Using Similarity Distance for Performance Prediction of the Query Optimization Process

Anisoara Nica
SAP AG
Waterloo, Ontario, Canada
Anisoara.Nica@sap.com

Stephen Chou
Department of Electrical and Computer Engineering, University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

Query optimization is a sophisticated process whose resource consumption and quality of the best execution plan depends on the query complexity, available resources of the RDBMS server, and the current instance of the database. For a self-managing RDBMS such as SAP SQL Anywhere, the query optimizer must adapt each optimization process to the current resources as these servers are run in highly diverse environments with no DBA for tuning, running mixed workloads with little correlation between query complexity, its execution time, or database size. The main goal of the work presented in this paper is to be able to dynamically choose a join enumeration algorithm among the algorithms available to the query optimizer based on the known characteristics of the algorithms, the query complexity and its expensiveness, and the available resources for the optimization process. First, the paper discusses the experimental results of the optimization time breakdown and the memory consumption for a set of join enumeration algorithms ranging from highly heuristics algorithms to dynamic programming algorithms with exhaustive bushy trees enumeration. Next, we present a novel technique to predict the optimization time and memory consumption for a current query by using similarity distances between its query graph and a set of etalon queries with known performance. The new technique can be used by any query optimizer for choosing among join enumeration algorithms, or optimization levels, based on such characteristics as query complexity and available resources.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query Processing*; H.2.3 [**Database Management**]: Languages—*Query Languages*

Keywords

Similarity Measure, Query Graph, Etalon Query, Query Optimization, Enumeration Algorithm, SQL Anywhere, Sybase

DBTest 2013 June 24, 2013, New York, NY, U.S.A

1. INTRODUCTION AND MOTIVATION

SQL Anywhere[1][1] is a self-managing RDBMS with high reliability, high performance, synchronization capabilities, small footprint, and a full range of SQL features running on highly diverse platforms. The SQL Anywhere Optimizer [9, 10, 11, 12] optimizes each query at the open time, and it is designed to be highly adaptable to the query workload, system resources available to the optimization process, and the state of the database server. The goal of this work is to introduce a new level of adaptability to the optimization process to balance the optimization time and resource consumptions during optimization, on one side, with the complexity and expensiveness of the query, on the other side. In our experience, the complexity of the query is uncorrelated to its execution time, hence, the optimization process must behave differently for the same query optimized under different conditions, e.g., the optimization time cannot exceed the execution time. The join enumeration algorithm, used for a particular optimization, is the dominant factor in both the optimization time and the memory consumption, hence, providing a set of join enumeration algorithms to choose from is a way to provide this extra dimension to adaptability. First, we need to quantify particular subphases of the optimization process and understand which properties depend on the complexity of the query (i.e., its query graph) and which properties depend on the expensiveness of the query. As the query complexity depends only of its query graph, properties of the optimization process which depend only of its complexity can be predicted just by analyzing the query graph. Hence, a cache of properties of previously optimized queries can be used to predict these properties for a current query based only of its query graph.

The rest of the paper is organized as follows. In Section 2 we present the findings of how different join enumeration algorithms use system resources during optimization process. One goal of detailed measurement of the resource consumption is to understand which operations of the query optimization depend exclusively on the query topology, and which operations depend mostly on the current database state such as table sizes, column statistics, etc.

Section 3 introduces a novel technique based on using previously observed resource consumption, for different join enumeration algorithms, to estimate the resource consumption for a current query if a particular algorithm will be run. We use the detailed measurements for a set of known *etalon*

[1]SQL Anywhere are trademarks of SAP AG. Other company or product names referenced in this paper are trademarks and/or servicemarks of their respective companies.

queries[2] to estimate performance of the join enumeration algorithms for current queries. By contrast, previous work on optimization time estimation is based on enumerating some part of the search space, before the actual optimization [4, 13]. We describe the *etalon similarity measure* and present the experimental results of applying this technique to a benchmark of random queries.

