Evaluating Rank Joins with Optimal Cost

Karl Schnaitter
UC Santa Cruz
Santa Cruz, CA, USA
karlsch@soe.ucsc.edu

Neoklis Polyzotis
UC Santa Cruz
Santa Cruz, CA, USA
alkis@cs.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. It is often the case in practice that the inputs may be accessed in ranked order and the scoring function is monotonic. These conditions allow for efficient algorithms that solve the rank join problem without reading all of the input. 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 optimal.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—query processing

General Terms

Algorithms, Performance, Theory

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.

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.

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, 10] that need to join relations with multiple score attributes. Overall, previous studies provide little insight into the comparative performance 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 to 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

PODS'08, June 9–12, 2008, Vancouver, BC, Canada.

algorithms can be arbitrarily bad in general because they use a loose upper bound on the scores of remaining join results. We show that it is possible to do better 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 $e_i \geq 0$, and signify the importance of the tuple according to criteria specified by the query. Base scores are aggregated using a *scoring function* $\mathcal{S}$ that computes the score of a join result τ as $\mathcal{S}(\mathbf{b}(\tau))$. We may also use $\mathcal{S}(\tau)$ as a shorthand for the score of τ. Following common practice, we assume that $\mathcal{S}$ is monotonic, i.e., $\mathcal{S}(x_1, \ldots, x_e) \leq \mathcal{S}(y_1, \ldots, y_e)$ if $x_i \leq y_i$ for all i.

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

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* $(R_1, \ldots, R_n, \mathcal{S}, K)$ *such that* $\mathcal{S}$ *is a monotonic scoring function, the relations* $R_1, \ldots, R_n$ *are accessed sequentially in decreasing order of* $\overline{\mathcal{S}}$*, and* $1 \leq K \leq |R_1 \bowtie \cdots \bowtie R_n|$.

We do not place any restrictions on the input relations except that each R_i is accessed sequentially and in decreasing order of $\overline{\mathcal{S}}$. More formally, we use $R_i[p]$ to denote the p-th tuple in R_i and assume that $\overline{\mathcal{S}}(R_i[p]) \geq \overline{\mathcal{S}}(R_i[q])$ for $q \geq p$. We note that this particular access model is a common assumption in works that study 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 $I = (R_1, \ldots, R_n, \mathcal{S}, K)$ is an ordered relation O that comprises the top K tuples of $R_1 \bowtie \ldots \bowtie R_n$ ordered by $\mathcal{S}$. 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 $\mathcal{S}(O[p]) = \mathcal{S}(O'[p])$ for all p. The last score, $\mathcal{S}(O[K])$, is referred to as the *terminal score*, and is denoted $\mathcal{S}^{\text{term}}$.

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 $\mathcal{S}$ is a weighted average. The top 10 join results are found by solving the instance $(H, R, \mathcal{S}, 10)$. 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 work is focused on deterministic algorithms for the rank join problem. Specifically, algorithms considered in this paper must be determined by the visible contents of the input relations and the values of the $\overline{\mathcal{S}}$ function on combinations of input tuples that have been read. This means an algorithm may not use any metadata, such as the domains of the attributes. These 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 $I = (R_1, \ldots, R_n, \mathcal{S}, K)$, 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 deter-

Algorithm template PBRJ$(R_1, \ldots, R_n, \mathcal{S}, K)$
Template parameters: pulling strategy P; bounding scheme B
Input: relations $R_1, \ldots, R_n$; scoring function $\mathcal{S}$; 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 ← ∞
2.  while |O| < K OR min_{ω∈O} S(ω) < t do
3.      i ← P.chooseInput()
4.      ρ_i ← next unseen tuple of R_i
5.      R ← HR_1 ⋈ ... HR_{i-1} ⋈ {ρ_i} ⋈ HR_{i+1} ⋈ ··· ⋈ HR_n
6.      Add each member of R to O, retaining only the top K tuples
7.      Add ρ_i to HR_i
8.      t ← B.updateBound(ρ_i)
9.  end while
10. return O
```

Figure 2: PBRJ Template.

