Accurate Query Optimization by Sub-plan Memoization

Ashraf Aboulnaga
University of Wisconsin - Madison[*]
ashraf@cs.wisc.edu

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

December 1999

Technical Report
MSR-TR-99-102

Query optimizers use approximate techniques such as histograms or sampling for result size and distinct value estimation, even though these techniques may incur high estimation errors, leading the optimizer to choose sub-optimal query execution plans. In this report, we propose a novel approach to query optimization that provides the query optimizer with *exact values* for the result size of operators and operator trees, which we call *sub-plans*, and for the number of distinct values in the output of these sub-plans. In our approach, the query optimizer optimizes the query and records all the sub-plans for which result size or distinct value estimates are required in a data structure that we call the *sub-plan memo*. After query optimization is completed, the sub-plans in the sub-plan memo are executed and their actual result sizes and the number of distinct values in their outputs are recorded in the memo. The optimizer then re-optimizes the query using the more accurate result size and distinct value information in the sub-plan memo, which results in potentially choosing a better query execution plan. The process is repeated if re-optimization encounters sub-plans that are not in the sub-plan memo. This technique can result in potentially increasing the optimization time sharply. However, it is very effective in choosing the optimal query execution plan. Therefore, this approach is primarily suitable for frequently executed queries embedded in application programs. We present an experimental evaluation of our proposed approach based on a prototype implementation in Microsoft SQL Server 2000.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com/

[*] Work done while the author was at Microsoft Research

1 Introduction

Estimating the cost of candidate query execution plans is an essential part of query optimization. The cost models used by query optimizers to estimate the cost of candidate query execution plans usually require estimating the *result sizes* (or *selectivities*) of the operators constituting these plans and the *number of distinct values* in the output of these operators. Result size and distinct value estimates play a very important role in cost estimation. More accurate information about these two quantities typically results in more accurate cost estimates, which helps the optimizer choose more efficient query execution plans.

The result size of an operator and the number of distinct values in its output depend on the data distribution of its inputs. To estimate these two quantities, database systems use various techniques to approximate input data distributions, such as *histograms* [PIHS96] or *sampling* [LNS90]. These approximate techniques, no matter how accurate, always incur some estimation error. Furthermore, the estimation error grows significantly as the estimates are propagated through the joins in the query execution plan. It is shown in [IC91] that the estimation error can grow exponentially in the number of joins. Also, distinct value estimation is an inherently difficult problem [CMN98], so it can be expected that any distinct value estimation technique will incur a significant estimation error.

Query optimizer cost models are highly sensitive to result size and distinct value estimates. Errors in estimating these quantities usually lead to errors in estimating plan costs, which can ultimately cause the optimizer to choose a sub-optimal query execution plan [ML86].

These traditional estimation techniques are used despite the possibility that they can lead to choosing sub-optimal plans because they allow the query optimizer to avoid extremely bad plans and choose plans that, while not necessarily optimal, are usually "good enough." Furthermore, these techniques satisfy an important requirement of any result size or distinct value estimation technique: they provide the required estimates *fast.* Query optimization, including all the necessary estimation steps, must not take more than a small fraction of the time to execute the query. Query optimizers, therefore, use these fast but possibly inaccurate estimation techniques since they lead to choosing acceptable query execution plans, even though these plans may not be truly optimal.

Requiring that query optimization be fast and accepting sub-optimal query execution plans may be the correct approach for *ad hoc* queries. It is not necessarily the correct approach for queries *embedded* in application programs, which comprise a large portion of the workloads handled by database systems. These queries are often optimized off-line to produce *compiled* query execution plans that are then used whenever the queries are executed. Optimization does not necessarily have to be fast since it is an off-line process. Furthermore, these queries are typically executed frequently since the applications that they are part of are typically executed frequently. Thus, the cost of optimization is amortized over many executions of the query. Moreover, finding the optimal execution plan in this setting is more important than for ad hoc queries, because the repeated execution of the queries will increase the effect of any savings in execution time. For these embedded queries, the user may be willing to spend more time optimizing the query, and obtaining more accurate cost estimates, if this results in choosing a more efficient query execution plan.

In this report, we propose a novel approach to query optimization that allows a query optimizer to estimate the cost of candidate query execution plans using *exact values* for the result sizes of the operators and operator trees encountered during optimization and the number of distinct values in their output. We call such operators or operator trees *sub-plans*. In our approach, query optimization is done in *phases*. In each phase, the query optimizer fully optimizes the query and produces a query execution plan. In the first phase, the optimizer optimizes the query using its traditional techniques for result size and distinct value estimation. During this optimization, the optimizer records all the sub-plans (operators or operator trees) for which result size or distinct value estimates are required in a data structure that we call the *sub-plan memo*. After the optimization is completed, the sub-plans in the sub-plan memo are executed and their actual result sizes and the actual number of distinct values in their outputs are determined and recorded in the sub-plan memo.

