Why Sort-Merge Gives the Best Implementation of the Natural Join.

T.H. Merrett
School of Computer Science
McGill University, Montreal

Technical Report SOCS-81-37
October 1981

WHY SORT-MERGE GIVES THE BEST IMPLEMENTATION OF THE NATURAL JOIN.

T.H. MERRETT
SCHOOL OF COMPUTER SCIENCE
MCGILL UNIVERSITY, MONTREAL

To join two relations efficiently, they must not only be clustered but *mutually clustered*. Sorting is the only known way to achieve mutual clustering. Once the relations are sorted, merging is the obvious way to implement the join. If the relations are known to be sorted appropriately, the most costly part of the process can be omitted. To know that a relation is sorted already, it is best to remember that we sorted it. Otherwise detecting that the relation is sorted requires inspection of each tuple, a cost comparable in many cases to the cost of sorting. Detecting any other form of mutual clustering is bound to be a combinatorial process of much greater cost than sorting. Cheaper forms of detection may be possible using sampling, but these are heuristics at best and there is no reliable foundation for them.

It is assumed that the sizes of the relations are large compared with the RAM memory available or, equivalently, compared with the number of processors available. Our conclusion that sort-merge is best is thus applicable to single-processor computers or to multiple-processor database machines. Discussing the cost of joining relations without making this assumption is outside the subject of databases.

Keywords: clustering, merging, mutual clustering, natural join, page-pair graphs, relational algebra, sorting.

C R Categories: 4.33

Introduction

A number of writers have approached the question of the cost of performing the operations of the relational algebra. The earliest papers ([G], [P]) did not suggest sort-merge techniques. Blasgen and Eswaran ([BE]) did a study of queries combining projection, restriction and join, which showed that sort-merge was superior in most cases. These works, while based on cost models, did not produce any general arguments for the superiority of any of the methods proposed. Many other papers have been devoted to the optimization of queries, but they are not of direct interest here.

In this paper we give general arguments for the superiority of sort-merge, based on a simple view of the requirements to be met by an optimal join algorithm and based on considerations of the cost of determining that some other method might compute a particular join more cheaply.

The circumstances under which natural joins may be formed are so diverse that in a few situations our argument may be detoured. There are also some areas bridged over by our discussion that may be properly paved by further research. Almost always, however, given what is known at this time, sort-merge should be used, and this paper shows why.

The next section introduces "mutual clustering" of the data of two operand relations as a criterion for optimality of join implementation. This measure is founded on the "page-pair graph" of a join, which is a general way to analyze the cost of any particular join. The next two sections discuss the cost of achieving mutual clustering and, important to our argument, the cost of detecting mutual clustering in a pair of relations. Then we give our conclusion, and, because of the occasional

looseness in the argument, some exceptions to the conclusion and some topics for further research.

Mutual Clustering

To aid our thought we present the join of two relations as a matrix. This is best described by a trivial example, which the reader can extend in his imagination. In Table 1.1, a relation WAREHOUSEMEN is shown as a *horizontal* table, the top row being the attribute STOCKMAN and the lower row being the FLOOR each stockman works on. The relation INVENTORY is shown to the left as a conventional vertical table, with the attributes ITEM and, again, FLOOR, this time meaning the floor on which the item is to be found. The natural join of WAREHOUSEMEN and INVENTORY on the common attribute FLOOR gives the relation RESPONSIBILITY, formed of tuples of WAREHOUSEMEN and tuples of INVENTORY with the same floor. This formation is shown in the body of Table 1.1 by the entries 1 connecting, say, (Computer, 2) from INVENTORY with (Dan, 2) from WAREHOUSEMEN.

		Dan	Jan	Joe	Moe	Nan
		2	2	1	1	2
Computer	2	1	1			1
Snowmobile	1			1	1	
Tractor	1			1	1	
Video Disk	2	1	1			1

Table 1.1 Join of Relations INVENTORY (ITEM, FLOOR) with WAREHOUSEMEN (STOCKMAN, FLOOR)

Since relations such as WAREHOUSEMEN, INVENTORY and RESPONSIBILITY are assumed to be too big for RAM and to require secondary storage, the connections between tuples do not interest us so much as the connections between the *pages* on which the tuples are stored. The dashed lines in Table 1.1 show how WAREHOUSEMEN and INVENTORY might be divided up on two pages each, and how these divisions affect the connections required to form RESPONSIBILITY. It is apparent that every page of WAREHOUSEMEN must be compared with every page of INVENTORY in order to effect the join. Fig. 1.2 shows the *page-pair graph* describing this situation. The pages of INVENTORY are denoted r_1, r_2 and those of WAREHOUSEMEN are s_1, s_2. A node of the graph corresponds to each pair of pages, (r_i,s_j). For the moment, the edges of the graph do not interest us.

