# **Evaluating Rank Joins with Optimal Cost**

Technical Report UCSC-CRL-07-10

Karl Schnaitter UC Santa Cruz karlsch@soe.ucsc.edu

### ABSTRACT

In the rank join problem, we are given a set of relations and a scoring function, and the goal is to return the join results with the top K scores. When the input relations are given in ranked order and the scoring function is monotonic, it is possible to solve the rank join problem without reading all of the input. This key observation allows for efficient rank join algorithms that may be used in real-world applications. In this paper, we present a thorough analysis of such rank join algorithms. A strong point of our analysis is that it is based on a more general problem statement than previous work, making it more relevant to the execution model that is employed by database systems. One of our results indicates that the well known HRJN algorithm has shortcomings, because it does not stop reading its input as soon as possible. We find that it is NP-hard to overcome this weakness in the general case, but cases of limited query complexity are tractable. We prove the latter with an algorithm that infers provably tight bounds on the potential benefit of reading more input in order to stop as soon as possible. As a result, the algorithm achieves a cost that is within a constant factor of the best possible.

### 1. INTRODUCTION

A relational ranking query (or a top-K join query) specifies a scoring function over the results of a join and returns the K tuples with the highest scores. As an example, Figure 1 presents a ranking query (written in an SQL-like language) that retrieves the top 10 hotels and restaurants located in the same city, giving priority to the cheap hotels and the best restaurants with live music. Ranking queries have become increasingly popular in many application domains, from multimedia retrieval [2] to uncertain databases [1], as they allow a user to focus on the most relevant query results.

In order to evaluate relational ranking queries efficiently, recent studies [1, 4, 7, 8] have introduced specialized *rank join* algorithms that access the input tuples in order of their importance and thus generate the top join results after reading only a fraction of the input data. In our example, this means that cheap hotels and highly rated restaurants with music would be read first. Rank join algorithms borrow ideas from aggregation algorithms for middleware [3], but they assume that aggregation is performed through a relational join that is more general than a merge of object lists.

The performance of rank join algorithms has been studied empirically, but each experiment has been limited to a subset of existing algorithms. Some works have also included analytical studies. These studies, however, generally assume that each input relation is ranked according to a single numerical attribute, and are thus inconsistent with recent works in ranking query optimization [6, 9] that need to join relations with multiple score attributes. Overall, previous studies provide little insight into the *comparative perfor*- Neoklis Polyzotis UC Santa Cruz alkis@cs.ucsc.edu

SELECT h.name, r.name FROM Hotel h, Restaurant r WHERE h.city = r.city RANK BY 0.4/h.price + 0.4 \* r.rating + 0.2 \* r.hasMusic LIMIT 10 Figure 1: Example ranking query.

*mance* of rank join algorithms under realistic assumptions, an issue that is crucial for the realization of ranking query processors.

**Our contributions.** Motivated by the previous observations, we embark on an extensive analysis of rank join algorithms. In contrast to earlier works, we consider a general model where input relations may have multiple scoring attributes. This model has direct applications in practice, as it lends itself to real-world queries and the query execution plans used by database systems.

In order to structure our analysis, we introduce Pull/Bound Rank Join (PBRJ), an algorithm template that generalizes previous rank join algorithms. The idea of PBRJ is to alternate between *pulling* tuples from input relations and *upper bounding* the score of join results that use the unread part of the input. The join results are collected as tuples are pulled, and the algorithm stops once the top K buffered results have a score at least equal to the upper bound. There are many possible ways that an algorithm may pull from its input and compute an upper bound on unseen join results—for instance, one simple pulling strategy is to read from all input relations evenly, but it may be possible prioritize access to the most promising relations. These decisions lead to a rich family of PBRJ instantiations that is general enough to model a wide variety of practical algorithms. The PBRJ template thus provides a solid foundation for a study of rank join processing.

Using the PBRJ framework, we first analyze the performance of the well known HRJN algorithm template [4]. Our results reveal that some HRJN variants have strong performance compared to other members of the HRJN family, but the performance of HRJN algorithms can be arbitrarily bad in general because they use a loose upper bound on the scores of remaining join results. We remedy this deficiency with a PBRJ algorithm that computes an upper bound that is provably tight. The significance of this algorithm is that it is the first to provide strong performance guarantees for the practical rank join model that we consider. The algorithm runs in polynomial time for queries with bounded size, but we show that finding a tight bound requires solving a problem that is NP-hard in general. Finally, we extend our analysis to additional cases, such as alternative cost metrics and pipelined query plans. One of our results provides theoretical justification for the use of multi-way join operators as opposed to plans of several operators. This result is interesting since it suggests that the search space of ranking query optimizers may be simplified without sacrificing performance.

In summary, this paper presents an in-depth analysis of rank join algorithms for relational ranking queries. Our work makes significant technical contributions on several fronts: we characterize the performance of existing algorithms, introduce a new algorithm with strong performance guarantees, and analyze the complexity of a key aspect of the problem, namely inferring tight score bounds. Moreover, the presented results are directly applicable to existing ranking query processors, and are thus of interest to application developers and system designers.

### 2. PRELIMINARIES

In this section, we formalize the rank join problem and state the assumptions of our analysis. We note that these assumptions are revisited in Section 6 when we discuss extensions to our work.

**Problem definition.** Consider a natural join of relations  $R_1, \ldots, R_n$  where each  $\tau_i \in R_i$  is composed of *named attributes* and *base scores*. The base scores are denoted as a vector  $\mathbf{b}(\tau_i) \in [0, 1]^{e_i}$  for some  $e_i$ , and signify the importance of the tuple according to criteria specified by the query. Base scores are aggregated using a *scoring function* S that computes the score of a join result  $\tau$  as  $S(\mathbf{b}(\tau))$ . We may also use  $S(\tau)$  as a shorthand for the score of  $\tau$ . Following common practice, we assume that S is monotonic, i.e.,  $S(x_1, \ldots, x_e) \leq S(y_1, \ldots, y_e)$  if  $x_i \leq y_i$  for all *i*. Let  $\tau$  be a join result such that  $\tau = \tau' \bowtie \rho$  for some intermediate

Let  $\tau$  be a join result such that  $\tau = \tau' \bowtie \rho$  for some intermediate results  $\tau'$  and  $\rho$ . We define  $\overline{S}(\tau')$  to be the value of S using the base scores of  $\tau'$ , and substituting 1 for any that are missing. The monotonicity of S implies that  $S(\tau) \le \overline{S}(\tau')$  since each base score of  $\rho$  is at most 1. Thus we call  $\overline{S}(\tau')$  the *score bound* of  $\tau'$  since it is an upper bound on the scores of join results derived from  $\tau'$ .

Loosely stated, the goal of the rank join problem is to find K tuples with the highest scores from the natural join of  $R_1, \ldots, R_n$ . We formally define an instance of the rank join problem as follows.

DEFINITION 2.1. An instance of the rank join problem is an (n+2)-tuple  $(R_1, \ldots, R_n, S, K)$  such that S is a monotonic scoring function, the relations  $R_1, \ldots, R_n$  are accessed sequentially in decreasing order of  $\overline{S}$ , and  $1 \le K \le |R_1| \bowtie \cdots \bowtie R_n|$ .

We do not place any restrictions on the input relations except that each  $R_i$  is accessed sequentially and in decreasing order of  $\overline{S}$ . More formally, we use  $R_i[p]$  to denote the *p*-th tuple in  $R_i$  and assume that  $\overline{S}(R_i[p]) \ge \overline{S}(R_i[q])$  for  $q \ge p$ . We note that this particular access model is a common assumption in previous studies on the evaluation of rank joins in database systems [4, 8]. The definition also requires that at least *K* join results exist, which guarantees that it is possible to fulfill a request for the top *K* results.

A solution of  $I = (R_1, \ldots, R_n, S, K)$  is an ordered relation O that comprises the top K tuples of  $R_1 \bowtie \ldots \bowtie R_n$  ordered by S. We note that there may be more than one possible solution O for a particular instance I if there are tie scores in the output, but the sequence of score bounds in O is completely determined by I. In other words, if O and O' are solutions of I, then S(O[p]) = S(O'[p]) for all p. The last score, S(O[K]), is referred to as the terminal score, and is denoted  $S^{\text{term}}$ .

Returning to the query in Figure 1, the base scores are given by  $\mathbf{b}(h) = \langle 1/h.price \rangle$ ,  $\mathbf{b}(r) = \langle r.rating, r.hasMusic \rangle$ , and the scoring function S is a weighted average. The top 10 join results are found by solving the instance (H, R, S, 10). It is interesting to note that the input relation R is associated with two base scores. This situation also occurs when an input relation is the output of a subquery that joins multiple tables in a pipelined execution plan.

Algorithms considered. Our theoretical analysis is focused on deterministic algorithms that solve the rank join problem. More

specifically, algorithms considered in this paper must be determined by the visible contents of the input relations and the values of the  $\overline{S}$  function on combinations of input tuples that have been read. This means an algorithm may not access any metadata, such as the domains of the attributes. These basic assumptions do not exclude any previously proposed rank join algorithms, so they provide reasonable boundaries for our analysis.

**Performance metrics.** A *cost metric* is a function cost(A, I) that yields the cost of solving instance I with algorithm A. The cost metrics in this paper are defined using the idea of *depth*. The depth on an input relation  $R_i$  is the number of tuples read sequentially from  $R_i$  before returning a solution. For a rank join algorithm A and instance  $I = (R_1, \ldots, R_n, S, K)$ , the depth on  $R_i$  is denoted depth(A, I, i). We define sumDepths(A, I) as the sum of depths on all inputs. Clearly, sumDepths is an interesting cost metric as it indicates the amount of I/O performed by an algorithm.

Given a class of algorithms  $\mathcal{B}$ , a class of problem instances  $\mathcal{J}$ , and a cost metric *cost*, we say that a rank join algorithm  $A \in \mathcal{B}$ is *optimal* if  $cost(A, I) \leq cost(A', I)$  for all rank join algorithms  $A' \in \mathcal{B}$  and problem instances  $I \in \mathcal{J}$ . Our analysis also uses a relaxed form of optimality known as instance optimality. We say that A is *instance-optimal* if there exist constants  $c_1$  and  $c_2$  such that  $cost(A, I) \leq c_1 \cdot cost(A', I) + c_2$  for all  $A' \in \mathcal{B}$  and  $I \in \mathcal{J}$ . The constant  $c_1$  is called the *optimality ratio*.

We define  $\mathcal{A}$  as the class of deterministic rank join algorithms described above, and  $\mathcal{I}$  as the set of all problem instances satisfying Definition 2.1. Unless otherwise specified, our optimality results are with respect to  $\mathcal{A}$  and  $\mathcal{I}$  with the cost metric *sumDepths*.

### **3.** PULL/BOUND RANK JOIN

In this section, we introduce the *Pull/Bound Rank Join* (PBRJ) algorithm template that forms the backbone of our analysis.

The PBRJ template, shown in Figure 2, is instantiated by a deterministic *pulling strategy* P and *bounding scheme* B. On each loop iteration of PBRJ, the pulling strategy P chooses a relation  $R_i$  to read, and the new tuple  $\rho_i$  is stored in an input buffer  $HR_i$  (typically a hash table). New join results are generated by joining  $\rho_i$  with the tuples in the other input buffers  $HR_j$  for  $j \neq i$ . The generated results are pushed to an output buffer O that holds the top K results seen so far. After each tuple is processed, it is given to the bounding scheme B via the method *updateBound*, which returns a new upper bound on the score of unseen join results. The results O are returned when the K-th buffered result has a score that is at least as large as the bound t provided by the bounding scheme, since this indicates that the buffered results cannot be improved by reading more tuples.