In the second phase, the query optimizer re-optimizes the query, but whenever it needs result size or distinct value estimates for a sub-plan for cost estimation, it looks for this sub-plan in the sub-plan memo. If the sub-plan is found, the optimizer uses the accurate result size and distinct value information in the sub-plan memo. The algorithm used by the query optimizer for searching the plan space when optimizing the query in the second phase is the same algorithm used in the first phase. Thus, most of the sub-plans that the optimizer encounters in the second phase will be ones that were already encountered in the first phase, so they will be found in the sub-plan memo. However, since the second phase uses the more accurate result size and distinct value information found in the sub-plan memo, the optimizer may search parts of the plan space not searched in the first phase and encounter new sub-plans that are not in the sub-plan memo. If the optimizer encounters sub-plans that are not in the sub-plan memo, their cost is estimated using traditional techniques, and they are added to the sub-plan memo. At the end of the phase, all these newly encountered sub-plans are executed and their actual result sizes and the number of distinct values in their outputs are recorded in the sub-plan memo.

This process is repeated until the optimizer goes through a phase in which it does not encounter any new sub-plans. The output query execution plan is the one chosen by this last phase. This plan is chosen using completely accurate result size and distinct value information obtained from the sub-plan memo. Our approach converts result size and distinct value estimation to *memo functions*. We, therefore, call it *query optimization by sub-plan memoization*.

Query optimization by sub-plan memoization solves several significant problems with traditional estimation techniques:

1. Since the result size and distinct value estimates are obtained by actually executing the sub-plans, estimation errors are completely eliminated.
2. The problem of error propagation is also solved. Our approach provides accurate result size and distinct value information for all sub-plans encountered during query optimization, regardless of the complexity of these sub-plans or the number of joins they contain. This information is as accurate for complex sub-plans involving multiple joins as it is for simple predicates on single tables.

3. The difficult problem of distinct value estimation is circumvented. The distinct value information found in the sub-plan memo is not based on estimates, but rather on actually executing the sub-plans.

The query execution plan chosen by sub-plan memoization is, therefore, a truly "optimal" plan with respect to result size and distinct value estimation. The obvious drawback of query optimization by sub-plan memoization is that to optimize a single query, we need to execute multiple sub-plans to determine their result sizes and the number of distinct values in their outputs. Thus, query optimization will take a long time, and potentially much longer than the execution time of the query being optimized. This makes query optimization by sub-plan memoization too expensive for many queries. Query optimization by sub-plan memoization is a suitable approach only for embedded queries that are optimized once and executed many times over. For this important class of queries, the potential for choosing more efficient query execution plans by using accurate result size and distinct value information makes the long query optimization times acceptable.

The rest of this report is organized as follows. In Section 2, we present an overview of related work. Section 3 describes query optimization by sub-plan memoization in detail. Section 4 presents an experimental evaluation of our approach based on a prototype implementation in Microsoft SQL Server 2000. Section 5 contains concluding remarks.

2 Related Work

Several techniques for estimating result sizes and distinct values have been proposed in the literature. One technique for estimating result sizes is sampling the data at query optimization time [LNS90]. The main disadvantage of sampling is the overhead it adds to query optimization. Furthermore, sampling cannot be used to accurately estimate the number of distinct values of an attribute [CMN98]. Sampling is more useful for other applications such as building histograms or approximate query processing.

Another technique for estimating result sizes is histograms. Histograms for database systems were introduced in [Koo80], and most commercial database systems now use them for result size estimation. Although one-dimensional equi-depth histograms are used in most systems, more accurate histograms have been proposed in [PIHS96]. In [PI97], the techniques of [PIHS96] are extended to multiple dimensions. A novel approach for building histograms based on wavelets is presented in [MVW98]. Efficient algorithms for constructing optimal histograms using dynamic programming, and for approximating these optimal histograms using heuristics, are presented in [JKM+98].

Histograms, by their nature, only capture an *approximation* of the data distribution, and they incur varying degrees of estimation errors. Our approach eliminates all estimation errors and provides the query optimizer with accurate values for the result sizes of sub-plans and the number of distinct values in their outputs.

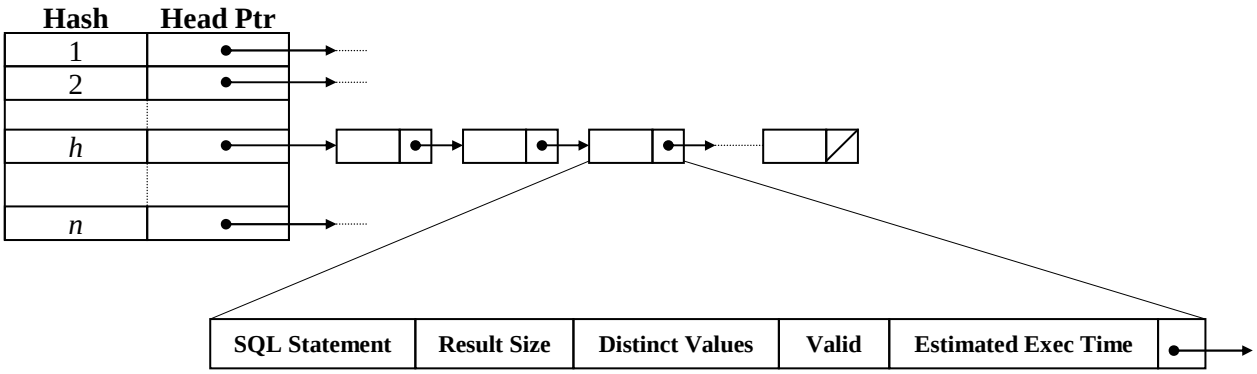

Figure 1: Structure of the sub-plan memo

Another approach to estimating result sizes is using feedback from the query execution engine, as in [CR94] and [AC99]. The techniques proposed in these papers exploit feedback information from the query execution engine to build accurate histograms with low overhead. These techniques are still approximate techniques that incur estimation errors.