ministic *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 ρ_i is stored in an input buffer HR_i (typically a hash table). New join results are generated by joining ρ_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 are returned when the K-th buffered score 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 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 ∞. 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 $\mathcal{A}$ 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 $\mathcal{S}(1, \ldots, 1)$. A call of $updateBound(\rho_i)$ sets $t_i = \overline{\mathcal{S}}(\rho_i)$ if R_i still has unseen tuples, or $t_i = -\infty$ if R_i is exhausted. The returned bound is $\max(t_1, \ldots, t_n)$.

The name of the corner bound comes from the fact that the score bound $\overline{\mathcal{S}}(\rho_i)$ in the derivation of t_i is computed by setting every base score outside of R_i to 1, which effectively puts the base scores of other relations in 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 (intuitively, R_i has the highest potential). Ties are broken in favor of the input with the least current depth, or the input with the least index.

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

4. THE CORNER BOUND

In Section 3, 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 $\mathcal{F}_{\mathsf{c}}$, which we define as the set of PBRJ algorithms that use the corner bound. $\mathcal{F}_{\mathsf{c}}$ is equivalent to the HRJN family of algorithms [4] that have received significant attention in previous studies [6, 7, 10], so any results that we can derive about $\mathcal{F}_{\mathsf{c}}$ also hold for HRJN. We also note that the "basic" HRJN operator and the HRJN* variant are essentially iterative implementations of $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ and $\text{PBRJ}_{\mathsf{c}}^{*}$, and hence these corresponding algorithms have the same performance.

The remainder of the section is structured in two parts. We first investigate the space of algorithms in $\mathcal{F}_{\mathsf{c}}$, effectively analyzing the impact of the pulling strategy when the corner bound is employed. We then expand our analysis to see how well $\mathcal{F}_{\mathsf{c}}$ measures up to other algorithms in $\mathcal{F}$. Before presenting our results, we observe that an algorithm in $\mathcal{F}_{\mathsf{c}}$ will terminate if and only if K results have been buffered and the last read tuple ρ_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 $\mathcal{F}_{\mathsf{c}}$

4.1.1 *Instance Optimality of* $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ *and* $\text{PBRJ}_{\mathsf{c}}^{*}$

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 $\mathcal{F}_{\mathsf{c}}$. 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 $\mathcal{F}_{\mathsf{c}}$.

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

THEOREM 4.1. $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ *is instance-optimal within algorithms $\mathcal{F}_{\mathsf{c}}$ 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 $A \in \mathcal{F}_{\mathsf{c}}$. Suppose that A pulls p_i tuples from each relation R_i and define $p_{\max} = \max_i(p_i)$. We claim that $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ cannot pull more than $p_{\max}$ tuples from any relation. We prove this by contradiction. Assume that $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ pulls $p_{\max}$ complete rounds, and then pulls an additional tuple. At the point that this additional tuple is read, $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ has seen at least all of the input tuples seen by A, so it has buffered the top K results returned by A. However $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ 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 $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ cannot pull more than $p_{\max}$ tuples from any relation, from which we have

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

Since A and I were chosen arbitrarily, we conclude that $\text{PBRJ}_{\mathsf{c}}^{\mathsf{RR}}$ is instance-optimal with an optimality ratio of n. ∎

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

Proof: By contradiction. Suppose that there is some input k such that $depth(\text{PBRJ}^*_\mathsf{c}, I, k) > depth(\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}, I, k)$. Let O^{RR} be the result returned by $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$. For each i, we define $p_i^{\mathsf{RR}} = depth(\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}, I, i)$. Also let p_i be the current depth of R_i when PBRJ^*_c decides to pull $R_k[p_k^{\mathsf{RR}} + 1]$. We derive a contradiction by showing that PBRJ^*_c must actually halt at this moment.

We first claim that PBRJ^* has seen all input tuples that participate in O^{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^{RR}. Since PBRJ^*_c pulls R_k, we know that R_k is also unexhausted and $\overline{\mathcal{S}}(R_k[p_k]) \geq \overline{\mathcal{S}}(R_i[p_i])$. We proceed in two cases.