We can easily show that PBRJ is correct if we require that P

```
Algorithm template PBRJ(R_1, \ldots, R_n, S, K)
Template parameters: pulling strategy P; bounding scheme B
Input: relations R_1, \ldots, R_n; scoring function S; result size K
Output: set of K join results with highest score
Data structures: input buffers HR_1, \ldots, HR_n; output buffer O
1. t \leftarrow \infty
    while |O| < K OR \min_{\omega \in O} S(\omega) < t do
2
3.
       i \leftarrow P.chooseInput()
       \rho_i \leftarrow \text{next unseen tuple of } R_i
4.
       R \leftarrow HR_1 \bowtie \dots HR_{i-1} \bowtie \{\rho_i\} \bowtie HR_{i+1} \bowtie \dots \bowtie HR_n
5.
       Add each member of R to O, retaining only the top K tuples
6.
7.
        Add \rho_i to HR_i
8.
       t \leftarrow B.updateBound(\rho_i)
9. end while
10. return O
```

```
Figure 2: PBRJ Template.
```

returns the index of an unexhausted relation, and that B returns a correct upper bound on the scores of join results that use at least one unread tuple. We define  $\mathcal{F}$  as the set of instantiations of PBRJ with a deterministic pulling strategy and bounding scheme that satisfy these requirements, and note that  $\mathcal{F}$  is infinite in principle.

We observe that  $\mathcal{F}$  is a proper subset of the class  $\mathcal{A}$  of rank join algorithms that we consider, since  $\mathcal{F}$  only contains deterministic rank join algorithms and some members of  $\mathcal{A}$  do not follow the PBRJ template. However, we can show that  $\mathcal{F}$  is equivalent to  $\mathcal{A}$  in terms of I/O performance, as stated by the following theorem.

THEOREM 3.1. Let  $A \in \mathcal{A}$ . There exists a PBRJ algorithm  $F \in \mathcal{F}$  such that depth(A, I, i) = depth(F, I, i) for all problem instances  $I = (R_1, \ldots, R_n, \mathcal{S}, K)$  and all *i*.

**Proof:** We sketch a description of the pulling strategy and bounding scheme of F. Initially, the pulling strategy accesses the relations in the same order as A, and the bounding scheme returns  $\infty$ . Once F has read all the tuples seen by A, the bounding scheme returns  $\mathcal{S}^{\text{term}}$ . This behavior is deterministic by virtue of the fact that A is deterministic. After reading the same data as A, the output buffer of F must contain a complete solution. Since the bound is  $\mathcal{S}^{\text{term}}$  at this moment, F will halt with the same depths as A.

In other words, we can chart the cost of algorithms in A by studying the cost of PBRJ for different choices of P and B. We therefore focus our study on instantiations of the PBRJ template.

**Choices for** *P* **and** *B***.** We now review some choices for the pulling strategy and bounding scheme that are used in existing rank join algorithms (primarily HRJN [4]).

To the best of our knowledge, only one previously studied bounding scheme applies to our formulation of the rank join problem.

Corner Bound: Maintain bounds  $t_i$  for each input relation  $R_i$  that are initially set to S(1, ..., 1). A call of  $updateBound(\rho_i)$ sets  $t_i = \overline{S}(\rho_i)$  if  $R_i$  still has unseen tuples, or  $t_i = -\infty$  if  $R_i$  is exhausted. The returned bound is  $\max(t_1, ..., t_n)$ .

The name of the corner bound comes from the fact that the bound  $t_i$  for a relation  $R_i$  is computed by setting every base score outside of  $R_i$  to 1, which effectively sets the base scores of other relations to the corner of their domain.

We distinguish the following two pulling strategies:

- *Round-Robin:* This simple strategy accesses the inputs in a round-robin fashion. Without loss of generality, we assume that the inputs are accessed in the following order:  $R_1, \ldots, R_n$ .
- *Corner-Bound-Adaptive:* This strategy prioritizes inputs based on the per-input corner bounds. The idea is to select the input  $R_i$  that has the highest bound  $t_i$ . Ties are broken in favor of the input with the least current depth, or the input with the least index. Given that  $t = \max(t_1, \ldots, t_n)$ , the intuition is to drop the bound below the terminal score and thus lead to termination faster.

In what follows, we use  $PBRJ_c^{RR}$  and  $PBRJ_c^*$  respectively to refer to the variants of PBRJ that use these strategies with the corner bound.

### 4. THE CORNER BOUND

In the previous section, we introduced the class  $\mathcal{F}$  comprising instantiations of the PBRJ template with any pulling strategy and bounding scheme. This section is focused on a smaller class  $\mathcal{F}_c$ , which we define as the set of PBRJ algorithms that use the corner bound. As noted earlier, the corner bound is the only previously studied scheme that can be applied to our formulation of the rank join problem. Moreover,  $\mathcal{F}_c$  is equivalent to the HRJN family of algorithms [4] that have received significant attention in previous studies [6, 7, 9], so any results that we can derive about  $\mathcal{F}_c$  also hold for HRJN. We also note that the "basic" HRJN operator and the HRJN\* variant are essentially iterative implementations of PBRJ<sup>R</sup><sub>c</sub> and PBRJ<sup>2</sup><sub>c</sub>, and hence these corresponding algorithms share the same performance guarantees.

In the first part of this section, we investigate the space of algorithms in  $\mathcal{F}_c$ , effectively analyzing the impact of the pulling strategy when the corner bound is employed. We then expand our analysis to see how well  $\mathcal{F}_c$  measures up to other algorithms in  $\mathcal{F}$ . Before presenting our results, we observe that an algorithm in  $\mathcal{F}_c$  will terminate if and only if K results have been buffered and the last read tuple  $\rho_i$  from each unexhausted  $R_i$  satisfies  $\overline{\mathcal{S}}(\rho_i) \leq \mathcal{S}^{\text{term}}$ . These conditions provide a convenient means to characterize the termination of PBRJ when the corner bound is employed, and we use them extensively in our analysis.

## 4.1 Analysis of Algorithms within $\mathcal{F}_c$

# 4.1.1 Instance Optimality of PBRJ<sup>RR</sup> and PBRJ<sup>\*</sup><sub>c</sub>

It would be preferable if we could find a pulling strategy that is superior in all cases, since this would yield an algorithm that is optimal within the class  $\mathcal{F}_c$ . Unfortunately, it can be shown that no such strategy exists, using the adversary argument provided by Fagin et al. [3]. The argument actually yields a stronger result: if we define  $\mathcal{I}^{n\text{-rel}} \subseteq \mathcal{I}$  to be the instances with *n* input relations, then no algorithm can have an optimality ratio less than *n* within instances  $\mathcal{I}^{n\text{-rel}}$  and algorithms  $\mathcal{F}_c$ .

We show below that both PBRJ<sub>c</sub><sup>RR</sup> and PBRJ<sub>c</sub><sup>\*</sup> achieve this lower bound. Our results are given by three theorems. First, we state the instance optimality of PBRJ<sub>c</sub><sup>RR</sup>. Then we show that the cost of PBRJ<sub>c</sub><sup>RR</sup> is never more than PBRJ<sub>c</sub><sup>RR</sup> on the same problem instance. These two results lead to the third theorem which states the instance optimality of PBRJ<sub>c</sub><sup>\*</sup>.

THEOREM 4.1. PBRJ<sub>c</sub><sup>RR</sup> is instance-optimal within algorithms  $\mathcal{F}_{c}$  and instances  $\mathcal{I}^{n\text{-rel}}$  with an optimality ratio of n.

**Proof:** Let  $I = (R_1, \ldots, R_n, \mathcal{S}, K)$  be any problem instance in  $\mathcal{I}^{n\text{-rel}}$  and choose any algorithm  $A \in \mathcal{F}_c$ . Suppose that A pulls  $p_i$  tuples from each relation  $R_i$  and define  $p_{\max} = \max_i(p_i)$ . We claim that PBRJ<sub>c</sub><sup>RR</sup> cannot pull more than  $p_{\max}$  tuples from any relation. We prove this by contradiction. Assume that PBRJ<sub>c</sub><sup>RR</sup> pulls  $p_{\max}$  complete rounds, and then pulls an additional tuple. At the point that this additional tuple is read, PBRJ<sub>c</sub><sup>RR</sup> has seen at least all of the input tuples seen by A, so it has buffered the top K results returned by A. However PBRJ<sub>c</sub><sup>RR</sup> does not halt, so there must be some unexhausted  $R_i$  such that  $\overline{\mathcal{S}}(R_i[p_{\max}]) > \mathcal{S}^{\text{term}}$ . This contradicts the definition of the corner bound, since A halts at a depth of  $p_i \leq p_{\max}$  on  $R_i$ . It follows that PBRJ<sub>c</sub><sup>RR</sup> cannot pull more than  $p_{\max}$  tuples from any relation, from which we have

 $sumDepths(\text{PBRJ}_{c}^{RR}, I) \leq n \cdot p_{\max} \leq n \cdot sumDepths(A, I).$ 

Since A and I were chosen arbitrarily, we conclude that  $PBRJ_c^{RR}$  is instance-optimal with an optimality ratio of n.

THEOREM 4.2. Let  $I = (R_1, \ldots, R_n, S, K)$  be any instance. Then  $depth(\text{PBRJ}_c^{\mathsf{R}}, I, k) \leq depth(\text{PBRJ}_c^{\mathsf{R}}, I, k)$  for all k.

*Proof:* By contradiction. Suppose that there is some input k such that  $depth(\text{PBRJ}_{c}^{\text{R}}, I, k) > depth(\text{PBRJ}_{c}^{\text{RR}}, I, k)$ . Let  $O^{\text{RR}}$  be the result returned by  $\text{PBRJ}_{c}^{\text{RR}}$ . For each *i*, we define  $p_{i}^{\text{RR}} =$ 

 $depth(\text{PBRJ}_{c}^{\text{RR}}, I, i)$ . Also let  $p_i$  be the current depth of  $R_i$  when  $\text{PBRJ}_{c}^{*}$  decides to pull  $R_k[p_k^{\text{RR}}+1]$ . We derive a contradiction by showing that  $\text{PBRJ}_{c}^{*}$  must actually halt at this moment.

We first claim that PBRJ<sup>\*</sup> has seen all input tuples that participate in  $O^{\text{RR}}$ . This is shown by considering any unexhausted relation  $R_i$  and checking that no tuples beyond  $R_i[p_i]$  may participate in  $O^{\text{RR}}$ . Since PBRJ<sup>\*</sup><sub>c</sub> pulls  $R_k$ , we know that  $R_k$  is also unexhausted and  $\overline{S}(R_k[p_k]) \ge \overline{S}(R_i[p_i])$ . We proceed in two cases.

*Case 1:*  $\overline{S}(R_k[p_k]) = \overline{S}(R_i[p_i])$ . In this case, there was a tie, and PBRJ<sup>c</sup><sub>c</sub> breaks it by pulling  $R_k$ . There are two ways the tie may be broken. The first possibility is that  $p_k < p_i$  which means

$$p_i \ge p_k + 1 = p_k^{\mathsf{RR}} + 1 \ge p_i^{\mathsf{RR}}.$$

The other possibility is that  $p_k = p_i$  and  $k \leq i$  which means

$$p_i = p_k = p_k^{\rm RR} \ge p_i^{\rm RR}$$

where the last inequality is based on the order that PBRJ\_c^{RR} pulls from relations. Now we have observed that  $p_i \ge p_i^{RR}$  so PBRJ\_c^\* has seen all the tuples in  $R_i$  that PBRJ\_R^{RR} has seen.

Case 2:  $\overline{S}(R_k[p_k]) > \overline{S}(R_i[p_i])$ . Since PBRJ<sup>RR</sup><sub>c</sub> terminates without reading more than  $p_k^{\text{RR}} = p_k$  tuples from  $R_k$ , we know  $S^{\text{term}} \ge \overline{S}(R_k[p_k]) > \overline{S}(R_i[p_i])$ . This means that no tuple past  $R_i[p_i]$  can contribute to a solution.

These cases show that  $PBRJ_c^*$  has seen every input tuple that participates in the solution  $O^{RR}$ , so it must have buffered K results. Furthermore, for each  $R_i$  not exhausted by  $PBRJ_c^*$ , we have

 $\overline{\mathcal{S}}(R_i[p_i]) \leq \overline{\mathcal{S}}(R_k[p_k]) = \overline{\mathcal{S}}(R_k[p_k^{\mathsf{RR}}]) \leq \mathcal{S}^{\text{term}}.$ 

Now the termination criteria of PBRJ<sub>c</sub><sup>\*</sup> are met, so it will halt before reading  $R_k[p_k^{RR} + 1]$ , contradicting our assumption.