The importance of estimating result sizes for query optimization is discussed in [ML86]. In this paper, it is noted that the cost model used by the R* System query optimizer for nested loop joins is very sensitive to the estimated result size of the join. Inaccurate result size estimation can lead to sub-optimal plans being chosen. This conclusion demonstrates the usefulness of our query optimization approach.

3 Query Optimization by Sub-plan Memoization

In this section, we describe the details of our proposed approach for query optimization. We describe the sub-plan memo data structure, and the algorithm for query optimization using this data structure.

Query optimization by sub-plan memoization requires recording the sub-plans encountered by the optimizer during query optimization, their result sizes, and the number of distinct values in their outputs. This information is stored in a hash table that we call the *sub-plan memo*. To record a sub-plan in the sub-plan memo, we construct a *SQL statement* that is logically equivalent to this sub-plan (i.e., a statement that produces the same set of result tuples as the sub-plan). We then hash this SQL statement (as a character string), and record it in the sub-plan memo. When the result size of this SQL statement and the number of distinct values in its output are determined, they are also recorded in its entry in the sub-plan memo. Figure 1 illustrates the structure of the sub-plan memo. It is a hash table in which each entry has five fields: a character string representing a *SQL statement* (the hash key), the *result size* of the SQL statement, the *number of distinct values* in the output of this statement, a flag indicating whether this hash table entry has valid values for the *result size* and *distinct values* fields, and the *estimated execution time* of the SQL statement as determined by the query optimizer using traditional result size and distinct value estimation techniques. This last field is used to avoid executing SQL statements that are prohibitively expensive. We now explain the role of each of these fields.

The sub-plan memo data structure is local to the query optimizer, and it is used for optimizing a single query. An initially empty sub-plan memo is created at the start of optimizing a query, and it is used until optimization completes. When an entry for a SQL statement is added to the sub-plan memo, its *valid* field is initially set to **false**. When the SQL statement is executed, and its result size and the number of distinct values in its output are determined, the corresponding fields in its entry in the sub-plan memo are updated, and the *valid* field is set to **true**.

Query optimization by sub-plan memoization is done in multiple *phases*. Each phase involves a full optimization of the query being optimized, starting with its SQL statement (or, equivalently, with the initial logical tree representing the SQL statement) and producing a query execution plan. In each phase, whenever the optimizer encounters a sub-plan for which it requires result size and/or distinct value estimates, it constructs a SQL statement that is logically equivalent to this sub-plan and looks it up in the sub-plan memo. If a valid entry for the SQL statement is found (i.e., an entry for which the *result size* and *distinct values* fields are valid), the optimizer uses the result size and distinct value information in this entry. If no entry for the SQL statement is found, the optimizer adds a new entry for this statement to the sub-plan memo, and it uses traditional estimation techniques such as histograms to obtain the required result size and distinct value estimates.

Each query optimization phase involves a complete search of the space of possible query execution plans. This search can encounter many sub-plans for which result size and distinct value estimates are required. At the end of each phase, we scan the sub-plan memo to find new entries that were added in this phase. These are the entries whose *valid* field is **false**. We execute the SQL statements in these entries and based on their execution, we determine their actual result sizes and the actual number of distinct values in their outputs. We record these values for each SQL statement that we execute in its entry in the sub-plan memo, and we set the *valid* field of this entry to **true**.

The SQL statements in the sub-plan memo are processed independently of the query being optimized. They are complete stand-alone queries that follow the code path followed by any query through the system. These queries must be optimized *without* using sub-plan memoization, since they are ad hoc queries for which traditional estimation techniques such as histograms work best.

After executing the SQL statements that were newly added to sub-plan memo and determining their result sizes and the number of distinct values in their outputs, we start a new query optimization phase in which the query is re-optimized. Since the query optimizer uses a deterministic algorithm to search the plan space, many of the sub-plans encountered in the new optimization phase will already be in the sub-plan memo from earlier phases, so their actual result sizes and the actual number of distinct values in their outputs will be available in the sub-plan memo, and these actual values will be used for cost estimation. However, since the new optimization phase uses more accurate result size and distinct value information than earlier phases, it may explore new parts of the plan space and encounter sub-plans that are not in the sub-plan memo. These new sub-plans are added to the sub-plan memo just as in earlier phases. We repeat the process of optimizing the query and executing new sub-plans until the optimizer goes through an optimization phase in which all encountered sub-plans are found in the sub-plan memo. The query

execution plan chosen in this last optimization phase is the output optimal plan returned by the optimizer. This plan is chosen based on completely accurate result size and distinct value information.

