DunceCap: Compiling Worst-Case Optimal Query Plans

[Extended Abstract]

Adam Perelman*
Stanford University
adamperelman@cs.stanford.edu

Christopher Ré†
Stanford University
chrismre@cs.stanford.edu

ABSTRACT

Modern data analytics workloads frequently involve complex join queries where the pairwise-join-based algorithms used by most RDBMS engines are suboptimal. In this study, we explore two algorithms that are asymptotically faster than pairwise algorithms for a large class of queries. The first is Yannakakis' classical algorithm for acyclic queries. The second is a more recent algorithm which works for any query and which is optimal with respect to the worst-case size of the output. We introduce a query compiler, DunceCap, which uses these two algorithms and variations on them to produce optimal query plans, and find that these plans can outperform standard RDBMS algorithms as well as simple worst-case optimal algorithms by an order of magnitude on a variety of queries.

1. INTRODUCTION

Traditional RDBMS join algorithms are based on pairwise joins. Modern OLAP and graph processing workloads, however, typically involve complex queries, often with many large tables and sometimes with cycles in the query graph, where traditional pairwise algorithms are suboptimal and more efficient techniques are needed[1].

Theoreticians have developed several algorithms that are asymptotically superior to any pairwise plan for a variety of queries. In this section, we introduce two such algorithms. The first is Yannakakis' classical algorithm for acyclic queries[5]. The second is a worst-case optimal algorithm recently introduced by Ngo, Porat, Ré, and Rudra (henceforth NPRR) which can be used for any query, whether cyclic or acyclic[3].

*This author is grateful to Susan Tu for many collaborations on this project, to Chris Aberger and Andres Nötzli for implementation advice, and to Rohan Puttagunta and Manas Joglekar for valuable insights into join theory.

†We gratefully acknowledge support from a National Science Foundation CAREER Award under No. IIS-1353606.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
ACM 978-1-4503-2758-9/15/05.
http://dx.doi.org/10.1145/2723372.2764945.

1.1 Yannakakis' Algorithm

For a given join query $Q = R_1 \bowtie \cdots \bowtie R_n$, we define its hypergraph representation as $\mathcal{F} = (V, E)$, where V is the set of attributes in the query $V = \bigcup R_i$, and E is the set of relations $E = \{R_i\}$. We define a query as acyclic if and only if its hypergraph representation is acyclic. If a query Q is acyclic, then there exists a *full reducer* for Q, which is a sequence of pairwise semi-joins such that after execution of this semi-join plan, all relations $R_i \in Q$ are both pairwise consistent and globally consistent. That is, none of the relations have *dangling tuples*: tuples that do not correspond to any tuple in the query result. Based on this insight, Yannakakis' algorithm works as follows: first, we run a full reducer. Then, we compute the output via a sequence of pairwise joins. At each step, every tuple in the intermediate result corresponds to at least one tuple in the final output, so the run-time of this algorithm is linear in the size of the input and the output[5].

However, this algorithm only works for acyclic queries. For cyclic queries, we can extend Yannakakis as follows: first, we compute the minimum-width tree decomposition for the query; this is a standard technique to modify algorithms for acyclic graphs so that they can run on cyclic graphs. Within each bag of the hypertree decomposition, we use a traditional pairwise plan to compute the join of the relations in the bag. Once all the within-bag joins are complete, we are left with a tree of relations, which we join via Yannakakis' classical algorithm.

1.2 Worst-Case Optimal Join Algorithm

More recently, NPRR introduced a new algorithm that is guaranteed to run in time proportional to the worst-case size of the output, regardless of whether the query is cyclic or acyclic. Rather than using relational algebra operations as its building blocks, the NPRR algorithm reduces any join query to a sequence of set-theoretic operations, in particular set intersections and set unions[3].

2. QUERY COMPILER

Our query compiler, DunceCap, takes as input a list of relations over which to perform a natural join. It then compiles the query into a C++ query plan which executes either Yannakakis, NPRR, or a HybridJoin algorithm.

Our HybridJoin algorithm is very similar to Yannakakis' algorithm. However, after computing the minimum-width hypertree decomposition for a query, HybridJoin uses NPRR instead of a pairwise plan to compute the join of the relations

Query	Relational Algebra Expression
Triangle (K_3)	$R(A,B) \bowtie S(B,C) \bowtie T(A,C)$
4-clique (K_4)	$R(A,B) \bowtie S(B,C) \bowtie T(C,D)$ $\bowtie U(A,D) \bowtie V(A,C) \bowtie W(B,D)$
Lollipop ($L_{3,1}$)	$R(A,B) \bowtie S(B,C) \bowtie T(C,A)$ $\bowtie U(A,D)$
Lollipop ($L_{4,1}$)	$R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(D,A)$ $\bowtie V(A,C) \bowtie W(B,D) \bowtie X(A,E)$

Table 1: Common graph-pattern queries.

