Proximity Rank Join

Davide Martinenghi Marco Tagliasacchi
Dipartimento di Elettronica e Informazione – Politecnico di Milano
Piazza Leonardo da Vinci, 32 – 20133 Milano, Italy
{martinen,tagliasa}@elet.polimi.it

ABSTRACT

We introduce the proximity rank join problem, where we are given a set of relations whose tuples are equipped with a score and a real-valued feature vector. Given a target feature vector, the goal is to return the K combinations of tuples with high scores that are as close as possible to the target and to each other, according to some notion of distance. The setting closely resembles that of traditional rank join, but the geometry of the vector space plays a distinctive role in the computation of the overall score of a combination. Also, the input relations typically return their results either by distance from the target or by score. Because of these aspects, it turns out that traditional rank join algorithms, such as the well-known *HRJN*, have shortcomings in solving the proximity rank join problem, as they may read more input than needed. To overcome this weakness, we define a tight bound (used as a stopping criterion) that guarantees instance optimality, i.e., an I/O cost is achieved that is always within a constant factor of optimal. The tight bound can also be used to drive an adaptive pulling strategy, deciding at each step which relation to access next. For practically relevant classes of problems, we show how to compute the tight bound efficiently. An extensive experimental study validates our results and demonstrates significant gains over existing solutions.

1. INTRODUCTION

Imagine a smartphone user wishing to organize the evening by finding a restaurant, a movie theater and a hotel that are nearby, close to each other, and recommended in terms of, respectively, price, user rating, and number of stars. This can be done by suitably collecting and assembling different pieces of information, as envisaged in search computing [4]. Here, one could, e.g., exploit the current geographical position available on the smartphone, as well as local information obtained from specific search services on the Internet (such as Yahoo! Local, IMDB, etc.).

This simple example captures the essence of *proximity rank join* problems, in which the best combinations of objects coming from different services (here: restaurants, theaters, and hotels) are sought, and each object is equipped with both a score and a real-valued feature vector. The aggregation function assigning a score to a combination depends on the individual scores, on the proximity of the individual vectors to a given vector (called query) and on their mutual proximity, according to some notion of distance in the feature space. Typically, objects are retrieved sorted by either distance from a given vector or by score.

The interest in proximity rank join is motivated by its generality and broad applicability. Indeed, it covers multi-domain search, as in the previous example, as well as several other scientific fields, such as: *i)* information retrieval, e.g., finding similar documents in different data collections given a set of keywords; *ii)* multimedia databases, e.g., requesting similar images from different repositories given a sample image; *iii)* bioinformatics, e.g., discovering orthologous genes from different organisms given a target annotation profile; and many others. In all these cases, proximity plays a crucial role, as it captures the mutual relationships between the objects in the feature space.

In the conventional rank join problem [10], instead, the overall score of a combination depends only on the scores of the single objects. Existing algorithms, such as *HRJN* [9], may be used to address proximity rank join. However, as will be shown in this paper, they fail to achieve optimality as they are completely oblivious of the proximity aspect.

Our contributions. We formally define the proximity rank join problem and propose algorithms that solve it. We identify the main differences with traditional rank join and revisit existing algorithms and templates in the new setting in order to assess their limitations and potential. To this end, as customary in top-k query answering, we refer to the notion of instance-optimality [6] to characterize I/O efficiency. We show that, for the proximity rank join problem, no algorithm using a stopping criterion based on the so-called corner bound (such as *HRJN* and *HRJN**) is instance-optimal, even with only two input relations.

We then introduce a different bound that is tight [13] and can thus be used in order to guarantee instance-optimality. We show how to compute the tight bound efficiently for practically relevant instances of the problem, in particular when a Euclidean distance is used.

In order to further improve efficiency in terms of I/O as well as CPU cost, we refine the exploration strategy used by the algorithm to pull tuples from the inputs. Also, we exploit geometrical properties to conclude that part of the

Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

search space is dominated and thus need not be explored.

Finally, we provide a thorough experimental evaluation of our approach by analyzing performance with regard to the main operating parameters. These include the number of desired top combinations, the number of dimensions of the feature space, the density of tuples in the space, its skewness, and the number of relations in the join. Our empirical study is carried out both on real and on synthetic data.

2. PRELIMINARIES

Consider a set of relations $R_1, \ldots, R_n$ where each tuple $\tau_i \in R_i$ is composed of named attributes, a real-valued feature vector $\mathbf{x}(\tau_i) \in \mathbb{R}^d$, and a score $\sigma(\tau_i) \in \mathbb{R}$. Let $\mathbf{q} \in \mathbb{R}^d$ denote a constant vector representing the query, and $\delta(\mathbf{x}(\tau), \mathbf{q})$ a metric distance between vectors $\mathbf{x}(\tau)$ and $\mathbf{q}$. We consider the two most common access kinds that arise in practice:

A. *Distance-based access*: The relations $R_1, \ldots, R_n$ are accessed sequentially in increasing order of $\delta(\cdot, \mathbf{q})$.

B. *Score-based access*: The relations $R_1, \ldots, R_n$ are accessed sequentially in decreasing order of $\sigma(\cdot)$.

We indicate with $\tau_i^{(r_i)} = R_i[r_i]$ the r_i-th tuple extracted from R_i according to the available access kind, and with $P_i \subseteq R_i$ the ordered relation containing the tuples already extracted from R_i. The number $p_i = |P_i|$ of such tuples is called *depth*.

Let $\tau = \tau_1 \times \cdots \times \tau_n \in R_1 \times \ldots \times R_n$ denote an element of the cross-product of the n relations, hereafter called *combination*. The *aggregate score* $\mathcal{S}(\tau)$ of τ is defined as

$$\mathcal{S}(\tau) = f\left(\mathcal{S}(\tau_1), \ldots, \mathcal{S}(\tau_n)\right) \tag{1}$$

where $\mathcal{S}(\tau_i) = g_i\left(\sigma(\tau_i), \delta(\mathbf{x}(\tau_i), \mathbf{q}), \delta(\mathbf{x}(\tau_i), \boldsymbol{\mu}(\tau))\right)$, and

- The function $f(x_1, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$ is monotonically non-decreasing in all of its arguments.
- The functions $g_i(x, y, z) : \mathbb{R} \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}$, for $i = 1, \ldots, n$, are monotonically non-decreasing in x and monotonically non-increasing in y and z.
- The vector $\boldsymbol{\mu}(\tau) \in \mathbb{R}^d$ denotes the *centroid* of a combination, computed as $\arg\min_{\boldsymbol{\omega}} \sum_{i=1}^n \delta(\mathbf{x}(\tau_i), \boldsymbol{\omega})$.

We refer to the functions g_i as the *proximity weighting functions* since they compute the *proximity weighted score* $\mathcal{S}(\tau_i)$ of the tuple τ_i participating in the combination τ. Intuitively, the proximity weighted score increases with the score and decreases with the distance from the query vector $\mathbf{q}$ and from the combination centroid $\boldsymbol{\mu}(\tau)$. Therefore, the top combinations are those whose constituent tuples: *i)* have high scores; *ii)* are close to the query vector; *iii)* are close to each other. We note that (1) generalizes the *Maximize-Over-Location* scoring function recently defined in [16].

Example 2.1. As a concrete example, we consider

$$\mathcal{S}(\tau) = \sum_{i=1}^{n} w_s \ln(\sigma(\tau_i)) - w_q \|\mathbf{x}(\tau_i) - \mathbf{q}\|^2 - w_\mu \|\mathbf{x}(\tau_i) - \boldsymbol{\mu}(\tau)\|^2 \tag{2}$$

which is obtained by setting $f(x_1, \ldots, x_n) = \sum_{i=1}^n x_i$ and $g_i(x, y, z) = w_s \ln(x) - w_q y^2 - w_\mu z^2$, i.e., adopting a Euclidean distance. The weights w_s, w_q and $w_\mu \in \mathbb{R}^+$ can be used to tune the proximity weighting functions based on user's preferences. Note that if the scores $\sigma(\tau_i) \in [0, 1]$ then $\mathcal{S}(\tau) \in (-\infty, 0]$. Alternatively, one might take $e^{\mathcal{S}(\tau)}$ to obtain an aggregate score in the range $[0, 1]$. Table 1 illustrates an example with three relations, when two tuples have been retrieved from each of them, and tuples are sorted according to the distance from the query $\mathbf{q} = \mathbf{0}$. Figure 1(a) shows the location $\mathbf{x}(\tau_i)$ of the tuples in $\mathbb{R}^2$ represented as discs whose radius is proportional to the score (blue for R_1, red for R_2, green for R_3). Figure 1(a) also shows three circles, whose radius is equal to the distance of the last accessed tuple from each relation. The cross-product consists of 8 combinations, also reported in Table 1, sorted by their aggregate score computed using (2), by setting $w_s = w_q = w_\mu = 1$. ■

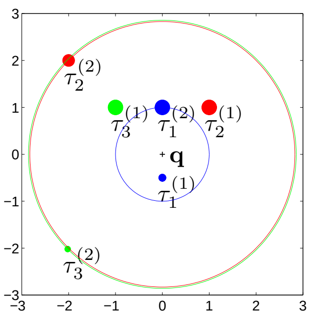
Table 1: Three relations with distance-based access and the formed combinations with their scores.