2. ON RESOURCE CONSUMPTION OF JOIN ENUMERATION ALGORITHMS

The SQL Anywhere Optimizer has a set of join enumeration algorithms with highly diverse properties related to what search space is generated by each algorithm, how logical enumeration, physical plan generation, and cost-based and logical pruning are implemented. These properties determine the resource consumption as well as the quality of the best plan generated by each algorithm. All algorithms share the same internal representation of a query (normalized join operator trees [10]), data structures such as the memoization table, special memory heaps, the cost model, etc. Because of these common characteristics, this implementation is ideal to analyze and compare the algorithms' resource consumption. The optimization process starts by obtaining an initial upper bound cost (i.e., the query's *expensiveness*) using the cheap *backtracking* algorithm - the upper bound cost is used for choosing a join enumeration algorithm, and for global cost-based pruning during join enumeration. In SQL Anywhere, statistics about the optimization process can be logged and include optimization time breakdown and memory usage for particular subphases[3].

The properties of the algorithms which determine the optimization time are described in Table 1, while experimental results of actual optimization time breakdown, memory consumption, and estimated runtime cost are shown in Figures 1 and 2. For optimization time breakdown, we measured time spent for operations related to the logical enumeration phase (enumerating logical partitions, memoization table management), the plan generation phase (building physical plans, costing, pruning, saving the best plans in the memoization table), miscellaneous times such as obtaining the initial upper bound, initializing the optimizer structures, partition management for algorithms such as *ordered-DPhyp* and *parallel-ordered-DPhyp*, cleanup time. A large body of work exists for optimizing the logical enumeration phase when exhaustive bushy three enumeration is performed [2, 7, 3]. However, as found by [4] and our own experiments, enumeration time is not the main component of the optimization time, but is only a small fraction, in the range of $5\% - 25\%$. Moreover, the plan generation phase (including costing and pruning of physical plans) is by far the most expensive part of the optimization process. In the past, we experimented with new algorithms trying to reduce as much as possible the plan generation time: *ordered-DPhyp* reduces this time by costing fewer partitions, while *parallel-ordered-DPhyp* reduces it to almost 0, by parallelizing the plan generation phase [9]. As expected, these improvements added new times for partition management, as well as extra memory consumption for *parallel-ordered-DPhyp* (Figure 1).

[2]From French, *etalon* means standard, something used as a measure for comparative evaluations.

[3]Visualization and comparison of logged data can be done using two research prototypes [11, 12].

3. RESOURCE ESTIMATION FOR A QUERY OPTIMIZATION PROCESS

In any query optimizer using dynamic programming-based algorithms the memory consumption and the runtime for logical enumeration depend on the number of connected subgraphs - denoted here as *csgs* - and the number of enumerated logical partitions - denoted as *csg-cmp-pairs* . Both measures *csgs* and *csg-cmp-pairs* depend only of the query graph, i.e., the query complexity, hence, any resource consumption directly related to these two measures will be similar for similar query graphs, regardless of other characteristics such as the query expensiveness. The problem of finding the number of connected subgraphs *csgs* and the number of *csg-cmp-pairs* for a random query graph is intractable. In contrast with previous work [4, 13], we propose here a new approach in estimating the memory and CPU consumption of a query optimization process based on properties of a set of $\mathcal{E}$ etalon queries for which the performance was previously observed. The only information needed for each etalon query is its degree sequence used for calculating etalon similarity measure, and a set of statistics for each join enumeration algorithm, such as CPU and memory consumption, obtained when an etalon query was optimized by each algorithm.

The set of etalon queries can be built, for example, during a calibration process when each join enumeration algorithm is run for each etalon query and optimization statistics related to resource consumption are saved in a catalog table. Note that the properties of the etalon queries depend on the implementation of the query optimizer, the machine the server is running on, etc., hence these properties must be recomputed for a specific server installation. Moreover, during the usual RDBMS operations, new queries can be added to the set of etalon queries $\mathcal{E}$ after it is optimized. If, for a current query Q, we can determine that its query graph is 'close' to an etalon query $e_Q \in \mathcal{E}$, then we can predict that Q, when optimized by a certain algorithm, will have the resource consumption similar to the optimization statistics of e_Q. This estimation process can be used to choose among the join enumeration algorithms based on the resources available to the query optimizer given the current state of the RDBMS server: e.g., if the server is very busy with concurrent connections and it is low in available memory, then we cannot use a parallel join enumeration algorithm or a join enumeration algorithm requiring more memory than the available amount. This technique can be used by any database system using dynamic programming-based algorithms for join enumeration, with the calibration process computing the properties specific to that server.