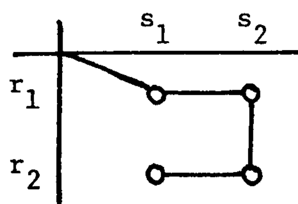

Fig. 1.2 Page-Pair Graph for Join of Table 1.1

For contrast, Fig. 1.3 shows the page-pair graph that arises if both INVENTORY and WAREHOUSEMEN were sorted on the attribute FLOOR and if the data pages were (conveniently) arranged so that each page held only one value of FLOOR. Here, nodes lie only on the *diagonal* (r_1', s_1') and (r_2', s_2').

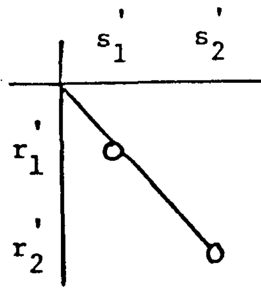

Fig. 1.3 Page-Pair Graph for Join of INVENTORY, WAREHOUSEMEN after Sorting.

The page-pair graphs show the amount of work necessary to complete the join, assuming that the information represented by a page-pair graph is somehow available to the join algorithm, as by an ideal indexing method. In practice, costs will be greater because of the cost of indexing. When a sort-merge technique is used, the page-pair graph takes on a block-diagonal form and the merge algorithm can process the operand relations at the cost given by the page-pair graph.

How do we find the cost from the page-pair graph? If we make the restrictive assumption that only two pages at a time can be processed in the fast memory of the computer, we can describe the join algorithm as a path connecting the nodes of the page-pair graph. As described in [MKY] a horizontal or vertical edge costs only one access because only one page is being replaced, while any other edge - such as diagonal- costs two accesses. Such paths are shown in Figs. 1.2 and 1.3. In [MKY] it is shown that the problem of finding the cheapest path is NP-complete. An NP-complete problem is traditionally considered to be hopeless, and in this case it really is hopeless since the probable exponential complexity is a function of the number of pages in the operand relations - a number assumed to be large.

If we allow more than two pages into RAM at a time, the problem of finding the optimum path is still more complicated.

Suppose nonetheless that we know the best path for some arbitrary arrangement of data on the pages, as, for example, in Fig. 1.2. It is plain that this path is more expensive than the path for the sorted data, in Fig. 1.3 (cost of 5, compared with 4). Clearly the fewer nodes there are in the page-pair graph, the cheaper the join will be. *The number of nodes in the page-pair graph depends on the arrangement of data on the pages.* In particular, sorted data will give the fewest nodes.

This leads us to define *mutual clustering* as an indication that a minimum-cost join may be possible. Two relations are said to be mutually clustered with respect to the arrangement of their data on pages if their page-pair graph has no more nodes than the page-pair graph of the same two relations when sorted. Mutual clustering is a necessary condition for a minimally cheap join.

In [BE] some results are given to suggest that clustered relations can be joined at minimum cost with the use of indexes. By *clustered* is meant that common values of the join attribute are grouped together on common pages: this is a property of individual relations, not a joint property of pairs of relations. That it is insufficient in general to give a cheap join is illustrated by any join in which the join attributes are the keys of their relations. In this case, both relations are clustered by default, but the data may still be arranged such that every possible node of the page-pair graph would consist only of one node in each row and one in each column.

(If the relations were sorted, these nodes would be on the diagonal of the page-pair graph). Only if the number of tuples per value of the join attributes is at least as large as the page size, does clustering become a guarantee of a cheap join.

2. The Cost of Mutual Clustering

How do we ensure that two relations to be joined are mutually clustered? To the knowledge of this author, there is only one way: sort them. This way is also remarkably cheap when the data occupies many times the RAM capacity of the computer. It is n log n accesses of course, where the relation has n pages of data, but the base of the logarithm is significantly larger than 2 - perhaps about 20 on a big machine - depending on the number of subfiles that can be merged simultaneously. Linear time sorting is also possible on special parallel hardware - see [OM] or [T]. If there were other ways of attaining mutual clustering, this section could be longer. However, it is unlikely that any cheaper way is available.

3. The Cost of Detecting Mutual Clustering

We can distinguish deterministic and sampling methods of detecting if two relations are already mutually clustered. Deterministic methods will have costs that are at least linear in the sizes of the relations, while sampling methods do not give guaranteed answers.

Detecting if a file is sorted requires a pass through it to confirm that each record fits into the monotonic sequence. Detecting if a file is clustered can be done in n log n accesses either by building a tree - say a B-tree- and checking that each leaf is never revisited after the filling of another leaf is started; or by sorting and comparing counts of the number of records for each value of the attribute whose clustering we want to verify. This does not detect mutual clustering, of course, but either method can also tell us if there are enough records for each value of the attribute to make clustering equivalent to mutual clustering.