	R_1	
	σ_1	$\mathbf{x}_1^T$
$\tau_1^{(1)}$	0.5	[0, -0.5]
$\tau_1^{(2)}$	1.0	[0, 1]
…	…	…

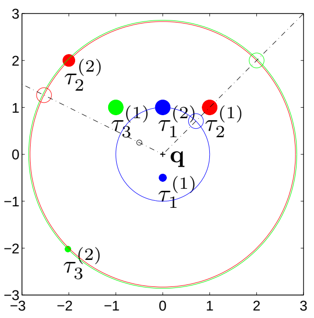
	R_3	
	σ_3	$\mathbf{x}_3^T$
$\tau_3^{(1)}$	1.0	[-1, 1]
$\tau_3^{(2)}$	0.4	[-2, -2]
…	…	…

	R_2	
	σ_2	$\mathbf{x}_2^T$
$\tau_2^{(1)}$	1.0	[1, 1]
$\tau_2^{(2)}$	0.8	[-2, 2]
…	…	…

$\tau = \tau_1 \times \tau_2 \times \tau_3$	$\mathcal{S}(\tau)$
$\tau_1^{(2)} \times \tau_2^{(1)} \times \tau_3^{(1)}$	-7.0
$\tau_1^{(1)} \times \tau_2^{(1)} \times \tau_3^{(1)}$	-8.4
$\tau_1^{(2)} \times \tau_2^{(2)} \times \tau_3^{(1)}$	-13.9
$\tau_1^{(1)} \times \tau_2^{(2)} \times \tau_3^{(1)}$	-16.3
$\tau_1^{(1)} \times \tau_2^{(1)} \times \tau_3^{(2)}$	-21.0
$\tau_1^{(2)} \times \tau_2^{(1)} \times \tau_3^{(2)}$	-22.6
$\tau_1^{(1)} \times \tau_2^{(2)} \times \tau_3^{(2)}$	-28.9
$\tau_1^{(2)} \times \tau_2^{(2)} \times \tau_3^{(2)}$	-29.5



(a) Tuples of Table 1



(b) Solution of problem (12)

Figure 1: (a) Location of the tuples represented as discs whose radius is proportional to the score. (b) Solution for: (*i*) partial combination $\tau_2^{(1)}$; (*ii*) partial combination $\tau_1^{(1)} \times \tau_3^{(1)}$ whose centroid is indicated by a black empty circle. The optimal locations of unseen tuples are represented by empty circles collinear with the the centroid and the query.

Definition 2.1. *The* proximity rank join *problem is an* $(n+2)$*-tuple* $(R_1, \ldots, R_n, \mathcal{S}, K)$ *such that* $\mathcal{S}$ *is an aggregation function defined as in* (1)*, all the relations* $R_1, \ldots, R_n$ *are accessed sequentially either in: A) increasing order of* $\delta(\cdot, \mathbf{q})$*, or B) decreasing order of* $\sigma(\cdot)$*, and* $1 \le K \le |R_1 \times \ldots \times R_n|$.

A solution is an ordered relation O containing the top K combinations from $R_1 \times \ldots \times R_n$ ordered by $\mathcal{S}$ (score ties are resolved using a tie-breaking criterion). A proximity rank join algorithm that on input $(R_1, \ldots, R_n, \mathcal{S}, K)$ returns the corresponding O upon termination is said to be *correct*.

Proximity rank join closely resembles *rank join* [9]. Unlike the latter, the former assumes that *i)* each tuple is equipped with a feature vector, *ii)* the aggregation function depends both on the score and on the feature vector, and *iii)* the relations are accessed as specified in Definition 2.1.

The proximity rank join problem can be tackled by adopting the *ProxRJ* template reported in Algorithm 1, which adapts the *Pull/Bound Rank Join* (*PBRJ*) template originally introduced for rank join in [13]. The *chooseInput* function (line 4) of a given *pulling strategy PS* decides at each step the next relation to be accessed. The *updateBound* function (line 9) of a given *bounding scheme BS* computes an upper bound on the aggregate score of the unseen combinations. The unseen combinations are those that can be formed with at least an unseen tuple $\tau_i \in R_i - P_i$.

The above-mentioned differences between rank join and proximity rank join on the assumptions regarding the relations and the aggregation function have an impact on the computation of the upper bound (and, consequently, on the pulling strategy). This will be further discussed in Section 3.

Algorithm 1: $ProxRJ(R_1, \dots, R_n, \mathcal{S}, K)$

```
    Input  : relations $R_1, \dots, R_n$; function $\mathcal{S}$ as in (1);
             result size $K$
    Output : $K$ combinations with highest aggregate score
    Data   : input buffers $P_1, \dots, P_n$; output buffer $O$
 1  begin
 2      $t \leftarrow \infty$;
 3      while $|O| < K$ or $\min_{\omega \in O} \mathcal{S}(\omega) < t$ do
 4          $i \leftarrow PS.chooseInput()$;
 5          $\tau_i \leftarrow$ next unseen tuple of $R_i$;
 6          $R \leftarrow P_1 \times \dots \times P_{i-1} \times \{\tau_i\} \times P_{i+1} \times \dots \times P_n$;
 7          Add each member of $R$ to $O$, retaining only
            the top $K$ combinations;
 8          Add $\tau_i$ to $P_i$;
 9          $t \leftarrow BS.updateBound(\tau_i)$;
10      end
11      return $O$
12  end
```

Unlike the general *PBRJ* template, *ProxRJ* focuses on cross-product (line 6), the notion of join being here implicitly captured by the proximity between the objects [15]. Yet, the results of this paper also hold when an additional join predicate is used to select a subset of the cross-product.

An algorithm complying with the *ProxRJ* template is correct (as *PBRJ* is for rank join [13]) if: *i) updateBound* returns a correct upper bound on the aggregate scores of the combinations that use at least one unread tuple; *ii) chooseInput* returns an index of an unexhausted relation.

We define the notion of tight bounding scheme as in [13], but we drop their condition (C3), which is only relevant when tuples may have multiple scores and when the behavior of $\mathcal{S}$ is only assumed to be known on the observed tuples.

DEFINITION 2.2. *Let $I = (R_1, \dots, R_n, \mathcal{S}, K)$ be a proximity rank join problem problem and let P_i be the extracted portion of R_i, $1 \le i \le n$:*

- *A continuation of I is a problem $(R'_1, \dots, R'_n, \mathcal{S}, K)$ that satisfies the following conditions.*
  (C1) *The first $|P_i|$ tuples in R'_i and R_i are identical.*
  (C2) *$|R'_i| = |R_i|$ for all i.*
- *If τ is a combination for 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* updateBound *returns a potential score or $-\infty$ whenever $|P_1 \times \dots \times P_n| \ge K$.*

In order to characterize the performance of proximity rank join algorithms, we introduce the *sumDepths* cost metric and the notion of instance optimality, which are common in top-k query answering. Let $depth(A, I, i)$ be the depth on input relation R_i explored by algorithm A on problem $I = (R_1, \dots, R_n, \mathcal{S}, K)$ before returning a solution. We define $sumDepths(A, I)$ as $\sum_{i=1}^{n} depth(A, I, i)$, which provides an indication of the amount of I/O performed by A to solve I. Let $\mathcal{A}$ be the class of correct, deterministic proximity rank join algorithms, and let $\mathcal{I}$ be the class of problems satisfying Definition 2.1. We say that A is *instance-optimal* wrt. $\mathcal{A}$ and $\mathcal{I}$ for the *sumDepths* cost metric if there exist constants c_1 and c_2 such that $sumDepths(A, I) \le c_1 \cdot sumDepths(A', I) + c_2$ for all $A' \in \mathcal{A}$ and $I \in \mathcal{I}$.