While searching the plan space during query optimization, the optimizer may require result size and/or distinct value estimates for sub-plans that are prohibitively expensive to execute. For example, the Microsoft SQL Server 2000 query optimizer sometimes requires result size information for the cross product of two relations. This cross product may appear in the final plan if the two relations are very small, but it is too expensive to compute if the relations are large. Since we do not place any restrictions on the plan space search, it is quite possible for the search to encounter such expensive sub-plans. These sub-plans are translated to SQL statements and added to the sub-plan memo. However, it would be prohibitively expensive to execute these statements to determine their actual result sizes and the number of distinct values in their outputs.

The SQL statements in the sub-plan memo are optimized (using traditional estimation techniques) before they are executed. To avoid executing prohibitively expensive SQL statements, we only execute a SQL statement from the sub-plan memo if its estimated execution time determined by the query optimizer is below a threshold, *t*. The threshold, *t*, can be an absolute number (e.g., 5 seconds). Alternatively, if the first query optimization phase – which uses only traditional estimation techniques since the sub-plan memo is still empty – estimates the execution time of the query being optimized to be *T*, we can set the threshold, *t*, to be a function of *T* (e.g., $t = 1.5*T$).

When the SQL statements in the sub-plan memo are optimized in preparation for execution, we record their estimated execution times in the *estimated execution time* fields of their sub-plan memo entries. This allows us to avoid unnecessarily re-optimizing SQL statements that are too expensive to execute at the end of later optimization phases. While scanning the sub-plan memo looking for newly added sub-plans, if an entry is invalid (i.e., its *valid* field is **false**) but has an estimated execution time above the threshold, *t*, the SQL statement in this entry is not considered for execution.

If a SQL statement in the sub-plan memo is considered too expensive to execute, its entry remains invalid throughout the query optimization process. The query optimizer, therefore, uses traditional estimation techniques for the sub-plan corresponding to this statement when it encounters this sub-plan in later optimization phases. This is not as accurate as using the exact information that is available for other sub-plans in the sub-plan memo. However, since this is an expensive sub-plan, it is highly unlikely to be part of the final query execution plan chosen by the optimizer. Thus, the lower estimation accuracy for this sub-plan is unlikely to result in choosing a sub-optimal query execution plan.

Query optimization by sub-plan memoization is an off-line process, so we are willing to tolerate long query optimization times. Nevertheless, it is still important to ensure that the time required to optimize a query does not become unacceptably large. Setting the threshold for estimated execution time, *t*, beyond which we do not allow SQL statements in the sub-plan memo to be executed is one way of controlling query optimization time. To further control query optimization time, we place an upper limit, *k*, on the number of optimization phases (e.g., $k = 4$). If after *k* phases, the optimizer is still searching new parts of

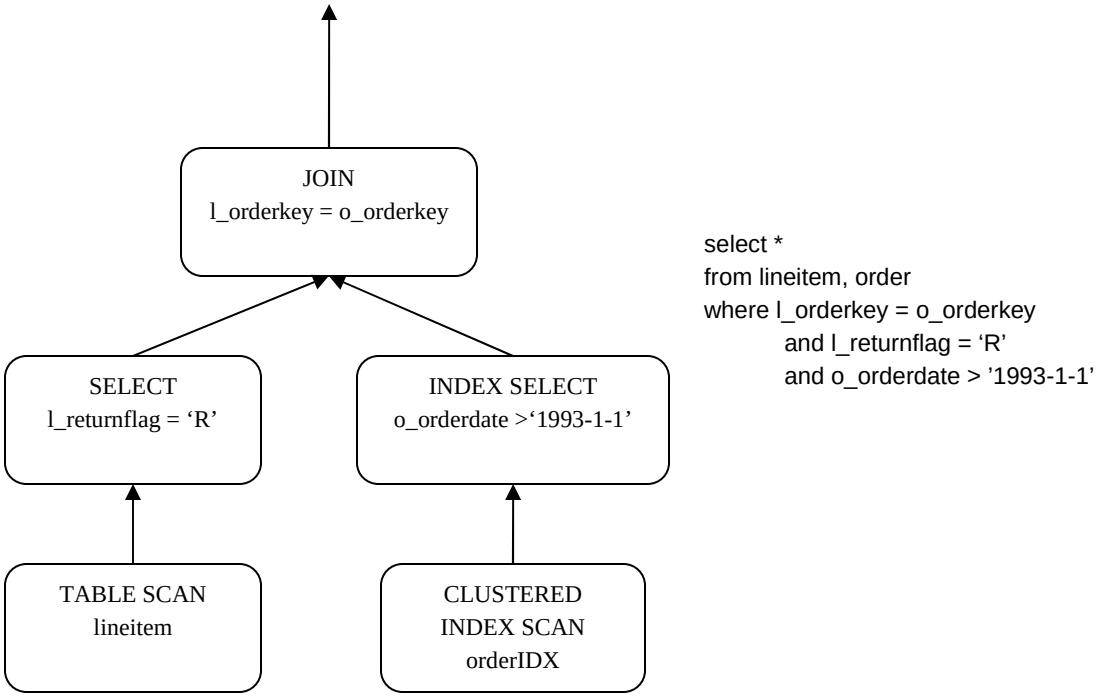

Figure 2: A sub-plan and the corresponding SQL statement

the plan space and adding new sub-plans to the sub-plan memo, we terminate the process and return the query execution plan chosen by the *k*th phase as the "optimal" plan.

