Towards a Worst-Case I/O-Optimal Algorithm for Acyclic Joins[*]

Xiao Hu Ke Yi

Hong Kong University of Science and Technology
{xhuam, yike}@cse.ust.hk

ABSTRACT

Nested-loop join is a worst-case I/O-optimal algorithm for 2 relations. Recently, a lot of efforts have been devoted to the "triangle query", for which an I/O-optimal algorithm is known. This paper extends these results to a fairly large class of acyclic joins. Acyclic joins can be computed optimally in internal memory using Yannakakis' algorithm from 1981, which simply performs a series of pairwise joins. However, no pairwise join algorithm can be I/O-optimal beyond 2 relations. To achieve I/O-optimality, the algorithm has to handle all the intermediate results carefully without writing them to disk. Unlike the optimal internal memory join algorithm which has a nice tight bound (the AGM bound), the I/O-complexity of joins turns out to be quite complex or even unknown. Yet, we are able to prove that our algorithm is I/O-optimal for certain classes of acyclic joins without deriving its bound explicitly.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Relational databases; F.2.2 [**Analysis of algorithms and problem complexity**]: Non-numerical Algorithms and Problems

Keywords

I/O-efficient algorithms, acyclic joins, worst-case optimal

1. INTRODUCTION

Evaluating join queries is one of the most central problems in relational databases, both in theory and practice. Yet surprisingly, the worst-case complexity of join evaluation has started to be unraveled just recently, largely thanks to the work of Atserials, Grohe, and Marx [2], who gave a worst-case bound on the join size. This then led to worst-case optimal[1] join algorithms [9, 13]. Ngo, Ré, and Rudra [10] presented a nice survey of these results, and also gave a simpler and unified proof for both the AGM bound and the running time of the algorithm.

These new optimal join algorithms crucially rely on retrieving tuples from a hash table, so they do not work well in external memory. On the other hand, most standard join algorithms used in existing database systems, like nested-loop join or sort-merge join, are specifically designed for limited main memory. Indeed, they achieve I/O-optimality for 2-table joins (more details to follow). Naturally, the intriguing question is if we can extend these algorithms to a broader class of joins in the external memory model.

1.1 Problem definition

Formally, a (natural) *join* is a triple $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$, where $\mathcal{V}$ is a set of *attributes*, $\mathcal{E} \subseteq 2^{\mathcal{V}}$ is a set of *relations*, and N is a function mapping each relation $e \in \mathcal{E}$ to a positive integer $N(e)$. Let $\text{dom}(v)$ be the *domain* of attribute $v \in \mathcal{V}$. An *instance* of $\mathcal{Q}$ is a function R that maps each $e \in \mathcal{E}$ to a set of tuples $R(e)$ with $|R(e)| \leq N(e)$, where each tuple $t \in R(e)$ specifies a value in $\text{dom}(v)$ for every attribute $v \in e$. The *join results* of $\mathcal{Q}$ on R, denoted by $\mathcal{Q}(R)$, consist of all combinations of tuples, one from each $R(e)$, that share common values for their common attributes. In this paper we assume that the query has constant size, namely, we consider the *data complexity* of computing the join. Indeed, the problem is intractable in terms of query size since $|\mathcal{Q}(R)|$ is exponential in $|\mathcal{E}|$ in the worst case. When the relations have subscripts as $e_1, e_2, \dots$, we use N_i, R_i, x_i, etc., as shorthands for $N(e_i), R(e_i), x(e_i)$, etc.

We use the standard external memory model [1], which has a main memory of size M with disk block size B. For simpler presentation, we will assume that the memory size is $c \cdot M$ for a sufficiently large constant c, which will not change our results by more than a constant factor. Note that since we assume a constant query size, the sizes of the tuples do not matter asymptotically, and we may assume $O(M)$ tuples from any relation fit in memory. Here are some standard bounds in this model: scanning N tuples needs $O(\frac{N}{B})$ I/Os, and sorting N tuples can be done in $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/Os. In this paper, we will ignore this log factor, and simply say that both can be done in $\widetilde{O}(\frac{N}{B})$ I/Os, where the $\widetilde{O}$ notation suppresses possibly one $\log_{\frac{M}{B}} \frac{N}{B}$ factor. For simplicity, we

[*]This work is supported by HKRGC under grants GRF-621413, GRF-16211614, and GRF-16200415.

[1]Henceforth "optimal" will always mean "worst-case optimal"; it will be stated explicitly otherwise, in particular when used together with other notions of optimality like "instance optimal".

PODS'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4191-2/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2902251.2902292

Join query	internal memory [2]	external memory	optimality	reference
Two relations	N_1N_2	$\frac{N_1N_2}{M^2} \cdot \frac{M}{B}$	yes	trivial
[1]Triangle C_3	$\sqrt{N_1N_2N_3}$	$\sqrt{\frac{N_1N_2N_3}{M^3}} \cdot \frac{M}{B}$	on equal N_i's	[7, 12]
[1]LW join LW_n	$\prod_i N_i^{\frac{1}{n-1}}$	$\prod_i (\frac{N_i}{M})^{\frac{1}{n-1}} \cdot \frac{M}{B}$	unknown	[6]
[2]General cyclic join	$\prod_i N_i^{x_i}$	open		
Line L_3	N_1N_3	$\frac{N_1N_3}{M^2} \cdot \frac{M}{B}$	yes	new
Line L_4	$\min\{N_1N_2N_4, N_1N_3N_4\}$	$\min\{\frac{N_1N_2N_4}{M^3}, \frac{N_1N_3N_4}{M^3}\} \cdot \frac{M}{B}$	yes	new
Line $L_n, n \geq 5$	$\prod_i N_i^{x_i}$	complex	yes for $n \leq 8$ on balanced N_i's for all n	new
Star T_n	$\prod_{i=1}^n N_i$	complex	yes	new
General acyclic join	$\prod_i N_i^{x_i}$	complex	varies	new
[3]Acyclic join with equal $N_i = N$	N^c	$(\frac{N}{M})^c \cdot \frac{M}{B}$	yes	new

Table 1: Worst-case complexity of join algorithms in internal and external memory

[1] The bounds listed in the table are actually equivalent to the AGM bound: it can be verified that when the AGM bound is smaller than $\prod N_i^{\frac{1}{n-1}}$, then $\prod N_i^{\frac{1}{n-1}} \leq \sum N_i$, and the running time will be dominated by the omitted linear term.

[2] The x_i's are the optimal fractional edge cover.

[3] $c = \sum x_i$ is the edge cover number.

will assume $N(e) \geq M$ for every e; otherwise proper ceiling will have to be carefully added to the I/O bounds.

For each join result, the algorithm should call an *emit* function with all the participating tuples, which must reside in memory at the time of the call, but does not have to write the result to disk. This agrees with most prior work on I/O-efficient join algorithms [6, 7, 12], and also reflects how join results are usually consumed in real database systems: they are often directly fed to a process down in the pipeline; sometimes the user just wants to apply an aggregation function (standard or an UDF) on the join results in a streaming fashion. For lower bounds, we apply the same *emit* requirement, and assume that the tuples are *indivisible*, which is a standard assumption in the external memory model. The indivisibility assumption is often justified by the fact that the user may be interested in the tuples as a whole entity, e.g., the user may need the tuples' other attributes that do not participate in the join.

As with prior work on acyclic joins [14, 11], we assume that the relations are *fully reduced*, i.e., there are no "dangling" tuples. More precisely, for any relation $R(e)$ and any attribute $v \in e$, for each value $a \in \text{dom}(v)$, there is at least one tuple $t \in R(e)$ such that t has value a on attribute v. For acyclic joins, the relations can be fully reduced in linear time and linear I/Os (ignoring a log factor) in external memory, using a series of semijoins, as described in [14]. In the rest of the paper, we omit the linear term when stating the running time or I/O cost of algorithms for brevity.

1.2 Previous results

In internal memory, there is a unified optimal algorithm [9, 13], based on the AGM bound [2], for all join queries. The AGM bound is based on the optimal fractional edge cover of the query hypergraph (details given in Section 2.1). It can be simplified when considering specific join queries, as listed in Table 1.

However, the situation is much more complicated in external memory. We know that nested-loop join uses $O(\frac{N_1N_2}{MB})$ I/Os to join two relations of sizes N_1 and N_2. Since $|\mathcal{Q}(R)| = N_1N_2$ in the worst case, and the B tuples brought in by one disk I/O can join with at most $O(M)$ tuples that are currently in memory, this bound is optimal.

The *triangle query* has received particular attention [7, 12], where $\mathcal{V} = \{v_1, v_2, v_3\}, e_1 = \{v_1, v_2\}, e_2 = \{v_1, v_3\}, e_3 = \{v_2, v_3\}$. An I/O-optimal algorithm is known, but only when $N_1 = N_2 = N_3$. Hu, Qiao, and Tao [6] extended this algorithm to *Loomis-Whitney joins*, a very special class of cyclic joins, but its optimality is still unclear, even when all relations have equal size.

Most database schemas are acyclic, due to its many desirable properties [3]. Indeed, in as early as 1981, Yannakakis [14] already gave an internal memory algorithm that evaluates any acyclic join in $O(|\mathcal{Q}(R)|)$ time, which is clearly optimal (in fact, instance optimal). This algorithm easily extends to external memory, as observed in [11], yielding an algorithm with $\widetilde{O}\left(\frac{|\mathcal{Q}(R)|}{B}\right)$ I/Os. This bound, however, is only optimal when the join results have to be written out to disk. In the emit model, it is even worse than a simple nested-loop join by a factor of M in the worst case for a 2-relation join, and the optimality gap gets even larger when more relations are joined, as we will see shortly.

After the relations have been fully reduced, Yannakakis' algorithm simply performs a series of pairwise joins between two relations sharing a common attribute, as in standard join processing in a database system. In internal memory, this works because these intermediate join sizes are no larger than the full join size. In external memory, however, we cannot afford to write any intermediate results in full to disk, so we will have to depart from this pairwise framework in order to achieve I/O-optimality. Note that simply pipelining the intermediate results to the next join operator will not work, as that join operator may use a nested-loop join, which requires reading one of its input relations multiple times. Thus, careful handling of the intermediate results with limited memory is a key challenge for an external memory algorithm.

1.3 Acyclicity

The acyclicity of the join $\mathcal{Q}$ is defined with respect to the hypergraph $(\mathcal{V}, \mathcal{E})$. Unlike ordinary acyclic graphs (i.e., trees or forests), there are several notions of acyclicity for hypergraphs. We adopt the most natural definition, given by Berge [4]. Consider the bipartite graph G, in which $\mathcal{V}$ corresponds to vertices on one side and $\mathcal{E}$ to vertices on the other side. There is an edge between $v \in \mathcal{V}$ and $e \in \mathcal{E}$ if $v \in e$. Then the hypergraph $(\mathcal{V}, \mathcal{E})$ is said to be acyclic if this bipartite graph is acyclic. This notion of acyclicity preserves many natural properties in ordinary acyclic graphs. For example, there is only one path between any two vertices $u, v \in \mathcal{V}$, and any subgraph of $(\mathcal{V}, \mathcal{E})$ is still acyclic. Note that this definition of acyclicity does not allow two relations to have two or more common attributes. But if these attributes always appear together in any relation, then they can be simply considered as one "combined" attribute.

We do point out that earlier work on acyclic queries, including Yannakakis' algorithm, has adopted a more relaxed notion of acyclicity, known as *α-acyclicity*. However, α-acyclicity does not preserve the two properties above. In particular, removing a hyperedge from an α-acyclic hypergraph can make it cyclic, which is quite counter-intuitive. In particular, two most commonly used types of joins, *line joins* and *star joins*, are both Berge-acyclic, to which we will pay special attention. Henceforth, "acyclic" will always mean "Berge-acyclic".

1.4 Our results

We observe from Table 1 that all existing external memory results "just" replace N_i with $\frac{N_i}{M}$ in their internal memory counterparts, plus an additional $\frac{M}{B}$ factor. Thus it is natural to make the same conjecture for other join queries. Indeed, our result on the line query L_3 confirms this, and it also trivially extends to L_4 (see Table 1). Recall that a line query L_n on n relations is a $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$, where $\mathcal{V} = \{v_1, v_2, ..., v_{n+1}\}$, $\mathcal{E} = \{e_1, e_2, ..., e_n\}$, and $e_i = \{v_i, v_{i+1}\}$. Interestingly, this conjecture breaks down on L_5. The AGM bound on L_5 is $N_1N_3N_5$, but its external memory counterpart, $\frac{N_1N_3N_5}{M^2B}$, is not sufficient to characterize the query's I/O complexity. It is easy to construct an instance in which every tuple in R_1 joins with every tuple in R_4, thus applying the same argument as for the 2-relation join yields a lower bound of $\Omega(\frac{N_1N_4}{MB})$. Note that we always have $N_1N_4 \le N_1N_3N_5$ (on fully reduced instances), but due to having different denominators, $\frac{N_1N_4}{MB}$ is in general not comparable to $\frac{N_1N_3N_5}{M^2B}$. The complication associated with the external memory model starts to manifest itself. A similar situation happens on star queries with at least 3 petals, namely, a part of the join results could impose a higher lower bound on the I/O complexity than the full join results.