In the following we present a method for determining if two query graphs are 'close' based on the similarity measure described in [5]. A similarity measure between two graphs is generally used to establish upper bounds for graph invariants: two graphs share a certain property if the corresponding similarity measure between them is high.

The similarity measure studied in [5] provides an upper bound of the size of the maximum common edge subgraph (MCES) between two graphs. The size of MCES is strongly connected with the number of *csgs* and the number of *csg-cmp-pairs* , hence, as our tests show, graphs with large etalon similarity measure have similar numbers for *csgs* and *csg-cmp-pairs* . As the formula for computing the similarity measure is only dependent of the degree vectors, the time to

Terminology: *LEP = Logical Enumeration Phase*, *PGP = Plan Generation Phase*
csgs = the number of connected subgraphs, *csg-cmp-pairs = the number of logical partitions*

– *Exhaustive Bushy Trees Enumeration* algorithms include *DPhyp*, *TopDownBranch*, *TopDown*, *ordered-DPhyp*,and *parallel-ordered-DPhyp*.These algorithms enumerate all *csg-cmp-pairs* and use a memoization table where each connected subgraph has an entry. *ordered-DPhyp* optimizes the time spent in *PGP* by reducing the number of enumerated *csg-cmp-pairs* for which physical plans are generated. *parallel-ordered-DPhyp* reduces the *PGP* time nearly to 0 by executing in parallel all physical plan generation. *parallel-ordered-DPhyp* uses the fact that algorithms such as *DPhyp*, *TopDownBranch*, *TopDown* enumerate a *csg-cmp-pair* (S_1, S_2) if an only if both S_1 and S_2 connected subgraphs have already enumerated all their partitions, hence the *PGP* phase can be applied to both S_1 and S_2 in parallel with the enumeration thread. The algorithm uses mutex-free memoization table. The main drawback of the current implementation is the increased memory usage of the *parallel-ordered-DPhyp* as more memory is used for each memoization table entry for parallel management. The memory usage comparison for cycle queries of size 17 is shown in Figure 1. The breakdown of the optimization time for these algorithms are depicted in Figures 1 and 4, with *PGP* time highly reduced for *ordered-DPhyp* and close to 0 for *parallel-ordered-DPhyp*.
– *Highly Heuristic* algorithms which enumerate bushy trees are *simplified-DPhyp* and *MinCutHyp*. Both enumerate, in *LEP* only a small subspace of the whole bushy trees space. Hence, both *LEP* time and *PGP* time can potentially be highly reduced. However, the quality of the best execution plan found by these algorithms can be very low: Figure 1 depicts examples of cycle queries of size 17 which have the estimated time for the best plans found by these two algorithms 1.5 to 3 times larger than the plans found by *ordered-DPhyp* for example.
– *Highly Heuristic* algorithms which enumerate only left-deep trees are *backtracking* and *backtrackingM*.Both algorithms employ heuristics for pruning whole subspaces when a subspace proves to be uninteresting. However, similar to *simplified-DPhyp* and *MinCutHyp*, the quality of the best plans can be very low. Figure 1 shows the estimated cost of the best plans found by these two algorithms up to 6 times larger than the best plans found by other algorithms.