Query optimization by sub-plan memoization requires being able to generate a SQL statement that is logically equivalent to any sub-plan for which result size or distinct value estimates are required. This SQL statement serves two purposes. It is executed to determine the actual result size of the sub-plan and the number of distinct values in its output. It is also used as a "key" to identify the sub-plan in later optimization phases, so that the accurate result size and distinct value information available for this sub-plan in the sub-plan memo can be used by the optimizer.

Generating a SQL statement that is logically equivalent to a sub-plan is a straightforward task. The logical and physical operators that can appear in a sub-plan have directly corresponding SQL constructs. As such, generating the SQL statement is simply a matter of traversing the sub-plan tree and creating the required SQL constructs. Creating these constructs may require consulting the catalog to determine attribute and table names. Figure 2 shows an example of a sub-plan and the corresponding SQL statement.

Although desirable, we do not require the generated SQL statements to be the "most compact" or the "most efficient" SQL. Executing these SQL statements does not affect the performance of the database system in production mode, since it is done only once as part of the off-line query optimization process.

```
ExecutionPlan OptimizeBySubPlanMemoization (Query q)
/* Input: Query to optimize by sub-plan memoization.
   Returns: Execution plan for q */
{
1   Create an empty sub-plan memo, spm;
2   numphases = 0;                  /* Counter for the number of optimization phases. */
3   do{
4       numphases++;
5       newsubplans = false;        /* Are new sub-plans encountered during this phase? */
6       execplan = Optimize (q, spm, newsubplans);
7       if (newsubplans){
8           Execute the new SQL statements in spm to get result size and distinct value information;
9       }
10  }while (newsubplans && numphases <= k);
11  return execplan;
}
```

Figure 3: Algorithm for query optimization by sub-plan memoization

Generating different SQL statements for sub-plans that are logically equivalent but are not identical (e.g., the outer and inner relations in a join are reversed) does not compromise the quality of the final plan. Not recognizing that these sub-plans are equivalent for the purpose of result size and distinct value estimation does not affect the quality of the optimizer decisions. It only means that the optimizer will do more work, since the sub-plan memo will contain logically equivalent SQL statements, and these statements will be independently executed giving identical results.

What we *do* require, though, is that exactly the same SQL statement be generated for a sub-plan each time it is encountered in the different phases of query optimization. The SQL statement corresponding to a sub-plan is the "key" that we use to identify this sub-plan. Generating the same SQL statement for a sub-plan each time it is encountered allows us to identify sub-plans in later optimization phases that were already encountered in earlier phases, which is a necessary condition for convergence.

Figure 3 presents the algorithm for query optimization by sub-plan memoization in C++-like pseudo-code. The input to the algorithm is a query expressed in the form used by the optimizer for initial queries. For example, it could be the SQL character string or an initial logical tree. The function *Optimize*, called in line 6 of the algorithm, is an invocation of the query optimizer to optimize query *q*. The optimizer is modified to use the sub-plan memo, *spm*, for estimation as outlined above, and to set the flag *newsubplans* to **true** if it adds any new entries to the sub-plan memo[1]. These modifications to the

[1] For example, *Optimize* can be declared as: ExecutionPlan Optimize (Query *q*, SubPlanMemo &*spm*, **bool** &*newsubplans*);

```
int ResultSizeEstimate (SubPlan s, SubPlanMemo &spm, bool &newsubplans)
/* Input: Sub-plan whose result size is required, and sub-plan memo to use for estimation.
   Returns: Result size estimate.  newsubplans is set to true if entries are added to spm. */
{
1   sql = SQL statement corresponding to s;
2   if (sql is successfully generated){
3       e = spm entry corresponding to sql;   /* Lookup SQL statement in hash table. */
4       if (An entry e is found){
5           if (e.valid){                     /* If entry e is valid, use it. */
6               return e.result_size;
7           }
8       }else{                                /* This is the first time to encounter sql. Add it to spm. */
9           Add a new (invalid) entry for sql to spm;
10          newsubplans = true;
11      }
12  }
13  res_size = Estimated result size of s using traditional estimation techniques;
14  return res_size;
}
```

Figure 4: Algorithm for result size estimation and updating the sub-plan memo if needed

optimizer require changing the result size and distinct value estimation routines. Figure 4 presents the modified result size estimation algorithm. The distinct value estimation algorithm is very similar. The modifications to the query optimizer for implementing sub-plan memoization are fairly simple and localized.

