Faster Join-Projects and Sparse Matrix Multiplications

Rasmus Resen Amossen
IT University of Copenhagen
Rued Langaardsvej 7
2300 Copenhagen S
Denmark
resen@itu.dk

Rasmus Pagh
IT University of Copenhagen
Rued Langaardsvej 7
2300 Copenhagen S
Denmark
pagh@itu.dk

ABSTRACT

Computing an equi-join followed by a duplicate eliminating projection is conventionally done by performing the two operations in serial. If some join attribute is projected away the intermediate result may be much larger than both the input and the output, and the computation could therefore potentially be performed faster by a direct procedure that does not produce such a large intermediate result. We present a new algorithm that has smaller intermediate results on worst-case inputs, and in particular is more efficient in both the RAM and I/O model. It is easy to see that join-project where the join attributes are projected away is equivalent to boolean matrix multiplication. Our results can therefore also be interpreted as improved sparse, output-sensitive matrix multiplication.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational databases; I.1.2 [**Computing Methodologies**]: SYMBOLIC AND ALGEBRAIC MANIPULATION—*Algebraic algorithms*

General Terms

Algorithms, Performance, Theory

Keywords

Matrix Multiplication, Collapsing Join-Project, Relational algebra

1. INTRODUCTION

Efficient computation of matrix multiplication and joins of database relations are both problems that have been studied in decades. What might not be obvious is that the two problems are related and below we shall see how computation time of both can be improved in some cases by combining techniques from the fields.

First, let us spend a moment motivating the need for improvements in computation of database joins—or rather, database joins followed by a duplicate eliminating projection: consider a set of movies and the set of actors in these movies. This data set can be described in a table with two columns, *movie* and *actor*, pairing related movies and actors. If we were interested in the unique set of actor pairs playing together in at least one movie we could join the table with itself on *movie*, project away the *movie* column and eliminate duplicates. In a small experiment on a subset of the Internet Movie Database (IMDB) we performed a join of 492,000 movie appearances, involving 37,000 actors and 8,100 movies. The number of actor pairs produced by the join-project was 70,000,000, while the size of the join (without projection) was much larger, having over 676,000,000 tuples.

As a more generel formulation, consider two tables $R_1(a, b)$ and $R_2(b, c)$ sharing the key b, and a join on b followed by a projection on a and c. In relational algebra this can be written as $\pi_{a,c}(R_1 \bowtie R_2)$. It is easy to see that an algorithm for this case can be used to solve the case where the relations may have more attributes, by considering several attributes as one (if needed, hashing can be used to produce a unique signature for large composite values). We will use N and Z to denote the input and output size, respectively. That is, $N = |R_1| + |R_2|$ and $Z = |\pi_{a,c}(R_1 \bowtie R_2)|$. As an example, assume $R_1 = R_2 = \{(x, y) \in \mathbb{N}^2 \mid 1 \leq x \leq n \text{ and } 1 \leq y \leq n\}$ so that $|R_1| = |R_2| = n^2$ for some $n \in \mathbb{N}$. Current database systems produce the final result by evaluating the operators in an evaluation tree. We will refer to this approach as the *classical algorithm* (see [8]). In our case, this implies two steps: First, the join $R_1 \bowtie R_2$ is performed, producing an intermediate result of a certain size. Next, the projection is carried out. In the join $R_1 \bowtie R_2$, each of the n^2 tuples in R_1 will match n tuples in R_2 resulting in n^3 unique tuples in total for the join operation. However, when performing the projection $\pi_{a,c}$ afterwards, the final result will only have n^2 unique tuples when duplicates are eliminated. In other words, the intermediate result had a factor $\Theta(\sqrt{N})$ tuples more than both the input and the final result which seems like a waste of costly I/O. Other cases are less trivial.

Let M and B denote the memory and block size respectively where the unit of measurement is a single relation entry. That is, we assume that the memory can hold M entries of a relation and a block can hold B entries [1]. Let furthermore $\tilde{\mathcal{O}}(f)$ be a shorthand for $f^{1+o(1)}$. Then the classical algorithm requires $\tilde{\mathcal{O}}(N\sqrt{Z}/B)$ I/Os (see Section 2). If the join attribute is not projected away the classical algorithm

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.

is good, running in $\tilde{\mathcal{O}}((N+Z)/B)$ I/Os.

