.* 1997), we say that an aggregate function F is distributive algebraic if there exists an $k - tuple$ of distributive aggregate functions $(G_1, \ldots, G_k)$ and a function M such that:

$$F(X) = M(G_1(X), \ldots, G_k(X))$$

Examples of distributive algebraic aggregate functions are all the (central) moments of the distribution of attribute values.

Finally, we define a *Data Mining Measure* as an $l - tuple$ of distributive algebraic aggregate functions.

Since a distribution is completely characterised by all its central moments, a *Data Mining Measure* is a true generalisation of a two-way table. All the more since the optimisation possibilities for Data Mining Measures are the same as those for two-way tables.

Currently experiments are underway with a version of KESO in which the quality functions are based on Data Mining Measures rather than two-way tables.

Conclusions

Mature data mining tools require interaction with existing DBMSs. This paper gives the concepts that underly the KESO data mining system focussed on this interaction. Moreover, three orthogonal approaches to the optimization of this interaction that follow naturally from the KESO framework are discussed.

All three approaches are based on simple observations on the laws that govern the special type of aggregated queries used by KESO, viz., two-way tables. Two of these approaches use a lattice structure on the set of two-way tables. The first approach stores intermediate results to minimise the number of tuples a two-way table has to be evaluated against. The second approach exploits communalities between batches of two-way tables, again to minimise the size of the input table for these queries.

The third approach is aimed at parallel and distributed databases. It shows how the the two-way computed on each of the fragments can be combined into the final two-way table.

Subsequently, we have taken this optimization framework as fixed and discussed how generalisations of the two-way table may optimize the database interaction further. The focuss in this discussion is on the communication between KESO and the underlying DBMS. It is shown that special types of aggregated queries, *distributive algebraic aggregates* cause very low communication overhead while keeping the optimization framework in tact.

The upshot of this discussion is that the extensions that KESO requires of Standard SQL to optimize data mining is limited to the carefull addition of aggregate functions; just as is argued in (Gray *et al.* 1997). This differs considerably from the requirements posed by approaches such as DMQL ((Han *et al.* 1996)) and M-SQL ((Imielinski, Virmani, & Abdulghani 1996)). The main reason is that KESO is based on an abstract view of data mining algorithms, whereas both DMQL and M-SQL are based on the direct implementation of data mining algorithms.

The work in this paper is closely related to the work on datacubes as presented in (Gray *et al.* 1997; Harinarayan, Rajaraman, & Ullman 1996). In fact, it combines the best from both papers. From (Harinarayan, Rajaraman, & Ullman 1996) we inherit the lattice on views on the datacube. The important difference with that paper is that we decide dynamically which "views" (=two-way tables) to store temporarily. Our distributive algebraic queries are completely in the spirit of (Gray *et al.* 1997). The important difference with that paper is that we do not pre-compute the complete cube, but only those portions that are actually used. An important difference with both papers is that this paper clearly shows the connections between the datacube and data mining.

Currently experiments are underway with a version of KESO which uses the generalisations of two-way tables introduced in this paper.

Acknowledgements

This work is sponsored by the EC under contract Esprit 30596.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1994. *Foundations of Databases.* Addison Wesley.

Fayyad, U. M., and Uthurusamy, R., eds. 1994. *AAAI-94 Workshop Knowledge Discovery in Databases.*

Fayyad, U. M., and Uthurusamy, R., eds. 1995. *AAAI-95 Conference on Knowledge Discovery and Data Mining.*

Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds. 1996. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press.

Gray, J.; Chaudhuri, S.; Bosworth, A.; Layman, A.; Reichart, D.; Venkatrao, M.; Pellow, F.; and Pirahesh, H. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Mining and Knowledge Discovery, An International Journal* 1.

Han, J.; Fu, Y.; Wang, W.; Koperski, K.; and Zaiane, O. 1996. Dmql: A data mining query language for relational databases. In *Proceedings of the SIGMOD-96 workshop on KDD*, ??

Harinarayan, V.; Rajaraman, A.; and Ullman, J. D. 1996. Implementing data cubes efficiently. In *Proceedings of the 1996 SIGMOD Conference*, 205 – 216.

Holsheimer, M.; Kersten, M.; and Siebes, A. 1996. Data surveyor: Searching the nuggets in parallel. In Fayyad et al. (1996).

Imielinski, T.; Virmani, A.; and Abdulghani, A. 1996. Datamine: Application programming interface and query language for database mining. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 256 – 261.

Siebes, A. 1996. Data mining and the KESO project. In *SOFSEM'96: Theory and Practice of Informatics*, volume 1175 of *Lecture Notes in Computer Science*, 161 – 177. Springer-Verlag.

Simoudis, E.; Han, J.; Fayyad, U. M.; and Uthurusam, R., eds. 1996. *AAAI-96 Conference on Knowledge Discovery and Data Mining.*