A FRAMEWORK FOR DATA-PARALLEL KNOWLEDGE DISCOVERY IN DATABASES.

Alex A. Freitas[1] and Simon H. Lavington[1].

1 Introduction and Overview of the Framework.

Despite the great demand for KDD (Knowledge Discovery in Databases) in large database and data warehouse systems, in general KDD algorithms have been applied to relatively small data samples (typically less than 10,000 tuples) and do not have any integration at all with Relational DBMS (RDBMS). The application of KDD algorithms to large databases faces serious scalability problems, particularly concerning unacceptably long processing times [Provost & Aronis 96].

This paper proposes a framework for data-parallel KDD, aiming mainly at improving the efficiency and scalability of KDD algorithms. The framework also has other advantages, such as software reusability and easy data-privacy control, but these are not discussed here. Our approach is based on generic, context-free, set-oriented primitives. The primitives are *generic* in the sense that they capture the core operations underlying a number of KDD algorithms. This is important because no single algorithm can be expected to perform well across all domains [Michie et al. 94], [Schaffer 94]. Moreover, the primitives are *set-oriented*, i.e. they perform operations on data elements independently of the order of those elements. This allows us to efficiently exploit data parallelism on cost-effective Parallel Database Servers (PDSs) through SQL database queries.

We assume that the RDBMS has a commonplace client-server architecture. The basic idea of the framework is illustrated in Figure 1. As shown in this Figure, the data is kept on a PDS, and the KDD algorithm does not have direct access to the data. That algorithm invokes our primitives by sending database queries to the PDS, which efficiently accesses the data and returns the query results to the Client. The database queries submitted by the Client are actually requests for the execution of set-oriented primitives on the PDS.

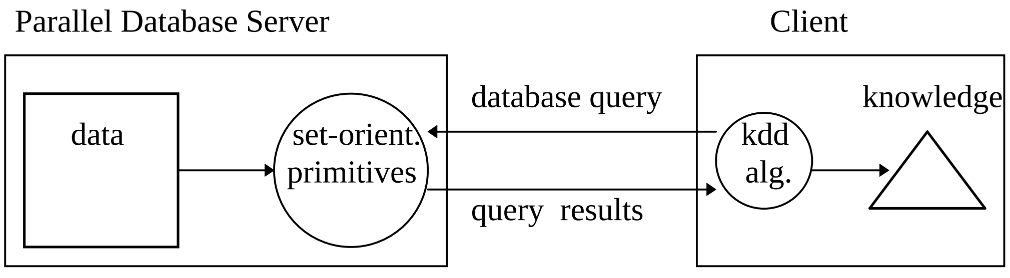

Figure 1: A framework for data-parallel KDD.

The challenge is to develop KDD primitives that are *(1) Well-defined; (2) Computationally significant* (taking a significant part of the total processing time of the KDD algorithm); *(3) Generic*; and *(4) Set-oriented.*

In this short paper we focus on briefly discussing the generality of the proposed primitives and their efficiency in the exploitation of data parallelism. A more detailed discussion about the primitives, their generality and parallelism-related results is presented in [Freitas & Lavington 96b]. To simplify the discussion, we assume that the KDD task is classification, but the proposed primitives can be used for other tasks as well. To measure the efficiency in the exploitation of data parallelism we did experiments comparing a MIMD machine, namely the White Cross WX9010 PDS, against an Ingres 6.4 DBMS running on a 25-

[1] {freial,lavis}@essex.ac.uk
University of Essex, Dept. of Computer Science, Colchester, CO4 3SQ, UK.
(The first author is supported by Brazilian government's CNPq, grant number 200384/93-7.)

MHz, 24-MBytes-RAM Sun IPC. The WX9010 (release 3.2.1.2) has 12 T425 transputers, each with 16 Mbytes RAM, each rated at about 12 MIPS and 25 MHz [Burwen 94]. It has a very high scanning rate: 3 million tuples/sec. Note that each transputer belongs to the same technology generation and has roughly the same MIP rate as the Sun IPC workstation. Ten out of the 12 transputers are actually used to process the query in parallel.

2 A Rule Induction (RI) Primitive: Generality and Summary of Results.

In essence, a RI algorithm iteratively selects the "best" candidate rule (CR), expands it (generating new CRs) and measures the quality of the just-generated CRs, until a satisfactory set of CRs is found. In our framework the client is in charge of selecting the next CR to be expanded and expanding it, but CR-quality measures are computed in parallel on the PDS, through a set-oriented primitive [Freitas & Lavington 96].