	DPhyp [7] *TopDownBranch* [3] *TopDown*[2]	*ordered-DPhyp*[9]	*parallel-ordered-DPhyp*[9]	*simplified-DPhyp* [8]	*MinCutHyp* [6]	*backtracking backtrackingM* [11]
Logical Search Space	all bushy trees	all bushy trees	all bushy trees	some bushy trees	some bushy trees	some left-deep trees
Logical Enumeration Phase (LEP)	interleaved with *PGP*	prior to *PGP*	parallel with *PGP*	interleaved with *PGP*	interleaved with *PGP*	no logical plans are enumerated
Logical Partition Pruning	none - *PGP* is applied to all *csg-cmp-pairs*	*PGP* is applied to a subset of all *csg-cmp-pairs*	*PGP* is applied to a subset of all *csg-cmp-pairs*	*PGP* is applied to the enumerated *csg-cmp-pairs*	*PGP* is applied to the enumerated *csg-cmp-pairs*	only *PGP*: physical plans are directly enumerated
Plan Generation Phase (PGP)	interleaved with *LEP*	follows *LEP*	parallel with *LEP*	interleaved with *LEP*	interleaved with *LEP*	only *PGP*
Cost-based Pruning	global and local cost-based pruning	global and local cost-based pruning	global and local cost-based pruning	global and local cost-based pruning	global and local cost-based pruning	global cost-based pruning
LEP Optimization Time	no optimization	no optimization	no optimization	highly reduced due to fewer enumerated *csg-cmp-pairs*	highly reduced due to fewer enumerated *csg-cmp-pairs*	no *LEP*
PGP Optimization Time	no optimization	highly reduced: *PGP* is done on a subset of enumerated *csg-cmp-pairs*	almost 0: *PGP* in parallel with *LEP*	highly reduced due to fewer enumerated *csg-cmp-pairs*	highly reduced due to fewer enumerated *csg-cmp-pairs*	highly reduced due to only left-deep trees
Comments	exhaustive bottom-up/top-down enumeration; first best plan is found very late;	exhaustive bottom-up enumeration; reduced *PGP* time	exhaustive bottom-up enumeration; *LEP* in parallel with *PGP*; highly reduced *PGP* time	simplification of the query graph before *LEP*; highly heuristic	top-down enumeration using the hypergraph min-cut algorithm; highly heuristic	left-deep trees only; reduced *PGP* time; local cost-based pruning with memoization table; highly heuristic

Table 1: Properties of the Join Enumeration Algorithms

compute these similarities is very small, e.g., less than 100ms for 1000 etalon queries. Moreover, the similarity distances are computed only for queries estimated to be expensive as discussed in Section 2.

Let $Q = (V^Q, E^Q)$ a query graph with $|V^Q| = N$ tables, $V^Q = \{v_1^Q, \ldots, v_N^Q\}$. The degree $d(v)$ of the vertex v is the number of vertices (tables) connected to v. The vertices in the set V^Q are sorted in non-increasing order of degree, i.e., $d(v_k^Q) \geq d(v_{k+1}^Q)$, for all $1 \leq k < N$: $(d(v_1^Q), \ldots, d(v_N^Q))$ is called the *degree sequence of the query* Q.

The $esim(Q, X)$ *etalon similarity measure* between $Q = (V^Q, E^Q)$ and an etalon query X, represented only by its degree sequence $(d(v_1^X), \ldots, d(v_N^X))$, is:

$$E(Q, X) = \lfloor \tfrac{1}{2} \times \textstyle\sum_{i=1,N}(min(d(v_i^Q), d(v_i^X))) \rfloor$$

$$esim(Q, X) = \frac{(N+E(Q,X))^2}{(N+|E^Q|)(N+|\frac{1}{2}\times\sum_{1\leq j\leq N} d(v_i^X)|)}$$

For a random query $Q = (V^Q, E^Q)$ with $|V^Q| = N$, its etalon query e_Q is chosen, from the etalon queries of size N: $esim(Q, e_Q) = max_{\{Y|Y\in\mathcal{E},|V^Y|=N\}} esim(Q, Y)$.
If $esim(Q, e_Q)$ is sufficiently large, the number of the connected subgraphs *csgs* and the number of *csg-cmp-pairs* of the query graph Q are closely related to e_Q, i.e., *csgs* $(Q) \sim$ *csgs* (e_Q) and *csg-cmp-pairs* $(Q) \sim$ *csg-cmp-pairs* (e_Q).