3.1 Discussion

- The sub-plans in the sub-plan memo can be simple (e.g., selections on single tables) or they can be complex (e.g., multiple selections and joins with aggregation). The result size and distinct value information provided for these sub-plans is perfectly accurate, regardless of their complexity. This is a significant improvement over traditional estimation techniques, which are generally inaccurate for complex sub-plans.
- Sub-plan memoization is not meant to replace traditional estimation techniques. Sub-plan memoization is an expensive technique that is only suitable for embedded queries that are optimized once and executed frequently. Traditional estimation techniques are the more suitable approach for ad hoc queries or queries that are not executed frequently. Furthermore, traditional estimation techniques are used in sub-plan memoization when the sub-plan is not found in the sub-plan memo or is too expensive to execute, or when the SQL statement corresponding to a sub-plan cannot be generated. Traditional estimation techniques are also used when optimizing

the SQL statements representing sub-plans to execute them to determine their result sizes and the number of distinct values in their outputs.

- The quality of the traditional estimation techniques used in query optimization by sub-plan memoization affects query optimization time. Having high quality traditional estimation techniques leads the optimizer to explore the correct part of the plan space even in the first optimization phase. The increased estimation accuracy in the second optimization phase and beyond will not cause the optimizer to generate too many new sub-plans, since it was in the right part of the plan space to begin with. This leads to fewer optimization phases and faster convergence (i.e., a shorter query optimization time).
- The modifications required to the query optimizer code to support sub-plan memoization are simple and localized. This makes it easy to incorporate sub-plan memoization into existing query optimizers.
- In our approach, we require a SQL statement corresponding to a given sub-plan to be generated. In some situations, this could be a limitation, e.g., there may not be sufficient information to identify an attribute name or the sub-plan may use a physical operator that has no corresponding SQL construct. In this case, the optimizer would use traditional estimation techniques for the sub-plan, which reintroduces estimation errors. Therefore, in our future work, we plan to modify the relational engine to develop techniques that do not rely on the generation of an explicit SQL statement.

4 Experimental Evaluation

We implemented a prototype of query optimization by sub-plan memoization in an internal version of Microsoft SQL Server 2000. In this section, we present the results of experiments using this prototype implementation and queries from the TPC-H benchmark [TPCH].

The experiments presented here use a 1GB TPC-H database. TPC-H data is uniformly distributed. Since traditional estimation techniques work best for uniformly distributed data, using TPC-H gives these techniques the best possible advantage when compared to sub-plan memoization.

Microsoft SQL Server 2000 uses sophisticated estimation techniques based on histograms and other database statistics. We built the relevant statistics for all single columns and all combinations of columns that take part in the queries. The statistics were built using full scans of the data, not samples, to provide the maximum possible accuracy. In our experiments, a rich set of clustered and non-clustered indexes on all tables is available for the optimizer to choose from.

Figures 5 and 6 show instances of Queries 5 and 10 of the TPC-H benchmark, respectively. Figure 7 shows the speedup in the query execution time of these queries resulting from using sub-plan memoization (speedup = execution time of plan chosen using traditional statistics / execution time of plan chosen using sub-plan memoization). The preciseness of sub-plan memoization helps the optimizer choose better query execution plans than using traditional statistics, even though the data is uniformly distributed and all the relevant statistics are available.

For TPC-H Query 5 (Figure 5), query optimization by sub-plan memoization required three optimization phases. The second optimization phase explored new parts of the plan space and generated new sub-plans, but the third phase did not. The join order and join methods chosen for this query using traditional statistics and using sub-plan memoization are shown in Figures 8(a) and (b), respectively. For this query, traditional statistics *underestimate* the result sizes of the sub-plans. Since the input sizes of the joins are inaccurately estimated to be small, the query optimizer chooses index nested loop join for all the joins in the query. Index nested loop join works well for joining small inputs. In reality, the input sizes of the joins are not small, so using index nested loop join is not optimal. When the query optimizer uses sub-plan memoization, it determines the true input and output sizes of the joins, so it reorders the joins and uses hash join and merge join instead of index nested loop join. Using accurate result size and distinct value information results in a six-fold speedup as seen in Figure 7.

For TPC-H Query 10 (Figure 6), query optimization by sub-plan memoization required two optimization phases. The second phase used the accurate result size and distinct value information determined at the end of the first phase, but it did not explore new parts of the plan space and did not generate any new sub-plans. The join order and methods chosen for this query using traditional statistics and using sub-plan memoization are shown in Figures 9(a) and (b), respectively. For this query, using sub-plan memoization causes the query optimizer to switch from using index nested loop join to using hash join for joining the "nation" and "customer" tables. The index nested loop join uses a non-clustered index to access the "customer" table, whereas the hash join uses the clustered index (i.e., a table scan). Using sub-plan memoization also causes the query optimizer to switch the role of the build and probe relations for all the hash joins. For this query, the speedup from using sub-plan memoization is 1.6 – not as large as for Query 5 but still quite significant.

