Integrating Query Processing and Data Mining in Relational DBMSs

Qiang Ding, William Perrizo, Victor Shi
Department of Computer Science
North Dakota State University
Fargo, ND 58105-5164
{Qiang.Ding, William.Perrizo, Victor.Shi}@ndsu.nodak.edu

Kirk Scott
Department of Mathematical Sciences
University of Alaska, Anchorage
Anchorage, Alaska 99508
afkas@uaa.alaska.edu

Abstract

In a database system, careful selection, project, and join (SPJ) optimisation methods are needed to achieve good performance. This is an area of much research in the past two decades, yet much remains to be done. Also, researchers have begun to view data mining as being an integral part of query processing, thus the two are intended to be jointly optimised. Data mining is at one end of the query spectrum and standard SPJ queries are at the other in terms of request definiteness (?). In SPJ queries, the desired result is fully describable ahead of time as one relation, while in data mining the desired result can only be described after the fact, as rules, decision trees, partitions or similar constructs (??). Nonetheless, in both cases the user desires to extract information from relational data and very often the desired information involves both SPJ querying and data mining (e.g., find all association rules on a relation that is the result of an SPJ query on several base relations). In this paper we introduce a mechanism to facilitate efficient SPJ query processing and data mining in a unified fashion. Using a compression method called Peano Trees (P-trees), I/O can be reduced to an absolute minimum (??), indexes can be eliminated entirely and query processing is optimized with data mining effectively.

1. INTRODUCTION

The concept of vertical partitioning has been studied within the context of both centralized and distributed database systems. Copeland et al presented an attribute level ***Decomposition Storage Model*** (DSM) [CK85]. DSM stores each column of a relational table into a separate table. Attribute level Vertical Decomposition is also used in Remotely Sensed Imagery (e.g., Landsat Thematic Mapper Imagery), where it is called ***Band Sequential*** (BSQ) format. Wong et al took advantage of encoded attribute values using a small number of bits to reduce the storage space [WLO$^+$85]. In this paper, we decompose attributes of relational tables into separate files by bit position. We refer to the proposed model as ***bit Sequential*** (bSQ). The bSQ model is similar to Wong et al Bit Transposed File model (BTF). In addition, we compress the vertical bit files using a query-and-data-mining-ready structure called the Peano Tree. Our approach to vertical partitioning only reads the data that are needed. We encode attribute values into bit vector format, which eases compression complexity. SPJ queries are formulated as Boolean expressions that facilitate fast hardware implementation. Our study shows that the proposed approach works well not only for query processing but also for data mining.

The P-tree and bSQ technologies discussed in this paper are patent pending by North Dakota State University.

This work was partially supported by GSA grant number ACT# K96130308.

2. PEANO TREES (P-TREES)

In [PDD$^+$01], a quadrant-based tree structures, called the Peano Trees of P-tree, was developed to facilitate compression and very fast processing (logical ANDing) of bit sequential (bSQ) data. P-trees can be multi-dimensional. If the data has a natural dimension (e.g., spatial data), the P-tree dimension is matched (??) to the data dimension. Otherwise, the dimension can be chosen to optimize compression.(??) In this paper we will use 2-dimensions throughout.

The Peano Tree is specially designed to facilitate very fast logical AND operations used in data mining and query processing. The most useful form of a P-tree is the *predicate*-P-tree in which a 1-bit appears at those tree nodes corresponding to quadrants for which the predicate holds. The predicate can be a particular bit-position of a particular attribute (Basic P-tree) or, more generally, a set of values for each of a set of attributes (value P-trees, tuple P-trees, etc.). In the figure below, a bSQ file with 64 rows is shown, the file is rearranged into 2-D Peano or Z order in the middle and the P-tree is on the right.