Figure 2 shows the actual and estimated numbers of *csgs* and *csg-cmp-pairs* for 60 randomly generated queries, and also the actual optimization times for the queries 18-24 and five etalon queries: chain, cycle, star, grid, and clique. The graphs of the queries are generated randomly for given numbers of vertices and edges. For each query Q, its corresponding x-axis label is of the form $Q(N, |E^Q|)\ e_Q(N)(esim(Q, e_Q))$ depicting its etalon query e_Q. The lines connecting two actual optimization time bars, in the bottom chart, link a query Q with its etalon query e_Q chosen as described above. The actual optimization times in Figure 2 are the times obtained using *DPhyp* algorithm, but all other algorithms which exhaustively enumerate bushy trees - *ordered-DPhyp*, *parallel-DPhyp*, *TopDown*, *TopDownBranch* - have the optimization time of the etalon query e_Q very close to the optimization time of Q for that particular algorithm. The highly heuristic algorithms such as *backtracking*, or *simplified-DPhyp* don't exhibit the same behavior as their enumeration time is not correlated to the number of *csgs* and *csg-cmp-pairs* .

4. CONCLUSION

In this paper, we present the experimental results of how different join enumeration algorithms use system resources during optimization process. All the studied join enumeration algorithms are implemented in the SQL Anywhere Optimizer, sharing the same internal representation of a query, and data structures, hence their performance can be analyzed and compared. Our findings revealed that enumeration time is only a small fraction of the total optimization time, in the range of $5\% - 25\%$, while the plan generation phase (including costing, pruning of physical plans, saving physical best plans) is by far the most expensive part of the optimization process. Secondly, we introduce a novel technique based on using previously observed resource consumption to estimate the resource consumption for a current query. The paper describes the *etalon similarity measure* between a query and the *etalon queries* and presents the experimental results of applying this technique to a benchmark of random queries. The results are very encouraging as the etalon query with the highest etalon similarity measure has similar optimization time with the random query for algorithms with exhaustive bushy trees enumeration. These techniques can be used by any query optimizer using dynamic programming-based algorithms for join enumeration.

5. REFERENCES

[1] M. Abouzour, I. T. Bowman, P. Bumbulis, D. E. DeHaan, A. K. Goel, A. Nica, G. N. Paulley, and J. Smirnios. Database self-management: Taming the monster. IEEE *Data Engineering Bulletin*, 34(4):3–11, 2011.

[2] D. DeHaan and F. W. Tompa. Optimal top-down join enumeration. In ACM SIGMOD *International Conference on Management of Data*, pages 785–796, Beijing, China, June 2007.

[3] P. Fender and G. Moerkotte. A new, highly efficient, and easy to implement top-down join enumeration algorithm. In *Proceedings,* IEEE *International Conference on Data Engineering*, pages 864–875, 2011.

[4] I. F. Ilyas, J. Rao, G. M. Lohman, D. Gao, and E. T. Lin. Estimating compilation time of a query optimizer. In ACM SIGMOD *International Conference on Management of Data*, pages 373–384, June 2003.

[5] M. Johnson. Relating metrics, lines and variables defined on graphs to problems in medicinal chemistry. In *Graph Theory and Its Applications to Algorithms and Computer Science*, New York, 1985. Wiley.

[6] R. Klimmek and F. Wagner. A simple hypergraph min cut algorithm. Technical Report B 96-02, Bericht FU Berlin Fachbereich Mathematik und Informatik, 1996.

[7] G. Moerkotte and T. Neumann. Dynamic programming strikes back. In ACM SIGMOD *International Conference on Management of Data*, pages 539–552, 2008.

[8] T. Neumann. Query simplification: graceful degradation for join-order optimization. In ACM SIGMOD *International Conference on Management of Data*, pages 403–414, 2009.

[9] A. Nica. A call for order in search space generation process of query optimization. In *Proceedings,* IEEE ICDE *Workshops (Self-Managing Database Systems SMDB)*, Apr. 2011.

[10] A. Nica. Incremental maintenance of materialized views with outerjoins. *Information Systems*, 37(5), 2012.

[11] A. Nica, D. S. Brotherston, and D. W. Hillis. Extreme visualisation of the query optimizer search spaces. In ACM SIGMOD *International Conference on Management of Data*, pages 1067–1070, June 2009.

[12] A. Nica, I. Charlesworth, and M. Panju. Analyzing query optimization process: Portraits of join enumeration algorithms. In *Proceedings, 28th* IEEE ICDE *International Conference on Data Engineering.* IEEE Computer Society Press, Apr. 2012.