As explained in more detail later, one way to improve the worst-case behavior in cases similar to the example above is to represent the input tuples of R_1 and R_2 as adjacency matrices of size $n \times n$ and construct the result by multiplying the matrices in $\tilde{\mathcal{O}}(n^{2.376})$ time [3].

This paper presents a way to evaluate these kind of expressions more efficiently, without the need for the large intermediate subresult, by using a hybrid of matrix multiplication and the classical algorithm. The hybrid technique implies a worst-case improvement in the computation time of conventional sparse matrix multiplications where both input and output is sparse. The improvement holds within the RAM model and the I/O model [1]. More specifically, we obtain a worst-case time complexity of $\tilde{\mathcal{O}}(N^{2/3}Z^{2/3} + N^{0.862}Z^{0.408})$ in the RAM model and $\tilde{\mathcal{O}}\left(\frac{N\sqrt{Z}}{BM^{1/8}}\right)$ I/Os in the I/O model where, as a side effect of the hybrid construction, our algorithm is at least as good in worst-case as any known algorithm for matrix multiplication for all possible combinations of N, n and Z.

We will refer to joins followed by a duplicate eliminating projection that projects away one or more join attributes as a *collapsing join-project*. Potentially, collapsing join-projects can be used as a single operator in query optimizers.

1.1 Related work

In 1984 Willard [6] presented an algorithm for evaluation of relational calculus expressions and analyzed the worst-case complexity in the RAM model. The shown time and space bounds had only the input size N as parameter. In 1990 Willard [7] presented an improved analysis which also took the output size Z into account. Willard did not consider projections and was therefore able to achive near-linear complexity in his algorithms. However, the results did not scale in the number of tables k used. Pagh and Pagh [5] introduced k as a third parameter in their analysis of acyclic joins and presented an algorithm that scales linearly with k. As seen, the analysis can be more precise when using more parameters. This paper considers k in Section 2 but we focus on the case $k = 2$ in our algorithm in Section 3. However, we introduce a fourth parameter, namely the number n of distinct attribute values in input.

We will refer to our generic algorithm as *Algorithm 1*. The generic algorithm has a number of instantiations, depending on how its steps are implemented (in the RAM or I/O model). Below, we compare the worst-case performance *analysis* of Algorithm 1 in the RAM model with the *analysis* of the classical sort-merge-join, the results by Coppersmith and Winograd [3] and Yuster and Zwick [9]. We emphasize *analysis* because the various analyses are not tight to the actual performance of the algorithms. The shown comparison is therefore not accurate. In the following, let n denote then number of distinct attribute values in the input, that is, $n = |\pi_a(R_1) \cup \pi_b(R_1) \cup \pi_b(R_2) \cup \pi_c(R_2)|$.

Algorithm 1 The analysis of this algorithm gives a complexity of $\tilde{\mathcal{O}}(N^{2/3}Z^{2/3} + N^{0.862}Z^{0.408})$.

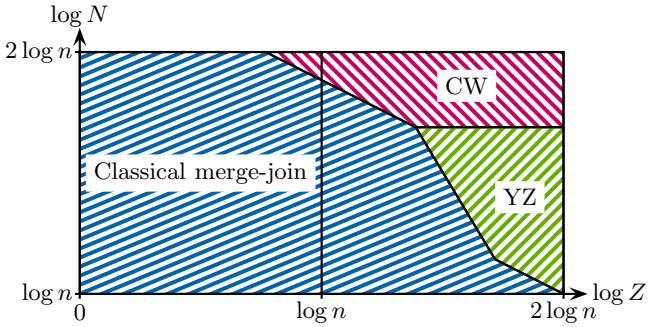

Figure 1: A comparison of the classical merge-join algorithm and the algorithms by Coppersmith and Winograd (CW) and Yuster and Zwick (YZ). The figure shows the previously fastest algorithm on a RAM model for different values of parameters N (input size) and Z (output size).

The classical algorithm Yannakakis [8] gave a worst-case complexity of $\tilde{\mathcal{O}}(NZ)$ for general acyclic join-projects on an arbitrary number of relations. For two relations, the worst-case complexity is $\tilde{\mathcal{O}}(N\sqrt{Z})$ as we show in Theorem 2.1. This anaysis is tight.

Coppersmith and Winograd We will refer to this result as *CW*. They obtained a matrix exponent of 2.376 giving a complexity of $\tilde{\mathcal{O}}(n^{2.376})$. This analysis is tight.

Yuster and Zwick We will refer to this result as *YZ*. Their complexity was $\tilde{\mathcal{O}}(N^{0.7}n^{1.2} + n^2)$ for $n \times n$ matrices with at most N nonzero elements but the analysis is not output sensitive. Notice that $n \leq N$.