The previous result provides an interesting tool for the analysis of PBRJ<sub>c</sub><sup>\*</sup>. Essentially, it is non-trivial to analyze the performance of PBRJ<sub>c</sub><sup>\*</sup> on an arbitrary input instance, as its depths depend on the distribution of scores among the inputs. The simplicity of the round-robin strategy, on the other hand, facilitates the derivation of optimality properties for the depths of PBRJ<sub>c</sub><sup>RR</sup>, e.g., the instanceoptimality property of Theorem 4.1. Theorem 4.2 makes possible to transfer the same properties to PBRJ<sub>c</sub><sup>\*</sup> by virtue of the fact that PBRJ<sub>c</sub><sup>\*</sup> can never exceed the depths of the round-robin strategy. The following result is a direct application of this idea.

THEOREM 4.3. PBRJ<sup>\*</sup><sub>c</sub> is instance-optimal within algorithms  $\mathcal{F}_{c}$  and instances  $\mathcal{I}^{n\text{-rel}}$  with an optimality ratio of n.

### 4.1.2 Optimal Cases for PBRJ<sup>\*</sup><sub>c</sub>

Our analysis has shown that  $PBRJ_c^{RR}$  and  $PBRJ_c^*$  are instanceoptimal within  $\mathcal{F}_c$ , but we are unable to show that any algorithm is optimal in all cases. The main obstacle to optimality comes from duplicate input scores, since ties create ambiguity in determining which relation will lead to termination more quickly. Fortunately, as we show below, duplicate input scores only cause difficulty if the repeated score bounds are equal to the threshold  $\mathcal{S}^{term}$ . The intuitive reason is that any algorithm in  $\mathcal{F}_c$  will need to read beyond the tuples whose bound is strictly greater than  $\mathcal{S}^{term}$ , and hence any ties in that part of the input will not affect performance.

We structure our analysis based on two different kinds of ties that may occur between input scores and  $S^{\text{term}}$ . We first consider *intra-input threshold ties* where  $\overline{S}(R[p]) = \overline{S}(R[q]) = S^{\text{term}}$  for some input relation R and distinct indices  $p \neq q$ . We also consider *inter-input threshold ties* where  $\overline{S}(R[a]) = \overline{S}(R'[a']) = S^{\text{term}}$ for some distinct relations  $R \neq R'$  and indices a, a'. We define  $\mathcal{I}^{\text{intra}} \subseteq \mathcal{I}$  and  $\mathcal{I}^{\text{inter}} \subseteq \mathcal{I}$  to be the class of instances with intraand inter-input threshold ties respectively. We show below that PBRJ<sup>\*</sup><sub>c</sub> is optimal within  $\mathcal{F}_c$  with a very light restriction on the ties occurring in the input. In particular, we show that optimality holds for any problem instance outside  $\mathcal{I}^{intra} \cap \mathcal{I}^{inter}$  where *both* intra- and inter-input threshold ties occur. Our proof uses a lemma showing that PBRJ<sup>\*</sup><sub>c</sub> minimizes the depth on each input when the problem instance is in  $\mathcal{I} - (\mathcal{I}^{intra} \cap \mathcal{I}^{inter})$ . The optimality result follows immediately.

LEMMA 4.1. Suppose that  $A \in \mathcal{F}_{c}$  uses the corner bound, and let  $I = (R_{1}, \ldots, R_{n}, \mathcal{S}, K)$  such that  $I \in \mathcal{I} - (\mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}})$ . Then  $depth(\text{PBRJ}_{c}^{*}, I, k) \leq depth(A, I, k)$  for all k.

**Proof:** Assume for contradiction that there exists some input k such that  $depth(\text{PBRJ}^*_c, I, k) > depth(A, I, k)$ . For all i, define  $p_i^A = depth(A, I, i)$  and let  $p_i$  denote the number of tuples that  $\text{PBRJ}^*_c$  pulls from  $R_i$  before pulling  $R_k$  past  $p_k^A$ . Let  $R_i$  be any relation not exhausted by  $\text{PBRJ}^*_c$ . Observe that

$$\overline{\mathcal{S}}(R_i[p_i]) \le \overline{\mathcal{S}}(R_k[p_k]) \le \mathcal{S}^{\text{term}}$$

since PBRJ<sup>\*</sup><sub>c</sub> pulls from  $R_k$  and A halts at depth  $p_k^A = p_k$  on  $R_k$ .

We next prove that PBRJ<sub>c</sub><sup>\*</sup> has buffered a complete solution, with two cases. If  $I \notin \mathcal{I}^{\text{inter}}$  then at least one of the above inequalities is strict when  $i \neq k$ , and hence  $\overline{\mathcal{S}}(R_i[p_i]) < \mathcal{S}^{\text{term}}$ . It follows that PBRJ<sub>c</sub><sup>\*</sup> has buffered a complete solution since it has observed every tuple in  $R_k$  seen by A, and it has observed every tuple in  $R_i \neq R_k$  that can contribute to a solution. Now if  $I \notin \mathcal{I}^{\text{intra}}$ then  $R_i$  has at most one tuple with a score bound equal to  $\mathcal{S}^{\text{term}}$ . Hence,  $\overline{\mathcal{S}}(R_i[p_i]) \leq \mathcal{S}^{\text{term}}$  implies that each tuple beyond  $R_i[p_i]$ has a score bound below  $\mathcal{S}^{\text{term}}$  and cannot contribute to a solution. It follows that a complete solution has already been seen.

Now we also have  $S^{\text{term}} \geq \overline{S}(R_i[p_i])$ , so PBRJ<sup>\*</sup><sub>c</sub> will halt without reading past  $R_k[p_k^A]$ , which contradicts our assumption.

THEOREM 4.4. PBRJ<sup>\*</sup><sub>c</sub> is optimal within algorithms  $\mathcal{F}_c$  and instances  $\mathcal{I} - (\mathcal{I}^{inter} \cap \mathcal{I}^{intra})$ .

### **4.2** Comparative Analysis of $\mathcal{F}_{c}$ and $\mathcal{A}$

In the previous subsection, we showed various optimality properties of PBRJ<sub>c</sub><sup>RR</sup> and PBRJ<sub>c</sub><sup>c</sup> compared to other algorithms in  $\mathcal{F}_c$ . Now we consider the performance of these algorithms compared to the general class of algorithms  $\mathcal{A}$ . We first state our positive results. We define  $\mathcal{I}^{e\text{-dim}}$  to be the class of problem instances with at most *e* base scores for each input relation, and recall that  $\mathcal{I}^{n\text{-rel}}$  is the class of problem instances with at most *n* input relations. If we intersect  $\mathcal{I}^{1\text{-dim}}$  and  $\mathcal{I}^{2\text{-rel}}$ , we get a very simple class of inputs where PBRJ<sub>c</sub><sup>RR</sup> and PBRJ<sub>c</sub><sup>R</sup> are instance optimal.

THEOREM 4.5.  $PBRJ_c^{RR}$  and  $PBRJ_c^*$  are instance-optimal within instances  $\mathcal{I}^{2\text{-rel}} \cap \mathcal{I}^{1\text{-dim}}$  with an optimality ratio of 2.

*Proof:* The proof for  $PBRJ_c^{RR}$  appears in Theorem 3 of [4], and instance optimality extends to  $PBRJ_c^*$  by Theorem 4.2.

The previous theorem places very restrictive conditions on the class of problem instances. If we weaken these conditions at all, the instance optimality property of PBRJ<sub>c</sub><sup>RR</sup> and PBRJ<sub>c</sub><sup>\*</sup> is lost. In fact, as we show in the next theorem, there is no instance optimal algorithm using the corner bound when we consider minimal generalizations of  $\mathcal{I}^{2\text{-rel}} \cap \mathcal{I}^{1\text{-dim}}$ .

THEOREM 4.6. No PBRJ algorithm using the corner bound is instance-optimal, either (a) within instances  $\mathcal{I}^{3\text{-rel}} \cap \mathcal{I}^{1\text{-dim}}$ , or (b) within instances  $\mathcal{I}^{2\text{-rel}} \cap \mathcal{I}^{2\text{-dim}}$ .

*Proof (a):* Consider a problem instance  $I = (R_1, R_2, R_3, S, 1)$  where each  $R_i$  has a base score  $b_i$  and  $S(b_1, b_2, b_3) = b_1 + b_2 + b_3$ . The relations are joined on their first attribute, and their contents are described as follows.

$R_1$	Join	$b_1$	$R_2$	Join	$b_2$	$R_3$	Join	$b_3$
	a	1.0		y	1.0		z	1.0
	x	0.9		a	0.7		a	0.8
		•		y	0.4		z	0.4
	x 0.6							
	x	0.4						

Let  $\tau$  be the result of joining the visible tuples with join attribute a for a score of 2.5. None of the hidden rows of  $R_2$  or  $R_3$  can lead to a higher score since the last shown score bounds are 2.4 in each relation. Therefore, any solution must involve the single known result from  $R_2 \bowtie R_3$ , but this partial result has a score bound of  $2.5 = S(\tau)$ . With this reasoning, an algorithm may correctly return  $\tau$  after reading the first row of  $R_1$  and the first three rows of  $R_2$  and  $R_3$ . Any algorithm in  $\mathcal{F}_c$  must read all of  $R_1$  in order to lower the corner bound below 2.5, leading to arbitrarily high cost.

*Proof (b):* Consider the instance  $(R_1, R_2 \bowtie R_3, S, 1)$ . The first score bound in  $R_2 \bowtie R_3$  is 2.5, so all join results have a score  $\leq 2.5$ . Hence an algorithm may correctly return the join result with score 2.5 after reading one tuple from each relation. Similar to proof (a), any algorithm in  $\mathcal{F}_c$  must read all of  $R_1$ .

We remark that the problem instances used in this proof did not fall in the classes  $\mathcal{I}^{\mathrm{intra}}$  or  $\mathcal{I}^{\mathrm{inter}}$  that we showed to cause suboptimality for PBRJ<sup>\*</sup><sub>c</sub> earlier in this section. Thus we cannot hope to gain any ground by forbidding threshold ties.

We can make two interesting observations at this point. First, the results in this section provide strong evidence in favor of the adaptive pulling strategy when the corner bounding scheme is assumed. In brief, PBRJ<sub>c</sub><sup>\*</sup> is instance-optimal within  $\mathcal{F}_c$  (and becomes optimal for specific classes of instances), and the choice of the specific strategy does not cancel any optimality properties within  $\mathcal{A}$ . Hence, system designers can safely focus on PBRJ<sub>c</sub><sup>\*</sup> for the implementation of a rank join operator based on  $\mathcal{F}_c$ . The second observation is that the corner bound is not always tight. For example, in the proof of Theorem 4.6 (b), the corner bounding scheme would yield a bound of 3 after reading a tuple from each relation, although there is sufficient evidence that no solution exists with a score above 2.5. The problem is that the corner bound for the unseen tuples in  $R_1$ , even though this score combination cannot appear in the other input.

### 5. TIGHT BOUNDING

The discussion at the end of the previous section indicated a need for a bounding scheme that is "tight" in some sense. In this section, we formalize the idea of tightness and show that a tight bound enables PBRJ to be instance-optimal. With this motivation, we develop a new bounding scheme and prove that it is tight. The section concludes with results on the hardness of computing a tight bound.

We define a tight bounding scheme as follows.

DEFINITION 5.1. Let  $I = (R_1, ..., R_n, S, K)$  be a problem instance in a context where only a prefix  $HR_i$  of each  $R_i$  is visible.

• A continuation of I is an instance  $I' = (R'_1, ..., R'_n, S', K)$ that satisfies the following conditions. (C1) The first  $|HR_i|$  tuples in  $R'_i$  and  $R_i$  are identical.

- (C2)  $|R'_i| = |R_i|$  for all *i*.
- (C3) S(x) = S'(x) for each  $x \in X_{i=1}^{n} \mathbf{b}[HR_i] \cup \{1\}^{e_i}$ where  $\mathbf{b}[HR_i]$  is defined as  $\{\mathbf{b}(\tau) \mid \tau \in HR_i\}$ .
- If τ is a join result of some continuation of I, and τ uses at least one unseen tuple, then we say τ is a potential result and S'(τ) is a potential score of I.
- A bounding scheme is tight if whenever | ⋈<sub>i=1</sub><sup>n</sup> HR<sub>i</sub>| ≥ K, updateBound returns a potential score or -∞.