Case 1: $\overline{\mathcal{S}}(R_k[p_k]) = \overline{\mathcal{S}}(R_i[p_i])$. In this case, there was a tie, and PBRJ^*_c 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

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

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

$$p_i = p_k = p_k^{\mathsf{RR}} \geq p_i^{\mathsf{RR}}$$

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

Case 2: $\overline{\mathcal{S}}(R_k[p_k]) > \overline{\mathcal{S}}(R_i[p_i])$. Since $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ terminates without reading more than $p_k^{\mathsf{RR}} = p_k$ tuples from R_k, we know $\mathcal{S}^{\text{term}} \geq \overline{\mathcal{S}}(R_k[p_k]) > \overline{\mathcal{S}}(R_i[p_i])$. 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^*_c are met, so it will halt before reading $R_k[p_k^{\mathsf{RR}} + 1]$, contradicting our assumption. ∎

The previous result provides an interesting tool for the analysis of PBRJ^*_c. Essentially, it is non-trivial to analyze the performance of PBRJ^*_c 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 $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$, e.g., the instance-optimality property of Theorem 4.1. Theorem 4.2 makes it possible to transfer the same properties to PBRJ^*_c by virtue of the fact that PBRJ^*_c can never exceed the depths of the round-robin strategy. The following result is a direct application of this idea.

THEOREM 4.3. PBRJ^*_c *is instance-optimal within algorithms $\mathcal{F}_\mathsf{c}$ and instances $\mathcal{I}^{n\text{-rel}}$ with an optimality ratio of n.* ∎

4.1.2 *Optimal Cases for* PBRJ^*_c

Our analysis has shown that $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ and PBRJ^*_c are instance-optimal within $\mathcal{F}_\mathsf{c}$, 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 $\mathcal{S}^{\text{term}}$. The intuitive reason is that any algorithm in $\mathcal{F}_\mathsf{c}$ will need to read beyond the tuples whose bound is strictly greater than $\mathcal{S}^{\text{term}}$, 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 $\mathcal{S}^{\text{term}}$. We first consider *intra-input threshold ties* where $\overline{\mathcal{S}}(R[p]) = \overline{\mathcal{S}}(R[q]) = \mathcal{S}^{\text{term}}$ for some input relation R and distinct indices $p \neq q$. We also consider *inter-input threshold ties* where $\overline{\mathcal{S}}(R[a]) = \overline{\mathcal{S}}(R'[a']) = \mathcal{S}^{\text{term}}$ 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 intra- and inter-input threshold ties respectively.

We show below that PBRJ^*_c is optimal within $\mathcal{F}_\mathsf{c}$ with a very light restriction on the ties occurring in the input. In particular, we show that optimality holds for any problem instance outside $\mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}}$ where both intra- and inter-input threshold ties occur. Our proof uses a lemma showing that PBRJ^*_c minimizes the depth on each input when the problem instance is in $\mathcal{I} - (\mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}})$. The optimality result follows immediately.

LEMMA 4.1. *Suppose that $A \in \mathcal{F}_\mathsf{c}$ 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 $depth(\text{PBRJ}^*_\mathsf{c}, I, k) \leq depth(A, I, k)$ for all k.*

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

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

since PBRJ^*_c pulls from R_k and A halts at depth $p_k^A = p_k$ on R_k.

We next prove that PBRJ^*_c 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^*_c 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.

We previously established that $\mathcal{S}^{\text{term}} \geq \overline{\mathcal{S}}(R_i[p_i])$ for each unexhausted relation R_i, so PBRJ^*_c will halt without reading past $R_k[p_k^A]$, contradicting our assumption. ∎

THEOREM 4.4. PBRJ^*_c *is optimal within algorithms $\mathcal{F}_\mathsf{c}$ and instances $\mathcal{I} - (\mathcal{I}^{\text{inter}} \cap \mathcal{I}^{\text{intra}})$.* ∎

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

In the previous subsection, we showed various optimality properties of $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ and PBRJ^*_c compared to other algorithms in $\mathcal{F}_\mathsf{c}$. 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 $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ and PBRJ^*_c are instance-optimal.

