Technical Perspective: Output-Optimal Evaluation of Conjunctive Queries

Dan Suciu
Universtity of Washington
suciu@cs.washington.edu

The best asymptotic runtime that we can hope for a query evaluation algorithm is $O(N + \texttt{OUT})$, where N is the size of the input database and OUT is the size of the query's output. Indeed, any algorithm must read the input, and must write the output, hence one cannot do better asymptotically. It is not immediately obvious how to evaluate a query in this time. The standard approach taken by all query engines is to compute one join at a time, but in that case the intermediate results may become much larger than both the input and the final output. Yannakakis' algorithm from the early 80's achieves a runtime of $O(N+\texttt{OUT})$ on all full, acyclic queries, by first removing all dangling tuples from all relations using a sequence of semijoins, then performing the joins.

The vast majority of SQL queries found in real workloads are acyclic, so the restriction of Yannakakis' algorithm to acyclic queries is not an important limitation. However, the restriction to *full* queries means that its runtime can only be guaranteed for `select * from ...` queries. In particular, Yannakakis' algorithm does not apply to queries with a `group by` clause, which form the backbone of day to day data analytics.

Most query engines compute a query with a `group by` clause by first performing all joins, then applying the group-by-aggregate operator at the end. In other words, they compute the full join, and compute the `group by` at the end. But the size of the full join is often much larger than the number of groups in the final output, which means that the runtime can be arbitrarily larger than both the input and the output.

A few query engines support an alternative approach: they push the group-by-aggregate operator down the query plan, thus aiming to reduce the size of the intermediate results. But even if one removes all dangling tuples before performing the joins, pushing group-by-aggregate down the query plan can only guarantee a runtime of $O\left(N \cdot \texttt{OUT}\right)$, which is a sharp increase over our desired $O(N + \texttt{OUT})$. The only exception are so called *acyclic, free-connex* queries. These are queries that satisfy a stringent syntactic restriction, where all `group by` variables form a connected component in the join tree. The only `group by` queries that can be evaluated in time $O(N+\texttt{OUT})$ are the acyclic, free-connex queries: these can be computed with this runtime by pushing the group-by-aggregate operators down the join tree.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2008 ACM 0001-0782/08/0X00 ...$5.00.

The following paper, by Hu et al., describes an algorithm that reduces the complexity of query evaluation for any acyclic query to $O\left(N + N \cdot \texttt{OUT}^{1-\varepsilon} + \texttt{OUT}\right)$, where $0 < \varepsilon \leq 1$ is a query-dependent constant. The running example in the paper is a chain query with two output variables (group by variables), where this constant is 2, and therefor the runtime is $O\left(N + N \cdot \texttt{OUT}^{1/2} + \texttt{OUT}\right)$. This runtime is better than $O\left(N \cdot \texttt{OUT}\right)$, without reaching the ideal $O(N + \texttt{OUT})$. When the query happens to be free-connex, then $\varepsilon = 1$ and the algorithm has the same runtime as Yannakakis' algorithm.

The paper proves, in Theorem 4, that this algorithm is optimal. The statement of the theorem is somewhat technical: it proves that the circuit needed to represent the query's output has size $\Omega\left(N \cdot \texttt{OUT}^{1-\varepsilon}\right)$, where ε is the same as in the algorithm. This implies that the algorithm presented in the paper has the best possible runtime among all "combinatorial" algorithms, understood here as those algorithm whose trace is a circuit (this precludes, e.g., algorithms using fast matrix multiplication).

So how does one compute an acyclic query in this optimal runtime? The algorithm is not straightforward, but the paper introduces it gently, by illustrating it on a chain query. At its core, the algorithm is based on partitioning each input relation into *light* and *heavy* tuples, then computing the query separately on the various combinations of light and heavy relations. This technique has been applied in the algorithm literature to various graph pattern matching algorithms, and it is also popular in distributed systems (for example *SkewJoin* hash-partitions the light elements, and broadcasts the heavy elements). One difference in the current paper is that the definition of light and heavy tuples is based on the size of the output, which needs to be computed (approximatively) in a separate step.

Finally, is this new algorithm ready for production? Not yet. Research, at its best, evolves from theory to practical implementation. Perhaps the best illustration of such an evolution is Yannakakis' algorithm itself, which was introduced in 1981, but whose practical adaptation has been investigated only recently. This paper makes a major first step in generalizing Yannakakis' algorithm to all queries with `group-by`, and solves this problem completely and definitively, by providing theoretical matching upper and lower bounds for the query evaluation problem. More, and significant research is still needed to incorporate these ideas into today's highly optimized query engines.