We will need the following two concepts in order to state our results for those queries marked as "complex" in Table 1. For any subset of relations $S \subseteq \mathcal{E}$, we call $\Join_{e \in S} R(e)$ the *subjoin* on S. On the other hand, define the *partial join* $\mathcal{Q}(R, S)$ as the projection of $\mathcal{Q}(R)$ on the attributes of S. Note that if the relations in S are connected, $\mathcal{Q}(R, S) = \Join_{e \in S} R(e)$ for an acyclic join on fully reduced relations. If S is not connected, then $\mathcal{Q}(R, S) \subseteq \Join_{e \in S} R(e)$. Recall that the natural join of two relations without a common attribute is simply their cross product.

Figure 1 shows an instance of a line query L_3, illustrating these concepts. In the figure, attribute values are vertices, and each tuple is an edge. Thus, any path from some vertex in A to some vertex in D is a join result. Note that these relations have been fully reduced. In this example, the partial join and subjoin on R_1 and R_2 are the same, since they are connected, and includes every partial path from A to C. The subjoin on R_1 and R_3 is simply their cross product: $R_1 \Join R_3 = R_1 \times R_3$, but the partial join on R_1 and R_3 is only a subset of $R_1 \times R_3$. For example, (t_1, t_2) belongs to both the subjoin and the partial join, but (t_1, t_3) only belongs to the subjoin but not the partial join.

Because the partial join $\mathcal{Q}(R, S)$ is a projection of the full join, the algorithm has to compute $\mathcal{Q}(R, S)$ for any S implicitly or explicitly. In internal memory, $|\mathcal{Q}(R, S)| \le |\mathcal{Q}(R)|$, so this is not an issue. However, in external memory, applying the same argument above leads to a lower bound of $\Omega\left(\frac{|\mathcal{Q}(R,S)|}{M^{|S|-1}B}\right)$. Each partial join contributes a term that is in general not comparable with others, and all of them should be taken into account to give the highest lower bound.

With this observation, we define

$$\psi(R, S) = \frac{|\mathcal{Q}(R, S)|}{M^{|S|-1}B}$$

as the minimum I/O cost for computing the partial join $\mathcal{Q}(R, S)$. Thus $\max_R \max_S \psi(R, S) = \max_S \max_R \psi(R, S)$ is a lower bound on the worst-case I/O cost for computing $\mathcal{Q}$.

Similarly, define

$$\Psi(R, S) = \frac{|\Join_{e \in S} R(e)|}{M^{|S|-1}B}$$

as the minimum I/O cost for computing the subjoin $\Join_{e \in S} R(e)$.

The main result of this paper is an algorithm for any acyclic join $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$, whose I/O cost on instance R is

$$\min_{\mathbf{S} \in \textsc{GenS}(\mathcal{Q})} \max_{S \in \mathbf{S}} \Psi(R, S),$$

where $\mathbf{S}$ is any set of subsets of $\mathcal{E}$ generatable by a nondeterministic process $\textsc{GenS}(\mathcal{Q})$, as described in Algorithm 3. Note that $\textsc{GenS}(\mathcal{Q})$ only depends on the structure of the query hypergraph.

The algorithm's worst-case I/O-cost is thus

$$\max_R \min_{\mathbf{S} \in \textsc{GenS}(\mathcal{Q})} \max_{S \in \mathbf{S}} \Psi(R, S), \tag{1}$$

which (for a given query hypergraph) is a function of the relation sizes, as well as M and B. Unfortunately, this function is very complex. To prove worst-case optimality, we make a connection to the partial joins directly, thus avoiding the

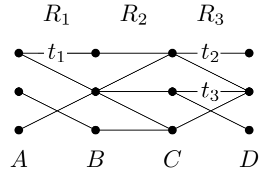


Figure 1: Illustration on subjoins and partial joins.

need to derive this function explicitly. We first rewrite (1):

$$\max_R \min_{\mathbf{S}\in\text{GENS}(\mathcal{Q})} \max_{S\in\mathbf{S}} \Psi(R,S) \le \min_{\mathbf{S}\in\text{GENS}(\mathcal{Q})} \max_R \max_{S\in\mathbf{S}} \Psi(R,S) = \min_{\mathbf{S}\in\text{GENS}(\mathcal{Q})} \max_{S\in\mathbf{S}} \max_R \Psi(R,S).$$

Then, it suffices to show that there exists an $\mathbf{S} \in \text{GENS}(\mathcal{Q})$ such that for any $S \in \mathbf{S}$, there is an instance I on which

$$\max_R \Psi(R,S) \le \psi(I,S),$$

or

$$\max_R |\bowtie_{e\in S} R(e)| \le |\mathcal{Q}(I,S)|. \quad (2)$$

Specifically, we obtain the following results (please also refer to Table 1):

(1) Our algorithm is optimal for any star join (Section 5).

(2) For any line join in which the relation sizes satisfy a certain *balancing condition*, the algorithm is optimal (Section 6). This condition is always satisfied on any 3-relation or 4-relation line join.

(3) For line joins with 5 or more relations, the balancing condition might break. In this case, our general algorithm is not optimal, but we have designed special algorithms that are optimal for line joins with up to 8 relations.

(4) In addition to line and star joins, our algorithm is also optimal on some other types of acyclic joins (Section 7).

(5) Our algorithm is optimal on any acyclic query in which all relations have equal size (Section 7). In this case, the I/O-complexity has a closed form $(\frac{N}{M})^c \cdot \frac{M}{B}$ where c is the edge cover number of the hypergraph. Very recently, Koutris et al. [8] have given an interesting reduction from their MPC model to the I/O model, which leads to algorithms that match this edge cover bound, but only for line joins and star joins on equal-sized relations.

2. PRELIMINARIES

2.1 The AGM bound

For a query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$, let x be a fractional edge cover for the hypergraph $(\mathcal{V}, \mathcal{E})$, i.e., for any $v \in \mathcal{V}$, $\sum_{e, v\in e} x(e) \ge 1$. Then the AGM bound [2] states that

$$\max_R |\mathcal{Q}(R)| = \min_x \prod_e N(e)^{x(e)}.$$

Note that the optimal fractional edge cover x can be found by solving a linear program on the $N(e)$'s, which takes $O(1)$ time for constant query size.

2.2 Properties of acyclic joins

Let $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$ be a join query whose corresponding hypergraph is acyclic. The following lemma can be easily proved by definition.

LEMMA 1. *In an acyclic query, there is always a relation that contains only one attribute, or an attribute that is contained in only one relation.*

This leads to the following important observation of the AGM bound on acyclic queries. Let x be the optimal fractional cover of $\mathcal{Q}$.

LEMMA 2. *For acyclic queries, $x(e) = 0$ or 1 for each e.*

PROOF. Let A be the incidence matrix of the linear program defined on x, which has a 1 at row e and column v if $v \in e$, and 0 otherwise. It is known that the optimal solution of the linear program is integral if the determinant of any square submatrix of A is ± 1 or 0. We prove this by induction on the size of the submatrix. For a 1×1 submatrix, this is trivially true. Suppose $\det(A') = \pm 1$ or 0 for any $k \times k$ submatrix A' of A. Consider any $(k+1) \times (k+1)$ submatrix A' of A, which corresponds to a subset of $k+1$ vertices $\mathcal{V}'$ and a subset of $k+1$ edges $\mathcal{E}'$ of the hypergraph. Since any sub-hypergraph of an acyclic hypergraph is acyclic, by Lemma 1 there is one relation in $\mathcal{E}'$ that contains only one attribute in $\mathcal{V}'$ or one attribute in $\mathcal{V}'$ that is contained in only one relation in $\mathcal{E}'$. In either case, A' can be rearranged as

$$A' = \begin{pmatrix} 1 & * \\ 0 & A'' \end{pmatrix},$$

where A'' is a $k \times k$ submatrix of A. Then by the induction hypothesis, $\det(A') = \pm 1 \cdot \det(A'') = \pm 1$ or 0. □

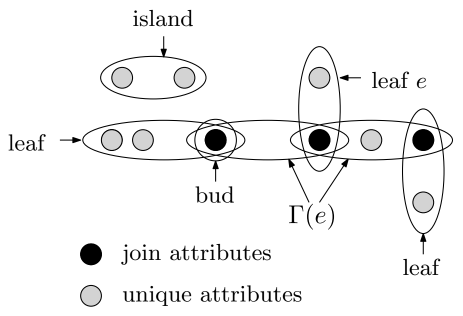


Figure 2: Attributes and edges in an acyclic hypergraph.

We classify the attributes and relations as follows. If an attribute $v \in \mathcal{V}$ appears in only one relation, it is called a *unique attribute*, otherwise a *join attribute*. Note that any two relations have at most one common join attribute due to the acyclicity requirement. A relation e is called an *island* if e has no join attribute. A *bud* is a relation with only one join attribute and no unique attribute. A relation e is called a *leaf* if it contains at least one unique attribute and exactly one join attribute. For a leaf e, let $\Gamma(e)$ be its *neighbors*, i.e., the set of other relations sharing the join attribute with e. These terms are illustrated in Figure 2.

By Lemma 1, there is always an island, a bud, or a leaf in an acyclic query. One may wonder why we consider islands and buds as they are not very meaningful in a join query. The reason, as we shall see later, is that they can appear during the recursive processing of the join in our algorithm, so we have to consider them for full generality.

2.3 Handling skew

Similar with the optimal join algorithms in internal memory [10], it is important to handle skew properly. In external memory, skew is defined with respect to the memory size M.

For a relation $R(e)$ and an attribute $v \in e$, the set of tuples in $R(e)$ with value a on attribute v is denoted as $R(e)|_{v=a}$, and let $N(e)|_{v=a} = |R(e)|_{v=a}|$.

A value a is *heavy* in $R(e)$ if $N(e)|_{v=a} \geq M$, otherwise *light*. Denote the set of heavy values in dom(v) with respect to $R(e)$ as

$$H(e, v) = \{a \in \text{dom}(v) : N(e)|_{v=a} \geq M\}.$$

Note that by sorting $R(e)$ on the v attribute, we can partition the tuples in $R(e)$ into those with heavy values on v and those with light values.

We will use the following operations in describing our algorithms. If $R(e)$ is sorted on v, we can handle the heavy and light values separately. For a heavy value a, the operation "load $R(e)|_{v=a}$ into memory as $M(e)$" means reading the next M tuples of $R(e)|_{v=a}$ into memory, and these tuples are denoted as $M(e)$. This operation is usually repeatedly invoked, and and last invocation may read less than M tuples from $R(e)|_{v=a}$. For light values, the operation "load $R(e)$ by v into memory as $M(e)$" means reading tuples from $R(e)$ by the order of v until M or more tuples are fetched, subject to the constraint that tuples with the same value on attribute v must be loaded into memory together. Note that since all values are light, no more than $2M$ tuples will be loaded to memory, with no more than M distinct values on v. This operation can also be repeatedly invoked, and the last invocation may read less than M tuples. If $R(e)$ is not sorted on any attribute, the operation "load $R(e)$ into memory as $M(e)$" means reading the next M tuples from $R(e)$ (again, the last invocation may read less than M tuples).

3. WARMING UP

Before presenting our general join algorithm, we start with a review on 2-relation join algorithms and show how we can extend them to the 3-relation line join.

For a join between 2 relations R_1 and R_2, we know that nested-loop join achieves $O(\frac{N_1 N_2}{MB})$ I/Os, which is worst-case optimal as $|\mathcal{Q}(R)|$ can be as large as $N_1 N_2$ in the worst case. In fact, by combining with sort-merge join, we can achieve instance optimality, i.e., an algorithm that runs in $\widetilde{O}(\frac{|\mathcal{Q}(R)|}{MB})$ I/Os on any instance R of $\mathcal{Q}$. More precisely, we sort both relations on the join attribute v, and then "merge" them together. For each value $a \in \text{dom}(v)$, if a is heavy in both R_1 and R_2, we run a nested-loop join on $R_1|_{v=a} \bowtie R_2|_{v=a}$, costing $O(\frac{N_1|_{v=a} N_2|_{v=a}}{MB})$ I/Os; otherwise the merge can be done with a single pass of $R_1|_{v=a}$ and $R_2|_{v=a}$. Note that $|\mathcal{Q}(R)| = \sum_a N_1|_{v=a} N_2|_{v=a}$, hence the total I/O cost is as claimed. In fact, this algorithm has been described in textbooks [5], but it appears that its (I/O) instance optimality has never been formally stated.

Next we consider a 3-relation line join L_3:

$$R_1(v_1, v_2) \bowtie R_2(v_2, v_3) \bowtie R_3(v_3, v_4).$$

The naive way to extend the nested-loop join to 3 relations would result in $O(\frac{N_1 N_2 N_3}{M^2 B})$ I/Os. However, by the AGM bound, the optimal edge cover for this query is $x_1 = 1, x_2 = 0, x_3 = 1$, thus its maximum join size is only $N_1 N_3$. Below, we present an algorithm that achieves $\widetilde{O}(\frac{N_1 N_3}{MB})$ I/Os (recall that we omit the linear term, so there is a hidden $\frac{N_2}{B}$ term).