Dataset	Algorithm	Query			
		K_3	K_4	$L_{3,1}$	$L_{4,1}$
Facebook	Yannakakis	0.159	10.6	0.65	148.6
	NPRR	0.086	2.51	18.2	54.7
	HybridJoin	0.086	2.51	0.354	12.7
	LogicBlox	0.52	4.13	11.4	241.5
Arxiv GR-GC	Yannakakis	0.032	0.259	0.075	1.34
	NPRR	0.027	0.120	0.087	0.715
	HybridJoin	0.027	0.120	0.054	0.363
	LogicBlox	0.49	0.55	0.73	2.14

Table 2: Runtime (in seconds) of each count query. LogicBlox, a commercial system that implements a worst-case optimal algorithm, uses 48 cores; all other algorithms are single-threaded.

within each bag. Once the within-bag joins are complete, we compute the output via Yannakakis' algorithm.

3. EXPERIMENTS

We ran the common graph-pattern queries described in Table 1 on two real-world graph datasets from the Stanford Network Analysis Project (SNAP): the Arxiv GR-GC dataset (5242 nodes, 14496 edges) and the Facebook dataset (4039 nodes, 88234 edges)[2].

3.1 Algorithm Comparison

Running times are listed in Table 2. For all queries except $L_{3,1}$, the NPRR algorithm is faster than the Yannakakis algorithm, implying that the cost of pairwise within-bag joins in the Yannakakis algorithm outweighs the lack of a full reducer in the NPRR algorithm. For the $L_{3,1}$ query, the opposite is true. However, for all queries, the HybridJoin algorithm achieves the best running time, implying that it effectively combines Yannakakis and NPRR.

We compare our running times with LogicBlox, the only commercial system which implements a worst-case optimal join algorithm[4]. The comparison is not perfect: LogicBlox is implemented in Java instead of C++, and incurs overhead for fault tolerance, task distribution, and so on. On the other hand, while the LogicBlox implementation uses all 48 cores on our machine, DunceCap use a single thread. By profiling our algorithms on different queries, we estimate that for long-running queries, differences in overhead account for a factor of about 2 in our runtime improvement over LogicBlox, and algorithmic differences account for a factor of about 10, although these vary depending on the query.

3.2 Attribute Ordering and NPRR

The order in which NPRR performs the set intersections for each attribute in a query depends on the *global attribute ordering* chosen by the algorithm. The theoretical worst-case optimal bound on the algorithm holds regardless of the attribute ordering. However, our results show that in practice, the choice of ordering can change runtime by over an order of magnitude. Whereas Table 2 assumes runtimes given the optimal attribute ordering, Table 3 shows the runtime for NPRR given two example attribute orderings (the orderings use the attribute names defined in Table 1).

Query	Attribute Order	Dataset	
		Facebook	Arxiv GR-GC
$L_{3,1}$	A,B,C,D	1.817	0.087
	D,A,B,C	25.64	0.495
$L_{4,1}$	A,B,C,D,E	54.68	0.715
	E,A,B,C,D	1358.0	6.729

Table 3: Runtime (in seconds) of NPRR on lollipop queries given each global attribute ordering.

In our experiments, the best attribute ordering for a query corresponds to its minimum-width hypertree decomposition. In particular, given the best attribute ordering for a lollipop query, NPRR begins by computing intersections for attributes that would have been placed in the same bag by a minimum-width hypertree decomposition. It does not consider the remaining attribute from the other bag until the very end of the algorithm. By staying within a single bag of the decomposition for as long as possible, this strategy keeps intermediate results relatively small. In contrast, the worst attribute orderings cause NPRR to begin with attributes that are from different bags of the decomposition, leading to larger intermediate results. Thus, the hypertree decomposition of a query is helpful for determining the optimal attribute ordering.

4. CONCLUSION AND FUTURE WORK

In this study, we compared the performance of Yannakakis' algorithm, a worst-case optimal algorithm, and a hybrid algorithm, finding that the hybrid algorithm effectively combines the best runtime characteristics of both Yannakakis and NPRR. This algorithmic refinement causes our query plans to outperform a commercial system that implements a worst-case optimal algorithm by an order of magnitude.

We see several promising directions for future work. First, we hope to incorporate work on fast set-intersections via SIMD algorithms into our query engine[1]. Second, a better understanding of these join algorithms requires an exploration of how each can be parallelized. Third, recent research has begun to explore "beyond worst-case" algorithms, which are instance-optimal rather than schema-optimal. There is potential for significant advances to be made by exploring the tradeoffs between our algorithms and these beyond-worst case algorithms.

5. REFERENCES

[1] C. R. Aberger, A. Nötzli, K. Olukotun, and C. Ré. EmptyHeaded: Boolean Algebra Based Graph Processing, Mar. 2015.

[2] J. Leskovec and A. Krevl. Snap datasets: Stanford large network dataset collection, June 2014.

[3] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms: [extended abstract]. In *PODS '12*, pages 37–48, New York, NY, USA, 2012. ACM.

[4] T. L. Veldhuizen. Leapfrog triejoin: a worst-case optimal join algorithm. *CoRR*, abs/1210.0481, 2012.

[5] M. Yannakakis. Algorithms for acyclic database schemes. VLDB '81, pages 82–94. VLDB Endowment, 1981.