Detecting mutual clustering in general is as much an open problem as creating two mutually clustered files by means other than sorting. It is likely to be a problem of complexity comparable to the problem of finding optimal paths, and is unlikely to be as cheap as sorting the files in the first place to attain a special form of mutual clustering.

The above methods are *deterministic* in the sense that once they have produced an answer we can be sure of that answer. They are bound to access every page of each relation at least once. The one method that accesses each page no more than once detects if the relation is sorted or not: since data is unlikely to be sorted by accident, it is cheaper just to remember that we sorted it. Other methods are at least as expensive as sorting in the first place.

To test for mutual clustering in less than linear time, we must resort to studying *samples* of the data in the relations. For instance, if we see that the data in the first ten pages of a relation is in ascending order, we can hazard the guess that the data has been sorted. Or if we build a page-pair graph for the first ten pages of each relation and find that there are nearer 10 nodes than 10 x 10 in it, we can surmize that the relations are mutually clustered. (The converse does not hold: mutually clustered relations can use all possible nodes of the page-pair graph).

The drawback of the sampling approach is that it is not theoretically legitimate, unlike, say, sampling methods of quality control, which can be used with precisely defined degrees of confidence. This is because, in simple terms, the sampling we propose is not the complete inspection of a small number of individuals in a well-specified population but is rather like passing an aircraft for safety on the basis of looking at its turbofans and tailplanes. A relation is a highly structured object and looking at part of it tells us little about the whole. We can go a little further than this by noting that, for instance, a relation is unlikely to be sorted by accident: thus finding the first ten pages sorted is a very strong indication that the whole is sorted. Similarly with mutual clustering - except that the only way we know how to achieve it is by sorting.

4. Conclusion

To find out if two unsorted relations are mutually clustered is at least as expensive as sorting them. Once they are sorted, to compute the natural join is essentially a merge process, of a cost which is linear in the sizes of the operands and result. (Unless some value of the join attribute is associated in both relations with more tuples than can fit in RAM: then the cost is necessarily worse than linear.) Thus, even joins which are in principle cheaper to implement using, say, indexes, the cost of discovering that they will be cheap makes it preferable just to use sort-merge in any case.

5. Exceptions

Some exceptions can be made to this argument. For instance, if a selection is made on one of the relations before joining, the result may be small enough that direct access to the other relation by indexes may be preferable to sorting the large relation.

If the join attribute is functionally dependent on the attribute which the relation is sorted on, the relation is clustered on the join attribute and is mutually clustered with another relation whose join attribute is either the sort attribute or is dependent on the sort attribute.

If either relation is *multipaged* (see [M], [MO]) the cost of sorting is marginally cheaper, making sort-merge techniques even more favourable. Multipaging is a data organization which pre-sorts a relation on several attributes simultaneously in O(n log n) accesses.

6. Topics for Further Research

The topics we propose for further research are just the holes in our argument. Thus, if anyone can find a method for mutual clustering which is cheaper than sorting, there may be a case for weakening our conclusion to include that method. If somebody can give a rigourous basis for a method of sampling individual relations to detect regularities such as sortedness or mutual clustering, then we might make a stronger case for methods other than sort-merge since detecting when they are applicable would be relatively cheap.

These holes may either be filled or widened by future research. They will probably be filled. In any case, given the state of the art and our present knowledge, the natural join is best implemented using sort-merge.

Acknowledgements

I am indebted to Yahiko Kambayashi and Luc Devroye for helpful discussions of this problem. The work was supported by the Natural Science and Engineering Research Council of Canada under grant NSERC-A4365.

References

[BE] M.W. Blasgen, K.P. Eswaran, Storage and access in relational data bases, IBM Systems Journal 16, 4 (1977) 363-77.

[G] L.R. Gotlieb, Computing joins of relations, Proc. ACM-SIGMOD International Conference on Management of Data (San Jose, May 14-16, 1975) 55-63.

[M] T.H. Merrett, Multidimensional paging for efficient database querying, Proc. ICMOD 78 International Conference on Data Base Management Systems, (Milan, 29-30 June 1978) 277-90.

[MKY] T.H. Merrett, Y. Kambayashi, H. Yasuura, Scheduling of page fetches in join operations, Proc. 7th International Conference on Very Large Data Bases (Cannes, 9-11 Sept 1981) 488-98.

[MO] T.H. Merrett, Ekow Otoo, Multidimensional paging for associative searching. McGill University Technical Report SOCS-81-18, May 1981.

[OM] J.A. Orenstein, T.H. Merrett, Linear sorting methods using log n processors, McGill University Technical Report SOCS-81-24, Oct. 1981.

[P] R.M. Pecherer, Efficient evaluation of expressions in a relational algebra, University of California at Berkeley. Memorandum No. ERL-M510, 19 Feb. 1975.

[T] S.J.P. Todd. Algorithm and hardware for a merge sort using multiple processors, IBM S. Research and Development 22 5 (Sept. 1978) 509-17.