3. BOUNDING SCHEME

We carry out our analysis for the case of distance-based access. First, we consider a simple bounding scheme that does not leverage the geometrical constraints imposed by the problem formulation; this bounding scheme is equivalent to that of *HRJN* [9]. Then, we show how to define a tight upper bound for the score of unseen combinations, which is of paramount importance in order to guarantee instance optimality, as proved in [13]. We show that the computation of the upper bound requires solving a set of quadratic programs, if the aggregation function is as in (2).

Similar results and algorithms are available for the simpler case of score-based access, as shown in Appendix C.

3.1 Corner bound

Under distance-based access, it is possible to compute a correct upper bound t_c by keeping track of the distance from the query $\mathbf{q}$ of the first and last accessed tuple from each relation, i.e., $\delta(\mathbf{x}(R_i[1]), \mathbf{q})$ and $\delta(\mathbf{x}(R_i[p_i]), \mathbf{q})$, respectively:

$$t_c = \max\{t_1, \dots, t_n\}, \text{ with } t_i = f\left(\bar{\mathcal{S}}_1, \dots, \underline{\mathcal{S}}_i, \dots, \bar{\mathcal{S}}_n\right) \quad (3)$$

In (3), $\bar{\mathcal{S}}_j$ is an upper bound on the proximity weighted score that can be attained by a tuple $\tau_j \in R_j$, i.e.

$$\bar{\mathcal{S}}_j = g_j\left(\sigma_j^{\max}, \delta(\mathbf{x}(R_j[1]), \mathbf{q}), 0\right) \quad (4)$$

where $\sigma_j^{\max}$ is the maximum score possible for R_j. Similarly $\underline{\mathcal{S}}_i$ denotes an upper bound on the proximity weighted score that can be attained by an unseen tuple $\tau_i \in R_i - P_i$:

$$\underline{\mathcal{S}}_i = g_i\left(\sigma_i^{\max}, \delta(\mathbf{x}(R_i[p_i]), \mathbf{q}), 0\right) \quad (5)$$

Note that when no object has been extracted from a relation R_i, i.e., $p_i = 0$, both distances $\delta(\mathbf{x}(R_i[1]), \mathbf{q})$ and $\delta(\mathbf{x}(R_j[p_i]), \mathbf{q})$ are conventionally set to zero.

The bound in (3), corresponding to the one used by *HRJN*, is a *corner bound* [13]. Bound (3) is not tight, since a possible combination whose aggregate score is equal to the bound might not exist. Here, non-tightness precludes instance optimality.

THEOREM 3.1. *Let A be an algorithm complying with the ProxRJ template for distance-based access using the corner bound* (3). *Then A is not instance optimal for problems with at least two relations.*

A similar result was shown in [13] for *PBRJ* algorithms using the corner bound on the rank join problem. However, Theorem 3.1 does not descend from their results. Indeed, the lack of tightness in rank join derives from failure of the join predicate, whereas here we have cross-products, and the geometry of the problem (absent in rank join) contributes to determining the bounds. Moreover, unlike theirs, our result also holds with two input relations ($n = 2$), as the corner bound turns out to be tight only in the extreme case $n = 1$.

3.2 Tight bound

Let $M = \{i_1, \ldots, i_m\}$ denote a proper subset of $\{1, \ldots, n\}$ and $m = |M|$, $0 \le m < n$. Let $PC(M) = P_{i_1} \times \cdots \times P_{i_m}$ denote the set of partial combinations that can be formed using seen tuples from P_i, $i \in M$. Clearly, $|PC(M)| = \prod_{i \in M} p_i$. Given a partial combination $\tau \in PC(M)$, we want to find the maximum aggregate score that can be achieved by completing τ with unseen tuples from $R_i - P_i$, $i \in \{1, \ldots, n\} - M$. Distance-based access imposes that $\delta(\mathbf{x}(R_i[r_i]), \mathbf{q}) \ge \delta_i$, for $r_i > p_i$, where $\delta_i = \delta(\mathbf{x}(R_i[p_i]), \mathbf{q})$ is the distance from the query of the last accessed tuple from R_i. The goal is to find the feasible locations $\mathbf{y}_i = \mathbf{x}(\tau_i)$, $i \in \{1, \ldots, n\} - M$ of the unseen tuples that maximize the aggregate score, i.e.,

$$\begin{array}{ll} \text{maximize} & f(\mathcal{S}_1, \ldots, \mathcal{S}_n) \\ \text{subject to} & \delta(\mathbf{y}_i, \mathbf{q}) \ge \delta_i, i \in \{1, \ldots, n\} - M \end{array} \tag{6}$$

where

$$\mathcal{S}_i = \begin{cases} g_i\left(\sigma(\tau_i), \delta(\mathbf{x}(\tau_i), \mathbf{q}), \delta(\mathbf{x}(\tau_i), \boldsymbol{\mu})\right), & i \in M \\ g_i\left(\sigma_i^{\max}, \delta(\mathbf{y}_i, \mathbf{q}), \delta(\mathbf{y}_i, \boldsymbol{\mu})\right), & i \in \{1, \ldots, n\} - M \end{cases} \tag{7}$$

Note that all the $n - m$ optimization variables $\mathbf{y}_i$ influence the functions $\mathcal{S}_i$, $i = 1, \ldots, n$, since they participate in the computation of the combination centroid $\boldsymbol{\mu}$.

Let $\mathbf{y}_i^*$, $i \in \{1, \ldots, n\} - M$, denote a solution of (6) for a partial combination $\tau \in PC(M)$, and f^* the value of the objective function computed at $\mathbf{y}_i^*$. Thus, the upper bound $t(\tau)$ on the aggregate score of the possible combinations formed by completing τ is given by $t(\tau) = f^*$.

Then, for each proper subset M of $\{1, \ldots, n\}$, we compute

$$t_M = \max\{t(\tau) | \tau \in PC(M)\} \tag{8}$$

and we obtain the final upper bound as

$$t = \max\{t_M | M \subset \{1, \ldots, n\}\} \tag{9}$$

The detailed pseudocode of the function *updateBound* computing t is reported in Appendix B.1.

THEOREM 3.2. *The bounding scheme* (9) *is tight.*

The simplest pulling strategy is the one that accesses the inputs in a *round-robin* fashion (e.g., in the order $R_1, \ldots, R_n$). Tightness and a round-robin strategy are sufficient to guarantee instance optimality.

THEOREM 3.3. *ProxRJ with the tight bounding scheme* (9) *and a round-robin pulling strategy is instance-optimal for proximity rank join with distance-based access.*

We observe that the computation of the tight upper bound (9) comes at a cost that is polynomial under *data complexity* (i.e., w.r.t. the number of retrieved tuples). In fact, it suffices to solve the problem (6) a number of times equal to $C = \sum_{m=0}^{n-1} \binom{n}{m} \prod_{j \in M, M \subset \{1,\ldots,n\}, |M|=m} p_j$, and it is easily shown that $2^n - 1 \le C \le p^n(2^n - 1)$, where $p = \max\{p_1, \ldots, p_n\}$. This is aligned with the findings in [13], where the authors also show polynomial data complexity of tight bounding for rank join. Observe that, here too, the number of relations n weighs exponentially on the cost.

In addition, solving each instance of the problem in (6) might be difficult, depending on the aggregation function and the distance in use. In Section 3.2.1, we show that, for some relevant cases, such as when using the aggregation function in (2), the problem in (6) can be efficiently solved.