[13] K. Ono and G. M. Lohman. Measuring the complexity of join enumeration in query optimization. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 314–325, Brisbane, Australia, Aug. 1990. Morgan Kaufmann.

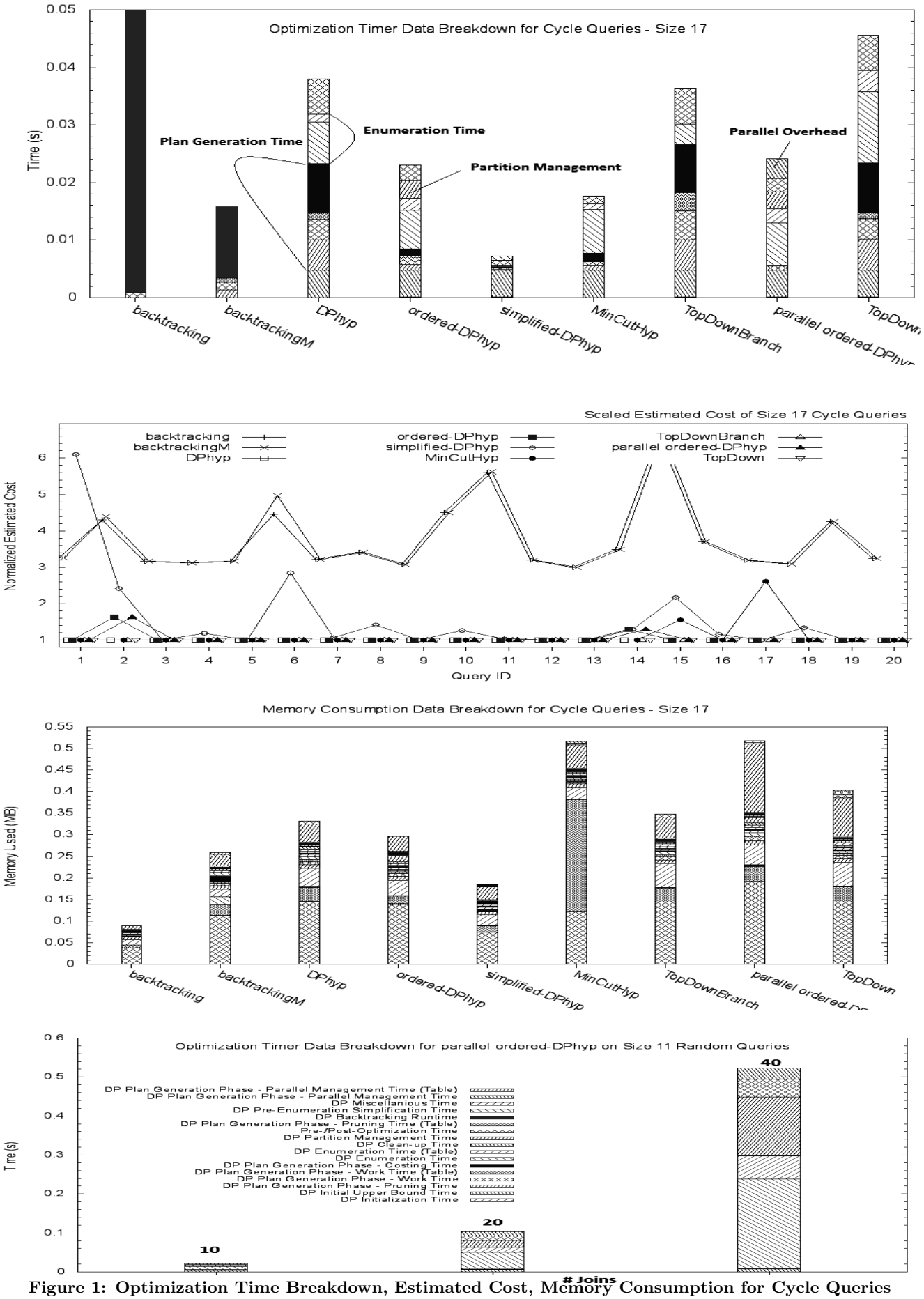

Figure 1: Optimization Time Breakdown, Estimated Cost, Memory Consumption for Cycle Queries