THEOREM 4.5. $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ *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 $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ 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 $\text{PBRJ}^{\mathsf{RR}}_\mathsf{c}$ and PBRJ^*_c 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 $I = (R_1, R_2, R_3, \mathcal{S}, 1)$ where each R_i has a base score b_i and $\mathcal{S}(b_1, b_2, b_3) = b_1 + b_2 + b_3$. The contents of the relations are as follows.

R_1	Join	b_1
	a	1.0
	x	0.9
	...	
	x	0.6
	x	0.4

R_2	Join	b_2
	y	1.0
	a	0.7
	y	0.4
	...	

R_3	Join	b_3
	z	1.0
	a	0.8
	z	0.4
	...	

Let τ 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 $2.5 = \mathcal{S}(\tau)$. With this reasoning, an algorithm may correctly return τ after reading the first row of R_1 and the first three rows of R_2 and R_3. Any algorithm in $\mathcal{F}_{\mathsf{c}}$ 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 $(R_1, R_2 \bowtie R_3, \mathcal{S}, 1)$. The first score bound in $R_2 \bowtie R_3$ is 2.5, so all join results have a score ≤ 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 $\mathcal{F}_{\mathsf{c}}$ must read all of R_1. ∎

We remark that the problem instances used in this proof did not fall in the classes $\mathcal{I}^{\text{intra}}$ or $\mathcal{I}^{\text{inter}}$ that we showed to cause suboptimality for PBRJ$^*_{\mathsf{c}}$ 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$^*_{\mathsf{c}}$ is instance-optimal within $\mathcal{F}_{\mathsf{c}}$ (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$^*_{\mathsf{c}}$ for the implementation of a rank join operator based on $\mathcal{F}_{\mathsf{c}}$. 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 substitutes $\langle 1, 1\rangle$ for $\langle b_2, b_3\rangle$ when computing the score 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 $I = (R_1, \ldots, R_n, \mathcal{S}, K)$ 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 $I' = (R'_1, \ldots, R'_n, \mathcal{S}', K)$ 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) *$\mathcal{S}(x) = \mathcal{S}'(x)$ for each $x \in \bigtimes_{i=1}^{n} \mathbf{b}[HR_i] \cup \{1\}^{e_i}$ 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 $\mathcal{S}'(\tau)$ is a* potential score *of I.*
- *A bounding scheme is* tight *if whenever $\left|\bowtie_{i=1}^{n} HR_i\right| \geq K$, updateBound returns a potential score or $-\infty$.* ∎

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 $\overline{\mathcal{S}}$ 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 an optimality ratio of n within the class of deterministic algorithms. As mentioned in Section 4, this is the best optimality ratio possible [3].

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 $I = (R_1, \ldots, R_n, \mathcal{S}, K)$ be an arbitrary rank join problem instance in $\mathcal{I}^{n\text{-rel}}$. 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 τ 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 τ 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

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

indicating the instance optimality of F. ∎

This result 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 $x \preceq y$ 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 $X \lhd Y$, if for each $x \in X$ there exists $y \in Y$ such that $x \preceq y$. We observe that the $\lhd$ relation is preserved by the $\subseteq$ relation in the

L	a	b_1	b_2
	x	0.3	1.0
	z	0.3	0.5
	...		

R	a	b_3	b_4
	x	1.0	0.6
	y	0.6	1.0
	y	0.6	0.6
	y	0.5	0.5
	...		

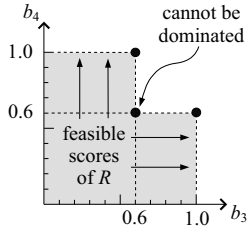

Figure 3: Illustration of Example 5.1.