Observe that (C1)-(C3) ensure that no algorithm in  $\mathcal{A}$  can differentiate between I and I' based on the observed contents of the input relations. The cross product in (C3) comes from the assumption that rank join algorithms may evaluate  $\overline{S}$  on any combination of visible tuples. The definition of tightness only constrains the bound when K join results have been found, because the bound does not affect PBRJ before that time. Also note that a tight bound may return  $-\infty$  to cover the case where there are no potential results.

Intuitively, a tight bound allows PBRJ to halt at the first opportunity. The following theorem shows that any tight bounding scheme with the round-robin pulling strategy is enough to ensure instance optimality within the class of deterministic algorithms.

THEOREM 5.1. Let F be an instantiation of PBRJ with the round-robin pulling strategy and a tight bounding scheme. F is instance-optimal within instances  $\mathcal{I}^{n\text{-rel}}$  with optimality ratio n.

**Proof:** Let  $I = (R_1, \ldots, R_n, S, K)$  be an arbitrary rank join problem instance in  $\mathcal{I}^{n \cdot rel}$ . Let  $A \in \mathcal{A}$  be an arbitrary deterministic rank join algorithm. For each *i*, we define  $p_i^F = depth(F, I, i)$ and  $p_i^A = depth(A, I, i)$ . We also define  $p_{\max}^F = \max_i p_i^F$  and  $p_{\max}^A = \max_i p_i^A$ . We want to show that  $p_{\max}^F \leq p_{\max}^A$ . Assume for contradiction that  $p_{\max}^F > p_{\max}^A$ , or in other words that F does not halt after reading  $p_{\max}^A$  complete rounds. At this point, F has seen at least the input tuples seen by A, so it must have buffered the K results returned by A. Since F does not return, we know the bound t is greater than  $\mathcal{S}^{\text{term}} > -\infty$ . Since the bounding scheme of F is tight, there is a potential result  $\tau$  in some continuation I'with a score  $\mathcal{S}(\tau) = t > \mathcal{S}^{\text{term}}$ . When A is executed on I', it will not return  $\tau$  because it must behave the same as it did for I. This is a contradiction because any rank join algorithm must return all tuples that are strictly greater than  $\mathcal{S}^{\text{term}}$ . The contradiction tells us that  $p_{\max}^F \leq p_{\max}^A$ , and it follows that

$$\sum_{i=1}^{n} p_i^F \leq n \cdot p_{\max}^F \leq n \cdot p_{\max}^A \leq n \cdot \sum_{i=1}^{n} p_i^A,$$

indicating the instance optimality of *F*.

This motivates the development of a tight bounding scheme, which we present after describing some necessary notation.

### 5.1 Notation

The analysis in this section requires some additional notation that is useful for manipulating vectors of base scores. As in Definition 5.1, we define  $\mathbf{b}[R] = {\mathbf{b}(\tau) \mid \tau \in R}$ . Whenever x is a vector, we use x[i] to denote the *i*-th coordinate of x and the notation  $x[i \mapsto a]$  represents the result of replacing x[i] with a. We concatenate two vectors x and y by writing x y.

If x, y are *m*-ary vectors of base scores, we say x is *dominated* by y or  $x \leq y$  when  $x[i] \leq y[i]$  for  $1 \leq i \leq m$ . We write  $x \prec y$ to mean that x is dominated by but not equal to y. The dominance relation is used to define the notion of a cover. Formally, given vector-sets X and Y, we say that Y is a *cover* of X, denoted as



Figure 3: Illustration of Example 5.1.

 $X \triangleleft Y$ , if for each  $x \in X$  there exists  $y \in Y$  such that  $x \preceq y$ . We observe that the  $\triangleleft$  relation is preserved by the  $\subseteq$  relation in the following sense: if  $X \subseteq Y \triangleleft C$  then  $X \triangleleft C$ , and if  $X \triangleleft C \subseteq D$ then  $X \triangleleft D$ . Finally, for convenience in our pseudocode, we define  $cut(R, x) = \{x' \in \mathbf{b}[R] \cup \{1\}^m \mid x \preceq x'\}$  which yields a set of vectors that dominate x, using base scores from a relation R or the maximal vector of all ones.

#### 5.2 The Feasible Region Bounding Scheme

We now present the *feasible region* (FR for short) bounding scheme that computes a tight bound on the scores of unseen join results. Before describing the computation of the FR bound, we provide the intuition behind it with the following small example.

EXAMPLE 5.1. Suppose we need the top result from the join of L, R in Figure 3, using the scoring function  $S(b_1, b_2, b_3, b_4) =$  $b_1 + b_2 + b_3 + b_4$ . After observing the shown tuples, one result  $\tau$ is known for a score of  $S(\tau) = 2.9$ . It is clear that no additional tuples need to be read from L, since the last tuple has a score bound less than  $S(\tau)$ . Hence, it may be possible to generate a higher scoring result only by joining an unseen tuple from R with a visible tuple from L. Now, let us attempt to "bound" the base scores of any unseen tuple  $\rho$  in R based on the observed score values (this is the only information about S that is available to an algorithm). We know that  $\rho$  cannot have base scores that dominate the vector (0.6, 0.6) since  $\rho$  would have a score bound at least 3.2 by the monotonicity of S, and this would violate the ordering of R. This observation implies that the base scores of unseen tuples in R are covered by the set  $C = \{ \langle 1.0, 0.6 \rangle, \langle 0.6, 1.0 \rangle \}$ . Hence, a join tuple that results from a visible tuple of L and an unseen tuple of R has a score of at most  $\max\{\mathcal{S}(\mathbf{b}(\lambda) b_R) \mid \lambda \in L \text{ is visible } \land b_R \in C\} =$  $2.9 \leq S(\tau)$ . Overall,  $\tau$  may safely be returned as the top result since neither of the two inputs contain tuples that can generate a higher result score. Note that PBRJ<sup>\*</sup><sub>c</sub> would need to read more from *R* since the bound for *R* is  $3.0 > S(\tau)$ .

#### 5.2.1 **Bound Computation**

Figure 4 shows the pseudocode for our new bounding scheme. We first discuss the global variables  $CR_i$ ,  $G_i$ , and  $g_i$ . Each  $CR_i$ stores a cover of  $\mathbf{b}[R_i - HR_i]$ . Intuitively speaking,  $CR_i$  delineates a region (defined as the space of vectors dominated by at least one member of  $CR_i$ ) that contains the feasible base score vectors from the unseen portion of  $R_i$ . The maintenance of  $CR_i$  requires additional variables  $G_i$  and  $q_i$ , which keep track of tuples from  $R_i$ that have the same score bound. More concretely,  $q_i$  is the most recently seen score bound from  $R_i$ , and  $G_i$  comprises all observed tuples from  $R_i$  with a score bound of  $q_i$ . This effectively divides  $R_i$  into contiguous groups of tuples with equal score bounds. Thus  $G_i$  acts as a buffer for the accessed tuples from the current group and  $q_i$  indicates their score.

The covers  $CR_i$  are maintained by updateBound on lines 1-4 of the pseudocode. The interesting case occurs when a new group

#### **Global Initialization**

1. for i = 1, ..., n do  $CR_i \leftarrow \{1\}^{e_i}; G_i \leftarrow \emptyset; g_i \leftarrow \infty$  end for

**Procedure**  $updateBound(\rho_i)$ 

- **Input:** tuple  $\rho_i$  from relation  $R_i$ 1.
- if  $\overline{\mathcal{S}}(\rho_i) < g_i$  then 2.
- $CR_i \leftarrow updateCR(CR_i, \mathbf{b}[G_i])$
- 3.  $G_i \leftarrow \{\rho_i\}; g_i = \overline{\mathcal{S}}(\rho_i)$ 4. else  $G_i \leftarrow G_i \cup \{\rho_i\}$  end if

**return** max{*resultBound*(W) |  $W \subseteq \{1, ..., n\} \land W \neq \emptyset$ } 5.

**Procedure** updateCR(C, Y)**Input:** a current cover C and a set of new base scores Y; both sets have e-ary base scores for some e

- if  $Y = \emptyset$  then return C end if 1.
- 2.  $y \leftarrow$  some element of Y
- 3.  $S \leftarrow updateCR(C, Y \{y\})$
- $4. \quad S^- \leftarrow \{s \in S \mid y \preceq s\}$
- 5.  $S^+ \leftarrow \bigcup_{i=1}^{e} \{s^-[i \mapsto y[i]] \mid s^- \in S^-\}$ 6. return  $(S S^-) \cup (S^+ \cap (0, 1]^e)$

**Procedure** resultBound(W)**Input:** a nonempty subset W of  $\{1, \ldots, n\}$ ;

for simplicity, assume  $W = \{1, \dots, w\}$ .

- 1. if  $|\bigotimes_{j>w} HR_j| = 0$  OR  $R_j$  is exhausted for some  $j \le w$  then
- 2. return  $-\infty$  end if
- $H \leftarrow \{\mathbf{b}(\eta) \mid \eta \in \bigotimes_{j>w} HR_j\}$ 3.
- 4.  $C \leftarrow \{ X_{j \le w} cut(HR_j, \widehat{c}_j) \mid \forall j \le w(\widehat{c}_j \in CR_j) \}$
- 5.  $t_{\text{cover}} \leftarrow \max\{\min\{\mathcal{S}(ch) \mid c \in X\} \mid X \in C \land h \in H\}$
- 6.  $t_{\text{order}} \leftarrow \min\{g_j \mid j \le w\}$
- 7. return  $\min\{t_{cover}, t_{order}\}$



is found, i.e.,  $\overline{S}(\rho_i) < g_i$ . If  $\gamma_i$  is in the current group  $G_i$  and  $\tau_i$  is an unseen tuple in  $R_i - HR_i$ , then the sorted access of  $R_i$ implies that  $\overline{S}(\tau_i) \leq \overline{S}(\rho_i) < \overline{S}(\gamma_i)$ , and hence it follows that  $\gamma_i \not\prec \tau_i$  by the monotonicity of S. The crucial point, therefore, is that the unseen tuples in  $R_i$  cannot dominate any tuple in  $G_i$ . The call to  $updateCR(CR_i, \mathbf{b}[G_i])$  uses this observation to compute a new cover of  $R_i - HR_i$ . The idea is to iterate over each base score vector y in  $G_i$  and cut the points that dominate y out of the feasible region. To do this, updateCR takes the set of cover points that dominate y (line 4) and projects them along each dimension to match the coordinate of y (line 5). As a final step, updateCR excludes cover points with a zero coordinate (line 6). This restriction is important in our context, as these points may allow the feasible region to contain score vectors that lead to a loose bound. (An example appears in the appendix.)

An example of a call to updateCR is shown in Figure 5. Observe that the cover points (0.4, 1.0) and (0.7, 0.8) in the first graph are projected along both dimensions to yield four new cover points, effectively cutting the feasible region by removing the vectors that dominate (0.3, 0.6). We note that (0.4, 0.6) and (0.3, 0.8) are redundant, as they do not affect the shaded region. Such redundant points can be removed,<sup>1</sup> but we do not consider this heuristic as it does not affect the analysis in the remainder of the paper.

Having discussed the maintenance of  $CR_i$ , we now describe the computation of the bound through the resultBound method. Given a nonempty subset W of  $\{1, \ldots, n\}$ , resultBound(W) returns an upper bound on the score of a hypothetical tuple  $\tau$  that results from joining unseen tuples  $\tau_j \in R_j - HR_j$  for  $j \in W$ , and visible tuples  $\tau_j \in HR_j$  for  $j \notin W$ . There is no such  $\tau$  if  $\bigotimes_{j \notin W} HR_j$  is empty, or if  $R_j$  is exhausted for some  $j \in W$ . For these corner cases we return  $-\infty$ . For all other cases, a bound on  $\mathcal{S}(\tau)$  is computed as the minimum of two correct upper bounds  $t_{order}$  and  $t_{cover}$ . We

<sup>&</sup>lt;sup>1</sup>This can be achieved by computing the skyline of the cover.



Figure 5: Cover update via updateCR. The dots represent the base score vectors, and the white dots signify the current group. The cover points are shown as crosses, and the shaded space that they dominate contains all unseen base scores. The first graph shows the state of the algorithm after observing three tuples, and the second shows the state after calling updateCR with the third tuple as input and observing two more tuples.

describe the computation of  $t_{\text{order}}$  and  $t_{\text{cover}}$  next, assuming for simplicity that  $W = \{1, \ldots, w\}$ .