We will make use of the 2-relation instance-optimal algorithm above. In fact, we will only need the observation that

Algorithm 1: Line join on 3 relations

```
1  sort $R_1$ by attribute $v_2$;
2  sort $R_2$ by attribute $v_2, v_3$ lexicographically;
3  sort $R_3$ by attribute $v_3$;
4  foreach $a \in H(e_1, v_2)$ do
5      compute $R_3' = R_2|_{v_2=a} \bowtie R_3$ by merge join, and
       write the results to disk;
6      compute $R_1|_{v_2=a} \bowtie R_3'$ by nested-loop join, and
       emit all results;
7      discard $R_1|_{v_2=a}$ and $R_2|_{v_2=a}$;
8  while load $R_1$ by $v_2$ into memory as $M_1$ do
9      compute $R_2(M_1) = R_2 \ltimes M_1$;
10     compute $R_2(M_1) \bowtie R_3$ by sort-merge join;
11     foreach tuple $t$ emitted do
12         find any matching tuples $t' \in M_1$ and emit$(t', t)$;
```

this algorithm has $\widetilde{O}(\frac{N_1}{B} + \frac{N_2}{B})$ I/Os when R_1 and R_2 have no common heavy values.

Our 3-relation line join algorithm is described in Algorithm 1. It handles heavy values (line 4–7) and light values (line 8–12) on attribute v_2 in R_1 separately. For each heavy value $a \in H(e_1, v_2)$, $R_2|_{v_2=a}$ is stored continuously on disk sorted by v_3. Since R_3 is sorted by v_3, we can compute $R_3' = R_2|_{v_2=a} \bowtie R_3$ by a merge join on line 5. Note that all tuples in $R_2|_{v_2=a}$ have the same value on v_2, they must have distinct values on v_3, so no value on v_3 is heavy. Thus by the observation above, this costs $O(\frac{N_2|_{v_2=a}}{B} + \frac{N_3}{B})$ I/Os. Also, R_3' has at most N_3 tuples, so we can afford to write the results to disk. Next, computing $R_1|_{v_2=a} \bowtie R_3'$ by nested-loop join on line 6 costs $O(\frac{N_1|_{v_2=a} N_3}{MB})$ I/Os (since $N_1|_{v_2=a} \geq M$, no $\lceil \cdot \rceil$ is needed). So the total cost to handle all the heavy values in R_1 is

$$\sum_{a \in H(e_1, v_2)} \left(\frac{N_2|_{v_2=a}}{B} + \frac{N_3}{B} + \frac{N_1|_{v_2=a} N_3}{MB} \right).$$

Since there are $O(\frac{N_1}{M})$ heavy values in $H(e_1, v_2)$, this becomes $O(\frac{N_1 N_3}{MB} + \frac{N_2}{B})$.

For tuples in R_1 with light values on v_2, we load them into memory, one chunk at a time. From previous analysis, each memory chunk contains at most $2M$ tuples with at most M distinct values on v_2. Since R_1 and R_2 are both sorted by v_2, computing all the semijoins on line 9 takes just one scan through R_1 and R_2. Since M_1 has no more than M distinct values on v_2, $R_2(M_1)$ does not have any heavy value on v_3. Then by the previous observation, doing a sort-merge on $R_2(M_1) \bowtie R_3$ on line 10 has cost $\widetilde{O}(\frac{|R_2(M_1)|}{B} + \frac{N_3}{B})$. Thus, the total cost to handle all the light values is

$$\widetilde{O}\left(\frac{N_1}{B} + \frac{N_2}{B} + \sum_{M_1} \left(\frac{|R_2(M_1)|}{B} + \frac{N_3}{B} \right) \right).$$

Since there are $O(\frac{N_1}{M})$ memory chunks M_1, and the $R_2(M_1)$'s are disjoint, this becomes $\widetilde{O}(\frac{N_1 N_3}{MB} + \frac{N_2}{B})$.

THEOREM 1. *Algorithm 1 has I/O cost $\widetilde{O}(\frac{N_1 N_3}{MB})$ for any 3-relation line join.*

To see why this is optimal, simply consider the instance in Figure 3. Taking $S = \{R_1, R_3\}$, we have $\psi(R, S) = \frac{N_1 N_3}{MB}$ as a lower bound.

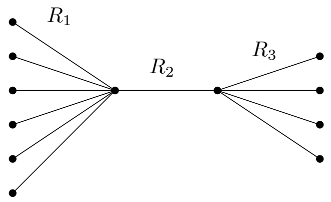


Figure 3: Worst-case instance for 3-relation line join.

4. ALGORITHM FOR ACYCLIC JOINS

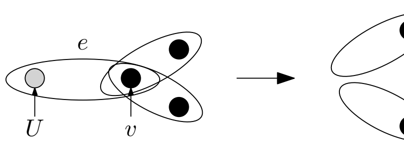
4.1 The algorithm

In this section, we generalize our 3-relation line join algorithm to handle arbitrary acyclic queries. The algorithm, described in Algorithm 2, recursively "peels off" a bud, an island, or a leaf.

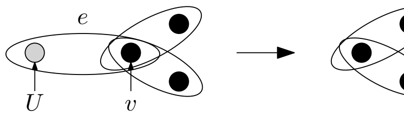
The base case is when only one relation remains, in which case we simply report all tuples (line 1–2). A bud can simply be ignored (line 3–4), as it has only one attribute and it appears in other relations. Since there are no dangling tuples, a bud cannot restrict other relations, either. Strictly speaking, however, ignoring a bud violates the emit model. But this can be easily fixed by attaching each tuple t in the bud to all tuples it joins with. This takes linear I/Os in total, so it is not an issue.

An island e can be peeled off by just using nested-loop join with $R(e)$ being the "outer relation" and the rest of $\mathcal{Q}$ as the "inner relation" (line 5–9). The rationale is that any tuple in $R(e)$ joins with any join result from the rest of $\mathcal{Q}$, so the I/O cost inevitably will have $\frac{N(e)}{M}$ multiplied with the cost of computing the rest of $\mathcal{Q}$.

To peel a leaf e, the idea is similar to the 3-relation line join algorithm, handling heavy (line 14–20) and light tuples (line 21–27) in $R(e)$ separately. There are two key differences, though. First, computing the join on (parts of) R_2 and R_3 will be replaced by recursion. Second, for a general acyclic join, a leaf e may share its join attribute with more than one relations, namely, there can be more than one "R_2".



Dealing with heavy tuples in $R(e)$



Dealing with light tuples in $R(e)$

Figure 4: Different strategies for dealing with heavy and light tuples.

We provide some more explanation on the algorithm for peeling off a leaf. First, a seemingly difficult decision is which leaf to peel. Indeed, different pealing strategies may lead to different costs. For example, on L_4, we will see later that different peeling strategies would lead to two different bounds, $\widetilde{O}(\frac{N_1N_3N_4}{M^3B})$ and $\widetilde{O}(\frac{N_1N_2N_4}{M^3B})$, respectively. Thus, a "smart" algorithm should first compare N_2 and N_3, and then choose the better peeling strategy. However, as we are not trying to optimize the dependency on n, the number of relations, we chose to simply pick a leaf e nondeterministically (line 11). To convert this nondeterministic algorithm to a deterministic one, we use the standard simulation technique to explore all nondeterministic branches in a round-robin fashion, terminating the simulation as soon as any branch terminates. In effect, we attain the cost of the best peeling strategy, up to a factor that might be exponential in n, which is a constant anyway.

Next, consider a peeling a leaf e. Let U be its set of unique attributes, v its join attribute, and Γ the set of neighboring relations of e. For a heavy value a, we first find $R(e')|_{v=a}$ for each $e' \in \Gamma$, i.e., the tuples in Γ that join with a. Now we can remove both e and v from $\mathcal{Q}$, as all tuples have the same value a on v. In the case Γ has more than one relation, this will decompose $\mathcal{Q}$ into disconnected components (see Figure 4). Note that this operation may generate new islands and buds, which is exactly why we need to deal with islands and buds in each recursive call. Then, for each memory chunk of $R(e)|_{v=a}$, we do the join recursively; for each

Algorithm 2: AcyclicJoin($\mathcal{V}, \mathcal{E}, R$)

```
1  if $\mathcal{E} = \{e\}$ then
2      emit all tuples in $R(e)$;
3  else if there is a bud $e \in \mathcal{E}$ then
4      AcyclicJoin($\mathcal{V}, \mathcal{E} - \{e\}, R$);
5  else if there is an island $e \in \mathcal{E}$ then
6      while load $R(e)$ into memory as $M_1$ do
7          acyclicJoin($\mathcal{V} - e, \mathcal{E} - \{e\}, R - \{R(e)\}$);
8          foreach tuple $t$ emitted do
9              find any matching tuples $t' \in M_1$ and
               emit($t', t$);
10 else
11     pick a leaf $e$ nondeterministically;
12     suppose $e$ has unique attributes $U$, join attribute $v$,
       and neighboring relations $\Gamma$;
13     sort $R(e)$ by attribute $v$;
14     sort $R(e')$ by attribute $v$ for each $e' \in \Gamma$;
15     foreach $a \in H(e, v)$ do
16         $R'(a) \leftarrow R - \{R(e)\} - \{R(e') : e' \in \Gamma\}$
           $+\{R(e')|_{v=a} : e' \in \Gamma\}$;
17         while load $R(e)|_{v=a}$ into memory as $M_1$ do
18             acyclicJoin($\mathcal{V} - U - \{v\}, \mathcal{E} - \{e\}, R'(a)$);
19             foreach tuple $t$ emitted do
20                 emit($t', t$) for each tuple $t' \in M_1$;
21         discard $R(e)|_{v=a}$ and $R(e')|_{v=a}$;
22     while load $R(e)$ by $v$ into memory as $M_1$ do
23         foreach $e' \in \Gamma$ do
24             compute $R(e')(M_1) = R(e') \ltimes M_1$;
25         $R'(M_1) \leftarrow R - \{R(e)\} - \{R(e') : e' \in \Gamma\}$
           $+\{R(e')(M_1) : e' \in \Gamma\}$;
26         AcyclicJoin($\mathcal{V} - U, \mathcal{E} - \{e\}, R'(M_1)$);
27         foreach tuple $t$ emitted do
28             finding any matching tuples $t' \in M_1$ and
               emit($t', t$);
```

join result returned from the recursion, we combine it with all the tuples in memory since they have the same value a on the join attribute v.

For light tuples in $R(e)$, we load them into memory, one chunk M_1 at a time. Then for each $e' \in \Gamma$, we find $R(e')(M_1)$, i.e., the tuples in $R(e)$ that can join with M_1. Next, we make the recursive call. Note that in this case, we only remove e and its unique attributes, but not v, before going into the recursion, so this does not disconnect the query (see Figure 4).

4.2 Analysis

We will bound the I/O-cost of Algorithm 4 in terms of $\Psi(R, S)$, the minimum cost of computing the subjoins. The first bound, stated below, uses all subsets S of the relations.

THEOREM 2. *For an acyclic join $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$ and instance R, the I/O cost of Algorithm 2 is*

$$\widetilde{O}\left(\max_{S \subseteq \mathcal{E}} \Psi(R, S)\right), \tag{3}$$

where S is over all subsets of $\mathcal{E}$.

The proof is by induction and naturally follows the recursion of the algorithm. The details are quite technical, though, and are given in Appendix A.1.

It is not hard to see that this bound cannot be tight. In particular, it does not depend on the peeling order, which means that this is actually a bound on the worst branch of the nondeterministic peeling process. The other source of looseness comes from a *star*. A *star* consists of a *core* e_0 that has no unique attributes, and k *petals* $e_1, \ldots, e_k, k \geq 1$, such that $e_i \cap e_0 \neq \emptyset$. Each petal contains one or more unique attributes and does not intersect with any relations except the core. The core connects with the rest of $\mathcal{Q}$ via exactly one join attribute. Please see Figure 5 for an illustration.

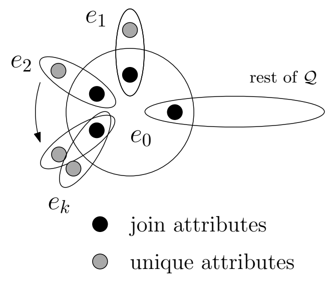


Figure 5: An illustration of a star.

When applied to a standalone star, Theorem 2 would generate all subsets of its relations. However, an observation is that we will always have

$$\Psi(R, \{e_0, e_1, \ldots, e_k\}) \leq \Psi(R, \{e_1, \ldots, e_k\}),$$

since $R_0 \bowtie R_1 \bowtie \cdots \bowtie R_k \subseteq R_1 \bowtie \cdots \bowtie R_k$ and $\Psi(R, \{e_0, e_1, \ldots, e_k\})$ has a larger denominator. This means that not all subsets S of relations have to be included in (3), and more importantly, this observation applies recursively. Specifically, whenever Algorithm 2 successively peels off the petals of a star, followed by its core, we can apply this argument: If the core is included in the recursive consideration, then not all its petals have to be included; all the petals have to be included in the same subjoin only if the core is not included.