In essence, this primitive counts the number of tuples in each partition (i.e. group of tuples with the same value for the *Group By* attributes) formed by a relational *Group By* statement. The input parameters of the primitive are the candidate attribute (possibly used to expand the current CR), the class attribute and a tuple-set descriptor (a logical conjunction of attribute-value pairs describing the tuples covered by the current CR). The primitive selects the tuples satisfying the tuple-set descriptor and partitions them into groups, with one group for each combination of values of the candidate and goal attributes. Here we assume that the candidate attribute is categorical. Continuous attributes are discretized in advance - see [Freitas & Lavington 96a]. Then the number of tuples in each group is counted, and this information is the output of the primitive.

The generality of the primitive stems from the fact that it can be used to compute several candidate-rule (CR) quality measures, such as Information Gain and Information Gain Ratio [Quinlan 93], Reduction of Gini Diversity Index [Breiman et al. 84], J-measure [Smyth & Goodman 92], Chi-Squared and Cramer's V Coefficient [Zytkow & Zembowicz 93], etc. Each of these CR-quality measures is used by several KDD algorithms, so that the primitive is truly generic.

To evaluate the efficiency in the exploitation of data parallelism, experiments were done with three databases, namely two randomly-generated databases, each of them generated twice with 100,000 and 200,000 training tuples (so that effectively four synthetic databases were produced), and the Labour Force Survey dataset produced by the UK's Department of Employment, with 103,219 training tuples.

The experiments consisted of applying two versions of a TDIDT (Top-Down Induction of Decision Tree) algorithm to each of the five databases. Averaging over the ten experiments, the data-parallel versions of the TDIDT algorithm achieved a speed up (Sp) of 8.1 over their sequential counterpart. There was a significant variance among the individual experiments. In general the Sp depends strongly on the number of tuples counted per database query (i.e. per run of the primitive). When the primitive is executed in high levels of the tree (close to the root) a large number of tuples is counted, but in lower levels of the tree (close to the leaves) few tuples are counted, which reduces the efficiency in the exploitation of data parallelism. Hence, the Sp of the data-parallel version depends on the number of nodes of the induced tree. The smaller this number, the larger the Sp.

3 An Instance-Based Learning (IBL) Primitive: Generality and Summary of Results.

In essence, an IBL algorithm compares the new tuple (to be classified) with all stored instances (tuples) and retrieves the "nearest" (most similar) - or the K nearest, where K is a user-specified parameter - tuple(s) to the new one, as determined by a distance metric. Then the class of the retrieved tuple, or the prevalent class in the K retrieved tuples, is assigned to the new tuple.

In our framework the computation of the distance metric between the new tuple and all stored tuples is done in parallel on the PDS, through a set-oriented primitive. This primitive is based on an abstract Minkowski metric [Salzberg 91], from which particular distance metrics such as Euclidean and Manhattan distances are easily derived by specifying a parameter (an exponent) in the Minkowski-metric abstract formula. The input parameters of the primitive are the attribute values of the new tuple, a parameter specifying a particular distance metric in the Minskowski-metric abstract formula, a list of attribute weights and a set of tuple weights. The last two parameters are optional, i.e. they are used only by attribute-weighted and tuple-weighted IBL algorithms, respectively. The distance between two attribute values is computed through arithmetic functions such as abs(), which are commonplace in major RDBMS. The output of the primitive is the set of distance values between the new tuple and each stored tuple. This output is then further processed by relational operations on the PDS (e.g. selecting the tuple with minimum distance), to avoid the large client/server communication overhead of returning all distances to the client.

The generality of the primitive stems from the fact that it can be used to compute several kinds of distance metrics and IBL algorithms, such as (a) conventional unweighted, attribute-weighted and tuple-weighted metrics, based directly on the similarity between corresponding pairs of attribute values in two tuples [Aha 92]; (b) rule-based similarity metrics, where the distance between two tuples depends on the number of rules satisfied by the two tuples [Sebag & Schoenauer 93]; (c) hybrid RI/IBL algorithms interpreting rules as instances [Agre 95]; and (d) loosely-coupled hybrid RI/IBL algorithms (where the IBL algorithms is kept separate from - but co-operate with - the RI algorithm) [Surma & Vanhoof 95].