EXAMPLE 3.1. For the relations in Table 1, the upper bound t, using (2), is -7 (see Appendix B.2), and can be potentially achieved by an unseen combination formed by completing $\tau_2^{(1)} \times \tau_3^{(1)}$ with an unseen tuple from R_1. Therefore, the seen combination $\tau_1^{(2)} \times \tau_2^{(1)} \times \tau_3^{(1)}$ in Table 1 can be guaranteed to be the top-1, since its aggregate score -7 is as high as t. Note that none of the seen combinations in Table 1 can be guaranteed to be the top-1 using the corner bound, since, by (3), we have $t_c = \max\{-5, -10.25, -10.25\} = -5$. ■

3.2.1 Tight bound for a case with Euclidean distance

We focus now on aggregation functions of the form given in (2). Initially, we make the simplifying assumption that the access kind allows retrieving all tuples within a target maximum distance δ from the query $\mathbf{q}$. Thus, all unseen tuples in $R_i - P_i$ are constrained to be at a distance of at least δ. W.l.o.g., let us assume $M = \{1, \ldots, m\}$ and consider a partial combination $\tau \in PC(M)$. The upper bound $t(\tau)$ is obtained by solving the following instance of (6).

$$\begin{array}{ll} \text{max.} & \sum_{i=1}^m w_s \ln(\sigma(\tau_i)) - w_q \|\mathbf{x}(\tau_i) - \mathbf{q}\|^2 - w_\mu \|\mathbf{x}(\tau_i) - \boldsymbol{\mu}\|^2 + \\ & \sum_{i=m+1}^n w_s \ln(\sigma_i^{\max}) - w_q \|\mathbf{y}_i - \mathbf{q}\|^2 - w_\mu \|\mathbf{y}_i - \boldsymbol{\mu}\|^2 \\ \text{s.t.} & \|\mathbf{y}_i - \mathbf{q}\| \ge \delta, i \in \{m+1, \ldots, n\} \end{array} \tag{10}$$

where $\boldsymbol{\mu} = \frac{1}{n}\left(\sum_{i=1}^m \mathbf{x}(\tau_i) + \sum_{i=m+1}^n \mathbf{y}_i\right)$. Problem (10) is a non-convex QCQP (quadratically constrained quadratic program). Let $\boldsymbol{\nu} = \frac{1}{m}\sum_{i=1}^m \mathbf{x}(\tau_i)$ denote the centroid of the partial combination τ. The solution can be easily found in closed form observing that the problem is symmetric in the optimization variables $\mathbf{y}_i$. Thus, the optimal solution must satisfy $\mathbf{y}_{m+1}^* = \ldots = \mathbf{y}_n^* = \mathbf{y}^*$. We have (see Appendix B.3):

$$\mathbf{y}^* = \begin{cases} \mathbf{q} + (\boldsymbol{\nu} - \mathbf{q})\frac{m w_\mu}{m w_\mu + n w_q} & \text{if } \|(\boldsymbol{\nu} - \mathbf{q})\frac{m w_\mu}{m w_\mu + n w_q}\| \ge \delta \\ \mathbf{q} + (\boldsymbol{\nu} - \mathbf{q})\frac{\delta}{\|\boldsymbol{\nu} - \mathbf{q}\|} & \text{otherwise} \end{cases} \tag{11}$$

Note that $\mathbf{y}^*$ lies on the ray from the query through $\boldsymbol{\nu}$.

In general, however, we need to solve the problem

$$\begin{array}{ll} \text{max.} & \sum_{i=1}^m w_s \ln(\sigma(\tau_i)) - w_q \|\mathbf{x}(\tau_i) - \mathbf{q}\|^2 - w_\mu \|\mathbf{x}(\tau_i) - \boldsymbol{\mu}\|^2 + \\ & \sum_{i=m+1}^n w_s \ln(\sigma_i^{\max}) - w_q \|\mathbf{y}_i - \mathbf{q}\|^2 - w_\mu \|\mathbf{y}_i - \boldsymbol{\mu}\|^2 \\ \text{s.t.} & \|\mathbf{y}_i - \mathbf{q}\| \ge \delta_i, i \in \{m+1, \ldots, n\} \end{array} \tag{12}$$

which differs from (10) because the distances δ_i are not constrained to be equal. This may occur, for example, when the access kind enables retrieving a target number of tuples from each relation R_i. Although the problem is no longer symmetric in the optimization variables, the following holds.

THEOREM 3.4. *In the optimal solution of* (12) *all the* $\mathbf{y}_i^*$*'s,* $i = m+1, \ldots, n$, *are collinear and lie on the ray from the query through the centroid of the partial combination.*

By Theorem 3.4, we can reduce (12) to a 1-dimensional problem with n scalar variables θ_i, $i = 1, \ldots, n$, representing the distance from $\mathbf{q}$. The first m variables are constrained to be equal to the length of the orthogonal projection of $\mathbf{x}(\tau_i)$ onto the line defined by the query and the centroid of the seen tuples, i.e., $\theta_i = \mathcal{P}(\mathbf{x}(\tau_i))$, $i = 1, \ldots, m$, where

$$\mathcal{P}(\mathbf{x}(\tau_i)) = \frac{(\mathbf{x}(\tau_i) - \mathbf{q})^T(\boldsymbol{\nu} - \mathbf{q})}{\|\boldsymbol{\nu} - \mathbf{q}\|} \tag{13}$$

The remaining $n - m$ variables are lower bounded by the current distances from $\mathbf{q}$, i.e., $\theta_i \geq \delta_i$, $m{+}1, \ldots, n$. The upper bound $t(\tau)$ is obtained by solving the following problem

$$\begin{array}{ll} \text{max.} & \sum_{i=1}^{m} w_s \ln(\sigma(\tau_i)) + \sum_{i=m+1}^{n} w_s \ln(\sigma_i^{\max}) \\ & - \sum_{i=1}^{n} w_q \theta_i^2 - \sum_{i=1}^{n} w_\mu (\theta_i - \frac{1}{n}\sum_{j=1}^{n}\theta_j)^2 \\ \text{s.t.} & \theta_i = \mathcal{P}(\mathbf{x}(\tau_i)), \quad i = 1, \ldots, m \\ & \theta_i \geq \delta_i, \quad i = m+1, \ldots, n \end{array} \tag{14}$$

In Appendix B.4 we show that (14) can be written as a convex quadratic program (QP) with linear constraints, thus it can be efficiently solved using off-the-shelf solvers. Let $\boldsymbol{\theta}^* = [\theta_1^*, \ldots, \theta_n^*]^T$ denote the optimal solution of (14). The solution of the original problem (12) is given by

$$\mathbf{y}_i^* = \mathbf{q} + \theta_i^* \frac{\boldsymbol{\nu} - \mathbf{q}}{\|\boldsymbol{\nu} - \mathbf{q}\|}, \quad i = m+1, \ldots, n \tag{15}$$

i.e., the i-th variable is at distance θ_i^* from the query $\mathbf{q}$ and on the ray that originates from $\mathbf{q}$ and goes through $\boldsymbol{\nu}$.

EXAMPLE 3.2. Assume Table 1 reports all the seen tuples. Thus, $\delta_1 = 1$, $\delta_2 = 2\sqrt{2}$ and $\delta_3 = 2\sqrt{2}$. Solving (12) for the partial combination $\tau_2^{(1)}$ gives $\mathbf{y}_1^* = [\sqrt{2}/2, \sqrt{2}/2]^T$ and $\mathbf{y}_3^* = [2, 2]^T$ (and $t(\tau_2^{(1)}) = -12.8$), which lie along the ray from $\mathbf{q}$ to $\mathbf{x}(\tau_2^{(1)})$. Solving (12) for $\tau_1^{(1)} \times \tau_3^{(1)}$ requires: *i)* computing the centroid of $\tau_1^{(1)} \times \tau_3^{(1)}$ ($\boldsymbol{\nu} = [-0.5, 0.25]^T$); *ii)* computing the projections on the line from $\mathbf{q}$ to $\boldsymbol{\nu}$ ($\theta_1 = -0.22$, $\theta_3 = 1.34$); *iii)* solving (14) to obtain $\theta_2^* = 2\sqrt{2}$; *iv)* computing $\mathbf{y}_2^* = [-2.53, 1.26]^T$ according to (15); *v)* computing $t(\tau_1^{(1)} \times \tau_3^{(1)}) = -16$. Figure 1(b) shows that the optimal locations of the unseen tuples are at the minimum allowed distances, but this does not hold in general. ■

3.2.2 *Dominance*

The solution of problem (12) for a partial combination $\tau \in PC(M)$ depends on the current distance lower bounds δ_i, $i \in \{1, \ldots, n\} - M$, and needs to be re-computed whenever any of these lower bounds may change, i.e., after every access to R_i, $i \in \{1, \ldots, n\} - M$. Nevertheless, in order to determine t_M as of (8), it is unnecessary to solve (12) for all partial combinations $\tau \in PC(M)$. Indeed, some of them might be dominated, in the sense that, for a dominated partial combination τ, it will never happen that $t_M = t(\tau)$.