The  $t_{\text{order}}$  bound is based on the ordering of the input relations, which tell us that, for  $j \leq w$ , the witness  $\tau_j$  of  $\tau$  from the unseen portion of  $R_j$  satisfies  $\overline{S}(\tau_j) \leq g_j$ . It follows that  $S(\tau) \leq g_j$ , so we set  $t_{\text{order}} = \min\{g_j \mid j \leq w\}$ .

To compute  $t_{\text{cover}}$ , we first recall that each unseen witness  $\tau_i$  for  $j \leq w$  is dominated by a member of  $CR_j$ , and thus some combination of cover points in  $X_{j \leq w} CR_j$  must dominate the join of the first w witnesses  $\tau_1 \bowtie \cdots \bowtie \tau_w$ . Given that the join of visible witnesses  $\tau_{w+1} \boxtimes \cdots \boxtimes \tau_n$  must be in  $\bigotimes_{j>w} HR_j$ , it follows that  $\mathcal{S}( au)$  can be bounded by the maximum score over the cross product of  $\sum_{j \le w} CR_j$  and  $\sum_{j \ge w} HR_j$ . We write this "ideal" bound as  $\widehat{t}_{\text{cover}} = \max\{\mathcal{S}(\widehat{c}\mathbf{b}(\eta)) \mid \widehat{c} \in \bigotimes_{j < w} CR_j \land \eta \in \bigotimes_{j > w} HR_j\}.$ We cannot use  $\widehat{t}_{\mathrm{cover}}$  in our algorithm, since algorithms in  $\mathcal A$  are only allowed to know the value of  $\overline{S}$  on combinations of base score vectors that have been read (but see Section 6.1. In other words, S may be evaluated using any vector among  $\mathbf{b}[HR_i] \cup \{1\}^{e_j}$  for the *j*-th input. In order to follow this rule, we want to replace each cover point with one of these admissible vectors that dominates it. To state this formally, assume that  $\hat{c}$  is the concatenation of cover points  $\hat{c}_1 \cdots \hat{c}_w$  where each  $\hat{c}_j \in CR_j$ . Now the admissible base score vectors that dominate  $\hat{c}_j$  are given by  $cut(HR_j, \hat{c}_j)$ , so it follows that any  $c \in \bigvee_{j \leq w} cut(HR_j, \hat{c}_j)$  is a possible replacement for  $\hat{c}$  in the expression for  $\hat{t}_{cover}$ . If we use the minimum over all values of c, we get the expression for  $t_{cover}$  used in the algorithm.

#### 5.2.2 Properties of the Feasible Region Bound

We now prove that the FR bound is correct and tight, meaning that PBRJ is instance-optimal when the FR bound is used with the round-robin pulling strategy. Before going into the details, we state a simple lemma that is useful for both proofs. It establishes an invariant that the  $CR_i$  covers will never contain a vector with a zero coordinate. The proof is trivial and is omitted.

LEMMA 5.1. If  $C \subseteq (0,1]^e$  then  $updateCR(C,Y) \subseteq (0,1]^e$ .

We prove the correctness of the FR bounding scheme in three steps. We first show a necessary property of updateCR, namely, that updateCR(C, Y) covers the points covered by C, given that none of them dominate a point in Y. Second, we show that the unseen tuples from each relation  $R_i$  are covered by  $CR_i$ . These two steps are captured in the following two claims, and we conclude with the correctness of the FR bound in Theorem 5.2.

CLAIM 5.1. Let  $X, Y \subseteq [0,1]^e$  and  $C \subseteq (0,1]^e$ . Suppose that  $X \triangleleft C$  and there do not exist  $x \in X, y \in Y$  such that  $y \preceq x$ . Then  $X \triangleleft updateCR(C, Y)$ .

**Proof:** By induction on the size of Y. If Y is empty, the claim holds since we return the cover C. Now assume that Y is not empty. We want to show that an arbitrary  $x \in X$  is covered by some element of  $(S-S^-) \cup (S^+ \cap (0,1]^e)$ . Clearly,  $X \triangleleft S$  by the inductive hypothesis, so we know that  $x \preceq s$  for some  $s \in S$ . It will suffice to show that if  $s \in S^-$  then x is dominated by some member of  $S^+ \cap (0,1]^e$ . Thus we assume that  $s \in S^-$ , meaning that  $y \preceq s$ . We also know  $y \preceq x$  by our assumptions on X and Y, so there is some j such that x[j] < y[j]. Let  $s^+ = s[j \mapsto y[j]] \in S^+$ . We know that  $s^+ \in (0,1]^e$  because  $s \in S \subseteq (0,1]^e$  by Lemma 5.1 and  $y[j] > x[j] \ge 0$ . Now we have

$$\begin{aligned} x[j] < y[j] = s^+[j] \\ \text{and} \ x[i] \leq s[i] = s^+[i] \text{ for all } i \neq j, \end{aligned}$$

which implies that  $x \leq s^+ \in S^+ \cap (0,1]^e$ , as desired.

CLAIM 5.2. After each call to the updateBound method of the FR bound,  $CR_k \triangleright \mathbf{b}[R_k - HR_k]$  for all k.

*Proof:* Choose any input k. Let  $CR_k^p$  and  $HR_k^p$  denote the values of  $CR_k$  and  $HR_k$  after p calls to updateBound, for  $p \ge 0$ . We show that  $CR_k^p \triangleright \mathbf{b}[R_k - HR_k^p]$  by induction on p. Considering the value of  $CR_k$  before the first iteration, we have  $CR_k^0 = \{1\}^{e_k} \triangleright$  $[0, 1]^{e_k} \supseteq \mathbf{b}[R_k - HR_k^0]$ . For the inductive case, let  $p \ge 1$ . First we observe that

$$CR_k^{p-1} \triangleright \mathbf{b}[R_k - HR_k^{p-1}] \supseteq \mathbf{b}[R_k - HR_k^p].$$

If  $CR_k$  is not updated by calling updateCR on line 2, we are done since  $CR_k^p = CR_k^{p-1}$ . Henceforth, we assume that  $CR_k$  gets updated, meaning that  $\rho_k \in R_k$  is passed to updateBound and  $\overline{S}(\rho_k) < g_k$ . We see that (before  $G_k$  is reset) for any  $\gamma \in G_k$  and unseen tuple  $\rho \in R_k - HR_k^p$  we have

$$\overline{\mathcal{S}}(\gamma) = g_k > \overline{\mathcal{S}}(\rho_k) \ge \overline{\mathcal{S}}(\rho)$$
 by the ordering of  $R_k$ ,

which implies that  $\mathbf{b}[\gamma] \not\preceq \mathbf{b}[\rho]$ . This, together with the fact that  $CR_k^{p-1} \triangleright \mathbf{b}[R_k - HR_k^p]$  allows us to apply Claim 5.1, yielding  $CR_k^p = updateCR(CR_k^{p-1}, \mathbf{b}[G_k]) \triangleright \mathbf{b}[R_k - HR_k^p]$ .

#### THEOREM 5.2. The FR bounding scheme is correct.

*Proof:* Suppose that the FR bound is t after observing part of an instance  $I = (R_1, \ldots, R_n, S, K)$ . Let  $\tau$  be an unseen join result with witnesses  $\tau_i \in R_i$ . At least one witness of  $\tau$  must not be visible, thus we assume without loss of generality that  $\tau_j$  is unseen for each  $j \leq w$  where  $w \geq 1$ .

It will suffice to show that  $S(\tau) \leq t$ . We start by considering the value of  $t_{\text{order}}$  computed by  $resultBound(\{1, \ldots, w\})$ . For each  $j \leq w$ , we know  $\overline{S}(\tau_j) \leq g_j$  by the ordering of  $R_j$ , which yields  $S(\tau) \leq t_{\text{order}}$ . Now we consider  $t_{\text{cover}}$ . For each  $j \leq w$ , choose any  $\hat{c}_j \in CR_j$  that dominates  $b(\tau_j)$ . This is made possible by Claim 5.2. By the monotonicity of S and the fact that each member of  $cut(HR_j, \hat{c}_j)$  dominates  $\hat{c}_j$ , we have

$$\begin{array}{rcl} \mathcal{S}(\tau) &\leq & \min\{\overline{\mathcal{S}}(c\,h) \mid c \in \bigvee_{j \leq w} cut(HR_j, \widehat{c}_j) \\ & \wedge h = \mathbf{b}(\tau_{w+1}) \dots \mathbf{b}(\tau_n) \} \\ &\leq & t_{\text{cover}} \end{array}$$

because  $t_{\text{cover}}$  is computed as a maximum with additional options for  $\hat{c}_1, \ldots, \hat{c}_w, \tau_{w+1}, \ldots, \tau_n$ . Now we have shown  $S(\tau) \leq t_{\text{order}}$ and  $S(\tau) \leq t_{\text{cover}}$ , so  $S(\tau) \leq resultBound(\{1, \ldots, w\}) \leq t$ .

We conclude our analysis of the FR bound with the following theorem that states the tightness of the bound.

#### THEOREM 5.3. The FR bounding scheme is tight.

*Proof:* Suppose that the FR bound returns t during execution on a problem instance  $I = (R_1, \ldots, R_n, S, K)$ . We know that t = resultBound(W) for some  $W \subseteq \{1, \ldots, n\}$ . We assume without loss of generality that  $W = \{1, \ldots, w\}$ . In the call to resultBound(W), let  $h \equiv \mathbf{b}(\eta) \in H, X \in C$ , and  $c \in X$  be the choices that yield the value of  $t_{cover}$ . Also suppose  $c = c_1 c_2 \cdots c_w$ and let  $\hat{c}_j \in CR_j$  such that  $c_j \in cut(HR_j, \hat{c}_j)$  for each  $j \leq w$ .

To check the definition of tightness, it will suffice to assume that I has K visible join results and  $t > -\infty$ , and then construct a continuation  $I' = (R'_1, \ldots, R'_n, S', K)$  that has a potential score of t. Here, we specify conditions that I' must satisfy; the tedious verification that such an instance exists is left to the appendix.

For j > w we define  $R'_j = R_j$ . For  $j \le w$ ,  $R'_j$  is constructed by replacing the first unseen tuple in  $R_j$  with a tuple  $\rho_j$  such that

(T1)  $\rho_1, \ldots, \rho_w$  join with  $\eta$ .

(T2) 
$$\mathbf{b}(\rho_j)[\ell] = \max(A_{j\ell})$$
 for  $j \le w$  and  $1 \le \ell \le e_j$ , where

 $A_{j\ell} = \{\frac{1}{2}(a_j[\ell] + \hat{c}_j[\ell]) \mid a_j \in \mathbf{b}[HR_j] \cup \{0\}^{e_j} \land a_j[\ell] < \hat{c}_j[\ell]\}.$ Condition (T2) intuitively ensures that  $\mathbf{b}(\rho_j)$  is a feasible unseen base score vector, since each member of  $A_{j\ell}$  falls under the cover point  $\hat{c}_j$  on every dimension. It also implies that a member of  $\mathbf{b}[HR_j]$  dominates  $\mathbf{b}(\rho_j)$  only if it dominates  $\hat{c}_j$ . We use  $\tau$  to denote the result of joining  $\rho_1, \ldots, \rho_w$  with  $\eta$ .

Let S' be any monotonic scoring function for  $R_1, \ldots, R_n$  that satisfies the following conditions:

(S1) 
$$\mathcal{S}'(x) = \mathcal{S}(x)$$
 for all  $x \in \bigotimes_{i=1}^{n} \mathbf{b}[HR_i] \cup \{1\}^{e_i}$ .  
(S2)  $\overline{\mathcal{S}}'(\rho_j) = g_j$  for  $j \leq w$ .

 $(S3) \mathcal{S}'(\tau) = resultBound(W).$ 

We see that I' is a valid problem instance, since the inserted tuples have the same score bound as their predecessors, S' is monotonic, and I' has at least the K visible join results of I. Then I' is clearly a continuation of I by construction. This means  $\tau$  is a potential result because the tuples  $\rho_1, \ldots, \rho_w$  used to construct  $\tau$  are not visible. Now  $S'(\tau) = t$  is a potential score of I so we conclude that the FR bound is tight.