following sense: if $X \subseteq Y \lhd C$ then $X \lhd C$, and if $X \lhd C \subseteq D$ then $X \lhd D$. 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 $\mathcal{S}(b_1, b_2, b_3, b_4) = b_1 + b_2 + b_3 + b_4$. After observing the shown tuples, one result τ is known for a score of $\mathcal{S}(\tau) = 2.9$. It is clear that no additional tuples need to be read from L, since the last tuple has a score bound less than $\mathcal{S}(\tau)$. 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 ρ in R based on the observed score values (this is the only information about $\mathcal{S}$ that is available to an algorithm). We know that ρ cannot have base scores that dominate the vector $\langle 0.6, 0.6\rangle$ since ρ would have a score bound at least 3.2 by the monotonicity of $\mathcal{S}$, 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 $\max\{\mathcal{S}(\mathbf{b}(\lambda)\, b_R) \mid \lambda \in L \text{ is visible} \wedge b_R \in C\} = 2.9 \leq \mathcal{S}(\tau)$. Overall, τ 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$^*_c$ *would need to read more from R since the bound for R is $3.0 > \mathcal{S}(\tau)$.* ∎

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 g_i, which keep track of tuples from R_i that have the same score bound. More concretely, g_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 g_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 g_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 is found, i.e., $\overline{\mathcal{S}}(\rho_i) < g_i$. If γ_i is in the current group G_i and τ_i is an unseen tuple in $R_i - HR_i$, then the sorted access of R_i implies that $\overline{\mathcal{S}}(\tau_i) \leq \overline{\mathcal{S}}(\rho_i) < \overline{\mathcal{S}}(\gamma_i)$, and hence it follows that

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

Procedure $updateBound(\rho_i)$
Input: tuple ρ_i from relation R_i
1. **if** $\overline{\mathcal{S}}(\rho_i) < g_i$ **then**
2. $\quad CR_i \leftarrow updateCR(CR_i, \mathbf{b}[G_i])$
3. $\quad G_i \leftarrow \{\rho_i\}$; $g_i = \overline{\mathcal{S}}(\rho_i)$
4. **else** $G_i \leftarrow G_i \cup \{\rho_i\}$ **end if**
5. **return** $\max\{resultBound(W) \mid W \subseteq \{1, \ldots, n\} \wedge W \neq \emptyset\}$

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
1. **if** $Y = \emptyset$ **then return** C **end if**
2. $y \leftarrow$ some element of Y
3. $S \leftarrow updateCR(C, Y - \{y\})$
4. $S^- \leftarrow \{s \in S \mid y \preceq s\}$
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 $W = \{1, \ldots, w\}$.
1. **if** $|\bowtie_{j>w} HR_j| = 0$ **OR** R_j is exhausted for some $j \leq w$ **then**
2. $\quad$ **return** $-\infty$ **end if**
3. $H \leftarrow \{\mathbf{b}(\eta) \mid \eta \in \bowtie_{j>w} HR_j\}$
4. $C \leftarrow \{\bigtimes_{j \leq w} cut(HR_j, \widehat{c}_j) \mid \forall j \leq w(\widehat{c}_j \in CR_j)\}$
5. $t_{\text{cover}} \leftarrow \max\{\min\{\mathcal{S}(c\, h) \mid c \in X\} \mid X \in C \wedge h \in H\}$
6. $t_{\text{order}} \leftarrow \min\{g_j \mid j \leq w\}$
7. **return** $\min\{t_{\text{cover}}, t_{\text{order}}\}$

Figure 4: The feasible-region bounding scheme.

$\gamma_i \not\preceq \tau_i$ by the monotonicity of $\mathcal{S}$. 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. (See the technical report [9] for an example.)

An example of a call to $updateCR$ is shown in Figure 5. Observe that the cover points $\langle 0.4, 1.0\rangle$ and $\langle 0.7, 0.8\rangle$ 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 $\langle 0.3, 0.6\rangle$. We note that $\langle 0.4, 0.6\rangle$ and $\langle 0.3, 0.8\rangle$ are redundant, as they do not affect the shaded region. Such redundant points can be removed,[1] 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 τ 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 τ if $\bowtie_{j \notin W} HR_j$ 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 describe the computation of t_{order} and t_{cover} next, assuming for simplicity that $W = \{1, \ldots, w\}$.