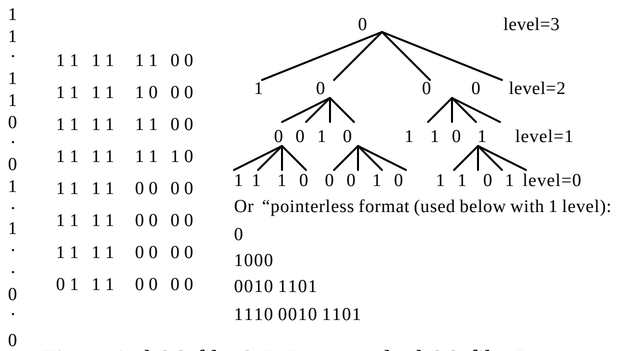

Figure 1. bSQ file, 2-D Peano order bSQ file, P-tree.

In Figure 1, the count of 1 bits in the entire file is called root count of the P-tree (equals 39 in this example). The root count or any other quadrant count can be computed quickly by summing from the bottom up. A P-tree is a type of quadrant tree.

If we compute all quadrant counts and place them at the nodes of a P-tree, it is called a Peano Count tree. In a Peano Count tree, the leaf sequence (depth-first) is a partial run-length compressed version of the original bit vector [DKR+02]. Therefore, P-trees can save substantial amounts of storage. Furthermore, P-tree Boolean operations (AND, OR, and NOT) can be conducted directly without decompression, eliminating a high CPU cost required in most compression algorithms. Each bit file is compressed and stored as a basic P-tree. As discussed in previous work [DKR+02], basic P-trees of different bit positions can be ANDed together resulting in a value P-tree (1 for each quadrant that has that value throughout) or tuple P-tree (1 for each quadrant that has that tuple throughout).

3. QUERY OPTIMIZATION

3.1 Select-Project-Join (SPJ) Queries

Without the loss of generality, we consider a SPJ query where multiple joins and multiple join attributes are involved (non-star join) and also GROUP BY, ORDER BY and DISTINCT clauses. It is shown in [Sco92] that the full elimination of all non-participants can be accomplished with a "two pass" algorithm. We organize our query trees using the "constellation" model in which one of the *fact* files is considered central and the others are points in a star around that central attribute. Each secondary *point* fact file can be the center of a "sub-star". It is useful to view the query tree as a wheel with the central fact file at the center and its dimension files as spokes of that wheel (any one or more of which can be fact files with query sub-wheels).

We apply the selection masks first at the rim of the query wheel. Then we perform semi-joins from the rim toward the central fact file. Finally we perform semi-joins back out again. The result is the full elimination of all non-participants [Sco92]. The following is an example of such a query with a central query wheel (around relation, O) and one sub-query-wheel (around relation, E).

Duplicate elimination after a projection (SQL DISTINCT clause) is one of the most expensive operations in query optimisation. In general, it is as expensive as the join operation. However, in our approach, it can automatically be done while forming the output tuples (since that is done in an order). While forming all output records for a particular value of the ORDER-BY-attribute, duplicates can be easily eliminated without need for an expensive algorithm.

The ORDER BY and GROUP BY clauses are very commonly used in queries and can require a sorting of the output relation. However, in our approach, if the central relation is chosen to be the one with the sort attribute and the surrogation is according to the attribute order (typically the case – always the case for numeric attributes), then the final output records can be put together and aggregated in the requested order without a separate sort step at no additional cost. Aggregation operators such as COUNT, SUM, AVG, MAX, and MIN can be implemented without additional cost during the output formation step and any HAVING decision can be made as output records are being composed, as well.

If the COUNT aggregate is requested by itself, we note that P-trees automatically provide the full counts for any predicate with just one multi-way AND operation. The following example illustrates all these points.

```
SELECT    DISTINCT C.c, R.capacity
FROM      S, C, E, O, R
WHERE     S.s=E.s AND C.c=O.c AND O.o=E.o AND
          O.r=R.r AND C.cred>1 AND (E.grade='B' OR
          E.grade='A') AND R.capacity>10,  DESC;
ORDER BY C.c;
```

S

s	n	gen
0 *000*	A	M *0*
1 *001*	T	M *0*
2 *010*	S	F *1*
3 *011*	B	F *1*
4 *100*	C	M *0*
5 *101*	J	F *1*