5 Conclusions

Query optimization by sub-plan memoization is a novel technique for query optimization that allows the optimizer to estimate costs using *exact* values for the result sizes of sub-plans and the number of distinct values in their outputs. Thus, this is an example of exploiting feedback from query execution for improving optimization. The cost estimates obtained using these exact values are more accurate than the cost estimates obtained using traditional estimation techniques such as histograms, which helps the optimizer choose more efficient query execution plans. However, obtaining these exact values considerably increases query optimization time. This makes query optimization by sub-plan memoization unsuitable for arbitrary ad hoc queries, but appropriate for embedded queries in high-performance applications. Developing techniques for determining whether sub-plan memoization is better than traditional estimation techniques for a particular query in a particular application is still an open issue. We also would like to explore alternative techniques that further reduce the optimization time overhead for sub-plan memoization, e.g., by using memoization only for "attractive" sub-plans that have the highest uncertainty and the possibility of the greatest impact on the choice of the execution plan. Of course, determining a quantitative measure for identifying such attractive sub-plans is a difficult challenge.

References

[AC99] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *Proceedings of the ACM SIGMOD Conference*, pages 181–192, 1999.

[CMN98] S. Chaudhuri, R. Motwani, and V.R. Narasayya. Random sampling for histogram construction: how much is enough? In *Proceedings of the ACM SIGMOD Conference*, pages 436–447, 1998.

[CR94] C.M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the ACM SIGMOD Conference*, pages 161–172, 1994.

[IC91] Y.E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of the ACM SIGMOD Conference*, pages 268–277, 1991.

[JKM+98] H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K.C. Sevcik, T. Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 275–286, 1998.

[Koo80] R.P. Kooi. *The optimization of queries in relational databases*. PhD thesis, Case Western Reserve University, September 1980.

[LNS90] R.J. Lipton, J.F. Naughton, and D.A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the ACM SIGMOD Conference*, pages 1–11, 1990.

[ML86] L.F. Mackert and G.M. Lohman. R* optimizer validation and performance evaluation for local queries. In *Proceedings of the ACM SIGMOD Conference*, pages 84–95, 1986.

[MVW98] Y. Matias, J.S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the ACM SIGMOD Conference*, pages 448–459, 1998.

[PIHS96] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the ACM SIGMOD Conference*, pages 294–305, 1996.

[PI97] V. Poosala and Y.E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Databases*, pages 486–495, 1997.

[TPCH] The Transaction Processing Council. *TPC-H Benchmark Specification*. Available from http://www.tpc.org.

```
select   n_name,
         sum(l_extendedprice*(1-l_discount)) as revenue
from     customer,
         orders,
         lineitem,
         supplier,
         nation,
         region
where    c_custkey = o_custkey
         and o_orderkey = l_orderkey
         and l_suppkey = s_suppkey
         and c_nationkey = s_nationkey
         and s_nationkey = n_nationkey
         and n_regionkey = r_regionkey
         and r_name = 'ASIA'
         and o_orderdate >= '1/1/1994'
         and o_orderdate < dateadd(yy, 1, '1/1/1994')
group by  n_name
order by  revenue desc
```

Figure 5: An instance of TPC-H Query 5

```
select   c_custkey,
         c_name,
         sum(l_extendedprice * (1 - l_discount)) as revenue,
         c_acctbal,
         n_name,
         c_address,
         c_phone,
         c_comment
from     customer,
         orders,
         lineitem,
         nation
where    c_custkey = o_custkey
         and l_orderkey = o_orderkey
         and o_orderdate >= '1993-10-1'
         and o_orderdate < dateadd(mm, 3, '1993-10-1')
         and l_returnflag = 'R'
         and c_nationkey = n_nationkey
group by  c_custkey,
            c_name,
            c_acctbal,
            c_phone,
            n_name,
            c_address,
            c_comment
order by  revenue desc
```