In order to determine whether a partial combination is dominated, we consider the unconstrained version of (12). Let $f^\alpha(\mathbf{y}) = -(a\mathbf{y}^T\mathbf{y} + 2[\mathbf{b}^\alpha]^T\mathbf{y} + c^\alpha)$ compactly represent the objective function of (12) for partial combination τ^α, where $\mathbf{y} = \mathbf{y}_{m+1} = \ldots = \mathbf{y}_n$ by the symmetry of the objective function (a, $\mathbf{b}^\alpha$, and c^α are computed as a, $\mathbf{b}$, and c in Appendix B.3). Let $\mathbf{y}^\alpha$ be the vector for which $f^\alpha(\mathbf{y})$ is maximum, i.e., the solution of this unconstrained problem for a partial combination $\tau^\alpha \in PC(M)$. For another partial combination $\tau^\beta \in PC(M)$, the set

$$\mathcal{D}(\tau^\alpha, \tau^\beta) = \{\mathbf{y} \in \mathbb{R}^d | f^\alpha(\mathbf{y}) \geq f^\beta(\mathbf{y})\} \tag{16}$$

intuitively represents the region of the feature space $\mathbb{R}^d$ where τ^α dominates τ^β. We observe that such a region is defined by $f^\alpha(\mathbf{y}) \geq f^\beta(\mathbf{y})$, i.e., $2(\mathbf{b}^\alpha - \mathbf{b}^\beta)^T\mathbf{y} \leq c^\beta - c^\alpha$, which is a half-space, since the quadratic terms disappear. The *dominance region* of $\tau^\alpha \in PC(M)$ is defined as

$$\mathcal{D}(\tau^\alpha) = \{\mathbf{y} \in \mathbb{R}^d | \forall \tau^\beta \in PC(M). f^\alpha(\mathbf{y}) \geq f^\beta(\mathbf{y})\} \tag{17}$$

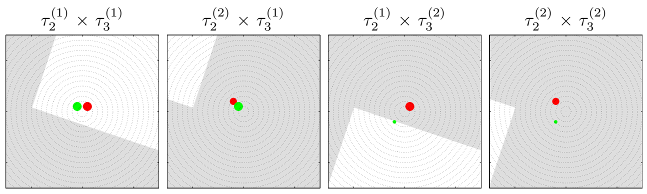


Figure 2: Dominance regions (in white) of the four partial combinations in $PC(\{2,3\})$ in Table 1.

We observe that dominance regions are polyhedra that define a partition of $\mathbb{R}^d$. If the dominance region $\mathcal{D}(\tau^\alpha)$ is empty, we say that τ^α is *dominated*. Emptiness of $\mathcal{D}(\tau^\alpha)$ can be checked by solving a feasibility linear program (LP), as shown in Appendix B.5. In practice, solving the LP might be too costly to be of practical interest. Indeed, we can either *i)* ignore dominance by solving the optimization problem (12) for each partial combination, i.e., up to $\prod_{i \in M} p_i$ times; or *ii)* determine the dominated partial combinations and solve (12) only for those that are not dominated. Note that, once a combination is dominated, it will remain so regardless of the newly formed combinations. In our experiments in Section 4, we show that a good heuristics consists of periodically solving the LP only after a fixed number of accesses.

EXAMPLE 3.3. Figure 2 shows the dominance regions of the four partial combinations $\tau \in PC(\{2, 3\})$ formable from Table 1. Here, no partial combination is dominated. ■

3.3 Pulling strategy

A pulling strategy identifies the relation R_i from which the next tuple is retrieved. We describe a *potential adaptive* (PA) strategy that is based on the current upper bounds t_M, defined for each set $M \subset \{1, \ldots, n\}$. We generalize the approach introduced in [7] to the case of $n \geq 2$ relations. Let pot_i denote the *potential* of relation R_i, defined as $pot_i = \max\{t_M | M \subset \{1, \ldots, n\} - \{i\}\}$, i.e., the upper bound on the aggregate score of combinations that can be formed with unseen tuples from R_i. The PA strategy is then defined as follows: access relation R_i such that pot_i is maximal (among $pot_1, \ldots, pot_n$), breaking ties in favor of the relation with the least depth p_i, then the relation with the least index i.

Let *TBPA* and *TBRR* denote algorithms complying with the *ProxRJ* template for distance-based access using the tight bound (6) and, resp., the PA and round-robin strategy.

THEOREM 3.5. *Let I be a proximity rank join problem. Then depth(TBPA, I, i) $\leq$ depth(TBRR, I, i) for all i.*

COROLLARY 3.6. *TBPA is instance optimal.*

4. EXPERIMENTAL STUDY

We investigate the following aspects: *i)* I/O cost reduction, in terms of the *sumDepths* metrics, that can be achieved using a tight bounding scheme and an adaptive pulling strategy, with respect to a simpler corner bound and a round-robin pulling strategy; *ii)* overhead, in terms of CPU time, due to the computation of a tight bound and potential savings that can be achieved when dominance is exploited;

Table 2: Operating parameters (defaults in bold).

full name	parameter	tested values
Number of results	K	1,**10**,50
Number of dimensions	d	1,**2**,4,8,16
Density	ρ	20,50,**100**,200
Skewness	ρ_1/ρ_2	**1**,2,4,8
Number of relations	n	**2**,3,4

iii) impact, on the aforementioned metrics, of the problem parameters, i.e., number of results, dimensionality of the feature space, tuple density in the feature space, skewness of the tuple density and number of relations.

4.1 Methodology

Data sets. First, we conduct our analysis on a synthetic data set whose generation is described in Appendix D.1. The relevant parameters are summarized in Table 2.

Then, we consider real data sets with relations containing hotels, theaters, and restaurants in five different American cities. In each such city, we run the example described in Section 1. Data are obtained as described in Appendix D.2.

Methods. We test four different instantiations of the *ProxRJ* template described in Algorithm 1. These are the result of combining the two bounding schemes (Corner Bound and Tight Bound) with the two pulling strategies (Round-Robin and Potential Adaptive). Therefore, we denote the tested algorithms as *CBRR*, *CBPA*, *TBRR*, and *TBPA*. We observe that *CBRR* and *CBPA* correspond to, resp., *HRJN* and *HRJN** [9].

Evaluation metrics. We adopt *sumDepths* as the primary metrics for comparing the different algorithms. This is especially relevant in application scenarios where the cost of fetching tuples from the relations largely dominates over computing the combinations and their respective bounds. This is the case, e.g., of search services invoked over the Web. We also report the *total CPU time*, in seconds, for the various experiments. We do not measure the time needed for fetching the tuples, as this is implicitly captured by the *sumDepths* metrics. Moreover, we indicate the fraction of time consumed by the calls to the function *updateBound* and, whenever applicable, by the evaluation of dominance. For fairness, we compute both metrics over ten different data sets and report the average.

4.2 Results

The results obtained on the synthetic data sets are summarized in Figure 3. In the stacked bar charts reporting the total CPU time, *i)* the darker bars at the bottom represent the cost of forming the combinations and computing their aggregate score. *ii)* the lighter bars on top represent the cost of executing *updateBound*. As a general trend, we observe that the use of a tight bounding always outperforms the simpler corner bound in terms of the *sumDepths* metrics by a noticeable margin, i.e., at least 15% in the worst case. This comes at the cost of an increased complexity when computing the bound. However, the average total CPU time in our experiments, for $n = 2$, is always less than 0.5 seconds. Thus, in a scenario where data is accessed by means of remote service invocations, the *total* CPU time is of the same order of magnitude as the time needed for a *single* access. Moreover, we note that in our experiments the tight bound is recomputed after every accessed tuple. While this guarantees to access the minimum number of tuples, in practical systems a good trade-off can be achieved by recomputing the tight bound only after retrieving blocks of tuples. We now comment on the effect of the individual parameters.

Number of results - K: As in conventional rank join problems, the number of results grows sublinearly with respect to the number of accessed tuples. The gain of *TBPA* over *CBPA* is between 25% and 45% and is larger for smaller values of K. The adaptive pulling strategy reduces the number of accessed tuples by 5-10%, although the two relations contain data sampled from identical distributions. *TBPA* requires approximately 4 times more CPU time than *CBPA*.

Number of dimensions - d: This is a characteristic parameter of the proximity rank join problem. The gain of *TBPA* over *CBPA* is between 15% and 45% and it is larger in higher dimensional spaces. Indeed, for the same density per volume unit, higher dimensional spaces are emptier, in the sense that the average inter-tuple distance is larger. This fact is neglected by the simpler corner bounding scheme adopted by *CBRR* and *CBPA* that always assume a zero distance from the centroid for unseen combinations. Note that the total CPU time scales favorably when d increases. This is due to two facts: first, the number of accessed tuples decreases as d increases, thus computing the tight bound needs to consider fewer partial combinations; second, for the same number of accessed tuples, the total CPU time does not depend on d, since computing the upper bound on the score of a partial combination requires solving a 1-dimensional problem, regardless of d.