C

c	n	cred
0 *00*	B	1 *01*
1 *01*	D	3 *11*
2 *10*	M	3 *11*
3 *11*	S	2 *10*

E

s	o	grade
0 *000*	1 *001*	B *10*
0 *000*	0 *000*	A *11*
3 *011*	1 *001*	A *11*
3 *011*	3 *011*	D *00*
1 *001*	3 *011*	D *00*
1 *001*	0 *000*	B *10*
2 *010*	2 *010*	B *10*
2 *010*	3 *011*	A *11*
4 *100*	4 *100*	B *10*
5 *101*	5 *101*	B *10*

O

o	c	r
0 *000*	0 *00*	0 *01*
1 *001*	0 *00*	1 *01*
2 *010*	1 *01*	0 *00*
3 *011*	1 *01*	1 *01*
4 *100*	2 *10*	0 *00*
5 *101*	2 *10*	2 *10*
6 *110*	2 *10*	3 *11*
7 *111*	3 *11*	2 *10*

R

r	capacity
0 *00*	30 *11*
1 *01*	20 *10*
2 *10*	30 *11*
3 *11*	10 *01*

```
                                             Sn
                                             A
                                             T
                                             S
Ss1  Ss2  Ss3       Sgen                     B
0011 0000 0101      0001                     C
00   11   01        11                       J

Es1  Es2  Es3     Eo1  Eo2  Eo3
0000 0000 0011    0000 0010 1010
0000 1111 1100    0000 0111 1101
11   00   01      11   00   01

Egrade1 Egrade2                                 Cn
1101 11 0100 00 Cc1 Cc2   Ccred1 Ccred2         B
1011    1001    00  01    01     11             D
                11  01    11     10             M
                                                S

Oo1  Oo2  Oo3      Oc1  Oc2       Or1  Or2
0011 0000 0101     0011 0000      0001 1100
0011 1111 0101     0011 1101      0011 0110

Rr1   Rr2          Rcap1    Rcap2
00    01           11       10
11    01           10       11
```

Apply selection masks:

```
mE =Egrade1    mR =Rcap1     mC =Ccred1
1101           11            01
1011           10            11
11
```

results in,

```
Es1   Es2   Es3     Eo1   Eo2   Eo3
00·0  00·0  00·1    00·0  00·0  10·0
0·00  1·11  1·00    0·00  0·11  1·01
11    00    01      11    00    01

Rr1    Rr2            Cc1     Cc2
00     01             ·0      ·1
1·     0·             11      01
```

The semi-join (toward center), E→O(on o=0,1,2,3,4,5), R→O(on r=0,1,2), C→O(on c=1,2,3), reduces

```
Oo1  Oo2  Oo3    Oc1  Oc2    Or1  Or2
0011 0000 0101   0011 0000   0001 1100
0011 1111 0101   0011 1101   0011 0110   to

Oo1  Oo2  Oo3     Oc1  Oc2     Or1  Or2
0011 0000 0101    ··11 ··00    0001 1100
00·· 11·· 01··    0011 1101    00·1 01·0
```

Thus, the participants are c=1,2; r=0,1,2; o=2,3,4,5. Semi-joining back again produces the following.

```
Cc1 Cc2   Rr1 Rr2
·0  ·1    00  01
1·  0·    1·  0·

Es1   Es2   Es3    Eo1   Eo2   Eo3
····  ····  ····   ····  ····  ····
··00  ··11  ··00   ··00  ··11  ··01
11    00    01     11    00    01
```

And thus the s-participants are s=2,4,5.

```
Ss1     Ss2     Ss3
··11    ··00    ··01
0·      1·      0·
```

Output tuples come from participating O.c P-trees as follows.