The space requirements for the above algorithms are generally determined by the size of the intermediate results and the size of the matrices involved.

Table 1 compares the time and space requirements for the above algorithms and in Figure 1 we show, for each $(N, Z) \in [n^1; n^2] \times [0; n^2]$, the fastest algorithm (excluding Algorithm 1) at that coordinate with respect to their analysis. Figure 2 shows where the analysis of Algorithm 1 is (strictly) best.

1.2 Outline

The rest of this paper is organized as follows: above we gave an example of suboptimal behavior of the classical algorithm and this behavior will be analyzed more formally in Section 2. Section 3 describes our algorithm in the RAM and I/O model.

2. THE CLASSICAL ALGORITHM

In this section we perform an analysis of the classical algorith. Let $\bowtie R_i$ denote a *natural join* of k relations $R_1, \ldots, R_k$ and Z denote the output size of the final projection $\pi(\bowtie R_i)$. Given a known input size $N = \sum |R_i|$ and output size Z we search for an upper bound for

$$\mathcal{U}(N, Z) = \max_{\substack{R_1 \ldots R_k \\ \Sigma|R_i|=N \\ |\pi(\bowtie R_i)|=Z}} |\bowtie R_i|.$$

Algorithm	Model	Time	Space
Classical alg.	RAM	$\tilde{\mathcal{O}}(N\sqrt{Z})$	$\tilde{\mathcal{O}}((N+Z)/w)$
Algorithm 1	RAM	$\tilde{\mathcal{O}}(N^{2/3}Z^{2/3} + N^{0.862}Z^{0.408})$	$\tilde{\mathcal{O}}(T/w)$
CW	RAM	$\tilde{\mathcal{O}}(n^{2.376})$	$\tilde{\mathcal{O}}(n^2/w)$
YZ	RAM	$\tilde{\mathcal{O}}(N^{0.7}n^{1.2} + n^2)$	$\tilde{\mathcal{O}}(T/w)$
Classical alg.	I/O	$\mathcal{O}(N\sqrt{Z}/B)$	T
Algorithm 1	I/O	$\tilde{\mathcal{O}}\left(\frac{N\sqrt{Z}}{BM^{1/8}}\right)$	T

Table 1: A comparison of worst-case time and space requirements for the algorithms mentioned in Section 1.1. The units for time and space in the RAM model are *steps* and *words* of size w, respectively, and in the I/O model, the units are number of I/Os and number of blocks of size B, respectively. T is a short-hand notation for the time complexity of the algorithm on the *same* line.

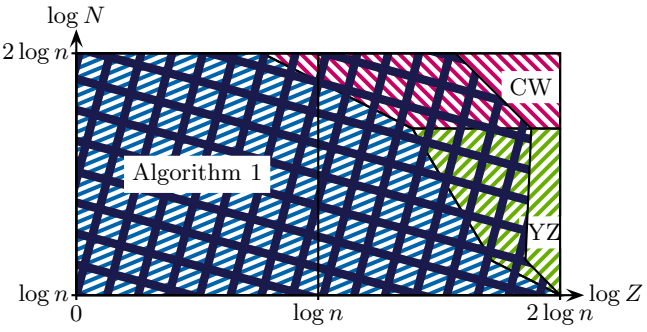

Figure 2: A comparison similar to Figure 1 but with the analysis of Algorithm 1 (the area under the grid) included. The graph shows the strictly fastest algorithm on a RAM model for different values of parameters N and Z. As seen, the analysis of Algorithm 1 completely dominates the merge-join and for some values of (N, Z) it also dominates the algorithms by Coppersmith and Winograd, and Yuster and Zwick.

In 1981 Yannakakis [8] showed that $\mathcal{U}(N, Z) \leq NZ$ by analyzing an algorithm that is identical to the classical algorithm when all relations share an attribute. But $\mathcal{U}$ depends on k as the following theorem shows. From now on we consider the case where all relations share an attribute.