The t_{order} bound is based on the ordering of the input relations,

[1] This can be achieved by computing the skyline of the cover.

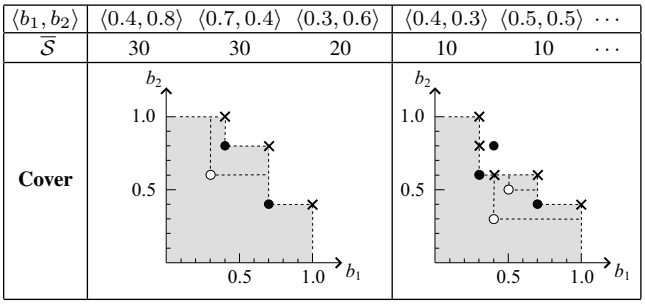

Figure 5: Cover update via updateCR. The dots represent the base score vectors, and the white dots signify the current group. The cover points are shown as crosses, and the shaded space that they dominate contains all unseen base scores. The first graph shows the state of the algorithm after observing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

which tell us that, for $j \leq w$, the witness τ_j of τ from the unseen portion of R_j satisfies $\overline{\mathcal{S}}(\tau_j) \leq g_j$. It follows that $\mathcal{S}(\tau) \leq g_j$, so we set $t_{\text{order}} = \min\{g_j \mid j \leq w\}$.

To compute t_{cover}, we first recall that each unseen witness τ_j for $j \leq w$ is dominated by a member of CR_j, and thus some combination of cover points in $\bigtimes_{j\leq w} CR_j$ must dominate the join of the first w witnesses $\tau_1 \bowtie \cdots \bowtie \tau_w$. Given that the join of visible witnesses $\tau_{w+1} \bowtie \cdots \bowtie \tau_n$ must be in $\bowtie_{j>w} HR_j$, it follows that $\mathcal{S}(\tau)$ can be bounded by the maximum score over the cross product of $\bigtimes_{j\leq w} CR_j$ and $\bowtie_{j>w} HR_j$. We write this "ideal" bound as

$$\widehat{t}_{\text{cover}} = \max\{\mathcal{S}(\widehat{c}\,\mathbf{b}(\eta)) \mid \widehat{c} \in \textstyle\bigtimes_{j\leq w} CR_j \wedge \eta \in \bowtie_{j>w} HR_j\}.$$

We cannot use $\widehat{t}_{\text{cover}}$ in our algorithm, since algorithms in $\mathcal{A}$ are only allowed to know the value of $\overline{\mathcal{S}}$ on combinations of base score vectors that have been read. In other words, $\mathcal{S}$ may be evaluated using any vector among $\mathbf{b}[HR_j] \cup \{1\}^{e_j}$ 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 $\widehat{c}$ is the concatenation of cover points $\widehat{c}_1 \cdots \widehat{c}_w$ where each $\widehat{c}_j \in CR_j$. Now the admissible base score vectors that dominate $\widehat{c}_j$ are given by $cut(HR_j, \widehat{c}_j)$, so it follows that any $c \in \bigtimes_{j\leq w} cut(HR_j, \widehat{c}_j)$ is a possible replacement for $\widehat{c}$ in the expression for $\widehat{t}_{\text{cover}}$. If we use the minimum over all values of c, we get the expression for t_{cover} used in the algorithm.[2]

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.

[2]A practical implementation of the FR bound should use $\widehat{t}_{\text{cover}}$ since it is no larger than t_{cover}, and simpler to compute.