This is more precisely defined by a nondeterministic recursive process GENS($\mathcal{Q}$) as described in Algorithm 3, which leads to the following tighter bound:

Algorithm 3: GENS($\mathcal{Q}$)

```
1  if $\mathcal{Q}$ is empty then
2      return $\{\emptyset\}$;
3  if $\mathcal{Q}$ contains a bud $e$ then
4      return GENS($\mathcal{Q} - \{e\}$);
5  else if $\mathcal{Q}$ contains a star then
6      pick a star $\mathcal{X}$ non-deterministically;
7      let $e_0$ be the core of $\mathcal{X}$;
8      $\boldsymbol{S_1} \leftarrow$ GENS($\mathcal{Q} - \mathcal{X} + \{e_0\}$);
9      $\boldsymbol{S_1} \leftarrow \boldsymbol{S_1} \cup \{S \cup f \mid S \in \boldsymbol{S_1}, f \subsetneq \mathcal{X} - \{e_0\}\}$;
10     $\boldsymbol{S_2} \leftarrow$ GENS($\mathcal{Q} - \mathcal{X}$);
11     $\boldsymbol{S_2} \leftarrow \boldsymbol{S_2} \cup \{S \cup f \mid S \in \boldsymbol{S_2}, f \subseteq \mathcal{X} - \{e_0\}\}$;
12     return $\boldsymbol{S_1} \cup \boldsymbol{S_2} \cup 2^{\mathcal{X}}$;
13 else
14     pick an island $e$ or a leaf $e$ non-deterministically;
15     $\mathbf{S} \leftarrow$ GENS($\mathcal{Q} - \{e\}$)};
16     return $\mathbf{S} \cup \{S \cup \{e\} \mid S \in \mathbf{S}\}$;
```

THEOREM 3. *For an acyclic join $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$ and instance R, the I/O cost of Algorithm 2 is*

$$\widetilde{O}\left(\min_{\mathbf{S} \in \text{GENS}(\mathcal{Q})} \max_{S \in \mathbf{S}} \Psi(S)\right), \tag{4}$$

where $\mathbf{S}$ is any set of subsets of $\mathcal{E}$ generatable by Algorithm 3.

The proof of Theorem 3 is given in Appendix A.2. Similar to the proof of Theorem 2, it is based on induction and the recursive structure of the algorithm. More precisely, we show that the I/O cost of any branch of the nondeterministic algorithm is captured by the corresponding branch of the nondeterministic GENS($\mathcal{Q}$) process, therefore achieving the I/O cost of the best branch by the round-robin simulation.

Here are some examples when applying Theorem 3 to some join queries.

Line query L_3: On L_3, we can either consider $\{e_1, e_2\}$ as a star (with only one petal), or $\{e_2, e_3\}$ as a star. Suppose we peel $\{e_1, e_2\}$ first. Then GENS($\mathcal{Q}$) will be recursively applied on $\{e_2, e_3\}$ and $\{e_3\}$, respectively. GENS($\{e_2, e_3\}$) will return all its subsets, and GENS($\{e_3\}$) returns $\{\{e_3\}, \emptyset\}$, each of which is unioned with $\{e_1\}$. Thus, the final $\mathbf{S}$ returned by GENS($\mathcal{Q}$) is

$$\{\{e_1, e_3\}, \{e_2, e_3\}, \{e_1, e_2\}, \{e_1\}, \{e_2\}, \{e_3\}, \emptyset\}.$$

It can be verified that if GENS($\mathcal{Q}$) peels $\{e_2, e_3\}$ first, it will generate the same $\mathbf{S}$.

On a fully reduced instance R, $\Psi(R, \{e_1, e_2\}), \Psi(R, \{e_2, e_3\})$ are dominated by $\Psi(R, \{e_1, e_3\})$, so the I/O-cost of the algorithm is $\widetilde{O}(\Psi(R, \{e_1, e_3\}) + \Psi(R, \{e_2\}))$, matching the result of Theorem 1.

Line query L_4: On L_4, GENS($\mathcal{Q}$) generates different $\mathbf{S}$ on different peeling orders. If we peel the star $\{e_1, e_2\}$ first, GENS($\mathcal{Q}$) will return (subjoins that are dominated by others are omitted)

$$\{\{e_1, e_3, e_4\}, \{e_1, e_3\}, \{e_1, e_4\}, \{e_2, e_4\}\}.$$

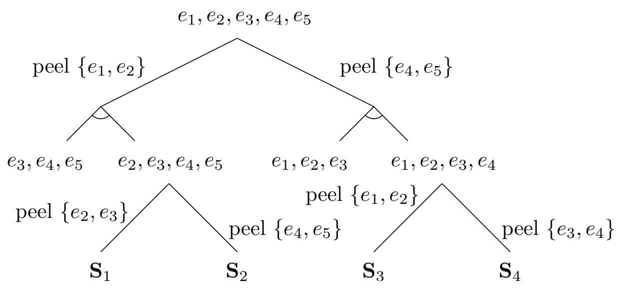


Figure 6: Applying $\textsc{GenS}(\mathcal{Q})$ **on** L_5.

If we peel $\{e_3, e_4\}$ first, $\textsc{GenS}(\mathcal{Q})$ will return

$$\{\{e_1, e_2, e_4\}, \{e_1, e_3\}, \{e_1, e_4\}, \{e_2, e_4\}\}.$$

Which one leads to a smaller bound will depend on all these subjoin sizes on the given instance. In the worst case, the former is dominated by $\Psi(R, \{e_1, e_3, e_4\}) = \frac{N_1 N_3 N_4}{M^2 B}$ and the latter dominated by $\Psi(R, \{e_1, e_2, e_4\}) = \frac{N_1 N_2 N_4}{M^2 B}$, so the I/O cost is as claimed in Table 1.

Line query L_5**:** On L_5, we can either peel off $\{e_1, e_2\}$ or $\{e_4, e_5\}$. In either case, $\textsc{GenS}(\mathcal{Q})$ generates two recursive calls, one on an L_3 and one on an L_4 (see Figure 6). As seen above, $\textsc{GenS}(\mathcal{Q})$ on an L_3 will generate a unique result, but can generate two different results on an L_4, so there are a total of 4 $\mathbf{S}$'s generatable by $\textsc{GenS}(\mathcal{Q})$ on L_5 (dominated subjoins are omitted):

$$\mathbf{S}_2 = \mathbf{S}_3 = \{\{e_1, e_3, e_5\}, \{e_2, e_5\}, \{e_1, e_4\}, \{e_2, e_4\}\}$$
$$\mathbf{S}_1 = \{\{e_1, e_3, e_5\}, \{e_2, e_4, e_5\}, \{e_2, e_5\}, \{e_1, e_4\}, \{e_2, e_4\}\}$$
$$\mathbf{S}_4 = \{\{e_1, e_3, e_5\}, \{e_1, e_2, e_4\}, \{e_2, e_5\}, \{e_1, e_4\}, \{e_2, e_4\}\}$$

Thus, two of the four peeling strategies are better than the others, and in terms of the worst case, they give a bound of $\widetilde{O}(\frac{N_1 N_3 N_5}{M^2 B} + \frac{N_2 N_5}{MB} + \frac{N_1 N_4}{MB} + \frac{N_2 N_4}{MB})$, which will be shown to be optimal in Section 6. This is an example where the I/O-complexity is not simply replacing N_i with $\frac{N_i}{M}$ in the AGM bound.

Star query: If $\mathcal{Q}$ by itself is a standalone star, we can remove it in one shot, which would generate all subjoins. However, we could also remove all but one petal, resulting in all subjoins except the full join, which agrees with our earlier observation that the full join is dominated by the subjoin defined by all petals, thus not needed.

5. OPTIMALITY ON STAR JOINS

On a star query with n petals, our analysis above immediately yields the following result.

Corollary 1. *On a star join $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$ and an instance R, Algorithm 2 has I/O cost*

$$\widetilde{O}\left(\frac{\prod_{i=1}^{n} N_i}{M^{n-1}B} + \max_{S \subsetneq \mathcal{E}, e_0 \in S} \psi(R, S)\right) \tag{5}$$

Proof. As argued, the I/O cost has a term $\Psi(R, S)$ for each proper subset S of $\mathcal{E}$. If S does not include e_0, this is captured by the first term in (5). If $e_0 \in S$, then the subjoin becomes a partial join, thus captured by the second term of (5). □

Theorem 4. *Algorithm 2 is worst-case optimal for any star join $\mathcal{Q}$.*

Proof. As the second term in (5) is already optimal (actually instance-optimal). To show the first term is optimal, it suffices construct an instance I such that its partial join size on the n petals is $\prod_{i=1} N_i$. We construct I by generalizing Figure 3: The domain of each join attribute v_i has only one value, and petal R_i is a one-to-many matching from the only value in $\mathrm{dom}(v_i)$ to the unique attribute of R_i. The core simply consists of a single tuple connecting all the (only) values in $\mathrm{dom}(v_i), i = 1, \ldots, n$. □

Although we have proved the optimality of our algorithm on star joins without deriving its complexity explicitly, it is still an interesting question as to what this complexity looks like, i.e., what is the second term of (5) on the worst R? The answer turns out to be a complicated function, intricately depending on N_0, the size of the core of the star, as compared to the sizes of the petals.

6. OPTIMALITY ON LINE JOINS

The optimality of Algorithm 2 on line joins is much more subtle than on star joins. Recall that a line join on n relations, L_n, is a query $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$, where $\mathcal{V} = \{v_1, v_2, ..., v_{n+1}\}$, $\mathcal{E} = \{e_1, e_2, ..., e_n\}$, and $e_i = \{v_i, v_{i+1}\}$, as illustrated in Figure 7.

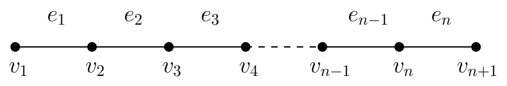


Figure 7: A line join on n **relations.**

6.1 Optimal edge cover

Let x be the optimal edge cover of a n-relation line join. First we give the characterization of x:

(1) $x_1 = x_n = 1$;

(2) there are no two consecutive 0's;

(3) there are no three consecutive 1's;

(4) there are no 5 consecutive x_i's being $(1, 1, 0, 1, 1)$.

The first 3 rules are obvious; the last rule follows from the observation that $(1, 0, 1, 0, 1)$ is always better than $(1, 1, 0, 1, 1)$, since the size of the middle relation is smaller than the product of its two neighbors on fully reduced relations.

Another simple observation is that x has "sub-optimality", i.e., for any $x_i = x_j = 1, i < j$, $(x_i, \ldots, x_j)$ must be an optimal edge cover for the subjoin $R_i \bowtie \cdots \bowtie R_j$. If not, we could replace $(x_i, \ldots, x_j)$ with a better one, which would improve x for the whole join.

For any $i \leq j$, $(x_i, \ldots, x_j)$ is called an *alternating interval* if $(x_i, \ldots, x_j) = (1, 0, 1, 0, ..., 0, 1)$. It is easy to see that x must consist of one or more alternating intervals. A single $x_i = 1$ is also an alternating interval, but it can only appear at the two ends. For example, for $n = 4$, $x = (1, 0, 1, 1)$ or $(1, 1, 0, 1)$.

6.2 Worst-case optimality

In this subsection we give sufficient conditions under which Algorithm 2 is optimal.

When n is odd.

For an odd n, we say that an n-relation line join $\mathcal{Q}$ is *balanced* if

$$N_i N_{i+2} \cdots N_{j-2} N_j \geq N_{i+1} N_{i+3} \cdots N_{j-3} N_{j-1}$$

for any $1 \leq i < j \leq n$ such that $j-i$ is an even number. Note that any 3-relation line join is always balanced after dangling tuples are removed. A 5-relation line join is balanced if $N_1 N_3 N_5 \geq N_2 N_4$.

Lemma 3. *On a balanced line join, the optimal edge cover x must be alternating.*

Proof. Suppose for contradiction that x is not alternating, then by previous analysis, x must consist of 3 or more alternating intervals. Thus, the pattern $(1,1,0,1,0,\dots,0,1,0,1,1)$ must appear somewhere, and suppose it is $(x_i,\dots,x_j)$. By sub-optimality, this pattern has appeared only because it is better than $(1,0,1,0,\dots,0,1,0,1)$ on the interval $(N_i,\dots,N_j)$, which means

$$N_{i+1} N_{i+3} \cdots N_{j-3} N_{j-1} < N_{i+2} N_{i+4} \cdots N_{j-4} N_{j-2},$$

violating the balancing condition on the interval $(N_{i+1},\dots,N_{j-1})$. $\square$

Applying Theorem 3 on a line join yields the following result.

Corollary 2. *Let $\mathcal{Q} = (\mathcal{V}, \mathcal{E}, N)$ be a line query on an odd number of relations. On instance R, Algorithm 2 has I/O cost*

$$\widetilde{O}\left(\max_S \Psi(S)\right) = \widetilde{O}\left(\max_S \frac{\prod_{e\in S} N(e)}{M^{|S|-1}B}\right),$$

where S is over all independent subsets of $\mathcal{E}$.

Proof. It suffices to show that $\text{GenS}(\mathcal{Q})$ can generate an $\mathbf{S}$ such that for each $S \in \mathbf{S}$, $\Psi(S)$ is dominated by $\Psi(S')$ for some independent subset S' of $\mathcal{E}$. Let $L(i,j)$ be *a* set of subsets generatable by $\text{GenS}(\mathcal{Q})$ on $e_i \bowtie e_{i+1} \bowtie \cdots \bowtie e_j$. The proof is by induction on the length of the interval (i,j), where the interval length must be odd. The base case is trivial.

To generate $L(i,j)$, we peel off (e_i, e_{i+1}) first, which yields

$$L(i,j) = (L(i+2,j) \circ \{e_i\}) \cup L(i+1,j) \cup \{\{e_i\}\},$$

where we use the notation $\mathbf{X} \circ Y = \{X \cup Y \mid X \in \mathbf{X}\}$. By the induction hypothesis, $L(i+2,j)$ only contains subsets that are dominated by independent subsets, so $L(i+2,j) \circ \{e_i\}$ retains this property.