### 5.3 Complexity of Tight Bounding

A tight bounding scheme enables PBRJ to stop as soon as possible and is thus crucial in achieving good I/O performance. The flip-side, of course, is the computational overhead of obtaining a tight bound and its impact on the running time of the rank join algorithm. We investigate this issue in what follows, by analyzing the complexity of tight bounding.

We first consider the problem of tight bounding under *data complexity*. In other words, we limit the problem instances to the class  $\mathcal{I}^{n\text{-rel}} \cap \mathcal{I}^{e\text{-dim}}$  where there are *n* relations with at most *e* base scores each. We note that this is a reasonable restriction in practice, as real-world queries are likely to have a bounded size. Under these assumptions, we show that the computation of a tight bound has polynomial complexity.

THEOREM 5.4. A tight bound can be computed in polynomial time for problem instances in  $\mathcal{I}^{n\text{-rel}} \cap \mathcal{I}^{e\text{-dim}}$ .

*Proof:* We consider the FR bound and the running time of the corresponding *updateBound* method. Suppose that  $\leq p$  tuples have been read from any relation. If we look at the *updateCR* method, every point that is added to a cover must be constructed from some combination of base scores from each dimension. There are at most  $p^e$  such combinations, so each cover has size  $O(p^e)$ . Based on this bound, a rudimentary analysis provides an upper bound of  $O(ep^{e+1} + p^{n(e+1)})$  on the running time of *updateBound*. ■

We next consider the problem under *query complexity*. Our analysis uses a formulation of tight bounding as a decision problem. Given an instance I with at least K visible join results and a real number s, the problem is to decide if I has a potential score > s. We refer to this decision problem as RESULTBOUND. In the context of rank join algorithms, the value of s that we are interested in is the score of the K-th best join result found so far. Clearly, a tight bounding scheme can provide a solution to RESULTBOUND. As we show next, however, RESULTBOUND is NP-hard under query complexity, so it follows that we cannot compute a tight bound in polynomial time unless P = NP.

THEOREM 5.5. The RESULTBOUND problem is NP-hard when restricted to rank joins over a particular fixed database.

*Proof:* We consider a database containing two relations T and  $\overline{T}$  with the attribute values and base scores shown below.

T	y	x	b	$\overline{T}$	x	y'	$\overline{b}$
	A	B	1.0		C	A	1.0
	D	D	0.3		D	D	0.3
	E	E	0.2		E	E	0.2
	F	F	0.1		F	F	0.1

We formulate a reduction from 3-SAT. Fix a 3-CNF formula  $\Phi$  over the variables  $v_1, \ldots, v_n$ . We may write  $\Phi$  as

$$\Phi = (L_{11} \lor L_{12} \lor L_{13}) \land (L_{21} \lor L_{22} \lor L_{23}) \land \dots \land (L_{n1} \lor L_{n2} \lor L_{n3})$$

where  $L_{k1}, L_{k2}, L_{k3}$  are distinct literals  $v_i$  or  $\overline{v}_i$  for each k.

In what follows, we define an instance  $I = (R_1, \ldots, R_{2n}, S, 1)$ based on the formula  $\Phi$ . For  $i \leq n$ , let  $R_{2i-1}$  be the result of renaming  $yxb \rightarrow y_{i-1}x_ib_i$  in T, and  $R_{2i}$  be the result of renaming  $xy'\overline{b} \rightarrow x_iy_i\overline{b}_i$  in  $\overline{T}$ . We also use mnemonic names  $T_i \equiv R_{2i-1}$ and  $\overline{T}_i \equiv R_{2i}$  when convenient.

For any base score vector  $b = \langle b_1, \overline{b}_1, b_2, \overline{b}_2, \dots, b_n, \overline{b}_n \rangle$ , define

$$f(b) = \min_{1 \le k \le n} \{ \max\{h(L_{k1}), h(L_{k2}), h(L_{k3})\} \}$$

where  $h(v_i) = b_i$  and  $h(\overline{v}_i) = \overline{b}_i$ . The idea is that f(b) indicates whether  $\Phi$  is satisfied if we interpret  $\langle b_i, \overline{b}_i \rangle = \langle 1, 0 \rangle$  as an assignment of  $v_i$  to true and  $\langle b_i, \overline{b}_i \rangle = \langle 0, 1 \rangle$  as an assignment of  $v_i$  to false. However, f is not this simple in reality since other assignments of base scores are possible. We also define

$$g(b) = \min\{b_1, \overline{b}_1, \dots, b_n, \overline{b}_n\}$$
  
and  $\mathcal{S}(b) = f(b) + g(b).$ 

Now we use I and s = 0.6 for the RESULTBOUND problem, with only the first three tuples visible in each relation. Note that this case is interesting since 0.6 is the score of the best known join result. We need to show that  $\Phi$  is satisfiable if and only if I has a potential score > 0.6.

First assume that  $\Phi$  has a satisfying assignment. We define an instance  $I' = (R'_1, \ldots, R'_{2n}, \mathcal{S}, 1)$  where each  $R'_i$  is the same as  $R_i$  except for the following modifications. If  $v_i$  is assigned to true, we change the join attributes in the last tuple of  $\overline{T}_i$  to  $x_i = B$  and  $y_i = A$ . Otherwise, we change the join attributes in the last tuple of

 $T_i$  to  $y_{i-1} = A$  and  $x_i = C$ . In the first case, the modified tuple in  $\overline{T}_i$  joins with the top tuple in  $T_i$ , and in the second case, the reverse holds. Each of these pairs of joining tuples join with each other on the value A to get a result tuple  $\tau$ . Since I' is clearly a continuation of I and  $\tau$  only occurs in the join results of I', we know that  $\tau$  is a potential result of I. The key property of  $\tau$  is that it involves the top tuple of  $T_i$  when  $v_i$  is true, or the top tuple from  $\overline{T}_i$  when  $v_i$  is false. When we compute  $f(\mathbf{b}(\tau))$ , we know for each k, the assignment will make one of  $L_{k1}, L_{k2}, L_{k3}$  true, and this means  $\max\{h(L_{k1}), h(L_{k2}), h(L_{k3})\} = 1$  and hence  $f(\mathbf{b}(\tau)) = 1$ . We also see that  $g(\mathbf{b}(\tau)) = 0.1$ , so  $S(\tau) = 1.1 > s$  as desired.

Now assume there is a potential result  $\tau$  from a continuation  $I' = (R'_1, \ldots, R'_{2n}, S', 1)$  such that  $S'(\tau) > 0.6$ . Let  $\tau_1, \overline{\tau}_1, \ldots, \tau_n, \overline{\tau}_n$  denote the witnesses of  $\tau$  and let  $b_1, \overline{b}_1, \ldots, b_n, \overline{b}_n$  denote their base scores. The relations  $R'_i$  can only differ from  $R_i$  on the fourth row, and the unseen base score must be less than 0.3 (after reading three rows of  $R'_i$ , the FR bound would have a cover of 0.3 and the current group would contain 0.2). We also know that, for each i, it is not possible for both  $\tau_i$  and  $\overline{\tau}_i$  to be the first tuples in their respective relations, since these tuples do not join. From these observations, either  $b_i \leq 0.3$  or  $\overline{b}_i \leq 0.3$ . In other words,  $\langle b_i, \overline{b}_i \rangle \leq \langle c_i, \overline{c}_i \rangle$  for some  $\langle c_i, \overline{c}_i \rangle \in \{\langle 1.0, 0.3 \rangle, \langle 0.3, 1.0 \rangle\}$ . If we define  $c = \langle c_1, \overline{c}_1, \ldots, c_n, \overline{c}_n \rangle$  then we have  $\mathbf{b}(\tau) \leq c$ . Moreover, the base scores in c are visible in I, so (C3) leads to

$$\mathcal{S}(c) = \mathcal{S}'(c) \ge \mathcal{S}'(\tau) > 0.6.$$

This implies f(c) > 0.3 since  $g(c) \le 0.3$ . Now suppose we assign  $v_i$  to true if and only if  $c_i = 1$ . Since f(c) > 0.3 we know that  $\max\{h(L_{k1}), h(L_{k2}), h(L_{k3})\} > 0.3$  for each k, and thus one of  $h(L_{k1}), h(L_{k2}), h(L_{k3})$ , say  $h(L_{k1}) > 0.3$ . This can only happen if either  $\langle c_i, \overline{c_i} \rangle = \langle 1.0, 0.3 \rangle$  and  $L_{k1} = v_i$ , or  $\langle c_i, \overline{c_i} \rangle = \langle 0.3, 1.0 \rangle$  and  $L_{k1} = \overline{v_i}$  for some i. In both cases,  $L_{k1}$  is satisfied, which implies that  $\Phi$  is satisfied by this assignment and we are done.

If we look closely at the previous proof, we see that the constructed instance is relatively simple, since each relation has only one base score. It follows that RESULTBOUND is NP-hard for a fixed database, even when restricted to instances in  $\mathcal{I}^{1\text{-dim}}$ . It is interesting to ask whether the theorem can be also be proved when restricted to instances in  $\mathcal{I}^{n\text{-rel}}$  for some *n*. We leave this as an open problem.

### 6. EXTENSIONS

#### 6.1 Generalizing the Problem

In our statement of the rank join problem, we made several assumptions to focus the presentation. We now consider alternative assumptions to generalize our results.

**Cost metrics.** The *sumDepths* metric that we have considered thus far is appropriate if the input is accessed one tuple at a time, with an equal cost for all relations. We can consider a more general case where a problem instance  $I = (R_1, \ldots, R_n, S, K)$  allows access to  $R_i$  in blocks of  $z_i$  tuples and each block access has a cost of  $c_i$ . The cost of a rank join algorithm A in this case is

$$blockCost(A, I) = \sum_{i=1}^{n} c_i \cdot \left[ \frac{depth(A, I, i)}{z_i} \right]$$

The optimality of PBRJ<sup>\*</sup><sub>c</sub>, stated by Theorem 4.4, immediately extends to the *blockCost* metric by Lemma 4.1. Also observe that each of our proofs of instance optimality for some algorithm A essentially showed that  $sumDepths(A, I) \leq n \cdot sumDepths(B, I)$  for any algorithm B under consideration and instance  $I \in \mathcal{I}^{n\text{-rel}}$ .

When this is the case, it is routine to show that

$$blockCost(A, I) \leq \frac{n \cdot \max(c_i/z_i)}{\min(c_i/z_i)} \cdot blockCost(B, I) + \sum_{i=1}^{n} c_i$$

and therefore the optimality ratio of n under the sumDepths metric translates to  $n \cdot \max(c_i/z_i) / \min(c_i/z_i)$  under blockCost.

**Join condition.** The results in this paper are presented for natural joins, but we may consider more general join conditions. Following the approach used by the J\* algorithm [8], we allow for a join condition that is accessed with a method valid(). Given tuples  $\tau_1, \ldots, \tau_k$  from a subset of the input relations,  $valid(\tau_1, \ldots, \tau_k)$  returns true if and only if  $\tau_1, \ldots, \tau_k$  may hypothetically participate in a join result with some selection of tuples from the other input relations. We can show that our analysis extends to this more general model. The only significant changes that we need to make are as follows: (a) the join condition  $\theta$  becomes part of the problem instance, and (b) a continuation can employ any join condition  $\theta'$  that gives the same answers as  $\theta$  on the visible portion of the input.

**Maximum base score.** Our definition of the score bound  $\overline{S}$  is based on the implicit assumption that each base score takes a maximum value of 1. It is important to note that none of our proofs use this assumption. Since we do not allow for a priori knowledge of the maximum base score, our results are based on the premise that any base score may take the value 1 until the observed input tuples prove otherwise.

There may be cases in practice where the maximum base scores are known in advance. If this is the case, it is easy to normalize the base scores to have maximum values of 1, then use a modified scoring function that undoes the normalization before passing the base scores to S. A rank join algorithm does not need to be changed to benefit from this normalization, since it simply enforces the implicit assumption on the range of base scores.

**Problems with no solution.** In our definition of a top-K rank join, we require that at least K join results exist. Although it is possible to lift this restriction and require a rank join algorithm to return all the join results if there are fewer than K in total, these cases would not be of much interest in our depth-based analysis since any algorithm would generally read all of its input to compute the join.

