Technical Perspective: Conjunctive Queries with Comparisons

Stijn Vansummeren
UHasselt, Data Science Institute
stijn.vansummeren@uhasselt.be

Query processing, the art of efficiently executing a relational query on a given database, is a foundational and core area in data management research. Established at the dawn of relational database systems in the 1970's, relational query processing remains a highly relevant and vibrant research topic today as recent work shows that, apart from its application in traditional database scenarios, it is also highly effective in optimizing machine learning workloads [1].

Join ordering is a crucial part of efficiently processing relational queries. In the join ordering problem, we are given a join query like

$$Q_1 = R(A, B) \bowtie S(B, C) \bowtie T(C, D),$$

and we need to determine the order in which we will execute this join. It could be, for example that the subquery $R \bowtie S$ yields a very large subresult compared to the full join output, while $S \bowtie T$ is very selective. In that case, we should first execute $S \bowtie T$, and then $R \bowtie (S \bowtie T)$ to avoid wastefully computing the unnecessary tuples in $R \bowtie S$ that do not join with any tuple in T. Most practical systems use statistics of the input relations to estimate the sizes of subresults, and reorder joins based on these estimates. Since size estimates are difficult to always get correct, however, we are not guaranteed to always obtain an optimal join order.

A seminal result by Yannakakis [2] states that instead of looking at data statistics we may obtain optimal join orders by looking at query structure. In particular, because the hypergraph of our query Q_1 above is *α-acyclic* [2] it can be executed in instance-optimal time $\widetilde{\mathcal{O}}(\mathrm{IN} + \mathrm{OUT})$ *on any database*, where IN denotes the size of the input database; OUT denotes the size of the query output; and the notation $\widetilde{\mathcal{O}}$ suppresses a $\log^{\mathcal{O}(1)}$ factor. The core insight required to obtain this result is that for α-acyclic join queries we may remove so-called "dangling tuples" from the input in $\widetilde{\mathcal{O}}(\mathrm{IN})$ time, after which any reasonable join order allows to compute Q_1 in $\widetilde{\mathcal{O}}(\mathrm{OUT})$ time. Here, an input tuple is dangling if it does not appear as part of a tuple in the output.

The paper *"Conjunctive Queries with Comparisons"* by Wang and Yi generalizes Yannakakis' result from equi-join queries to theta-joins involving comparisons. Join queries with comparisons naturally appear in OLAP and spatio-temporal queries, but also in machine learning over relational data. Consider some simple examples.

$$Q_2 = R(A, B) \bowtie S(B, C) \bowtie T(C, D) \text{ s.t. } A \le C$$
$$Q_3 = R(A, B) \bowtie S(B, C) \bowtie T(C, D) \text{ s.t. } A \le D$$

The comparison $A \le C$ in Q_2 is called a *short* comparison because it can be applied after joining two relations, while $A \le D$ in Q_3 is a *long* comparison because more than two relations need to be joined before it can be applied. Long comparisons are particularly important in the context of machine learning over relational data. Note that processing queries like Q_2 and Q_3 by first computing the equi-join between R, S, T and then filtering afterwards yields a complexity of $\widetilde{\mathcal{O}}(\mathrm{IN} + \mathrm{EJ})$ where EJ is the size of the equi-join result, which may be significantly larger than OUT.

Wang and Yi generalize Yannakakis' result to join queries containing multiple comparisons, both short and long. They require that the query is α-acyclic in terms of equi-joins (cf. Yannakakis) while the comparisons must be Berge-acyclic. Using this structure, they propose a novel way for removing dangling tuples in the presence of comparisons in $\widetilde{\mathcal{O}}(\mathrm{IN})$ time. This non-trivial insight hinges on grouping the data of input relations on equi-join attributes; computing minimum or maximum values of the comparison attributes per group; propagating the minima/maxima to other relations; and filtering the dangling tuples using orthogonal range searching. Once dangling tuples are removed, the minima/maxima are again used to enumerate the output tuples *with constant delay*, which implies $\widetilde{\mathcal{O}}(\mathrm{OUT})$ total time for output construction. The total time is hence $\widetilde{\mathcal{O}}(\mathrm{IN} + \mathrm{OUT})$.

While we have restricted ourselves in our discussion so far to join queries, i.e., queries without projections, Wang and Yi also discuss join queries with projections (a.k.a. conjunctive queries) and they show how queries that are not acyclic according to their definition can be transformed, based on generalized hypertree decompositions, into acyclic queries. Furthermore, they empirically show that while the complexity analysis above focuses on asymptotic complexity, and may therefore hide large constant, the algorithm is very efficient in practice.

In short, this is an exemplary paper that proposes foundational innovation for query processing on the highly relevant class of conjunctive queries with comparisons.

References

[1] D. Olteanu. The Relational Data Borg is Learning. *Proc. VLDB Endow.*, 13(12):3502–3515, 2020.
[2] M. Yannakakis. Algorithms for Acyclic Database Schemes. In *Very Large Data Bases*, pages 82–94. IEEE, 1981.