To deal with $L(i+1,j)$, we peel off (e_{j-1}, e_j), which gives

$$L(i+1,j) = (L(i+1,j-2) \circ \{e_j\}) \cup L(i+1,j-1) \cup \{\{e_j\}\}.$$

$L(i+1,j-1)$ is an interval of odd length and we can apply the induction hypothesis directly. For $L(i+1,j-2)$, which is an interval of even length, we peel off $\{e_{j-3}, e_{j-2}\}$:

$$L(i+1,j-2) = (L(i+1,j-4) \circ \{e_{j-2}\}) \cup L(i+1,j-3) \cup \{\{e_j-2\}\}.$$

Again, $L(i+1,j-3)$ is an interval of odd length and we apply the induction hypothesis directly. Then we repeat the same process on $L(i+1,j-4)$, until its length reduces to 2. Thus, the only subjoin that we have not accounted for is

$$\{e_{i+1}, e_{i+2}, e_{i+4}, e_{i+6}, \dots, e_j\}.$$

On a fully reduced instance, this subjoin is dominated by that of $\{e_i, e_{i+2}, e_{i+4}, \dots, e_j\}$. $\square$

Theorem 5. *For odd n, Algorithm 2 is optimal for any balanced n-relation line join.*

Proof. It suffices to show that for every independent subset S of $\mathcal{E}$, there exists an instance I such that the partial join size on S is the same as largest subjoin size on S, i.e., $|\mathcal{Q}(I,S)| = \prod_{e\in S} N(e)$.

It is easy to see that an instance I in which each relation is the cross product of its two domains satisfies this requirement. However, we still need to ensure that such a construction is feasible. More precisely, suppose attribute v_i has a domain size $|\text{dom}(v_i)| = z_i$. Then the following is the sufficient and necessary condition for the feasibility of such an I:

$$\begin{cases} z_i z_{i+1} = N_i, & i = 1,\dots,n, \\ z_i \leq N_i, & i = 1,\dots,n, \\ z_i \leq N_{i-1}, & i = 2,\dots,n+1, \\ z_i \geq 1, & i = 1,\dots,n+1. \end{cases}$$

Using the n equalities, we can represent all the z_i's using z_1:

$$\begin{cases} z_2 = \frac{N_1}{z_1}, \\ z_3 = \frac{z_1 N_2}{N_1}, \\ z_4 = \frac{N_1 N_3}{z_1 N_2}, \\ \dots, \\ z_n = \frac{z_1 N_2 N_4 \cdots N_{n-1}}{N_1 N_3 \cdots N_{n-2}}, \\ z_{n+1} = \frac{N_1 N_3 \cdots N_n}{z_1 N_2 N_4 \cdots N_{n-1}}. \end{cases}$$

All the inequalities on the z_i's thus translate to those on z_1:

$$\left.\begin{array}{l} 1 \\ \frac{N_1}{N_2} \\ \frac{N_1 N_3}{N_2 N_4} \\ \frac{N_1 N_3 N_5}{N_2 N_4 N_6} \\ \dots \\ \frac{N_1 N_3 \cdots N_{n-2}}{N_2 N_4 \cdots N_{n-1}} \end{array}\right\} \leq z_1 \leq \left\{\begin{array}{l} N_1 \\ \frac{N_1 N_3}{N_2} \\ \frac{N_1 N_3 N_5}{N_2 N_4} \\ \frac{N_1 N_3 N_5 N_7}{N_2 N_4 N_6} \\ \dots \\ \frac{N_1 N_3 \cdots N_n}{N_2 N_4 \cdots N_{n-1}}. \end{array}\right.$$

By the balancing condition, any term on the right-hand side is greater than any term on the left-hand side. Thus, this construction is feasible, which concludes the proof. $\square$

When n is even.

Theorem 6. *For an n-relation line join $\mathcal{Q}$ where n is even, Algorithm 2 is optimal if there is an odd k such that the two subjoins $e_1 \bowtie \cdots \bowtie e_k$ and $e_{k+1} \bowtie \cdots \bowtie e_n$ are both balanced.*

Proof. First, observe that under the above condition, the optimal edge cover consists of two alternating intervals $(x_1,\dots,x_k)$ and $(x_{k+1},\dots,x_n)$.

The construction of R is the same as before, except that we set $z_{k+1} = 1$, i.e., the common attribute of R_k and R_{k+1} has only one value. Feasibility can be verified by going through the same exercise as in the proof of Theorem 5. $\square$

The optimal bound in this case is thus $\max_S \frac{\prod_{e\in S} N(e)}{M^{|S|-1}B}$ where S is over all independent subsets of $\mathcal{E}$, except that e_k and e_{k+1} can be chosen into S together.

It may seem that the construction above can be generalized to the case where $\mathcal{Q}$ can be decomposed into any number of balanced subjoins. However, this case can never happen:

If $\mathcal{Q}$ is decomposed into 3 or more subjoins, then the pattern $(1,1,0,1,0,\ldots,0,1,0,1,1)$ must appear somewhere in the optimal edge cover, and by previous analysis, this means that one of the subjoins is not balanced.

6.3 When Algorithm 2 is not optimal

When $n = 5$.

As mentioned, L_3 is always balanced; a L_4 join can be split into an L_1 and and an L_3, both of which must be balanced. Thus, the smallest n on which Algorithm 2 might not be optimal is $n = 5$. When $N_1N_3N_5 < N_2N_4$, the construction of R above is not feasible. We have to allow R_3 to be any mapping from $\text{dom}(v_3)$ onto $\text{dom}(v_4)$, while R_2 and R_4 remain as cross products. Thus, the I/O lower bound becomes $\psi(\mathcal{Q}) = \frac{N_1N_3N_5}{MB} + \frac{N_2}{B} + \frac{N_4}{B}$, which is smaller than the bound $\frac{N_1N_3N_5}{M^2B} + \frac{N_1N_4}{MB} + \frac{N_2N_5}{MB} + \frac{N_2N_4}{MB}$ in the balanced case. For this case, we have designed a special algorithm that achieves the optimal bound, as described in Algorithm 4.

Algorithm 4: LineJoinUnbalanced5$(R_1,\ldots,R_5)$

```
1 call Algorithm 1 on $(R_1,R_2,R_3)$, and write the results
  to disk as $S$;
2 call Algorithm 1 on $(R_3,R_4,R_5)$, and write the results
  to disk as $T$;
3 sort $R_3$ by $v_3, v_4$ lexicographically;
4 sort $S$ and $T$ by $v_3, v_4$ lexicographically;
5 foreach $t \in R_3$ do
6     compute $S(t) = S \ltimes t$;
7     compute $T(t) = T \ltimes t$;
8     compute $S(t) \bowtie T(t)$ by nested-loop join, and emit
      all results;
```

The costs of the two line joins in line 1–2 are $\widetilde{O}(\frac{N_1N_3}{MB} + \frac{N_2}{B})$ and $\widetilde{O}(\frac{N_3N_5}{MB} + \frac{N_4}{B})$, but writing S and T to disk costs $\widetilde{O}(\frac{N_1N_3}{B} + \frac{N_3N_5}{B})$. The loop in line 4–7 has N_3 iterations. We can compute all the semijoins in line 5–6 with one scan of S, T and R_3 after sorting them by v_3, v_4 lexicographically. Finally, every $S(t)$ has size at most N_1 and every $T(t)$ has size at most N_5, so the nested loop join on line 8 takes $O(\frac{N_1N_5}{MB})$ I/Os. Adding up all these costs yields the desired bound.

When $n = 6$.

The only possibility where Algorithm 2 is not optimal for an L_6 is when its optimal edge cover is $(1,0,1,0,1,1)$ (or $(1,1,0,1,0,1)$), and the first 5 relations are not balanced. In this case, we can run a nested-loop join with R_6 as the outer relation and $R_1 \bowtie \cdots \bowtie R_5$ as the inner relation, computed by Algorithm 4.

When $n = 7$.

On an L_7, the optimal edge cover can be either $(1,1,0,1,0,1,1)$ or $(1,0,1,0,1,0,1)$.

The former case can be reduced to the unbalanced 5-relation case. More precisely, when the optimal edge cover is $(1,1,0,1,0,1,1)$, then the middle 5 relations must be unbalanced, i.e., $N_3N_5 > N_2N_4N_6$. Now we simply run nested-loop join on $R_1 \bowtie (R_2 \bowtie \cdots \bowtie R_6) \bowtie R_7$, while using Algorithm 4 for the 5-relation join in the middle. The I/O cost is $\widetilde{O}(\frac{N_1N_2N_4N_6N_7}{M^3B} + \frac{N_1N_3N_7}{M^2B} + \frac{N_1N_5N_7}{M^2B})$.

We can show that this bound is optimal. We construct an instance R in which R_1 and R_6 are many-to-one mappings from $\text{dom}(v_1)$ to $\text{dom}(v_2)$ and from $\text{dom}(v_6)$ to $\text{dom}(v_7)$, respectively. R_2 and R_7 are one-to-many mappings from $\text{dom}(v_2)$ to $\text{dom}(v_3)$ and $\text{dom}(v_7)$ to $\text{dom}(v_8)$, respectively. R_3 and R_5 are cross products and R_4 remains as a many-to-many mapping. It is feasible due to the condition $N_3N_5 > N_2N_4N_6$. On such an instance, we have

$$\begin{aligned}
|\mathcal{Q}(R,\{e_1,e_3,e_7\})| &= N_1N_3N_7,\\
|\mathcal{Q}(R,\{e_1,e_5,e_7\})| &= N_1N_5N_7,\\
|\mathcal{Q}(R,\{e_1,e_3,e_5,e_7\})| &= N_1N_2N_4N_6N_7.
\end{aligned}$$

When the optimal edge cover is $(1,0,1,0,1,0,1)$, 3 balancing conditions might break: $N_1N_3N_5N_6 \geq N_2N_4N_6$, $N_1N_3N_5 \geq N_2N_4$ and $N_3N_5N_7 \geq N_4N_6$. We have also designed a different algorithm, described below, which is optimal when any of these 3 balancing conditions break. Its optimality proof involves a careful case-by-case analysis, which we leave in Appendix A.3.

Algorithm 5: LineJoinUnbalanced7$(R_1,\ldots,R_7)$

```
1 call Algorithm 1 on $(R_3,R_4,R_5)$, and write the results
  to disk as $S$;
2 Construct a corresponding acyclic join query $(\mathcal{V},\mathcal{E},R)$
  with $R = \{R_1,R_2,S,R_6,R_7\}$;
3 AcyclicJoin$(\mathcal{V},\mathcal{E},R)$ ;
```

$n = 8$ *and beyond.*

An L_8 can be reduced to smaller joins, so can be solved optimally under all cases. However, the situation gets highly complex for $n \geq 9$. A general algorithm that can solve all the unbalanced cases optimally remains elusive.

7. GENERAL ACYCLIC JOINS

In this section, we study the optimality of Algorithm 2 on other classes of acyclic joins.

7.1 When all relations have equal sizes

Let x the optimal edge cover of $\mathcal{Q} = (\mathcal{V},\mathcal{E},N)$ and let $c = \sum x(e)$. When all relations have equal sizes, x is just the minimum edge cover of the hypergraph $(\mathcal{V},\mathcal{E})$. For acyclic queries, the minimum edge cover can be found by the following simple greedy strategy:

Algorithm 6: Minimum edge cover C

```
1 $C \leftarrow \emptyset$;
2 while $\mathcal{Q}$ has uncovered attributes do
3     let $e$ be any edge containing unique attributes;
4     $C \leftarrow C \cup \{e\}$;
5     remove $e$ and its attributes from $\mathcal{Q}$;
```

We will show the Algorithm 2 is optimal in this case.

Theorem 7. *Let $\mathcal{Q} = (\mathcal{V},\mathcal{E},N)$ be an acyclic query where $N(e) = N$ for all $e \in \mathcal{E}$. Let c be the minimum edge cover number of $(\mathcal{V},\mathcal{E})$. Algorithm 2 has I/O cost $\widetilde{O}\left(\left(\frac{N}{M}\right)^c \frac{M}{B}\right)$ on $\mathcal{Q}$ and this is optimal.*

PROOF. To prove the upper bound, we need to find an $\mathbf{S}$ generatable by GENS($\mathcal{Q}$), such that for any $S \in \mathbf{S}$, $\Psi(R,S) \le \left(\frac{N}{M}\right)^c \frac{M}{B}$ on any instance R. It turns out if all relations have equal sizes, nondeterminism is not needed, and we can allow GENS($\mathcal{Q}$) to peel any star, and peel any leaf or island if no stars exist. Buds can always be ignored as they do not appear in any $S \in \mathbf{S}$ or in the minimum edge cover.

The proof is by induction on the size of $\mathcal{Q}$. The base case when $\mathcal{Q}$ has only one relation is trivial. For the inductive step, let us first consider the case when GENS($\mathcal{Q}$) peels off a star $\mathcal{X}$ with core e_0. Let c_1 and c_2 be the minimum edge cover number of $\mathcal{Q} - \mathcal{X} + \{e_0\}$ and $\mathcal{Q} - \mathcal{X}$, respectively. Following the greedy algorithm above for finding a minimum edge cover, we have

$$c_2 + |\mathcal{X}| - 1 = c, \quad c_1 \le c_2 + 1 \le c.$$

Let $\mathbf{S}_1$ and $\mathbf{S}_2$ respectively be the set of subsets of relations returned by GENS($\mathcal{Q} - \mathcal{X} + \{e_0\}$) and GENS($\mathcal{Q} - \mathcal{X}$), following any nondeterministic branch. Any S returned by GENS($\mathcal{Q}$) must fall into one of the following cases.