THEOREM 2.1. *Let $k > 1$ be an integer. For k relations on the form $R_i(a_i, b)$ we have*

$$\mathcal{U}(N, Z) = \Theta(NZ^{1-\frac{1}{k}}).$$

PROOF. We first show the upper bound on $\mathcal{U}$. For each possible b-value x, define $s_i(x)$ as the number of tuples in R_i having $b = x$. That is, $s_i(x) = |\sigma_{b=x}(R_i)|$. The tuples in $\bowtie R_i$ having $b = x$ for some value x will all be unique and thus have a representative in the final projected output. Therefore

$$s_1(x)s_2(x)\cdots s_k(x) \leq Z. \tag{1}$$

For any i, define S_i as the subset of b-values occurring in more than $Z^{\frac{1}{k}}$ tuples of R_i, that is $S_i = \{x \mid s_i(x) > Z^{\frac{1}{k}}\}$. Each $x \in S_i$ will match at most $Z^{1-\frac{1}{k}}$ tuples in total in the other tables due to (1), and as $|S_i| \leq |R_i|$ we have that $x \in S_i$ will induce at most $|R_i|Z^{1-\frac{1}{k}}$ tuples in the final projected output. A similar argument can be applied for all i resulting in

$$\mathcal{U}(N, Z) \leq \sum_{i=1}^{k} |R_i|Z^{1-\frac{1}{k}} = NZ^{1-\frac{1}{k}}.$$

For the lower bound of $\mathcal{U}$ let $[q]$ be a general notation for the set $\{x \in \mathbb{N} \mid 1 \leq x \leq q\}$ and define k relations $R_i(a_i, b)$ with tuples $[Z^{\frac{1}{k}}] \times [\frac{N}{k}/Z^{\frac{1}{k}}]$. Note that $|R_i| = \frac{N}{k}$ and that every tuple $r \in R_1$ will match exactly $Z^{\frac{1}{k}}$ tuples in each of the $k-1$ other relations producing a total of $(Z^{\frac{1}{k}})^{k-1} = Z^{1-\frac{1}{k}}$ tuples in the join containing r. As $|R_1| = \frac{N}{k}$ the total join size is $\frac{N}{k}Z^{1-\frac{1}{k}}$. □

3. COMPUTING THE JOIN-PROJECT

We will show how to compute the collapsing join-project efficiently for $k = 2$. For $R_1(a, b)$ and $R_2(b, c)$ let V_a, V_b and V_c be sets of all distinct a, b and c values represented in such a way that $v \in V_i$ and $u \in V_j$ where $u = v$ are treated as equal if $i = j$ but distinct if $i \neq j$. The join $R_1 \bowtie R_2$ can be represented as a *sequentially tripartite* graph $G = (V_a, V_b, V_c, E)$ where $E \subseteq (V_a \times V_b) \cup (V_b \times V_c)$. Notice, that in contrast to a conventional tripartite graph we have that $V_a \times V_c \cap E = \emptyset$. We consider undirected graphs, where it is understood that an edge (u, v) is considered identical to the edge (v, u).

Let $(v_a, v_b) \in V_a \times V_b$ be an edge in E if and only if (v_a, v_b) is a tuple in $R_1(a, b)$ and similarly $(v_b, v_c) \in V_b \times V_c$ an edge in E if and only if (v_b, v_c) is a tuple in $R_2(b, c)$. See Figure 3. Notice that (a, c) is a tuple in $\pi_{a,c}(R_1 \bowtie R_2)$ if and only if there is a path of length 2 from a to c in G. The edges in $V_a \times V_b$ and $V_b \times V_c$ can be represented as two *adjacency matrices* M^{ab} and M^{bc}. We have the following easy lemma:

LEMMA 3.1. *A tuple $(a, c) \in \pi_{ac}(R_1 \bowtie R_2)$ if and only if $(M^{ab}M^{bc})_{a,c} > 0$.*

PROOF. As M^{ab} and M^{bc} are adjacency matrices over a graph their product will, by definition of matrix multiplication, contain a non-zero entry at row a column c exactly if a and c are connected by a path of length 2. □

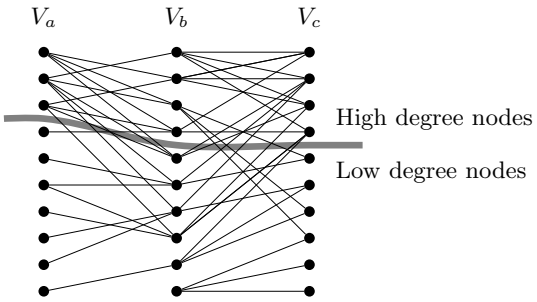

Figure 3: Two relations R_1 and R_2 represented as a graph.

Rather than just using matrix multiplication we will compute the result by decomposing the join and projection into several parts, defined by how much redundancy they are candidate to produce in the classical algorithm.