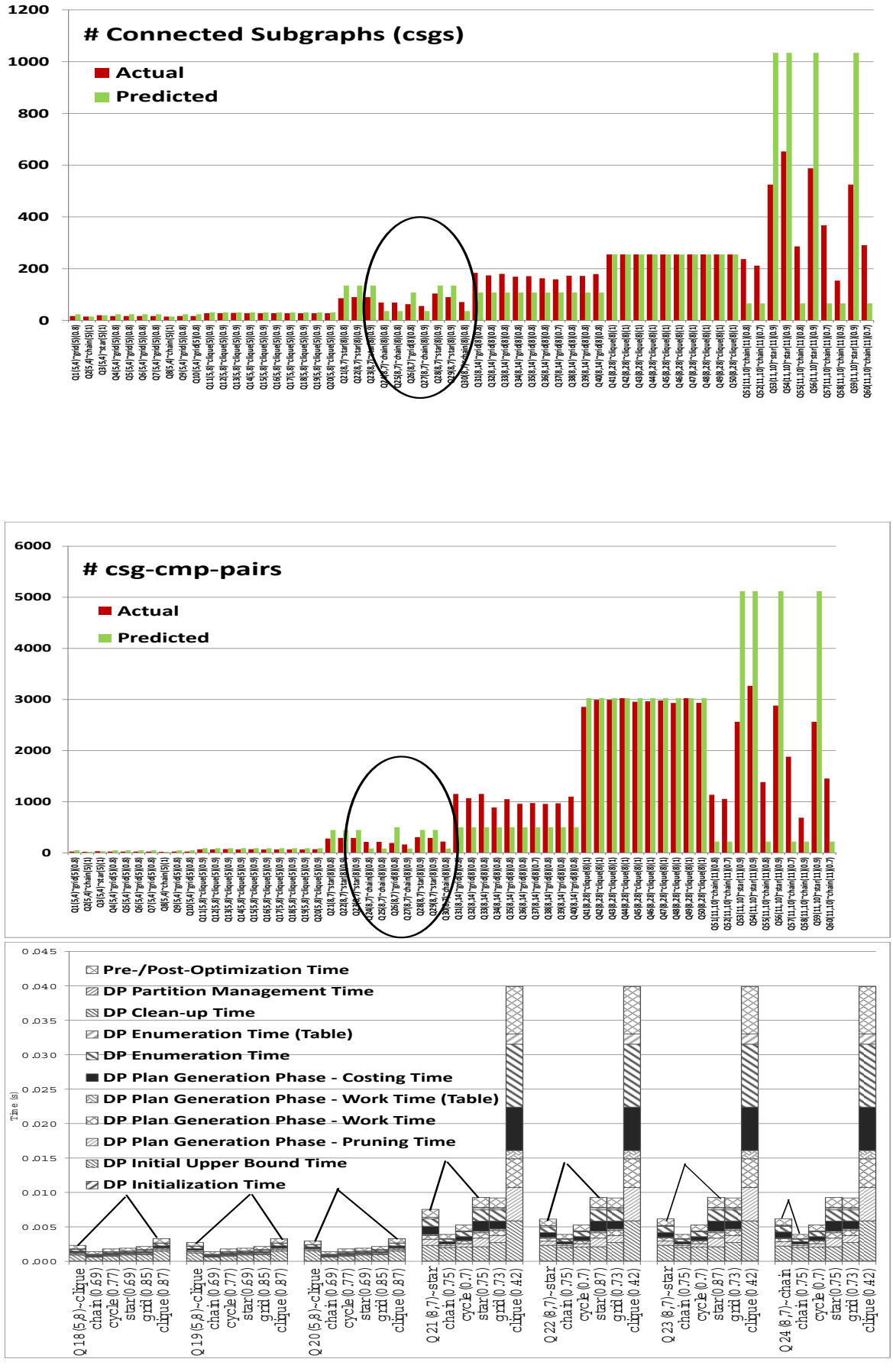

Figure 2: Similarity Measure: estimated and actual *csgs* , *csg-cmp-pairs* , and actual optimization times