To evaluate the efficiency in the exploitation of data parallelism, experiments were done with three databases, namely the Letter dataset and the Shuttle dataset used in the Statlog project [Michie et al. 94], with respectively 15,000 and 43,500 training tuples, and the Labour Force Survey dataset produced by the UK's Department of Employment, with 113,432 training tuples. The experiments consisted of applying two IBL algorithms, an unweighted and an attribute-weighted one (both of them using the Manhattan distance), to each of the three databases. Averaging over the six experiments, the data-parallel versions of the algorithms achieved a speed up (Sp) of 8.6 over the sequential version. There was a significant variance among the individual experiments. In all three databases, the Sp for the unweighted algorithm was much larger than the Sp for the attribute-weighted algorithm (which uses more complex arithmetic operations to compute the distance metric). Hence, although the WX9010 can rapidly perform relational *selection* (due to its very high scanning rate), it is very sensitive to the arithmetic complexity of the distance metric.

4 Conclusions.

We briefly described primitives for two major KDD paradigms, namely Rule Induction and Instance-Based Learning. The primitives are generic, i.e. they find use in a number of algorithms, and set-oriented, i.e. they access many-tuples-at-a-time in an unordered fashion. We used the primitives to implement data-parallel versions of KDD algorithms that achieved a roughly linear speed up over their sequential counterparts. We also discussed some characteristics of KDD algorithms that significantly affected the speed up.

Acknowledgments.

We thank Steve Hassan for his help in using the WX9010 and Dominicus R. Thoen for his help in the IBL experiments.

References.

[Agre 95] G. Agre. Knowledge-base maintenance as learning two-tiered domain representation. *Proc. 1st Int. Conf. CBR (ICCBR-95). LNAI 1010*, 109-120. 1995.

[Aha 92] D.W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *Int. J. Man-Machine Studies*, 36, 1992, 267-287.

[Breiman et al. 84] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone. *Classification and Regression Trees*. Pacific Groves, CA: Wadsworth, 1984.

[Burwen 94] M.P. Burwen. The White Cross parallel database servers. *The Superperformance Computing Service. Product/Technology Review* No. 145. (2685 Marine Way, Suite 1212, Mountain View, CA, USA)

[Freitas & Lavington 96] A.A. Freitas and S.H. Lavington. Using SQL primitives and parallel DB servers to speed up knowledge discovery in large relational databases. R. Trappl. (Ed.) *Cybernetics and Systems'96: Proc. 13th Europ. Meet. on Cyb. & Syst. Res*, 955-960. Vienna: Austrian Soc. for Cybern. Studies, 1996.

[Freitas & Lavington 96a] A.A. Freitas and S.H. Lavington. Speeding up knowledge discovery in large relational databases by means of a new discretization algorithm. R. Morrison & J. Kennedy (Ed.) *Advances in Databases: Proc. 14th British. Nat. Conf. on Databases*. 1996. LNCS 1094, 124-133.

[Freitas & Lavington 96b] A.A. Freitas and S.H. Lavington. Exploiting data parallelism in generic data mining primitives. Submitted to *Parallel Computing Journal*, Special Issue on Parallel Data Servers and Applications. 1996.

[Michie et al. 94] D. Michie, D.J. Spiegelhalter and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[Provost & Aronis 96] F.J. Provost and J.M. Aronis. Scaling up inductive learning with massive parallelism. *Machine Learning* 23(1), Apr./96, 33-46.

[Quinlan 93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Salzberg 91] S. Salzberg. Distance metrics for instance-based learning. *Proc. 6th Int. Symp. Methodologies for Intelligent Systems (ISMIS'91)*. LNAI 542, 399-408. 1991.

[Schaffer 94] C. Schaffer. A conservation law for generalization performance. *Proc. 11th Int. Conf. Machine Learning,* 259-265. 1994.

[Sebag & Schoenauer 93] M. Sebag & M. Schoenauer. A rule-based similarity measure. *Proc. 1st European Workshop on CBR (EWCBR-93).* LNAI 837, 119-131. 1993.

[Surma & Vanhoof 95] J. Surma and K. Vanhoof. Integrating rules and cases for the classification task. *Proc. 1st Int. Conf. CBR (ICCBR-95). LNAI* 1010, 325-334. 1995.

[Smyth & Goodman 92] P. Smyth and R.M. Goodman. An information theoretic approach to rule induction from databases. *IEEE Trans. Knowledge and Data Engineering*, 4(4), 301-316. Aug./92.

[Zytkow & Zembowicz 93] J. Zytkow and R. Zembowicz. Database exploration in search of regularities. *Journal of Intelligent Information Systems*, 2(1), Mar. 1993, 39-81.