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

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, $X \lhd S$ 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 $y \not\preceq x$ 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 $y[j] > x[j] \geq 0$. 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 $x \preceq s^+ \in S^+ \cap (0,1]^e$, as desired. ∎

CLAIM 5.2. *After each call to the updateBound method of the FR bound, $CR_k \rhd \mathbf{b}[R_k - HR_k]$ 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 $p \geq 0$. We show that $CR_k^p \rhd \mathbf{b}[R_k - HR_k^p]$ by induction on p. Considering the value of CR_k before the first iteration, we have $CR_k^0 = \{1\}^{e_k} \rhd [0,1]^{e_k} \supseteq \mathbf{b}[R_k - HR_k^0]$. For the inductive case, let $p \geq 1$. First we observe that

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

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 $\overline{\mathcal{S}}(\rho_k) < g_k$. 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

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

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

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

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

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

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

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

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 $I = (R_1, \ldots, R_n, \mathcal{S}, K)$. 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 $\widehat{c}_j \in CR_j$ such that $c_j \in cut(HR_j, \widehat{c}_j)$ 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 $I' = (R'_1, \ldots, R'_n, \mathcal{S}', K)$ that has a potential score of t. Here, we specify conditions that I' must satisfy; the verification that such an instance exists is left to the technical report [9].

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

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

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

$A_{j\ell} = \{\frac{1}{2}(a_j[\ell] + \widehat{c}_j[\ell]) \mid a_j \in \mathbf{b}[HR_j] \cup \{0\}^{e_j} \wedge a_j[\ell] < \widehat{c}_j[\ell]\}$.

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 $\widehat{c}_j$ on every dimension. It also implies that a member of $\mathbf{b}[HR_j]$ dominates $\mathbf{b}(\rho_j)$ only if it dominates $\widehat{c}_j$. We use τ to denote the result of joining $\rho_1, \ldots, \rho_w$ with η.

Let $\mathcal{S}'$ 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 $x \in \bigtimes_{i=1}^{n} \mathbf{b}[HR_i] \cup \{1\}^{e_i}$.

(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, $\mathcal{S}'$ 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 τ is a potential result because the tuples $\rho_1, \ldots, \rho_w$ used to construct τ are not visible. Now $\mathcal{S}'(\tau) = t$ 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
	A	B	1.0
	D	D	0.3
	E	E	0.2
	F	F	0.1

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

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

$$\Phi = (L_{11} \vee L_{12} \vee L_{13}) \wedge (L_{21} \vee L_{22} \vee L_{23}) \wedge \cdots \wedge (L_{n1} \vee L_{n2} \vee L_{n3})$$

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 $I = (R_1, \ldots, R_{2n}, \mathcal{S}, 1)$ based on the formula Φ. For $i \leq n$, let R_{2i-1} be the result of renaming $yxb \to y_{i-1}x_ib_i$ in T, and R_{2i} be the result of renaming $xy'\overline{b} \to x_iy_i\overline{b}_i$ 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 $b = \langle b_1, \overline{b}_1, b_2, \overline{b}_2, \ldots, b_n, \overline{b}_n \rangle$, define

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

where $h(v_i) = b_i$ and $h(\overline{v}_i) = \overline{b}_i$. The idea is that $f(b)$ indicates whether Φ 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

$$\begin{aligned} g(b) &= \min\{b_1, \overline{b}_1, \ldots, b_n, \overline{b}_n\} \\ \text{and } \mathcal{S}(b) &= f(b) + g(b). \end{aligned}$$

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 Φ is satisfiable if and only if I has a potential score > 0.6.

First assume that Φ 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 τ. Since I' is clearly a continuation of I and τ only occurs in the join results of I', we know that τ is a potential result of I. The key property of τ 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 $\mathcal{S}(\tau) = 1.1 > s$ as desired.

Now assume there is a potential result τ from a continuation $I' = (R'_1, \ldots, R'_{2n}, \mathcal{S}', 1)$ such that $\mathcal{S}'(\tau) > 0.6$. Let $\tau_1, \overline{\tau}_1, \ldots, \tau_n, \overline{\tau}_n$ denote the witnesses of τ 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 τ_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, $\langle b_i, \overline{b}_i \rangle \preceq \langle c_i, \overline{c}_i \rangle$ 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 $\mathbf{b}(\tau) \preceq c$. Moreover, the base scores in c are visible in I, so (C3) leads to

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

This implies $f(c) > 0.3$ since $g(c) \leq 0.3$. 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 Φ is satisfied by this assignment and we are done. ∎

Looking at the above 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 $I = (R_1, \ldots, R_n, \mathcal{S}, K)$ 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

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

The optimality of PBRJ$^*_\mathsf{c}$, 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 that (a) the join condition θ becomes part of the problem instance, and (b) a continuation can employ any join condition θ' that gives the same answers as θ on the visible portion of the input.