Figure 6: An instance of TPC-H Query 10

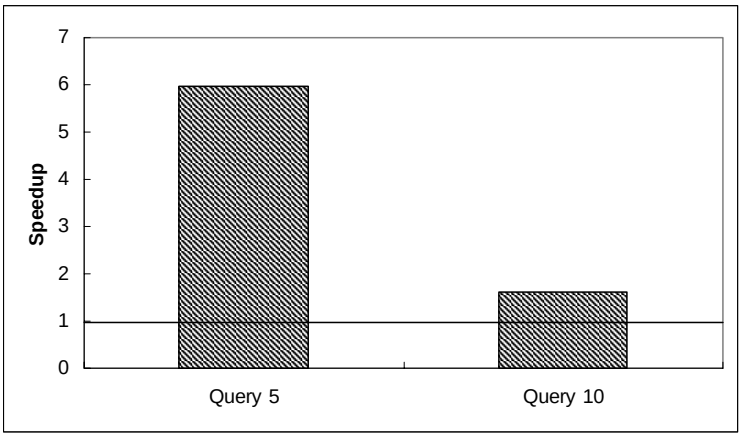

Figure 7: Speedup from using sub-plan memoization

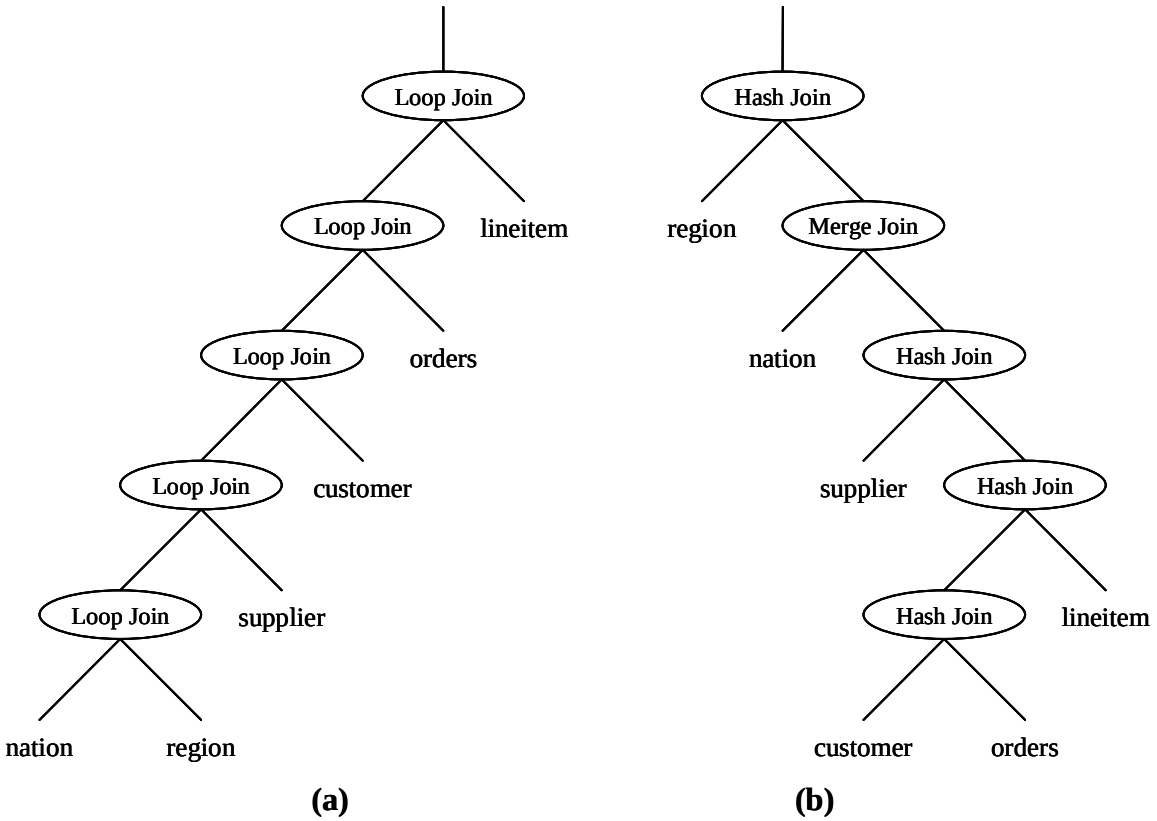

Figure 8: Plans chosen for Query 5 by (a) traditional statistics, and (b) sub-plan memoization

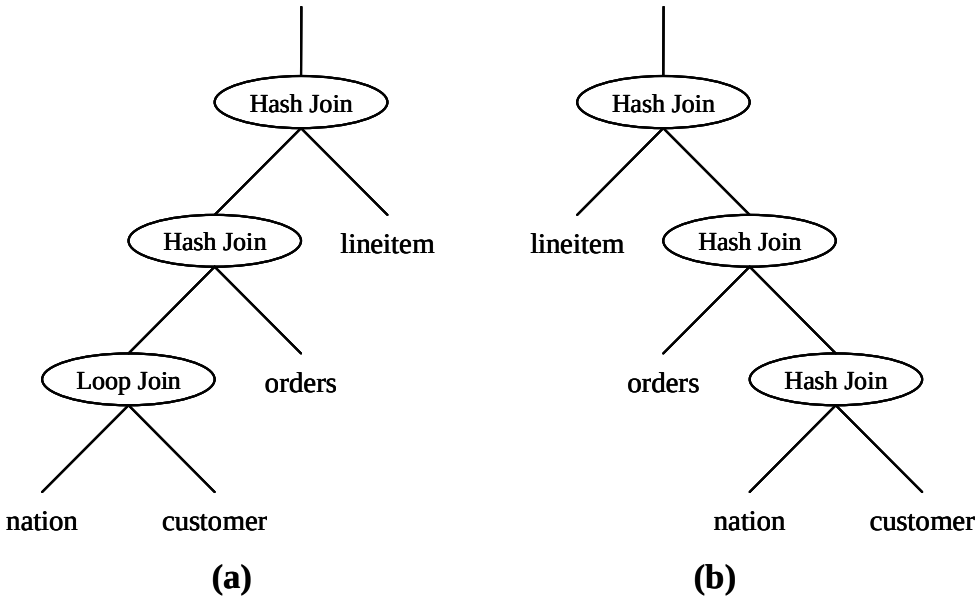

Figure 9: Plans chosen for Query 10 by (a) traditional statistics, and (b) sub-plan memoization