DEFINITION 3.2 (DEGREE). *Let $\delta(v) : V \to \mathbb{N}$ denote the* degree *of the node $v \in V = V_a \cup V_b \cup V_c$ defined as the size of $(\{v\} \times V) \cap E$.*

Using the degree, we can give a simple upper bound on the number of occurrences of every tuple (a, c) in the join $R_1 \bowtie R_2$:

LEMMA 3.3. *Let $B_1(a) = \{b \in V_b \mid (a, b) \in E\}$ and $B_2(c) = \{b \in V_b \mid (b, c) \in E\}$. Then the a tuple of the form $(a, \cdot, c)$ will occur exactly $r = |B_1(a) \cap B_2(c)|$ times in the join $R_1 \bowtie R_2$ and $r \leq \min(\delta(a), \delta(c))$.*

We will split the nodes in V_a, V_b and V_c in low and high degree nodes using two thresholds $\Delta_{ac}, \Delta_b \in \mathbb{N}$. According to Lemma 3.3 the values from V_a and V_c with a degree smaller than Δ_{ac} are guaranteed to occur with multiplicity at most Δ_{ac} in the join. Tuples in R_1 and R_2 containing such a and c values can therefore be joined by using a conventional merge join and removing duplicates using either a dictionary or by sorting, depending on model of computation. The output multiplicity cannot be deduced from the degree of values in V_b but tuples having $\delta(b) < \Delta_b$ will occur at most $N\Delta_b$ times in the join and handling these small-degree tuples by a merge-join will imply a smaller input to the following more time-consuming step: The rest of the tuples are represented as two adjacency matrices which are multiplied using an efficient conventional matrix multiplication algorithm [9] in order to find paths of length 2 between a and c nodes. The algorithm is summarized in Algorithm 1.

OBSERVATION 3.4. *Conventional matrix multiplication is a special case of Algorithm 1 for $\Delta_{ac} = \Delta_b = 0$. The algorithm by Yuster and Zwick [9] is a special case of Algorithm 1 for $\Delta_{ac} = n+1$. The classical merge-join algorithm is a special case of Algorithm 1 for $\Delta_{ac} = \Delta_b = n+1$.*

In particular, there exist values Δ_{ac} and Δ_b so that Algorithm 1 is at least as good as any other known algorithm for sparse boolean matrix multiplication.

Also notice, that the algorithm will produce the correct output (but the running times may differ) for any values of Δ_{ac} and Δ_b.

Algorithm 1 Computing $\pi_{ac}(R_1 \bowtie R_2)$ or equivalently: computing the product of M^{ab} and M^{bc}.

1: $\mathcal{R}_1' \leftarrow \{(a, b) \in R_1 \mid \delta(a) < \Delta_{ac}\}$ ▷ Low multiplicity output
2: $\mathcal{R}_2' \leftarrow \{(b, c) \in R_2 \mid \delta(c) < \Delta_{ac}\}$
3: $S \leftarrow \pi_{ac}(\mathcal{R}_1' \bowtie \mathcal{R}_2)$ using the classical algorithm
4: $S \leftarrow S \cup \pi_{ac}(\mathcal{R}_1 \bowtie \mathcal{R}_2')$ using the classical algorithm
5: $\mathcal{R}_1'' \leftarrow \{(a, b) \in R_1 \mid \delta(b) < \Delta_b\}$ ▷ Low $\delta(b)$
6: $\mathcal{R}_2'' \leftarrow \{(b, c) \in R_2 \mid \delta(b) < \Delta_b\}$
7: $S \leftarrow S \cup \pi_{ac}(\mathcal{R}_1'' \bowtie \mathcal{R}_2'')$ using the classical algorithm
8: $M' \leftarrow$ adjacency matrix for $\{(a, b) \in R_1 \mid \delta(a) \geq \Delta_{ac}$ and $\delta(b) \geq \Delta_b\}$
9: $M'' \leftarrow$ adjacency matrix for $\{(b, c) \in R_2 \mid \delta(c) \geq \Delta_{ac}$ and $\delta(b) \geq \Delta_b\}$
10: $\mathcal{M} \leftarrow M' M''$ using multiplication algorithm of choice
11: $S \leftarrow S \cup \{(a, c) \mid \mathcal{M}_{a,c} > 0\}$
12: Eliminate duplicates in S
13: Output S

3.1 Complexity in the RAM model

In the following we assume that Z is known. This assumption will be justified later.