Density - ρ (number of tuples per volume unit): *TBPA* and *CBPA* are similarly affected by the tuple density, and the gain of the former is always in the 20-30% range. The *sumDepths* metrics increases with the density. This is due to the higher probability of generating combinations with a large aggregate score by means of random sampling.

Skewness - ρ_1/ρ_2 (ratio of densities for two relations): Generating skewed data sets highlights the benefits of an adaptive pulling strategy. The gain over round-robin can be as large as 25-30% when $\rho_1/\rho_2 \geq 4$.

Number of relations - n: The gain of *TBPA* over *CBPA* is significantly larger when joining more than two relations, attaining more than 50% for $n = 3$. We observe that, in this case, *TBPA* outperforms *CBPA* in terms of both *sumDepths* and total CPU time. Indeed, the additional computational burden related to the tight bounding scheme is more than compensated by the significantly smaller number of combinations (approximately 4000 as opposed to 32000) that need to be formed. This observation is further confirmed for $n = 4$. In this case, *CBPA* was unable to report the top-10 results (within five minutes), due to the extremely large number of combinations that need to be formed.

Dominance: As discussed in Section 3.2.2, checking dominance can be costly. A good trade-off between cost and benefits is achieved by testing dominance only periodically. Figures 3(m) and 3(n) illustrate the total CPU time of computing the result when the dominance test is enabled. A dominance period equal to ∞ corresponds to disabling the dominance test. In each stacked bar chart, the lightest topmost bar represents the time needed to check dominance. For $n = 2$, we observe that testing dominance after every accessed tuple increases the total CPU time, despite the fact that the time needed to compute the tight bound is decreased. When the dominance period is between 8 and 16, a marginal 4% improvement is achieved. For $n = 3$ the

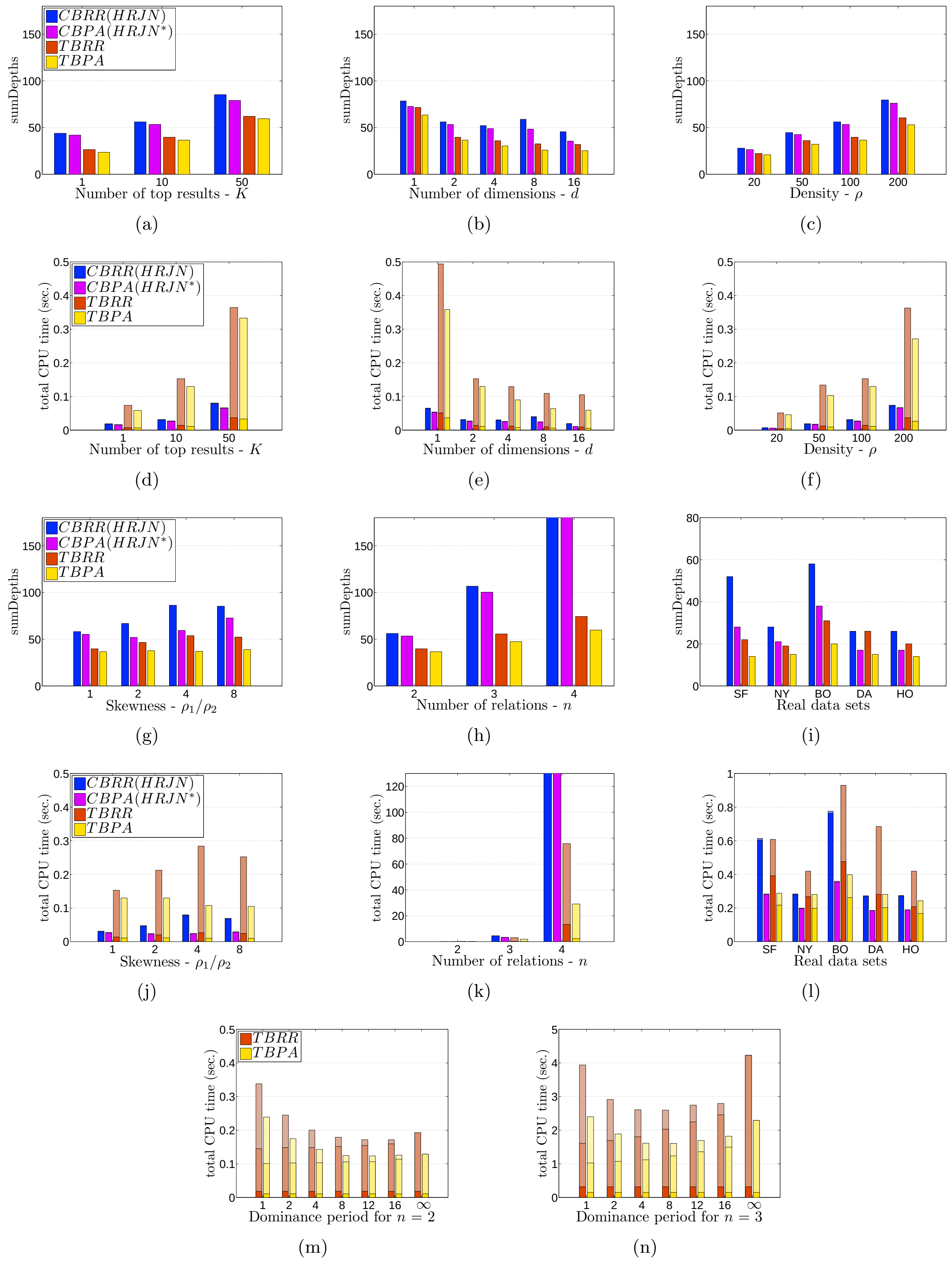


Figure 3: Performance of the tested algorithms in terms of the *sumDepths* and total CPU time metrics. In each figure one parameter is varied, while the others are set to the default values in Table 2. The stacked bars in (d),(e),(f),(j),(k),(l) show the total CPU time and, with lighter shading, the fraction of time required to compute the tight bound. The stacked bars in (m) and (n) show, for $n = 2$ and $n = 3$, the total CPU time when dominance is exploited and, with lighter shading, the fraction required to compute the tight bound (middle) and to compute the dominance (top).

dominance test is always beneficial; a significant 35% improvement is attained with a dominance period equal to 8.

Finally, Figures 3(i) and 3(l) report the results obtained on the real data sets. *TBPA* outperforms *CBPA* by a large margin (on average, 35%). The adaptive pulling strategy is always beneficial, regardless of the bounding scheme, reducing by 30% the number of accessed tuples.

5. RELATED WORK

Proximity rank join is a significant extension of rank join, a class of problems recently discussed in [9, 14, 13, 7, 10].

State-of-the-art rank join algorithms [9, 13, 7] inherit the idea of using an upper bound from the threshold-based stopping condition of the well-known *threshold algorithm* [6] (TA). TA addresses rank aggregation, which is the problem of combining several ranked lists into a single consensus ranking.

Our main starting point is [13], where the rank join *PBRJ* template is introduced, which we have specialized and extended into *ProxRJ* for proximity rank join. This template encompasses the well-known *HRJN* and *HRJN** operators [9], corresponding to our *CBRR* and *CBPA* algorithms. In [13], rank join is studied in the case where each relation may be equipped with even more than one score attribute, while we have focused on relations with a single score. As was discussed in Section 3, the results in [13] do not always carry over to proximity rank join, due to the geometry of the problem (absent in rank join). These include, e.g., the fact that *HRJN* is instance optimal with two relations (with a single score) for rank join but not for proximity rank join.

In [7], the authors propose an efficient way for computing an approximation of the tight bound, in order to find a good trade off between I/O cost and CPU cost.

In [2], the authors study (two-way) rank join problems, in which memory availability is limited, and propose algorithms that are instance-optimal also with this restriction.

We have considered a scenario in which the objects' feature vectors are deterministic. Others have addressed related problems (e.g., nearest neighbors) when the objects have uncertain features, such as approximate positions [3].

Weighted proximity join is introduced in [16], where algorithms are proposed to find combinations of words in documents that appear close to each other and match different query terms. Their approach differs from ours in the following aspects: *i)* the aggregation function does not depend on the distance from the query, and all the input needs to be read to find the best combinations; *ii)* the proposed algorithms are specifically tailored to 1-dimensional spaces. Similarity join is promoted to a first-class operator in [15].