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 $\mathcal{T}_\mathsf{c}$ denote the algorithms in $\mathcal{T}$ that use these plans. The following lemma shows that if $T \in \mathcal{T}_\mathsf{c}$, then there is a single PBRJ operator using the corner bound that outperforms the tree T on all instances.

LEMMA 6.1. *For any $T \in \mathcal{T}_\mathsf{c}$ there exists $F_T \in \mathcal{F}_\mathsf{c}$ 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 $\mathcal{S}^{\text{term}}$, 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 $\mathcal{T}_\mathsf{c}$. For example, we see that PBRJ$^*_\mathsf{c}$ is optimal within algorithms $\mathcal{T}_\mathsf{c}$ and instances $\mathcal{I} - (\mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}})$, because otherwise there would be an algorithm $T \in \mathcal{T}_\mathsf{c}$ outperforming PBRJ$^*_\mathsf{c}$ on some instance $I \notin \mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}}$. Then Lemma 6.1 implies that $F_T \in \mathcal{F}_\mathsf{c}$ also outperforms PBRJ$^*_\mathsf{c}$ on I, which contradicts Theorem 4.4. We can similarly show that PBRJ$^{\mathsf{RR}}_\mathsf{c}$ and PBRJ$^*_\mathsf{c}$ are instance-optimal in $\mathcal{T}_\mathsf{c}$. Recalling that PBRJ$^*_\mathsf{c}$ is equivalent to the HRJN* algorithm, these observations suggest that existing optimizers that consider plans

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

- $\text{PBRJ}^*_{\mathsf{c}}$ never costs more than $\text{PBRJ}^{\text{RR}}_{\mathsf{c}}$
- *For* $\mathcal{I}^{n\text{-rel}}$: $\text{PBRJ}^*_{\mathsf{c}}$ and $\text{PBRJ}^{\text{RR}}_{\mathsf{c}}$ are instance-optimal
- *For* $\mathcal{I} - (\mathcal{I}^{\text{intra}} \cap \mathcal{I}^{\text{inter}})$: $\text{PBRJ}^*_{\mathsf{c}}$ is optimal

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

- *For* $\mathcal{I}^{2\text{-rel}} \cap \mathcal{I}^{1\text{-dim}}$: $\text{PBRJ}^*_{\mathsf{c}}$ and $\text{PBRJ}^{\text{RR}}_{\mathsf{c}}$ are instance-optimal
- *For more general problems*: no algorithm in $\mathcal{F}_{\mathsf{c}}$ 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* $\mathcal{I}^{n\text{-rel}} \cap \mathcal{I}^{\text{e-dim}}$: the feasible-region bound takes polynomial time
- *Under query complexity*: RESULTBOUND is NP-hard

Figure 6: Summary of Main Results

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 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 Ilyas 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 $\text{PBRJ}^{\text{RR}}_{\mathsf{c}}$ and $\text{PBRJ}^*_{\mathsf{c}}$. 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, 10] 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 [10] 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 $\text{PBRJ}^*_{\mathsf{c}}$ is never more than $\text{PBRJ}^{\text{RR}}_{\mathsf{c}}$. 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.

This work was supported in part by the National Science Foundation under Grant No. IIS-0447966p and by an IBM Faculty Development Award.

9. REFERENCES

[1] 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 and Neoklis Polyzotis. Evaluating rank joins with optimal cost. Technical report, UC Santa Cruz, 2007. Available at http://www.soe.ucsc.edu/research/reports/UCSC-CRL-07-10.pdf.

[10] Karl Schnaitter, Joshua Spiegel, and Neoklis Polyzotis. Depth estimation for ranking query optimization. In *International Conference on Very Large Databases*, pages 902–913, 2007.