Access to the scoring function. One of the restrictions in our study is that algorithms may only evaluate the score bound  $\overline{S}$  on combinations of input score vectors. This is necessary to allow a tight bound to be computed: the intuitive reason is that a bound must guarantee correctness based on what is known about the scoring function, and arbitrary knowledge of the scoring function can be learned by extra evaluations of S.

On the other hand, a practical implementation of a rank join algorithm may evaluate  $\overline{S}$  on some base scores that do not occur in the input. In particular, an implementation of the *resultBound* method of the skyline bound could replace  $t_{cover}$  with the alternate  $\hat{t}_{cover}$  bound that is described in Section 5.2.1. This change can only help in practice, since  $\hat{t}_{cover}$  is simpler to compute and never larger than  $t_{cover}$ .

### 6.2 Ranking Query Plans

Earlier work on ranking queries has developed *incremental* operators [4, 8] that produce ranked join results one at a time through an iterator interface. The PBRJ template can be made incremental by buffering all join results, and returning the top buffered result when its score is at least as large as the bound t. The practical advantage of incremental operators is that the number of required results K does not need to be known in advance, allowing for pipelined execution plans involving a tree of rank join operators. More recent work has developed optimization algorithms [5, 6] to select an efficient execution plan involving various physical operators.

Our analysis has interesting implications for the selection of ranking query plans. Observe that we can view a tree-structured plan holistically as an algorithm that solves instances of the rank join problem with a particular number of input relations. Hence we may define  $\mathcal{T} \subseteq \mathcal{A}$  as the class of rank join algorithms that solve input instances using an appropriate query plan. Then it follows from our analysis in Section 5 that a member of  $\mathcal{T}$  cannot improve the cost of solving an instance in  $\mathcal{I}^{n\text{-rel}}$  more than a factor of *n* over a single PBRJ instance. In other words, a multi-level plan can only provide a limited advantage over a single operator. In addition, multi-level plans do not have performance guarantees when viewed as a single algorithm.

We can reach stronger conclusions if we only consider plans where each operator is an instantiation of PBRJ with the corner bound. Let  $\mathcal{T}_c$  denote the algorithms in  $\mathcal{T}$  that use these plans. The following lemma shows that if  $T \in \mathcal{T}_c$ , then there is a single PBRJ operator using the corner bound that outperforms the tree T on all instances.

LEMMA 6.1. For any  $T \in \mathcal{T}_{c}$  there exists  $F_{T} \in \mathcal{F}_{c}$  such that  $sumDepths(F_{T}, I) \leq sumDepths(T, I)$  for all  $I \in \mathcal{I}^{n\text{-rel}}$ .

**Proof sketch:** Let  $F_T$  be an instantiation of PBRJ with the corner bound and a pull strategy that starts by accessing the input relations in the same order as T, and continues in round-robin order if more tuples are needed. It is possible to show that the last tuple emitted by each node of T has a score bound of at most  $S^{\text{term}}$ , by induction on the distance of the operator from the root of the plan. The lemma then follows by directly checking the corner bound's termination conditions after pulling all the tuples seen by T.

This lemma allows us to extend our results of Section 4 to  $\mathcal{T}_c$ . For example, we see that  $\text{PBRJ}_c^*$  is optimal within algorithms  $\mathcal{T}_c$ and instances  $\mathcal{I} - (\mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}})$ , because otherwise there would be an algorithm  $T \in \mathcal{T}_c$  outperforming  $\text{PBRJ}_c^*$  on some instance  $I \notin \mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}}$ . Then Lemma 6.1 implies that  $F_T \in \mathcal{F}_c$  also outperforms  $\text{PBRJ}_c^*$  on I, which contradicts Theorem 4.4. We can similarly show that  $\text{PBRJ}_c^{\text{RR}}$  and  $\text{PBRJ}_c^*$  are instance-optimal in  $\mathcal{T}_c$ . Recalling that  $\text{PBRJ}_c^*$  is equivalent to the HRJN\* algorithm, these observations suggest that existing optimizers that consider plans of pipelined HRJN\* operators are most likely wasting work: the pipelined operators cannot perform better than a single HRJN\* operator unless the input has both intra- and inter-input threshold ties, and even in those cases, the performance difference is at most a constant factor.

### 7. RELATED WORK

In this section, we briefly review the previous studies of ranking queries and their relevance to our main results (see Figure 6 for a summary). Early studies in relational ranking queries focused on top-K selection queries, which involve selection over a single table with several base scores. Each base score is assumed to have an index that provides either *sorted* access to the scores of all tuples or *random* access to the score of a given tuple, or possibly both. This can be considered a special case of a rank join by viewing the indexes as separate relations that are joined on a common primary key, however it differs from our formulation of the problem because we do not allow random access. The top-K selection algorithms of Fagin et al. [2, 3] have been the most influential. Their strategy is

#### Optimality restricted to algorithms $\mathcal{F}_{\mathsf{c}}$ (or $\mathcal{T}_{\mathsf{c}})$

- PBRJ<sup>\*</sup><sub>c</sub> never costs more than PBRJ<sup>RR</sup><sub>c</sub>
- For  $\mathcal{I}^{n-\mathrm{rel}}$ : PBRJ<sup>\*</sup><sub>c</sub> and PBRJ<sup>RR</sup> are instance-optimal
- For  $\mathcal{I} (\mathcal{I}^{intra} \cap \mathcal{I}^{inter})$ : PBRJ<sup>\*</sup><sub>c</sub> is optimal

### Optimality among all algorithms ${\mathcal A} \ (\text{or} \ {\mathcal F})$

- For  $\mathcal{I}^{2\text{-rel}} \cap \mathcal{I}^{1\text{-dim}}$ : PBRJ<sup>\*</sup><sub>c</sub> and PBRJ<sup>RR</sup><sub>c</sub> are instance-optimal
- For more general problems: no algorithm in  $\mathcal{F}_{c}$  is instance-optimal
- PBRJ is instance optimal with round-robin access and a tight bound

### Investigation of Tight Bounding

- The feasible region bounding scheme is tight
- For  $\mathcal{I}^{n\text{-}\mathrm{rel}} \cap \mathcal{I}^{e\text{-}\mathrm{dim}}$ : the feasible region bound takes polynomial time
- Under query complexity: RESULTBOUND is NP-hard

#### Figure 6: Summary of Main Results

fundamentally similar to PBRJ since they read their input in sorted order, and use bounds on unseen scores to terminate.

The more recent work in top-K join queries is the most relevant to our study. The various proposals all follow Fagin's strategy of reading the input streams in order of base score. The first rank join algorithm, named J\* [8], is based on an A\* optimization strategy. The rank join algorithm proposed by Ilvas et al. [4] is equivalent to our PBRJ template when the corner bounding scheme is used, and the basic HRJN and HRJN\* implementations correspond to  $PBRJ_c^{\mbox{\tiny RR}}$  and  $PBRJ_c^*.$  Their experiments showed that  $HRJN^*$  can run faster than J\*, while performing roughly the same amount of I/O. Another general-purpose rank join algorithm was presented in the recent work of Agrawal and Widom [1] in the context of uncertain databases. The main novelty of their algorithm is that it operates with limited memory. Finally, the LARA-J algorithm [7] is an algorithm that uses ideas similar to J\*: both algorithms materialize an exponential number of partial join results to compute a tight bound for problem instances in  $\mathcal{I}^{1\text{-dim}}$ , which is consistent with our hardness result for RESULTBOUND.

These rank join algorithms were shown to be instance-optimal under various assumptions. In particular, the proofs assume that there is only one base score per input relation, but our data model follows recent work in query optimization [6, 9] that assumes several base scores per relation. The proof of instance optimality presented by Ilyas et al. [4] only works for multi-way joins when the join condition is a "black box," which is not usually the case, since natural joins are common in practice. Agrawal and Widom's proof of instance optimality [1] only covers the case of binary joins.

Our analysis is related to the theoretical study that was done in our development of the DEEP framework [9] for estimating the depths of operators in ranking query plans. This work provides an alternate proof of Theorem 4.3 that does not use the fact that the cost of  $PBRJ_c^{r}$  is never more than  $PBRJ_c^{RR}$ . The only other relevant theorem in this work is essentially a weaker version of Theorem 4.4 since it prohibits ties on values other than the threshold.

### 8. CONCLUSIONS

This paper was motivated by a desire to develop a complete and detailed characterization of possible solutions to the rank join problem and their relative theoretical performance. We have accomplished this goal for a formulation of the problem that is more general than prior studies of rank join algorithms. Some of our results have very practical applications, such as the fact that a single HRJN\* operator is superior to an HRJN\* pipeline in nearly all cases. We also make a significant algorithmic contribution with the introduction of the feasible region bounding scheme, which enables the PBRJ algorithm to have instance-optimal I/O cost.

It would be interesting to consider special cases that were not

treated in our analysis, such as particular scoring functions and cases where data statistics and integrity constraints are known. In another direction, our work could be generalized further to employ indexes for random access on join attributes.

Acknowledgements. We recognize Joshua Spiegel of BEA Systems, who contributed to our early discussions about the material in this paper. We also wish to thank Tova Milo and Wang-Chiew Tan for their feedback on previous drafts of this paper.

### 9. **REFERENCES**

- Parag Agrawal and Jennifer Widom. Confidence-aware joins in large uncertain databases. Technical report, Stanford University, 2007. Available at http://dbpubs.stanford.edu/pub/2007-14.
- [2] Ronald Fagin. Combining fuzzy information from multiple systems. Journal of Computer and System Sciences, 58(1):83–99, 1999.
- [3] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.
- [4] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13(3):207–221, 2004.
- [5] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Hicham G. Elmongui, Rahul Shah, and Jeffrey Scott Vitter. Adaptive rank-aware query optimization in relational databases. ACM Transaction on Database Systems, 31(4):1257–1304, 2006.
- [6] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In ACM SIGMOD International Conference on Management of Data, pages 131–142, 2005.
- [7] Nikos Mamoulis, Man Lung Yiu, Kit Hung Cheng, and David W. Cheung. Efficient top-k aggregation of ranked inputs. ACM Transaction on Database Systems, 32(3):19, 2007.
- [8] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *International Conference on Very Large Databases*, pages 281–290, 2001.
- [9] Karl Schnaitter, Joshua Spiegel, and Neoklis Polyzotis. Depth estimation for ranking query optimization. In *International Conference on Very Large Databases*, pages 902–913, 2007.

### APPENDIX A. TIGHTNESS OF THE FR BOUND

This appendix completes the proof of tightness of the feasible region bound. Our first step shows that a partial monotonic scoring function can be extended to a total monotonic scoring function as long as it contains a mapping for the maximal score vector.

CLAIM A.1. Suppose  $\{1\}^k \subseteq A \subseteq [0,1]^k$  and let  $f : A \to \mathbb{R}$ be monotonic. There exists a monotonic function  $f' : [0,1]^k \to \mathbb{R}$ such that f'(a) = f(a) for all  $a \in A$ .

*Proof:* For any  $b \in [0, 1]^k$  define  $S(b) = \{a \in A \mid b \leq a\}$ . We know that S(b) is nonempty since  $A \supseteq \{1\}^k \triangleright [0, 1]^k$ . This allows us to define  $f'(b) = \inf_{s \in S(b)} f(s)$ . To show that f' is monotonic, consider any  $x, y \in [0, 1]^k$  such that  $x \leq y$ . Clearly,  $S(y) \subseteq S(x)$  by the definition of S, so we have

$$f'(x) = \inf_{s \in S(x)} f(s) \le \inf_{s \in S(y)} f(s) = f'(y)$$

thus proving the monotonicity of f'.

We say  $x \prec y$  or x is *strictly dominated by y* to mean x[i] < y[i] for each dimension *i*. The next claim intuitively states that the cover points are pushed towards the origin far enough to guarantee that they only strictly dominate scores in the current group.

CLAIM A.2. After each call to updateBound, if  $\hat{c}_i \in CR_i$  and  $\tau_i \in HR_i$  such that  $\mathbf{b}(\tau_i) \ll \hat{c}_i$ , then  $\tau_i \in G_i$ .