THEOREM 3.5. *Let $f(N, Z)$ denote the time complexity of Algorithm 1. Then $f(N, Z)$ is $\tilde{\mathcal{O}}(N^{2/3}Z^{2/3} + N^{0.862}Z^{0.408})$ for suitable choice of Δ_{ac} and Δ_b.*

PROOF. Algorithm 1 produces the output in three steps that account for the superlinear work with respect to the $\tilde{\mathcal{O}}$ notation:

1. Tuples generated in line 3 and 4: There are Z unique tuples in $\pi_{ac}(R_1 \bowtie R_2)$ and each tuple corresponds to at most Δ_{ac} tuples in $R_1 \bowtie R_2$ according to Lemma 3.3 so this step produces at most $O(Z\Delta_{ac})$ tuples in total.

2. Tuples generated in line 7: Each b node can be reached from at most Δ_b different a or c nodes. Therefore this step contributes with at most $O(N\Delta_b)$ tuples.

3. Tuples generated in line 10 and 11: The size of the matrices will be at most[1] $\frac{N}{\Delta_{ac}} \times \frac{N}{\Delta_b}$ and $\frac{N}{\Delta_b} \times \frac{N}{\Delta_{ac}}$ so this step can be handled in $\tilde{\mathcal{O}}(M(\frac{N}{\Delta_{ac}}, \frac{N}{\Delta_b}, \frac{N}{\Delta_{ac}}))$ time where $M(x, y, z)$ denotes the minimum number of arithmetic operations needed in order to multiply an $x \times y$ with a $y \times z$ matrix. This can be implemented in $\tilde{\mathcal{O}}(M(x, y, z))$ time in the RAM model.

Huang and Pan [4] proved that

$$M(x, y, x) = x^{2-\alpha\beta+o(1)}y^{\beta} + x^{2+o(1)}$$

[1]The matrix dimensions are also bounded by n but as seen in figure 2 we still obtain a near-optimal result by simplifying the analysis.

is an upper bound on M, where α and β are constants given by the matrix multiplication algorithm. With bounds proved by Coppersmith and Winograd [3] and Coppersmith [2] the currently best known algorithm has $\alpha = 0.294$ and $\beta = 0.533$.

The duplicate elimination in line 12 can be done in $\tilde{\mathcal{O}}(Z)$ time using sorting or hashing.

We now have

$$\begin{aligned} f(N,Z) &= \tilde{\mathcal{O}}\left(M(\tfrac{N}{\Delta_{ac}},\tfrac{N}{\Delta_b},\tfrac{N}{\Delta_{ac}}) + N\Delta_b + Z\Delta_{ac}\right) \\ &= \tilde{\mathcal{O}}\Big((\tfrac{N}{\Delta_{ac}})^{2-\alpha\beta+o(1)}(\tfrac{N}{\Delta_b})^{\beta} + (\tfrac{N}{\Delta_{ac}})^{2+o(1)} \\ &\qquad + N\Delta_b + Z\Delta_{ac}\Big) \qquad (2) \\ &= \tilde{\mathcal{O}}\Big((\tfrac{N}{\Delta_{ac}})^{k}(\tfrac{N}{\Delta_b})^{\beta} + (\tfrac{N}{\Delta_{ac}})^{2} \\ &\qquad + N\Delta_b + Z\Delta_{ac}\Big). \qquad (3) \end{aligned}$$

where (3) is a simplified expression obtained by setting $k = 2 - \alpha\beta$.

We are interested in values for Δ_{ac} and Δ_b so that $f(N,Z)$ is minimized. For simplicity, rewrite (3) to $\max\{\dots\}$ of the involved terms

$$f(N,Z) = \tilde{\mathcal{O}}\left(\max\left\{(\tfrac{N}{\Delta_{ac}})^{k}(\tfrac{N}{\Delta_b})^{\beta}, (\tfrac{N}{\Delta_{ac}})^{2}, N\Delta_b, Z\Delta_{ac}\right\}\right).$$

It is now safe to assume that $N\Delta_b = Z\Delta_{ac}$ and therefore we can simplify the above equation by setting $\Delta_b = Z\Delta_{ac}/N$:

$$f(N,Z) = \tilde{\mathcal{O}}\left(\max\left\{(\tfrac{N}{\Delta_{ac}})^{k}(\tfrac{N^2}{Z\Delta_{ac}})^{\beta}, (\tfrac{N}{\Delta_{ac}})^{2}, Z\Delta_{ac}\right\}\right)$$

Notice that the two first parameters decrease with Δ_{ac} while the last one increases. This means that minimum exists where either $(\frac{N}{\Delta_{ac}})^{k}(\frac{N^2}{Z\Delta_{ac}})^{\beta} = Z\Delta_{ac}$ or $(\frac{N}{\Delta_{ac}})^{2} = Z\Delta_{ac}$.