The study of join predicates that depend on the spatial proximity among the objects has been thoroughly investigated in the past literature. The work in [8] presents incremental distance joins, e.g., how to compute the spatial join between two lists of objects and reporting the closest pairs early. A non-incremental algorithm for the K-closest-pair problem is investigated in [5]. This is extended in [12], where additional spatial constraints are imposed on the domain of the objects belonging to the two lists (e.g., all points of the first list that are part of the answer are enclosed in a given region). The top-k spatial join described in [11] is a binary join operator that returns, for each list, the object with the largest number of overlapping objects in the other list, thus extending the previous approaches to objects covering finite regions. Nevertheless, in all of the above-mentioned works, the authors assume that input relations are indexed by structures of the R-tree family. This contrasts our setting, where indexes are unavailable and relations are accessed either by distance or by score.

6. CONCLUSION

We introduced proximity rank-join as the problem of finding the best combinations of heterogeneous objects that are close to a given target object (the query) and to each other. We gave a general formulation of the problem and proposed an instance optimal algorithm that solves it based on the computation of a tight upper bound on the aggregate score of the combinations to be formed. Our solution proves superior to existing rank join approaches, also experimentally.

We plan to extend proximity rank join to the case of relations that can be accessed not only by sorted access but also by random access. Similarly to what we did for Euclidean distance, we also intend to specialize the tight bounding scheme to the case of proximity based on cosine similarity.

Acknowledgments. The authors acknowledge support from the Search Computing (SeCo) project, funded by the ERC.

7. REFERENCES

[1] YQL console. http://developer.yahoo.com/yql/console/.

[2] P. Agrawal and J. Widom. Confidence-aware join algorithms. In *ICDE*, pages 628–639, 2009.

[3] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1):326–339, 2008.

[4] M. Brambilla and S. Ceri, editors. *Search Computing - Challenges and Directions*, volume 5950 of *LNCS*. Springer, March 2010.

[5] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Algorithms for processing k-closest-pair queries in spatial databases. *Data Knowl. Eng.*, 49(1):67–104, 2004.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[7] J. Finger and N. Polyzotis. Robust and efficient algorithms for rank join evaluation. In *SIGMOD Conference*, pages 415–428, 2009.

[8] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD Conference*, pages 237–248, 1998.

[9] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3):207–221, 2004.

[10] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top- query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[11] N. Mamoulis, Y. Theodoridis, and D. Papadias. Spatial joins: Algorithms, cost models and optimization techniques. In *Spatial Databases*, pages 155–184. 2005.

[12] A. N. Papadopoulos, A. Nanopoulos, and Y. Manolopoulos. Processing distance join queries with constraints. *Comput. J.*, 49(3):281–296, 2006.

[13] K. Schnaitter and N. Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, pages 43–52, 2008.

[14] K. Schnaitter, J. Spiegel, and N. Polyzotis. Depth estimation for ranking query optimization. In *VLDB*, pages 902–913, 2007.

[15] Y. N. Silva, W. G. Aref, and M. H. Ali. The similarity join database operator. In *ICDE*, 2010.

[16] R. Thonangi, H. He, A. Doan, H. Wang, and J. Yang. Weighted proximity best-joins for information retrieval. In *ICDE*, pages 234–245, 2009.

APPENDIX

A. PROOFS

Proof of Theorem 3.1. Consider $\mathcal{S}$ as in (2) with $w_s = 0$, $w_q = 1$, $w_\mu = 1$, and $\mathbf{q} = \mathbf{0}$, and a problem $I = (R_1, R_2, \mathcal{S}, 1)$ with

$$\mathbf{x}(\tau_1^{(1)}) = [0, -0.5]^T \quad \mathbf{x}(\tau_2^{(1)}) = [0, 2]^T$$
$$\mathbf{x}(\tau_1^{(2)}) = [0, 1]^T \quad \mathbf{x}(\tau_2^{(2)}) = [-2, 2]^T$$

(Tuple scores are immaterial to $\mathcal{S}$ as $w_s = 0$.)

When $p_1 = 2$ and $p_2 = 1$, by reasoning on the geometry of the seen tuples, an algorithm may correctly return $\tau = \tau_1^{(2)} \times \tau_2^{(1)}$ as the top combination for I, since no formable combination can have a higher score than $\mathcal{S}(\tau) = -5.5$. The same conclusion can also be drawn by finding a tight bound as shown in Section 3.2.1.

Conversely, we show that, in order to terminate, A needs to explore R_1 until a depth that is a priori unbounded. Indeed, when $p_1 = p_2 = 2$, A would compute the upper bounds $t_1 = -5$, $t_2 = -8.25$, and thus $t_c = -5$. With this, τ cannot be guaranteed to be the top combination for I, as $t_c > \mathcal{S}(\tau)$. Since deepening on R_2 cannot lower t_c, we discuss how t_c varies as p_1 grows. Indeed, t_c remains above -5.5 until the first tuple $\tau_1^{(j)}$ in R_1 is seen whose distance from $\mathbf{q}$ is at least $\sqrt{5.5 - \|\mathbf{x}(\tau_2^{(1)}) - \mathbf{q}\|^2} = \sqrt{1.5}$. Since there can be an arbitrary number of tuples in R_1 between $\tau_1^{(2)}$ and $\tau_1^{(j)}$, the value of *depth*$(A, I, 1)$ upon termination of A is unbounded, hence the claim. □

Proof of Theorem 3.2. In order to verify Definition 2.2 for any given proximity rank join problem I and extracted portions of the relations, it suffices to exhibit a continuation of I in which a combination using at least one unseen tuple has a score that equals the bound given by (9). To do that, let $\mathbf{y}_i^*$, $i \in \{1, \ldots, n\} - M$, denote a solution of problem (6) for the partial combination $\tau \in PC(M)$ maximizing (8) with the set M maximizing (9). It suffices then to extend each relation R_i, $i \in \{1, \ldots, n\} - M$, with a tuple with score $\sigma_i^{\max}$ and feature vector $\mathbf{y}_i^*$. With this, *i)* the value of the aggregation function for the combination made by τ and the new tuples equals by construction the bound (9), and *ii)* the requirements posed by distance-based access are met by satisfaction of the constraints in (6). □

Proof of Theorem 3.3. Follows immediately from tightness of (9) and Theorem 5.1 of [13]. □

Proof of Theorem 3.4. W.l.o.g., let $\mathbf{q} = \mathbf{0}$. We prove the claim by contradiction. Assume that the optimal solution is such that $\mathbf{y}_{m+1}^*, \ldots, \mathbf{y}_n^*$ are not collinear. We show that there is a feasible solution of (12) that attains a larger value of the objective function, thus contradicting the statement that $\mathbf{y}_{m+1}^*, \ldots, \mathbf{y}_n^*$ is the optimal solution. The objective function of problem (12) can be rewritten as a function $f(\mathbf{y}_{m+1}, \ldots, \mathbf{y}_n)$ given by

$$\sum_{i=1}^{m} w_s \ln(\sigma(\tau_i)) - w_q \|\mathbf{x}_i\|^2 + \sum_{i=m+1}^{n} w_s \ln(\sigma_i^{\max}) - w_q \|\mathbf{y}_i\|^2 + \\ - w_\mu h(\mathbf{y}_{m+1}, \ldots, \mathbf{y}_n)$$

where we adopt the shorthand notation $\mathbf{x}_i = \mathbf{x}(\tau_i)$ and $h(\mathbf{y}_{m+1},\ldots,\mathbf{y}_n)$ is given by

$$\begin{aligned} & \textstyle\sum_{i=1}^{m} \|\mathbf{x}_i - \boldsymbol{\mu}\|^2 + \sum_{i=m+1}^{n} \|\mathbf{y}_i - \boldsymbol{\mu}\|^2 \\ = & \textstyle\sum_{i=1}^{m} \mathbf{x}_i^T \mathbf{x}_i + \sum_{i=m+1}^{n} \mathbf{y}_i^T \mathbf{y}_i - \frac{m^2}{n} \boldsymbol{\nu}^T \boldsymbol{\nu} + \\ & -\tfrac{1}{n} \big(\textstyle\sum_{j=m+1}^{n} \mathbf{y}_j\big)^T \big(\sum_{j=m+1}^{n} \mathbf{y}_j\big) - 2 \frac{m}{n} \boldsymbol{\nu}^T \big(\sum_{j=m+1}^{n} \mathbf{y}_j\big) \end{aligned}$$

Let $\mathbf{y}_{m+1}^*, \ldots, \mathbf{y}_n^*$ denote the optimal solution, consisting of non-collinear points. Let $\mathbf{y}'_{m+1}, \ldots, \mathbf{y}'_n$ denote a possible solution defined as follows:

$$\mathbf{y}'_i = \|\mathbf{y}_i^*\| \frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|} \quad \text{for } m+1 \le i \le n \tag{18}$$

Thus, $\mathbf{y}'_i$ has the same distance from the origin as $\mathbf{y}_i^*$ (and thus is feasible) but it lies along the direction of $\boldsymbol{\nu}$.

