Technical Perspective: DPconv: Super-Polynomially Faster Join Ordering

Guido Moerkotte
University of Mannheim

On the floors of database conferences, one can often listen to database researchers talking about the join ordering problem. However, *the* join ordering problem does not exist! In fact, depending on the properties of the query graph, the cardinality estimation method, the cost function, and the resulting output join tree, there exist multiple join ordering problems. For example, if the query graph is connected and acyclic, the cardinality estimator uses the independence assumption, the cost function has the ASI property, and the resulting join tree is a left-deep tree without cross products, then the problem can be solved in polynomial time [3].

The join ordering problem discussed in the paper is specified by no restriction on the query graph (it can even have no edges), no restriction on the cardinality estimation method, and for the resulting join trees can be bushy with cross products. The framework $\mathrm{DP}_{\mathrm{conv}}$ transforms the join ordering problem to some kind of subset space, solves the problem therein, and transforms the solution back; this is similar in spirit to discrete FFT. Thereby, $\mathrm{DP}_{\mathrm{conv}}$ exhibits a reduced complexity compared to traditional ones working on the same join ordering problem (such as $\mathrm{DP}_{\mathrm{sub}}$): It is the first approach to beat the $O(3^n)$ complexity barrier.

On the cost function side, the authors instantiate their framework with two cost functions: $C_{\max}$ and C_{out}. A direct instantiation of the framework with $C_{\max}$ yields a complexity of $O(2^n n^3)$, where a factor of n (the number of relations) can be saved by a simpler algorithm. Thus, from a complexity point of view things look great. From a practicability point of view, assume a query whose result size is at least the size of any intermediate result (e.g., cross products only). Then, *every* plan under $C_{\max}$ has the same cost. In general, since the number of possible plans ($\frac{2(n-1)!}{(n-1)!}$ by far exceeds the number of possibilites for $C_{\max}$ (2^n-1), many plans will end up with the same costs. Consequently, we believe that a direct application of $C_{\max}$ for plan generation may fail. Nonetheless, the authors point to another interesting practical application of $C_{\max}$: use its result to prune the search space. To this end, they present some experiments using a pruning version of $\mathrm{DP}_{\mathrm{sub}}$. Since top-down algorithms like $\mathrm{TD}_{\mathrm{mcc}}$ [1] are typically better suited for pruning, it will be interesting to see how effective this pruning measure will be compared to and combined with existing ones.

The instantiation of $\mathrm{DP}_{\mathrm{conv}}$ with C_{out} results in a complexity of $O(2^n n^2 W n \log W n)$, where W is the cardinality of the largest intermediate result. For query 29c of the join order benchmark [4], the factor $W \log W$ alone is more than 1000 times larger than 3^n, the complexity of $\mathrm{DP}_{\mathrm{sub}}$. If we do not consider cross products, the factor $W \log W$ is already about 500K times larger than the number of ccps. This renders the approach infeasible for practice. However, remember that C_{out} was originally proposed to study the complexity of different join ordering problems. Only later it turned out that it is a good proxy for real costs whose application results in join orders which are quite good. Thus, an important line of future work opens up: Can we define properties of cost functions and families of cost functions satisfying (some of) these properties such that (a) $\mathrm{DP}_{\mathrm{conv}}$ allows for an efficient instantiation and (b) the cost functions are good proxies for real costs if applied to join ordering? Another alternative to improve the efficiency of $\mathrm{DP}_{\mathrm{conv}}$ would be to take the query graph into account by excluding cross products. This would require some progress in sparse subset convolution.

Parallelizing join ordering is another important topic. The candiate of choice has thus far been $\mathrm{DP}_{\mathrm{sub}}$ [2, 5]. Now, $\mathrm{DP}_{\mathrm{conv}}$ offers an alternative. A closer look at $\mathrm{DP}_{\mathrm{conv}}$ reveals that it is very well suited for massive parallelization. Hence, I look forward to a first implementation and evaluation of $\mathrm{DP}_{\mathrm{conv}}$ on a GPU.

1. REFERENCES

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

[2] W. Han, W. Kwak, J. Lee, G. Lohman, and V. Markl. Parallelizing query optimization. *Proc. of the VLDB Endowment (PVLDB)*, 1(1):188–200, 2008.

[3] T. Ibaraki and T. Kameda. Optimal nesting for computing n-relational joins. *ACM Trans. on Database Systems*, 9(3):482–502, 1984.

[4] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *VLDB Journal*, 27:643–668, 2018.

[5] R. Mancini, S. Karthik, B. Chandra, V. Mageirakos, and A. Ailamaki. Efficient massively parallel join optimization for large queries. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 2022.