RootCount$P_{O.c}$(2)=RootCountOc_1^Oc_2'=2, since

```
Oc1 ^ Oc2'
··11  ··11  =   ··11
00··  00··      00··
```

Since the 1-bits are in positions 4 and 5, the two O-tuples have O.o surrogate values 4 and 5. The r-values at positions 4 and 5 of O.r are 0 and 2. Thus, we retrieve the R.capacity values at offsets 0 and 2. However, both of these R.capacity values are 30. This duplication is found without sorting or additional processing. Output is (2,30). Similarly, RootCount$P_{O.c}$(1) = RootCountOc_1'^Oc_2=2,

```
Oc1' ^ Oc2
··00   ··00  =   ··00
11··   11··      11··
```

Since the 1-bits are in positions 2 and 3 this time, the two O-tuples have O.o surrogate values 2 and 3. The r-values at positions 2 and 3 of O.r are 0 and 1. We retrieve the R.capacity values 30 and 20 at R.capacity offsets 0 and 1. The output is (1,30) and (1,20). The final output is:

c	capacity
2	30
1	30
1	20

Finally we note, if the ORDER BY clause is over an attribute which is not in the relation O (e.g., over student number, s) then we center the query tree (or *wheel*) on a fact file that contains the ORDER BY attribute (e.g., on E in this case). If the ORDER BY attribute is not in any fact file (in a dimension file only) then the final query tree can be re-arranged to center on the dimension file containing that attribute. Since output ordering and duplicate elimination are traditionally very expensive sub-operations of SPJ query processing, the fact that our model and the P-tree data structure provide a fast and efficient way to accomplish these operations is a very favorable aspect of the approach.

In our implementation, the basic P-trees files for the join and selection attributes are striped across a cluster of computer systems (Beowulf cluster). AND operations are extremely fast parallel operations which scale well to very large relations (e.g., ~1 billion tuples). In the performance section of this paper, it is shown that large AND operations take only a few milliseconds.

3.2 Data Mining Operations

In other papers, Association Rule Mining Algorithms, Classifiers and Clustering algorithms have been developed and studied which use P-trees for very fast accurate mining [KDP02, PDD+01]. In all these cases, P-trees serve as data-mining-ready (counts pre-computed), compressed data structures. Using P-tree-based algorithms, it is shown that ARM, Classification, and Clustering can be done extremely quickly and with high accuracy. In fact, Closed P-KNN in [KDP02] is both faster and more accurate than existing algorithms.

Many data mining request involve pre-selection, pre-join, and pre-projection of a database to isolate the specific data subset to which the data mining algorithm is to be applied. For example, in the above database, one might be interested in all Association Rules of a given support threshold and confidence threshold across all the relations of the database. The brute force way to do this is to first join all relations into one universal relation and then to mine that gigantic relation. This is not a feasible solution in most cases due to the size of the resulting universal relation. Furthermore, often some selection on that universal relation is desirable prior to the mining step.

Our approach accommodates combinations of querying and data mining without necessitation the creation of a massive universal relation as an intermediate step. Essentially, the full vertical partitioning and P-trees provide a selection and join path, which can be combined with the data mining algorithm to produce the desired solution without extensive processing and massive space requirements. The collection of P-trees and BSQ files constitute a lossless, compressed version of the universal relation. Therefore the above techniques, when combined with the required data mining algorithm can produce the combination result very efficiently and directly. The following graphs show the speed of a multi-way AND operation, the speed and accuracy of ARM and KNN data mining based on P-trees. In the graphs, the operations are parallelized over 16 high end Pentiums (900-MHz with 256 MB main memory, running Windows 2000).

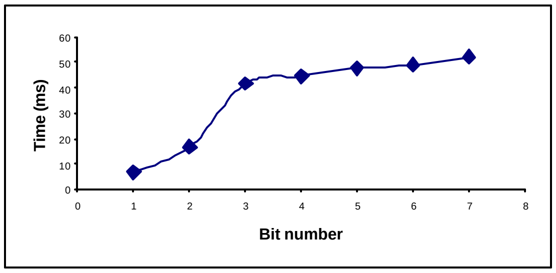

Figure 2. Average AND time.