*Proof:* Define the notation  $A \nearrow B$  (pronounced A is below B) to mean that  $a \not\cong b$  for all  $a \in A$  and  $b \in B$ . One way to visualize this relation in two dimensions is to imagine a staircase-shaped border dividing  $\mathbb{R}^2$  with A on the lower left and B on the upper right, while allowing any points on the border. Using this notation, we can restate the claim as  $CR_i \nearrow HR_i - G_i$ . It will suffice to show that if  $C \nearrow X$  then  $updateCR(C, Y) \nearrow X \cup Y$ . Then the invariant stated in the claim is clearly preserved by lines 2-3 of updateBound, which is the only time  $HR_i - G_i$  and  $CR_i$  are modified.

The proof is by induction on the size of Y. If Y is empty, we are done since updateCR(C, Y) = C. Now suppose Y is not empty, so we have the inductive hypothesis  $S \nearrow X \cup (Y - \{y\})$ . We need to show that  $(S - S^-) \cup (S^+ \cap (0, 1]^e) \nearrow X \cup Y$ . To show this, we assume that  $s \in (S - S^-) \cup (S^+ \cap (0, 1]^e)$  and  $z \in X \cup Y$ , then we check that  $z \not\prec s$ .

First suppose that  $s \in S - S^-$ . If  $z \neq y$ , then  $z \not\prec s$  by induction. We also have  $y \not\prec s$  because  $s \in S - S^-$  and thus  $y \not\prec s$  by the definition of  $S^-$ .

Now suppose that  $s \in S^+ \cap (0, 1]^e$ . We know  $s = s^-[i \mapsto y[i]]$ for some  $s^- \in S^-$ ,  $i \le e$  by the definition of  $S^+$ . This also means that  $s \preceq s^-$  since  $y[i] \le s^-[i]$  by the definition of  $S^-$ . Now if  $z \ne y$ , then  $z \not\prec s^-$  by induction, which implies that  $z \not\prec s$  since  $s \preceq s^-$ . We also have  $y \not\prec s$  since s[i] = y[i].

Now we finally prove that the construction in Theorem 5.3 is sound.

CLAIM A.3. The instance I' in the proof of Theorem 5.3 exists.

*Proof:* We need to check that  $R'_1, \ldots, R'_w$  and S' exist. As long as  $\rho_j$  exists for each  $j \leq w$ , our construction of  $R'_j$  that replaces the first unseen tuple in  $R_j$  is valid because  $R_j$  is not exhausted. Otherwise, we would get  $t = resultBound(W) = -\infty$ , violating the assumptions of the proof. Now to show  $\rho_j$  exists, first note that it is easy to satisfy (T1) by choosing join attributes that match. To address (T2), we need to show that each  $A_{j\ell}$  is nonempty. This follows because  $\hat{c}_j[\ell] > 0$  by Lemma 5.1, implying that  $0 \in A_{j\ell}$ . Hence, the tuples  $\rho_j$  and  $\tau$  are well-defined. We also note that it is clear from the definition of  $A_{j\ell}$  that  $\mathbf{b}(\rho_j)[\ell] < \hat{c}_j[\ell]$ , which implies that  $\mathbf{b}(\rho_j) \prec \hat{c}_j$ . We now show that S' exists. Suppose that u, v are vectors whose scores under S' are specified by (S1)-(S3). In other words, u and v may be members of the cross product of (S1), equal to  $\mathbf{b}(\tau)$ , or the result of filling in 1 for the missing base scores in  $\mathbf{b}(\rho_j)$  for some  $j \leq w$ . We write the scores specified for u and v as S'(u)and S'(v), although this is an abuse of notation since we have not shown that S' exists. We need to show two things: first, that it is possible to satisfy (S1)-(S3) (meaning that they don't conflict), and second, that this is possible when S' is monotonic. Our strategy is to check the monotonicity condition  $u \leq v \Rightarrow S'(u) \leq S'(v)$  for all u, v. Since u, v are chosen symmetrically, this entails  $v \leq u \Rightarrow$  $S'(v) \leq S'(u)$ . Then  $u = v \Rightarrow S'(u) = S'(v)$ , so (S1)-(S3) do not make conflicting assignments, and S' exists by Claim A.1.

We know (S1) is compatible with monotonicity because S is monotonic. It is also trivial to show that (S2) and (S3) respect monotonicity when considered in isolation. Now we check that each pair of (S1), (S2), (S3) are compatible.

(S1,S2): Let  $x \in \sum_{i=1}^{n} \mathbf{b}[HR_i] \cup \{1\}^{e_i}$  and let  $x_i$  denote the portion of x from input i. Take any  $j \leq w$  and let r denote the result of filling in 1 for the missing base scores in  $\mathbf{b}(\rho_j)$ . If  $r \leq x$ , then every coordinate of x is 1, except possibly those corresponding to  $x_j$ . Then  $S(x) = \overline{S}(x_j)$  by the definition of  $\overline{S}$ , and we have

$$\mathcal{S}'(x) = \mathcal{S}(x) = \overline{\mathcal{S}}(x_j) \ge g_j = \mathcal{S}'(r)$$

For the opposite case, assume that  $x \leq r$ . In particular, this means that  $x_j \leq \mathbf{b}(\rho_j)$ . Since  $\mathbf{b}(\rho_j) \ll \hat{c}_j$  by construction, we have  $x_j \ll \hat{c}_j$  by transitivity. We also know that  $x_j = \mathbf{b}(\gamma_j)$  for some  $\gamma_i \in HR_j$  and then  $\gamma_j \in G_j$  by Claim A.2. We conclude that

$$\mathcal{S}'(x) = \mathcal{S}(x) \le \overline{\mathcal{S}}(x_j) = \overline{\mathcal{S}}(\gamma_j) = g_j = \mathcal{S}'(r)$$

(S2,S3): Note that  $\tau$  is constructed from  $\rho_j$ , so if r is the result of filling in 1 for the missing base scores in  $\rho_j$ , then  $\mathbf{b}(\tau) \leq r$ . Thus we only need to check that  $\mathcal{S}'(\tau) \leq \mathcal{S}'(r)$ . Indeed,

$$\mathcal{S}'(\tau) = t \le t_{\text{order}} = \min\{g_1, \dots, g_w\} \le g_j = \mathcal{S}'(r).$$

(S1,S3): Before addressing this case, we claim that

$$\mathbf{b}(\rho_j) \preceq x_j \Rightarrow \widehat{c}_j \preceq x_j \text{ for all } j \leq w \text{ and } x_j \in \mathbf{b}[HR_j]$$

We prove the contrapositive. Let  $x_j \in \mathbf{b}[HR_j]$  such that  $\hat{c}_j \not\preceq x_j$ . This means  $x_j[\ell] < \hat{c}_j[\ell]$  for some  $\ell$ . Applying (T2), we have  $\mathbf{b}(\rho_j) \geq \frac{1}{2}(x_j[\ell] + \hat{c}_j[\ell]) > x_j[\ell]$ , which means that  $\mathbf{b}(\rho_j) \not\preceq x_j$  as desired.

Let  $x \in \bigotimes_{i=1}^{n} \mathbf{b}[HR_i] \cup \{1\}^{e_i}$  and let  $x_i$  denote the portion of x from input i. First suppose that  $\mathbf{b}(\tau) \preceq x$ . In particular, this means that  $\mathbf{b}(\rho_j) \preceq x_j$  for all  $j \leq w$ , and the observation above implies that  $\hat{c}_j \preceq x_j$ . It follows by the definition of X that  $x_1 \dots x_w \in X$ . Since  $c \in X$  was chosen to minimize S(ch), we have  $S(ch) \leq S(x_1 \dots x_w h)$ . It is also clear that  $h \preceq x_{w+1} \dots x_n$  since  $\mathbf{b}(\tau) \preceq x$ . Combining this with the previous observation yields  $S(ch) \leq S(x)$ . We conclude that

$$\mathcal{S}'(\tau) = t \le t_{\text{cover}} = \mathcal{S}(ch) \le \mathcal{S}(x) = \mathcal{S}'(x).$$

For the opposite case, suppose  $x \leq \mathbf{b}(\tau)$ . We want to check that  $t_{\text{order}}$  and  $t_{\text{cover}}$  are both upper bounds for  $\mathcal{S}(x)$ . Using the same reasoning as the (S1,S2) case, for  $j \leq w$ , we have  $x_j = \mathbf{b}(\gamma_j)$  for some  $\gamma_j \in G_j$ . This says that x is derived from tuples with score bounds  $g_1, \ldots, g_w$ , so  $\mathcal{S}(x) \leq \min\{g_1, \ldots, g_w\} = t_{\text{order}}$ . Now in order to check  $t_{\text{cover}}$  we first observe that  $\mathbf{b}(\tau) \leq ch$  since  $\mathbf{b}(\rho_j) \ll \hat{c}_j \leq c_j$  for each  $j \leq w$ . This yields  $x \leq ch$  by transitivity, and so  $\mathcal{S}(x) \leq \mathcal{S}(ch) = t_{\text{cover}}$ . We conclude that

$$\mathcal{S}'(x) = \mathcal{S}(x) \le \min\{t_{\text{order}}, t_{\text{cover}}\} = t = \mathcal{S}'(\tau).$$

### B. EXCLUSION OF COVER POINTS WITH A ZERO COORDINATE

As discussed in Section 5.2.1, method updateCR excludes cover points with a zero coordinate. We now show that this restriction is necessary in order to achieve a tight bound.

We consider the instance  $(R_1, R_2, S, 1)$ , where  $R_1, R_2$ , and S are defined as shown below:

$R_1$	a	$b_1$	$b_2$	$R_2$	а	$b_3$	$b_4$
	x	1.0	0		y	1.0	1.0
	z	0	1.0		x	1.0	0.9
	z	0.2	0.2		y	0	0.8
	z	0.1	0.1		y	0.3	0.3

$$\mathcal{S}(b_1, b_2, b_3, b_4) = \begin{cases} 10 + b_3 + b_4 & \text{if } b_1 = b_2 = 1\\ b_1 + b_2 + b_3 + b_4 & \text{otherwise} \end{cases}$$

The visible portion of the two inputs yields one join result  $\tau$ with score  $S(\tau) = 2.9$ . First, we observe that, if  $\{1\} \subseteq W$ , then  $resultBound(W) \leq g_1 = 2.2$  since resultBound(W) is no larger than  $t_{order}$  and the latter is defined as  $\min\{g_j \mid j \in W\}$ . Next, we consider the computation of resultBound(W) when W = $\{2\}$ . We observe that

$$CR_2 = \{ \langle 1, 0.8 \rangle \},\$$

and

$$cut(HR_2, \langle 1, 0.8 \rangle) = \{ \langle 1, 1 \rangle, \langle 1, 0.9 \rangle \}$$

It follows that

$$t_{\text{cover}} = \max\{\min\{\mathcal{S}(h\,c) \mid c \in cut(HR_2, \langle 1, 0.8 \rangle)\} \mid h \in \mathbf{b}[HR_1]\}$$
  
= 2.9

At the same time, we know that  $t_{order} = g_2 = 10.6$ . Hence,  $resultBound(\{2\}) = \min(t_{order}, t_{cover}) = 2.9$ . Overall, the previous observations imply that the FR bound is equal to 2.9, since it is the maximum of resultBound(W) on all possible choices of W.

Now, we consider the computation of the FR bound if we alter updateCR to include cover points with a zero coordinate. We refer to this version as the FR<sup>0</sup> bound. It is straightforward to verify again that  $resultBound(W) \leq 2.2$  for  $\{1\} \subseteq W$ . For  $W = \{2\}$ , we first observe that  $CR_2$  is computed as follows:

$$CR_2 = \{ \langle 1, 0.8 \rangle, \langle 0, 1 \rangle, \langle 0, 0.9 \rangle \}$$

Recall that  $t_{\text{cover}}$  is computed as the maximum over cover points in  $CR_2$ . Hence, we can state the following:

$$t_{\text{cover}} \geq \max\{\min\{\mathcal{S}(h\,c) \mid c \in cut(HR_2, \langle 0, 1 \rangle)\} \mid h \in \mathbf{b}[HR_1]\}$$
  
= 3

Given that  $g_2 = 10.6$  and  $resultBound(\{2\}) = \min\{t_{order}, t_{cover}\}$ , it follows that  $resultBound(\{2\}) \ge 3$ . In turn, the FR<sup>0</sup> bound is at least 3, making it larger than the provably tight FR bound (=2.9).