1. $S \in \mathbf{S}_1$ or $S \in \mathbf{S}_2$: In this case, we have $\Psi(R,S) \le \left(\frac{N}{M}\right)^{c_1 \text{ or } c_2} \frac{M}{B} \le \left(\frac{N}{M}\right)^c \frac{M}{B}$ by invoking the induction hypothesis on $\mathcal{Q} - \mathcal{X} + \{e_0\}$ and $\mathcal{Q} - \mathcal{X}$.

2. $S \subseteq \mathcal{X}$: For such an S, we have $\Psi(R,S) \le (\frac{N}{M})^{|S|} \frac{M}{B} \le (\frac{N}{M})^c \cdot \frac{M}{B}$, since $|S| \le |\mathcal{X}| \le c$.

3. $S = S_1 \cup f$ where $S_1 \in \mathbf{S}_1, f \subsetneq \mathcal{X} - \{e_0\}$: We have

$$\begin{aligned}\Psi(R,S) \le& \frac{N^{|f|} \cdot |\bowtie_{e \in S_1} R(e)|}{M^{|f|+|S_1|}} \cdot \frac{M}{B} \\ =& \left(\frac{N}{M}\right)^{|f|} \cdot \Psi(R,S_1) \\ \le& \left(\frac{N}{M}\right)^{|f|+c_1} \cdot \frac{M}{B}.\end{aligned}$$

   Since $|f| + c_1 \le |\mathcal{X}| - 2 + c_1 \le c$, this is bounded by $\left(\frac{N}{M}\right)^c \cdot \frac{M}{B}$.

4. $S = S_2 \cup f$ where $S_2 \in \mathbf{S}_2, f \subseteq \mathcal{X} - \{e_0\}$: Similar to above, we have

$$\Psi(R,S) \le \left(\frac{N}{M}\right)^{|f|+c_2} \cdot \frac{M}{B},$$

   and the upper bound holds by the fact that $|f| + c_2 \le |\mathcal{X}| - 1 + c_2 = c$.

Next, consider the case when GENS($\mathcal{Q}$) peels off a leaf (the case when it peels off an island is trivial). We observe that when a leaf e is peeled off by GENS($\mathcal{Q}$), any of its neighboring edge must contain unique attributes. Otherwise, suppose e has a neighboring edge e' that does not have unique attributes, then it will become the core of a star, with e being a petal. However, when GENS($\mathcal{Q}$) decides to peel off a leaf, it must be the case that $\mathcal{Q}$ does not have stars, which is contradictory.

Now suppose GENS($\mathcal{Q}$) peels off a leaf e, and let c' be the minimum edge cover of $\mathcal{Q} - \{e\}$. Since every neighboring relation of e has unique attributes, all of them must be in selected in the edge cover of $\mathcal{Q} - \{e\}$. Thus we have $c'+1 = c$. Any S returned by GENS($\mathcal{Q}$) must take the form of $S' \cup \{e\}$ where S' is generated by GENS($\mathcal{Q} - \{e\}$). Similar to the analysis above, we have

$$\Psi(R,S) \le \left(\frac{N}{M}\right)^{1+c'} \cdot \frac{M}{B} = \left(\frac{N}{M}\right)^c \cdot \frac{M}{B},$$

as desired.

To show that this bound is optimal, we need to construct an instance in which there are c relations whose partial join size is N^c. By LP duality, the minimum edge cover corresponds to a vertex packing such that an edge is in the cover if and only if it covers a vertex in the packing. We set the domain size of each vertex in the packing to be N, and that of each vertex not in the packing to be 1. Each relation will simply be the cross product of their domains. Note that each edge can have at most one vertex in the packing, so its relation size is at most N. On this instance, the partial join size of all relations in the edge cover is N^c. □

7.2 Lollipop joins

A *lollipop join* is illustrated in Figure 8, which is a star join with one petal, say e_n, extending to another relation e_{n+1}. It is the mixture of a star join and a an L_4, so it is intuitive that Algorithm 2 should be optimal for this type of joins.

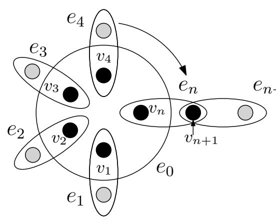


Figure 8: A lollipop join.

A lollipop join contains two stars, with e_0 and e_n as the core, respectively. Similar to the L_4 case, different peeling strategies leads to different I/O bounds, and which one is better depends on the relation sizes. When $N_0 \le N_n$, we peel off the star with e_n as the core first, otherwise we peel the star with e_0 as the core.

When $N_0 \le N_n$: In this case, we peel off the star with e_n being the core. The $\mathbf{S}$ returned by GENS($\mathcal{Q}$) will then contain the following types of subsets: (i) $S \subseteq \{e_n, e_{n+1}\}$ (ii) $S \cup \{e_{n+1}\}$ with $S \subseteq \mathcal{E} - \{e_n, e_{n+1}\}$; (iii) $S \subseteq \mathcal{E} - \{e_{n+1}\}$. Recall that to prove optimality, for each $S \in \mathbf{S}$, we need to construct an instance I whose partial join size on S matches the maximum subjoin size, i.e.,

$$\max_R |\bowtie_{e \in S} R(e)| = |\mathcal{Q}(I,S)|.$$

Case (i) is trivial. For case (ii), the instance I is constructed as follows: Set $|\mathrm{dom}(v_n)| = N_0$, and the domain of all the other join attributes have size 1. Each relation is simply the cross product of the domains of its attributes. Note that $|I(e_i)| = N_i$, except that $|I(e_n)| = N_0 \le N_n$. On this I, the partial join on $S \cup \{e_{n+1}\}$ has size

$$|\mathcal{Q}(I, S \cup \{e_{n+1}\})| = \prod_{e \in S \cup \{e_{n+1}\}} N(e),$$

which is clearly also the maximum subjoin size.

For (iii), if $e_0 \in S$, then the partial join size always matches the subjoin size. If $e_0 \notin S$, we construct I as follows: Set $|\text{dom}(v_n)| = N_0$, $|\text{dom}(v_{n+1})| = \frac{N_n}{N_0}$, and the domain of all the other join attributes have size 1. Each relation is the cross product of the domains of its attributes, and we have $|I(e_i)| = N_i$ for all i. Note that we will need $N_{n+1} \geq \frac{N_n}{N_0}$ for this construction to be feasible, but this is always the case on fully reduced instances.

When $N_0 \geq N_n$**:** In this case, we peel off the star with core e_0 first. The **S** generated by GenS($\mathcal{Q}$) contains the following types of subsets: (i) $S \subseteq \mathcal{E} - \{e_n, e_{n+1}\}$ always containing e_0; (ii) $S \cup \{e_0, e_{n+1}\}$ with $S \subsetneq \mathcal{E} - \{e_0, e_n, e_{n+1}\}$; (iii) $S \cup f$ with $S \subseteq \mathcal{E} - \{e_0, e_n, e_{n+1}\}, f \subseteq \{e_n, e_{n+1}\}$.

For any S in (i), the partial join size always matches the subjoin size. For any $S \cup \{e_0, e_{n+1}\}$ in case (ii), its subjoin size is at most

$$N_{n+1} \cdot \max_R |\mathcal{Q}(R, S \cup \{e_0\})|. \tag{6}$$

Let $R_{\max}$ be the R that maximizes $|\mathcal{Q}(R, S \cup \{e_0\})|$. Note that in $R_{\max}$, we must have $|\text{dom}(v_n)| \leq N_n$. We construct the instance I based on $R_{\max}$: For any $e \in S \cup \{e_0\}$, $I(e) = R_{\max}(e)$; $I(e_n)$ is a many-to-one mapping from $\text{dom}(v_n)$ to $\text{dom}(v_{n+1})$; $I(e_{n+1})$ is a one-to-many mapping of size N_{n+1} from $\text{dom}(v_n)$ to $\text{dom}(v_{n+1})$. On such an I, its partial join size on $S \cup \{e_0, e_n\}$ is exactly (6).

For any $S \cup f$ in (iii), if $f = \{e_{n+1}\}$ or $f = \emptyset$, the case can be easily handled by setting $|\text{dom}(v_1)| = \cdots = |\text{dom}(v_{n+1}| = 1$. On such an I, its partial join size is simply $\prod_{e \in S \cup f} N(e)$. If $f = \{e_n\}$ or $\{e_n, e_{n+1}\}$, we just need to change $|\text{dom}(v_n)|$ to N_n. This is feasible due to condition $N_0 \geq N_n$, so that $|R(e_0)|$ does not violate its size constraint. The partial join size is then still $\prod_{e \in S \cup f} N(e)$.

We do not have a closed-form worst-case bound for lollipop joins, as it contains a star.

7.3 Dumbbell joins

A *dumbbell join* is illustrated in Figure 9, which consists of two stars connected by a common petal. One star has e_0 at its center, with n petals $e_1, \ldots, e_n$. The other star has petals $e_n, \ldots, e_{m-1}$, with e_m at its center.

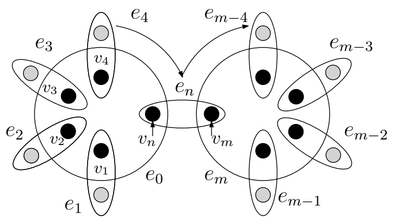


Figure 9: A dumbbell join.

Note that a dumbbell join generalizes an L_5, so Algorithm 2 is not always optimal. The optimality condition also generalizes that for L_5, becoming

$$N_i N_n N_j \geq N_0 N_m, \tag{7}$$

for any $1 \leq i \leq n-1, n+1 \leq j \leq m-1$.

The proof of worst-case optimality on dumbbell join query is a generalization of the analysis on the lollipop join, but more involved. The details are given in Appendix A.4. Again, we do not have a closed-form bound on its optimal I/O complexity, as it contains a star.

8. CONCLUDING REMARKS

In this paper, we have designed a worst-case I/O-optimal algorithm for computing a fairly large class of acyclic joins. Interestingly, we have been able to prove its optimality without deriving its worst-case bound explicitly. This is done by relating the I/O cost of the algorithm to all the subjoins sizes, and then showing that the subjoin sizes match the corresponding partial join sizes in the worst case.

We conclude by mentioning a few interesting open problems.

1. For cases where our general algorithm is not optimal, in particular for line joins with 5 to 8 relations, we have designed special algorithms to achieve optimality. Is there a more unified way to tackle these cases?

2. Yannakakis' algorithm is instance optimal in internal memory. However, we conjecture that such an algorithm does not exist in external memory, even on 3 relations. Is there a formal proof? If it indeed does not exist, are there any conditions under which it does?

3. How about cyclic joins? So far, worst-case I/O-optimal algorithms are known only for triangle queries on relations of equal size.

9. REFERENCES

[1] A. Aggarwal, J. Vitter, et al. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 739–748, 2008.

[3] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[4] C. Berge and E. Minieka. *Graphs and hypergraphs*, volume 7. North-Holland publishing company Amsterdam, 1973.

[5] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2008.

[6] X. Hu, M. Qiao, and Y. Tao. Join Dependency Testing, Loomis-Whitney Join, and Triangle Enumeration. In *Proc. ACM Symposium on Principles of Database Systems*, 2015.

[7] X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2013.

[8] P. Koutris, P. Beame, and D. Suciu. Worst-case optimal algorithms for parallel query processing.

[9] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *Proc. ACM Symposium on Principles of Database Systems*, pages 37–48, 2012.

[10] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *ACM SIGMOD Record*, 42(4):5–16, 2014.

[11] A. Pagh and R. Pagh. Scalable computation of acyclic joins. In *Proc. ACM Symposium on Principles of Database Systems*. ACM, 2006.

[12] R. Pagh and F. Silvestri. The input/output complexity of triangle enumeration. In *Proc. ACM Symposium on Principles of Database Systems*, 2014.

[13] T. Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *Proc. International Conference on Database Theory*, 2014.

[14] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. International Conference on Very Large Data Bases*, pages 82–94, 1981.

APPENDIX

A. DETAILED PROOFS

A.1 Proof of Theorem 2

PROOF. Recall that the join between two relations with no common attribute is simply their cross product. So (3) can be written as

$$\widetilde{O}\left(\sum_{S\subseteq\mathcal{E}}\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')|}{M^{|S|-1}B}\right). \tag{8}$$

where $\mathcal{C}(S)$ consists of all the connected components of S.

Let the I/O cost of Algorithm 2 be $f(\mathcal{Q})$. We will prove that $f(\mathcal{Q})$ is bounded by (8) by induction on $|\mathcal{E}|$. The base case $|\mathcal{E}|=1$ is trivial. Now suppose $f(\mathcal{Q})$ is (8) for any $\mathcal{Q}$ and R with $|\mathcal{E}|=k$ relations, and we will show that the claim holds on $k+1$ relations as well. To simplify the presentation, the analysis below hides a constant factor for each level of induction / recursion. Since we assume the query size is a constant, this does not change the asymptotic result.