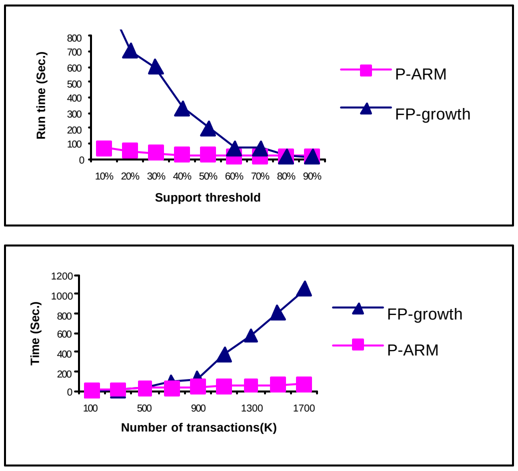

Figure 3. ARM versus support and transactions.

We implemented the classical KNN classifier with the Euclidian distance metric and two different KKN classifiers using P-trees – one with *higher order bit similarity* and another with *perfect centering* [KDP02].

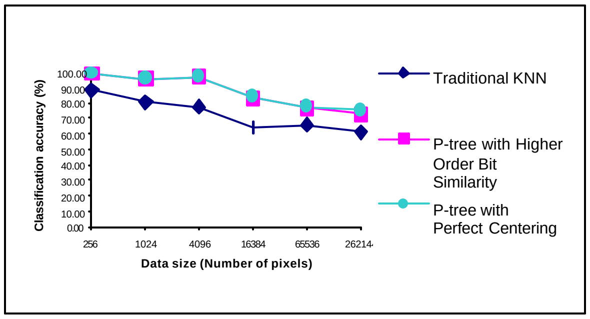

Figure 4. KNN accuracy for different dataset size.

Both classifiers outperform the classical KNN classifier in terms of both classification time and accuracy. We tested our methods using different datasets with different sizes.

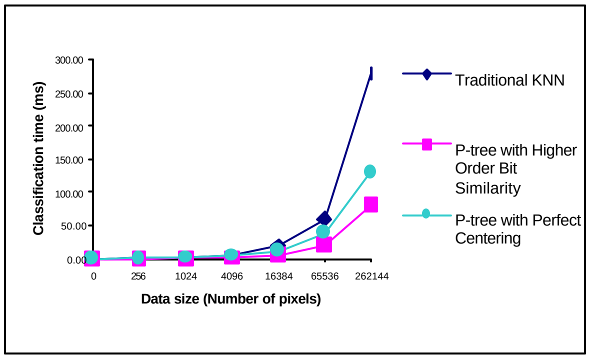

Figure 5. Average KNN time per test sample.

4. CONCLUSION

In this paper we have shown that exceptional selection, project, and join (SPJ) strategies can be unified with proven data mining strategies. It is no longer satisfactory (??) to separate data mining operations from query processing. Data mining is at one end of the query spectrum and standard SPJ queries are at the other. We used P-trees, bit-level structures to facilitate efficient SPJ query processing and data mining. Through complete vertical decomposition of data files and field-level access, only those participating fields are retrieved. We showed that the query and data mining processing are very fast and accurate, the number of I/Os can be minimized (??) and indexes can be completely eliminated.

REFERENCE

[CK85] G.Copeland, S.Khoshafian Decomp. Storage Model. *ACM SIGMOD*, Austin, May 1985.

[DKR+02] Q.Ding, M.Khan, A.Roy, W.Perrizo. P-tree algebra, *ACM-SAC* pp.426, Madrid, 2002.

[KDP02] M.Khan, Q.Ding, W.Perrizo. KNN Using P-trees,*PAKDD*, *LNAI 2336*, Springer, May 02

[PDD+01] W.Perrizo, Qin Ding, Qiang Ding, A.Roy, Deriving high confidence rules from spatial data using Peano Count Trees. *WAIM 2001*, pp.91-102, Xi'an, China, 2001.

[Sco92] K. A. Scott. *Multi-way equijoin query acceleration using hit-lists.* Ph.D. dissertation, North Dakota State U., 1992.

[SR02] Distinguished Database Profiles. SIGMOD Record, 31(2), pp.50, June 2002.

[WLO+85] H. K. T. Wong, H.-F. Liu, F. Olken, D. Rotem, and L. Wong. Bit Transposed Files. *Proc. Int. Conf. on Very Large Data Bases (VLDB'85)*, pp.448-457, Stockholm, 1985.