In order to deduce Δ_{ac}, consider the first case:

$$\begin{aligned} (\tfrac{N}{\Delta_{ac}})^{k}(\tfrac{N^2}{Z\Delta_{ac}})^{\beta} = Z\Delta_{ac} \quad &\Rightarrow \\ \Delta_{ac} = N^{\frac{k+2\beta}{1+k+\beta}} Z^{\frac{-2}{1+k+\beta}}&. \end{aligned}$$

With this value of Δ_{ac} we obtain the minimum

$$\tilde{\mathcal{O}}(Z\Delta_{ac}) = \tilde{\mathcal{O}}\left(N^{\frac{k+2\beta}{1+k+\beta}} Z^{1-\frac{2}{1+k+\beta}}\right).$$

Similarly, for the second case where $(\frac{N}{\Delta_{ac}})^2$ dominates we have the minimum

$$\tilde{\mathcal{O}}(Z\Delta_{ac}) = \tilde{\mathcal{O}}(N^{2/3}Z^{2/3})$$

using $\Delta_{ac} = N^{2/3}/Z^{1/3}$.

Finally the sum

$$\begin{aligned} f(N,Z) &= \tilde{\mathcal{O}}\left(N^{\frac{k+2\beta}{1+k+\beta}} Z^{1-\frac{2}{1+k+\beta}} + N^{2/3}Z^{2/3}\right) \\ &\approx \tilde{\mathcal{O}}(N^{0.862}Z^{0.408} + N^{2/3}Z^{2/3}). \qquad (4) \end{aligned}$$

must be an upper bound for the minimum, where the last line is obtained by using the currently best values of $\alpha = 0.294$ and $\beta = 0.533$ as described above. □

OBSERVATION 3.6. *If the exponent in matrix multiplication is $2 + o(1)$ as conjectured by many, the worst-case complexity of Algorithm 1 is $\tilde{\mathcal{O}}(N^{2/3}Z^{2/3})$ for suitable values of Δ_{ac} and Δ_b.*

As noted in the proof of Theorem 3.5 our analysis is simplified and our choice of Δ_{ac} and Δ_b not optimal: we do not take into account that $n \times n$ is an upper bound on matrix sizes and $\Delta_{ac}, \Delta_b \le n$. The real optimum is found by minimizing

$$\begin{aligned} &\tilde{\mathcal{O}}\Big(M\left(\min(\tfrac{N}{\Delta_{ac}},n), \min(\tfrac{N}{\Delta_b},n), \min(\tfrac{N}{\Delta_{ac}},n)\right) \\ &\qquad + N\Delta_b + Z\Delta_{ac}\Big) \\ &= \tilde{\mathcal{O}}\Big(M\left(\min(\tfrac{N}{\Delta_{ac}},n), \min(\tfrac{N^2}{Z\Delta_{ac}},n), \min(\tfrac{N}{\Delta_{ac}},n)\right) \\ &\qquad + Z\Delta_{ac}\Big) \end{aligned}$$

for $\Delta_{ac} \le n$.

OBSERVATION 3.7. *The optimal values of Δ_{ac} and Δ_b can be found efficiently assuming Z is known.*

3.2 Output sensitivity

When executing the algorithm, Z is not known in advance but it can be found iteratively without altering the complexity of the algorithm. This is done by iteratively guessing a value of $Z \in [Z'; 2Z'[$ for $Z' = 2^i$ in iteration i and noticing that the algorithm still works correctly when using an upper bound of Z. In each iteration the algorithm is stopped when the execution time exceeds the bound described in Theorem 3.5. As the execution time decreases geometrically, the latest execution time will dominate.

The time bound above, however, is given in big-oh notation which makes it impossible in practice to compare the actual evaluation time with the theoretical bound. A practical comparison would require an analysis of the involved constant. Another approach could be to estimate the value of Z by sampling: let S denote a sample obtained by picking q nodes from $\pi_a(R_1)$, q nodes from $\pi_c(R_2)$ and computing all paths of length 2 between these nodes. This sample S would have an expected size $(q/N)^2 Z$ and thus $Z = \mathbf{E}[(N/q)^2|S|]$, i.e. we have an unbiased estimator for Z.