First, the total costs of peeling all the buds is $\widetilde{O}(\frac{N}{B})$, which is included in (8).

Next, consider peeling off an island e. Let $\mathcal{Q}'$ be the query fed in to the recursive calls after the peeling. We have

$$f(\mathcal{V},\mathcal{E},R)=\frac{N(e)}{B}+\left\lceil\frac{N(e)}{M}\right\rceil\cdot f(\mathcal{Q}').$$

The $\frac{N(e)}{B}$ term is clearly contained in (8). If $N(e)<M$, then the algorithm just makes one recursive call, and we have $f(\mathcal{Q})=\frac{N(e)}{B}+f(\mathcal{Q}')=(8)$ by the induction hypothesis. When $N(e)>M$, we can drop the ceiling. By the induction hypothesis, $f(\mathcal{Q}')$ has a term $\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')|}{M^{|S|-1}B}$ for each subset S of $\mathcal{E}-\{e\}$. Since e is disconnected from S and $S\cup\{e\}$ is a subset of $\mathcal{E}$, $\frac{N(e)}{M}\cdot\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')|}{M^{|S|-1}B}$ is included in (8).

In the rest of the proof, we consider peeling off a leaf e, with join attribute v and neighboring relations Γ. The cost of the sorting in line 12–13 is $\widetilde{O}(\frac{N(e)}{B}+\sum_{e'\in\Gamma}\frac{N(e')}{B})$, which is included in (8).

Heavy tuples. Consider the heavy tuples in $R(e)$ first (line 14–20). For each a, the loop (line 16–19) has $\frac{N(e)|_{v=a}}{M}$ iterations (we must have $N(e)|_{v=a}\geq M$ because a is heavy), each recursively computing a query $\mathcal{Q}'(a)$. So the total cost of handling all the heavy tuples is

$$\sum_{a\in H(e,v)}\frac{N(e)|_{v=a}}{M}f(\mathcal{Q}'(a)).$$

Recall that for heavy values on v, we delete v before going into the recursion, which decomposes the query into $|\Gamma|$ components. And in the modified instance $R'(a)$ fed into the recursive calls, each $R(e'),e'\in\Gamma$ has been replaced with $R(e')|_{v=a}$. For any $e'\notin\Gamma$, $R(e')$ remains unchanged, i.e., $R(e')|_{v=a}=R(e')$. Thus, by the induction hypothesis, the total cost to handle all the heavy values is

$$\begin{aligned}&\sum_{a\in H(e,v)}\frac{N(e)|_{v=a}}{M}\sum_{S\subseteq\mathcal{E}-\{e\}}\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')|_{v=a}|}{M^{|S|-1}B}\\ =&\sum_{S\subseteq\mathcal{E}-\{e\}}\sum_{a\in H(e,v)}\frac{N(e)|_{v=a}}{M}\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')|_{v=a}|}{M^{|S|-1}B}.\end{aligned}$$

Thus, it suffices to show that, for any $S\subseteq\mathcal{E}-\{e\}$,

$$\sum_{a\in H(e,v)}N(e)|_{v=a}\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')|_{v=a}| \tag{9}$$

$$\leq\prod_{S'\in\mathcal{C}(S\cup\{e\})}|\Join_{e'\in S'}R(e')|. \tag{10}$$

Note that (9) is on the recursive calls on $\mathcal{Q}'(a)$, in which the attribute v has been removed, so the relations in Γ are not connected, while (10) considers the whole query $\mathcal{Q}$, in which all relations in Γ are connected, together with e.

First, observe that any $S'\in\mathcal{C}(S)$ that does not include any edge in Γ must be disconnected from e, thus also disconnected from any component of $\mathcal{C}(S\cup\{e\})$. Thus, it contributes the same factor

$$|\Join_{e'\in S'}R(e')|_{v=a}|=|\Join_{e'\in S'}R(e')|$$

to both (9) and (10), which cancel out. Note that if all connected components of S are like this, after all cancellation, we have

$$(9)=\sum_{a\in H(e,v)}N(e)|_{v=a}\leq N(e)=(10).$$

Thus, we can assume that each connected component of S includes a relation of Γ. Let them be $S_1,\dots,S_r$. Note that they are disconnected in $\mathcal{Q}'(a)$, but are all connected in $\mathcal{Q}$, together with e, so $\mathcal{C}(S\cup\{e\})$ has only one component.

We then have

$$\begin{aligned}(9)&=\sum_{a\in H(e,v)}N(e)|_{v=a}\prod_{i=1}^{r}|\Join_{e'\in S_i}R(e')|_{v=a}|\\ &=\sum_{a\in H(e,v)}\left|R(e)|_{v=a}\Join\left(\Join_{e'\in S}R(e')|_{v=a}\right)\right|\\ &=(10).\end{aligned}$$

Light tuples. Finally, consider the light tuples (line 21–27). Different memory chunks M_1 of $R(e)$ partition each $R(e')$ into disjoint parts $R(e')(M_1)$. The total cost of computing and sorting all the $R(e')(M_1)$'s on line 22–23 is $\widetilde{O}(\frac{N}{B})$. It only remains to consider the recursive calls. By the induction hypothesis, this is

$$\begin{aligned}\sum_{M_1}f(\mathcal{Q}')&=\sum_{M_1}\sum_{S\subseteq\mathcal{E}-\{e\}}\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')(M_1)|}{M^{|S|-1}B}\\ &=\sum_{S\subseteq\mathcal{E}-\{e\}}\sum_{M_1}\frac{\prod_{S'\in\mathcal{C}(S)}|\Join_{e'\in S'}R(e')(M_1)|}{M^{|S|-1}B}.\end{aligned}$$

Here we define $R(e')(M_1)=R(e')$ if $e'\notin\Gamma$.

Thus, it suffices to show that, for any $S \subseteq \mathcal{E} - \{e\}$,

$$\sum_{M_1} \prod_{S' \in \mathcal{C}(S)} |\bowtie_{e' \in S'} R(e')(M_1)| \tag{11}$$
$$\leq \prod_{S' \in \mathcal{C}(S)} |\bowtie_{e' \in S'} R(e')|. \tag{12}$$

Note that S is also a subset of $\mathcal{E}$, so (12) is included in (8).

For any S' that does not include a relation in Γ, $R(e')(M_1) = R(e')$, it contributes the same factor to both (11) and (12), thus cancels out. Any connected component that remains after the cancellation includes some relation of Γ. Recall that for light tuples, the join attribute v is included in the recursive calls, so all relations in Γ are still connected, which means that all that is left is one connected component S. Therefore,

$$(11) = \sum_{M_1} |\bowtie_{e' \in S} R(e')(M_1)|$$
$$= |\bowtie_{e' \in S} R(e')| = (12),$$

which concludes the proof. $\square$

A.2 Proof of Theorem 3

By the round-robin simulation of the nondeterministic algorithm, it is sufficient to prove the following:

LEMMA 4. *Let $\mathcal{Q}$ be an acyclic query, $\mathcal{A}$ be any branch of* GenS($\mathcal{Q}$)*, and* $\mathbf{S}$ *be the resulting set of subsets of relations returned by* GenS($\mathcal{Q}$) *following* $\mathcal{A}$*. There is a branch of Algorithm 2 whose I/O cost is* $\widetilde{O}\left(\max_{S \in \mathbf{S}} \Psi(R, S)\right)$.

PROOF. The proof is by induction on the size of the hypergraph of $\mathcal{Q}$. The base case when the hypergraph is empty is trivial. If $\mathcal{A}$ first peels off is a leaf, an island, or a bud, the proof is almost identical to the proof of Theorem 2, thus we will focus on the case when peeling off a star. Suppose $\mathcal{A}$ peels off a star $\mathcal{X}$ with k petals $e_1, \dots, e_k$, and the core is e_0. We assume that the petals are disjoint for now.

The set of subsets returned by GenS($\mathcal{Q}$) following $\mathcal{A}$ is

$$\text{GenS}(\mathcal{Q}) = 2^{\mathcal{X}} + 2^{\mathcal{X}-\{e_0\}} \times \text{GenS}(\mathcal{Q} - \mathcal{X}) + \left(2^{\mathcal{X}-\{e_0\}} - \{\mathcal{X} - \{e_0\}\}\right) \times \text{GenS}(\mathcal{Q} - \mathcal{X} + \{e_0\}). \tag{13}$$

Here we redefine the notation $\mathbf{X} \times \mathbf{Y} = \{X \cup Y \mid X \in \mathbf{X}, Y \in \mathbf{Y}\}$.

First consider the case where $\mathcal{X}$ has only one petal e_1. We ask Algorithm 2 to peel off e_1 and then e_0. Let v_1 be the join attribute between e_0 and e_1. Following similar analysis as in the proof of Theorem 2 and invoking the induction hypothesis on $\mathcal{Q} - \mathcal{X} = \mathcal{Q} - \{e_0, e_1\}$, the I/O-cost of handing the heavy tuples in $R(e_1)$ is

$$\sum_{a \in H(e_1, v_1)} \frac{R(e_1)|_{v_1=a}}{M} \cdot \left(\frac{R(e_0)|_{v_1=a}}{B} + \max_{S \in \text{GenS}(\mathcal{Q}-\mathcal{X})} \Psi(R, S) \right)$$
$$= \Psi(R, \{e_0, e_1\}) + \max_{S \in \{\{e_1\}\} \times \text{GenS}(\mathcal{Q}-\mathcal{X})} \Psi(R, S).$$

These two terms are captured by the first two terms of (13), respectively. The cost to handle the light tuples is (invoking the induction hypothesis on $\mathcal{Q} - \{e_1\}$)

$$\sum_{M_1 \in R(e_1)} \max_{S \in \text{GenS}(\mathcal{Q}-\{e_1\})} \Psi\left(R(M_1), S\right) = \sum_{S \in \text{GenS}(\mathcal{Q}-\{e_1\})} \Psi(R, S).$$

This is captured by the last term of (13), which degenerates to just GenS($\mathcal{Q} - \{e_1\}$) when $\mathcal{X}$ has only one petal.

If $\mathcal{X}$ has $k \geq 2$ petals, we ask Algorithm 2 to peel off the petals $e_k, e_{k-1}, \dots, e_1$ one by one, and finally e_0. Let v_k be the join attribute between e_0 and e_k. Recall that when dealing with heavy tuples in $R(e_k)$, we remove v_k before making the recursive call on $\mathcal{Q} - \{e_k\} - \{v_k\}$. Thus, we invoke the induction hypothesis on $\mathcal{Q} - \{e_k\} - \{v_k\}$ to bound the cost of handling the heavy tuples in $R(e_k)$:

$$\sum_{a \in H(e_k, v_k)} \frac{R(e_k)|_{v_k=a}}{M} \cdot \max_{S \in \text{GenS}(\mathcal{Q}-\{e_k\}-\{v_k\})} \Psi(R|_{v_k=a}, S)$$
$$= \Psi(R, \{e_k, e_0\}) + \max_{S \in \{\{e_k\}\} \times \text{GenS}(\mathcal{Q}-\{e_k\}-\{v_k\})} \Psi(R, S).$$

The first term is captured by the first term of (13). For the second term, observe that $e_{k-1}, \dots, e_1, e_0$ still forms a star in $\mathcal{Q} - \{e_k\} - \{v_k\}$. Consider any $S \in \{\{e_k\}\} \times$ GenS($\mathcal{Q} - \{e_k\} - \{v_k\}$). If $e_0 \notin S$, it is captured by the second term of (13); if $e_0 \in S$, then S cannot have all of $e_1, \dots, e_{k-1}$ due to the property of GenS($\mathcal{Q} - \{e_k\} - \{v_k\}$), thus captured by the third term of (13).

When dealing with the light tuples in $R(e_k)$, recall that Algorithm 2 does not remove the join attribute v_k before making the recursive call, so we invoke the induction hypothesis on $\mathcal{Q} - \{e_k\}$ to bound the cost to handle the light tuples:

$$\sum_{M_k \in R(e_k)} \max_{S \in \text{GenS}(\mathcal{Q}-\{e_k\})} \Psi(R(M_k), S)$$
$$= \Psi(R, \{e_k\}) + \max_{S \in \text{GenS}(\mathcal{Q}-\{e_k\})} \Psi(R, S).$$

The first term is captured by the first term of (13). For the second term, note that in $\mathcal{Q} - \{e_k\}$, $\mathcal{X} - \{e_k\}$ is no longer a star as e_0 contains a unique attribute v_k, and GenS will peel each of them as a leaf. So we have

$$\text{GenS}(\mathcal{Q} - \{e_k\}) = 2^{\{e_0, e_1, \dots, e_{k-1}\}} \times \text{GenS}(\mathcal{Q} - \mathcal{X}).$$

This must be a subset of GenS($\mathcal{Q}$) since it does not contain e_k.

Finally, if there are two or more petals in $\mathcal{X}$ joining with e_0 on the same join attribute, we ask Algorithm 2 to peel off the extra petals first, to transform $\mathcal{X}$ into a simple star. These extra petals will be included in every S. But since all we need to show is that not all petals can be included in S if e_0 is included, which is already guaranteed by the analysis on the simple star, this does not affect the theorem. $\square$

A.3 Optimality of Algorithm 5

In Algorithm 2, the cost of AcyclicJoin in line 1 is $\widetilde{O}(\frac{N_3 N_5}{MB} + \frac{N_4}{B})$, but writing S to disk costs $\widetilde{O}(\frac{N_3 N_5}{B})$. The cost of AcyclicJoin in line 2 is $\widetilde{O}(\frac{N_1 N_3 N_5 N_7}{M^2 B} + \frac{N_2 N_6}{MB} + \frac{N_1 N_6}{MB} + \frac{N_2 N_7}{MB})$. There is $\frac{N_4}{B} \leq \frac{N_3 N_5}{B} \leq \frac{N_1 N_3 N_5 N_7}{M^2 B}$ due to relation size constraints and assumptions. The upper bound is exactly $\widetilde{O}(\frac{N_1 N_3 N_5 N_7}{M^2 B} + \frac{N_2 N_6}{MB} + \frac{N_1 N_6}{MB} + \frac{N_2 N_7}{MB})$.

Under this scenario, 3 balancing conditions might break: (a) $N_1 N_3 N_5 N_7 \geq N_2 N_4 N_6$, (b) $N_1 N_3 N_5 \geq N_2 N_4$, and (c) $N_3 N_5 N_7 \geq N_4 N_6$. There are eight combinations about these three conditions, but in fact only following four situations needing consideration by the dependency and symmetry among those balancing conditions.

(i) all (a)(b)(c) broken;

(ii) only (a)(b) broken, symmetric with only (a)(c) broken;

(iii) only (a) broken;

(iv) only (b) broken, symmetric with only (c) broken;

Note that if both (b)(c) are broken, (a) will definitely be broken and this case will degenerate to case (i). Next we will construct an instance for each case discussed above and show worst-case optimality of Algorithm 5.

For case (i) (ii)(iii), construct an instance I similarly as previously. Relations e_1 and e_7 are both one-to-one mapping from dom(v_1) onto dom(v_2) and from dom(v_7) onto dom(v_8) respectively. Relations e_2, e_4 and e_6 are cross products while e_3 and e_5 remain as many-to-many mappings. It is feasible by the constraints with implication $N_2 > N_1$ and $N_6 > N_7$. For such an I, only e_4 is not cross product. Thus there is $|\mathcal{Q}(I)| = N_1N_3N_5N_7$ with partial joins below.

$$\begin{aligned}|\mathcal{Q}(I, \{e_2, e_4, e_6\})| &= N_1N_3N_5N_7,\\ |\mathcal{Q}(I, \{e_2, e_7\})| &= N_2N_7,\\ |\mathcal{Q}(I, \{e_1, e_6\})| &= N_1N_6,\\ |\mathcal{Q}(I, \{e_2, e_6\})| &= N_2N_6.\end{aligned}$$

For case (iv), construct an instance I in a similar way. Relation e_1 is a one-to-one mapping from dom(v_1) onto dom(v_2). For attribute v_6, we set $|\text{dom}(v_6)| = 1$. Relations e_2, e_4, e_5 and e_6 are cross products while e_3 and e_7 remain as many-to-many mappings. It is feasible due to the relation size constraints and its implications $N_2 > N_1$, $N_7 > N_6$ and $N_4 > N_5$. For such an I, there is $|\mathcal{Q}(I)| = N_1N_3N_5N_7$ with partial joins below.

$$\begin{aligned}|\mathcal{Q}(I, \{e_2, e_4, e_7\})| &= N_1N_3N_5N_7,\\ |\mathcal{Q}(I, \{e_2, e_7\})| &= N_2N_7,\\ |\mathcal{Q}(I, \{e_1, e_6\})| &= N_1N_6,\\ |\mathcal{Q}(I, \{e_2, e_6\})| &= N_2N_6.\end{aligned}$$

A.4 Optimality of dumbbell joins

For a dumbbell join query $\mathcal{Q}$, we peel off the star with e_m being the core first. This generates two recursive calls, one on an lollipop join ($e_0 \bowtie \cdots \bowtie e_n \bowtie e_m$) and one on a star join ($e_n \bowtie e_{n+1} \bowtie \cdots \bowtie e_m$). For the lollipop join ($e_0 \bowtie \cdots \bowtie e_n \bowtie e_m$), we choose to peel off the star (with e_0 being the core), which generates two recursive calls on $L_3(e_0 \bowtie e_n \bowtie e_m)$ and $L_2(e_n \bowtie e_m)$. The $\mathbf{S}$ generated by GenS($\mathcal{Q}$) contains the following types of subsets. Recall that to prove optimality, for each $S \in \mathbf{S}$, we need to construct an instance I whose partial join size on S matches the maximum subjoin size, i.e.,

$$\max_R |\bowtie_{e\in S} R(e)| = |\mathcal{Q}(I, S)|$$

(i) $S = S_1 \cup S_2 \cup f$ with $S_1 \subsetneq \{e_{n+1}, e_{n+2}, \ldots, e_{m-1}\}$, $S_2 \subseteq \{e_1, \ldots, e_{n-1}\}$, $f \subseteq \{e_n, e_m\}$.

If $f = \{e_n\}$, the domain of join attribute is the solution of the following inequalities.

$$\begin{cases} N_0 = \prod_{i=1}^{n} |\text{dom}(v_i)|,\\ N_n = |\text{dom}(v_n)| \cdot |\text{dom}(v_m)|,\\ N_m = \prod_{i=n+1}^{m} |\text{dom}(v_i)|,\\ 1 \le |\text{dom}(v_i)| \le N_i, i = 1, 2, \cdots, m-1 \end{cases}$$

Tuples in relation $e_i, i \in [1, m-1] - \{n\}$ can be an arbitrary mapping from unique attributes onto dom(v_i). This construction is always feasible by balancing condition (7). Under this instance, we obtain

$$\max_R |\bowtie_{e\in S} R(e)| = \prod_{e\in S} N(e) = |\mathcal{Q}(I, S)|.$$

If $f = \{e_m\}$, the maximum subjoin size is

$$\max_R |\bowtie_{e\in S} R(e)| \le \prod_{e\in S_2} N(e) \cdot \max_R |\bowtie_{e\in S_1\cup f} R(e)|. \quad (14)$$

Let $R_{\max}$ be the R that maximizes $|\bowtie_{e\in S_1\cup\{e_m\}} R(e)|$ under the constraint $|\text{dom}(v_m)| \le N_n$. We construct instance I by extending $R_{\max}$. The domain of join attribute v_i for $i \in [1, n]$ has only one value. Any relation $e_i, i \in [1, n-1]$ is a one-to-many matching from the only value in dom(v_i) to the unique attribute of R_i. This construction is always feasible as $1 \cdot |\text{dom}(v_m)| \le N_n$ in a fully reduced instance. Under this instance, we have

$$\prod_{e\in S_2} N(e) \cdot \max_R |\bowtie_{e\in S_1\cup f} R(e)| = |\mathcal{Q}(I, S)|. \quad (15)$$

If $f = \{e_n, e_m\}$, there is also (14). Let $R_{\max}$ be the R that maximizes $|\bowtie_{e\in S_1\cup f} R(e)|$ under the constraint $|\text{dom}(v_n)| \le N_0$. We construct instance I by extending $R_{\max}$. The domain of join attribute v_i for $i \in [1, n-1]$ has only one value. Any relation $e_i, i \in [1, n-1]$ is a one-to-many matching from the only value in dom(v_i) to the unique attribute of R_i. This construction is feasible as $|\text{dom}(v_n)| \le N_0$ and (15) holds for every S.

(ii) $S = S_1 \cup S_2$ with $S_1 \subseteq \{e_{n+1}, e_{n+2}, \cdots, e_{m-1}\}$, $S_2 \subsetneq \{e_0, e_1, \cdots, e_n\}$. The instance construction I depends on two cases. If $e_0 \notin S$, the proof is the same as (i) when $f = \{e_n\}$. If $e_0 \in S$, the proof is almost the same as (i) when $f = \{e_m\}$.

(iii) $S \subseteq \{e_{n+1}, e_{n+2}, \cdots, e_{m-1}, e_m\}$. The subjoin of any S containing e_m always matches its partial join. For S not containing e_m, we construct the instance I by setting the domain of any join attribute with only one value. Each leaf can be a one-to-many matching from the only value in dom(v_i) to the unique attributes in R_i. Under this instance, we achieve the same result in (i) with $f = \{e_n\}$.

(iv) $S = S_1 \cup S_2 \cup f$ with $S_1 \subsetneq \{e_{n+1}, e_{n+2}, \cdots, e_{m-1}\}$, $S_2 \subsetneq \{e_1, \cdots, e_{n-1}\}$, $f \in \{\{e_0, e_m\}, \{e_0\}, \{e_m\}, \{e_n\}\}$. If $f \in \{\{e_0\}, \{e_m\}, \{e_n\}\}$, the case degenerates to (i). If $f = \{e_0, e_m\}$, the case is different. Without loss of generality, we assume $N_i \le N_j$ when $i \le j$ and the balancing condition (7) becomes $N_1N_nN_{n+1} \ge N_0N_m$. For any subjoin, there is a upper bound below

$$\max_R \Psi(R, S_1 \cup \{e_m\}) \cdot \max_R \Psi(S_2 \cup \{e_0\}) \cdot \frac{B}{M}$$

under constraints $|\text{dom}(v_n)| \le N_n$ and $|\text{dom}(v_m)| \le N_n$. We focus on $\max_R \Psi(S_2 \cup \{e_0\})$ under $|\text{dom}(v_n)| \le N_n$ first. When $N_0 < N_1N_n$, we know that

$$\max_{S_2} \max_R \Psi(R, S_2 \cup \{e_0\}) \le \frac{N_0 \prod_{k=2}^{n-1} N_k}{M^{n-2}B}$$

Otherwise, $\max_{S_2} \max_R \Psi(R, S_2 \cup \{e_0\})$ is bounded by

$$\max_{S_2} \frac{\min\{N_0, \prod_{e\notin S_2} N(e)\} \cdot \prod_{e\in S_2} N(e)}{M^{|S_2|}B}$$
$$\le \max\{\frac{\prod_{j=1}^{n} N_j}{M^{n-i}B}, \frac{N_0 \prod_{j=i+1}^{n-1} N_j}{M^{n-i-1}B}\}$$

where i is the index such that $\prod_{j=1}^{i-1} N_j \leq \frac{N_0}{N_p} < \prod_{j=1}^{i} N_j$. Note that at least one of $N_0 < N_n N_1$ and $N_m < N_n N_{n+1}$ occurs otherwise balancing condition (7) will be broken.

When $N_0 < N_n N_1$ and $N_m < N_n N_{n+1}$, the maximum subjoin is

$$\max_S \max_R \Psi(R, S) = \frac{N_0 \prod_{k=2}^{n-1} N_k}{M^{n-2} B} \cdot \frac{N_m \prod_{k=n+2}^{m-1} N_k}{M^{m-n-2} B}$$

by our analysis above. We construct instance I by setting the domain as

$$|\text{dom}(v_i)| = \begin{cases} \min\{N_1, N_0\}, & i = 1 \\ \max\{\frac{N_0}{N_1}, 1\}, & i = n \\ \min\{N_{n+1}, N_m\}, & i = n+1 \\ \max\{\frac{N_m}{N_{n+1}}, 1\}, & i = m \\ 1, & \textit{otherwise} \end{cases}$$

Relation R_i for $i \in [1, m-1] - \{n\}$ can be an arbitrary mapping from unique attributes onto $\text{dom}(v_i)$. The construction is feasible with $|\text{dom}(v_n)| \cdot |\text{dom}(v_m)| \leq N_n$ implied by balancing condition (7). Under this instance, there is

$$\max_S \max_R \Psi(R, S) = \psi(I, S)$$

where $S = S_1 \cup S_2 \cup f$ with $S_1 = \{e_{n+2}, e_{n+3}, \cdots, e_{m-1}\}$, $S_2 = \{e_2, e_3, \cdots, e_{n-1}\}$, $f = \{e_0, e_m\}$.

When $N_0 < N_n N_1$ and $N_m \geq N_n N_{n+1}$, there is $\frac{N_0}{N_1} \leq \frac{N_n N_{n+1}}{N_m} \leq 1$. The maximum subjoin is

$$\max_S \max_R \Psi(R, S) = \frac{N_0 \prod_{k=2}^{n-1} N_k}{M^{n-1}} \cdot \max_R \Psi(R, S_1 \cup \{e_m\}$$

Let $R_{\max}$ be the R that maximizes $|\bowtie_{e \in S_1 \cup \{e_m\}} R(e)|$ under the constraint $|\text{dom}(v_m)| \leq N_n$. We construct instance I by extending $R_{\max}$. We set $|\text{dom}(v_1)| = N_0$ and the domain of join attribute v_j for $j \in [2, n]$ with only one value. This construction is always feasible as $|\text{dom}(N_1)| \leq N_1$ and $|\text{dom}(v_m)| \leq N_n$. We know that

$$\max_S \max_R \Psi(R, S) = \psi(I, S)$$

where $S = S_1 \cup S_2 \cup f$ with S_1 as $\arg\max_{S_1} \max_R \Psi(R, S_1 \cup \{e_m\})$, $S_2 = \{e_2, e_3, \cdots, e_{n-1}\}$, $f = \{e_0, e_m\}$.

When $N_0 \geq N_n N_1$ and $N_m < N_n N_{n+1}$, the instance construction is the same as the case $N_0 < N_n N_1$ and $N_m \geq N_n N_{n+1}$.