3.3 Complexity in the I/O model

We can obtain results analogous to those in Section 3.1 for the I/O model by using I/O efficient algorithms for the steps of Algorithm 1, including an I/O efficient version of fast matrix multiplication. However, as we will see below even a very simple matrix multiplication algorithm, a cache-aware version of the cubic algorithm, yields worst-case complexity better than the classical algorithm.

Matrix multiplications in the I/O model can be performed by grouping the matrix elements in squares of size $\sqrt{M}\times\sqrt{M}$ and performing a conventional matrix multiplication using these squares as element units. With a block size of B, such a matrix multiplication requires $(N/\sqrt{M})^3\frac{M}{B} = N^3/(B\sqrt{M})$ I/Os.

THEOREM 3.8. *The number of I/Os required by Algorithm 1 when using a cache-aware cubic matrix multiplication algorithm is* $\tilde{\mathcal{O}}\left(\frac{N\sqrt{Z}}{BM^{1/8}}\right)$.

PROOF. Consider the three steps mentioned in the proof of Theorem 3.5. Step 1 requires $\tilde{\mathcal{O}}(Z\Delta_{ac}/B)$ I/Os and step 2 requires $\tilde{\mathcal{O}}(N\Delta_b/B)$ I/Os using sorting to compute the join and eliminate duplicates. We use the simple cubic-time matrix multiplication algorithm described above for step 3 resulting in a requirement of $N^3/(\Delta_{ac}^2\Delta_b\sqrt{M}B)$ I/Os for that step. Summarized, the number of I/Os required is

$$\tilde{\mathcal{O}}\left(\frac{Z\Delta_{ac}}{B} + \frac{N\Delta_b}{B} + \frac{N^3}{\Delta_{ac}^2\Delta_b\sqrt{M}B}\right). \quad (5)$$

Using similar arguments as in the proof of Theorem 3.5 we can assume that the three terms are equal at optimum. Setting the two first terms equal gives $\Delta_b = Z\Delta_{ac}/N$ which can be inserted into (5):

$$\tilde{\mathcal{O}}\left(2\frac{Z\Delta_{ac}}{B} + \frac{N^4}{\Delta_{ac}^3 Z\sqrt{M}B}\right) \quad (6)$$

Similarly, equating the two remaining terms gives $\Delta_{ac} = N/(\sqrt{Z}M^{1/8})$ which can be inserted into (6) in order to achive the desired result

$$\tilde{\mathcal{O}}\left(\frac{N\sqrt{Z}}{BM^{1/8}}\right) \text{ I/Os.}$$

□

Notice that even though we are using the naive cubic-time multiplication algorithm, this result improves the complexity with a factor $M^{1/8}$ compared to the classical algorithm.

4. CONCLUSION

We presented an output-sensitive algorithm for collapsing join-projects and sparse boolean matrix multiplication that is more efficient worst-case in both the RAM and I/O model than currently known algorithms. As we only deal with worst-case analysis in this paper the algorithm is not meant to replace current algorithms in database systems but might be implemented and used by the query optimizer as an alternative operator if the used join-project plan is slow. The presented algorithm can be modified to be used for conventional matrix multiplication over an arbitrary ring as well.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Comm. ACM*, 31(9):1116–1127, 1988. ISSN 0001-0782.

[2] D. Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, 1997. ISSN 0885-064X. doi: http://dx.doi.org/10.1006/jcom.1997.0438.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: http://doi.acm.org/10.1145/28395.28396.

[4] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complex.*, 14(2):257–299, 1998. ISSN 0885-064X. doi: http://dx.doi.org/10.1006/jcom.1998.0476.

[5] A. Pagh and R. Pagh. Scalable computation of acyclic joins. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 225–232, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-318-2.

[6] D. E. Willard. Efficient processing of relational calculus expressions using range query theory. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 164–175, New York, NY, USA, 1984. ACM. ISBN 0-89791-128-8. doi: http://doi.acm.org/10.1145/602259.602281.

[7] D. E. Willard. Quasilinear algorithms for processing relational calculus expressions (preliminary report). In *PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 243–257, New York, NY, USA, 1990. ACM. ISBN 0-89791-352-3. doi: http://doi.acm.org/10.1145/298514.298570.

[8] M. Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94. IEEE Computer Society, 1981.

[9] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. ISSN 1549-6325. doi: http://doi.acm.org/10.1145/1077464.1077466.