We can observe that

$$\boldsymbol{\nu}^T \left(\sum_{j=m+1}^{n} \mathbf{y}'_j \right) > \boldsymbol{\nu}^T \left(\sum_{j=m+1}^{n} \mathbf{y}_j^* \right) \text{ since } \sum_{j=m+1}^{n} \boldsymbol{\nu}^T \mathbf{y}'_j > \sum_{j=m+1}^{n} \boldsymbol{\nu}^T \mathbf{y}_j^*$$

where the strict inequality stems from the observation that the inner products $\boldsymbol{\nu}^T \mathbf{y}_j$ are maximized when $\mathbf{y}_j$ has the same direction as $\boldsymbol{\nu}$. Also

$$\left(\sum_{j=m+1}^{n} \mathbf{y}'_j \right)^T \left(\sum_{j=m+1}^{n} \mathbf{y}'_j \right) > \left(\sum_{j=m+1}^{n} \mathbf{y}_j^* \right)^T \left(\sum_{j=m+1}^{n} \mathbf{y}_j^* \right)$$

$$\text{since } \left\| \sum_{j=m+1}^{n} \mathbf{y}'_j \right\| > \left\| \sum_{j=m+1}^{n} \mathbf{y}_j^* \right\|$$

where the strict inequality is due to the fact that, given two sets of vectors having pairwise the same lengths (i.e., $\|\mathbf{y}'_i\| = \|\mathbf{y}_i^*\|$, for $m+1 \le i \le n$), the length of the sum is maximized when the vectors are parallel. Thus we have

$$h(\mathbf{y}_{m+1}^*, \ldots, \mathbf{y}_n^*) > h(\mathbf{y}'_{m+1}, \ldots, \mathbf{y}'_n) \tag{19}$$

$$f(\mathbf{y}_{m+1}^*, \ldots, \mathbf{y}_n^*) < f(\mathbf{y}'_{m+1}, \ldots, \mathbf{y}'_n) \tag{20}$$

which contradicts the statement that $\mathbf{y}_{m+1}^*, \ldots, \mathbf{y}_n^*$ is the optimal solution, hence the optimal solution is collinear. □

Proof of Theorem 3.5 (sketch). The proof of this theorem is very similar to the proof of Theorem 4.2 in [13]. There, the authors show that their potential adaptive version of the *PBRJ* template with a corner bound ($PBRJ_c^*$) always terminates with a depth less than or equal to the depth of the round robin execution ($PBRJ_c^{RR}$), for every input relation. Their argument seamlessly adapts to our case by replacing *i)* $PBRJ_c^*$ with *TBPA*, *ii)* $PBRJ_c^{RR}$ with *TBRR*, and *iii)* their upper bound $\bar{\mathcal{S}}(R_i(p_i))$ with the potential pot_i of relation R_i. The proof proceeds by contradiction, assuming an index k for which *depth*$(TBPA, I, k) >$ *depth*$(TBRR, I, k)$. Let $p_i^{RR} = depth(TBRR, I, i)$ and let p_i be the depth of R_i when *TBPA* decides to pull $R_k[p_k^{RR} + 1]$. Then, it can be shown that, for each i, either *TBPA* has seen all tuples from R_i seen by *TBRR* or no tuple past $R_i[p_i]$ can participate in the solution. Thus, *TBPA* has already formed all combinations found by *TBRR*. Also, the score of the last combination found by *TBRR* is at least pot_i, for every i, and $\max\{pot_1, \ldots, pot_n\}$ coincides with our tight bound. Therefore the termination condition of *TBPA* is met. Contradiction. □

Proof of Corollary 3.6. Trivial, by instance optimality of *TBRR* and Theorem 3.5. □

B. DISTANCE-BASED ACCESS

B.1 Updating the bound

Algorithm 2 provides the pseudocode that is executed for updating the value of the tight bound after each retrieved tuple τ_i. The loop at line 3 evaluates (9). The loop at line 5 evaluates (8) for all partial combinations $\tau' \in PC(M)$. Both the upper bound $t(\tau')$ and the dominance test need to be evaluated. When $M = \varnothing$, the set $PC(M)$ conventionally contains exactly one tuple (the so-called *empty tuple* $\langle\rangle$) and the algorithm may proceed as in all other cases, since the corresponding combinations are those formed using unseen tuples from all relations. Note that, when invoking *updateBound* after accessing a tuple $\tau_i \in R_i$, these operations need not be performed for all partial combinations. Indeed, the upper bound $t(\tau')$ is computed if: *i)* partial combination τ' has not been flagged as dominated (line 6); *ii)* either τ' is a newly formed partial combination that uses τ_i or τ' is a seen partial combination that can be completed with tuples from R_i (line 7). In all other cases, the value of $t(\tau')$ is not recomputed; instead, we reuse the previously computed value, that we store as a global variable.

The dominance test is executed, by checking emptiness of $\mathcal{D}(\tau')$ via (35), if: *i)* partial combination τ' has not been flagged as dominated (line 6); *ii)* τ' is a partial combination that uses seen tuples from R_i (line 10). Note that, as discussed in Section 3.2.2, the dominance test can also be performed only periodically, instead of after each access.

Algorithm 2: *updateBound*(τ_i) distance-based case

```
   Input  : last seen tuple $\tau_i = R_i[p_i]$; seen tuples $P_j$,
            $j = 1, \ldots, n$, curr. values of $t(\cdot)$ for all seen
            combinations
   Output: Tight upper bound $t$
 1 begin
 2     $t \leftarrow -\infty$;
 3     for $M \subset \{1, \ldots, n\}$ do
 4         $t_M \leftarrow -\infty$;
 5         for $\tau' \in PC(M)$ do
 6             if $\tau'$ is not dominated then
 7                 if $i \in M \wedge \tau'_i = \tau_i \vee i \notin M$ then
 8                     Compute $t(\tau')$ solving (6);
 9                 end
10                 if $i \in M$ then
11                     if $\mathcal{D}(\tau') = \varnothing$ then
12                         Flag $\tau'$ as dominated;
13                     end
14                 end
15                 $t_M \leftarrow \max\{t_M, t(\tau')\}$;
16             end
17         end
18         $t = \max\{t, t_M\}$;
19     end
20     return $t$
21 end
```

B.2 Partial combinations and their upper bounds

For the relations in Table 1, Table 3 shows the partial combinations that need to be examined, for each of the subsets M of $\{1, 2, 3\}$. For each partial combination τ, we also report the value of $t(\tau)$, which is obtained by solving (6) with the aggregation function (2), as was shown in Section 3.2.1. The values of t_M are computed as in (8).

Table 3: Partial combinations formed with the tuples of Table 1.

M	$\tau \in PC(M)$	$t(\tau)$	t_M
$\varnothing$	$\langle\rangle$	-19.2	-19.2
$\{1\}$	$\tau_1^{(1)}$	-20.6	-19.2
	$\tau_1^{(2)}$	-19.2	
$\{2\}$	$\tau_2^{(1)}$	-12.8	-12.8
	$\tau_2^{(2)}$	-19.4	
$\{3\}$	$\tau_3^{(1)}$	-12.8	-12.8
	$\tau_3^{(2)}$	-20.1	
$\{1, 2\}$	$\tau_1^{(1)} \times \tau_2^{(1)}$	-16.0	-13.5
	$\tau_1^{(1)} \times \tau_2^{(2)}$	-24.0	
	$\tau_1^{(2)} \times \tau_2^{(1)}$	-13.5	
	$\tau_1^{(2)} \times \tau_2^{(2)}$	-20.4	
$\{1, 3\}$	$\tau_1^{(1)} \times \tau_3^{(1)}$	-16.0	-13.5
	$\tau_1^{(1)} \times \tau_3^{(2)}$	-22.0	
	$\tau_1^{(2)} \times \tau_3^{(1)}$	-13.5	
	$\tau_1^{(2)} \times \tau_3^{(2)}$	-26.4	
$\{2, 3\}$	$\tau_2^{(1)} \times \tau_3^{(1)}$	-7.0	-7.0
	$\tau_2^{(1)} \times \tau_3^{(2)}$	-21.0	
	$\tau_2^{(2)} \times \tau_3^{(1)}$	-13.1	
	$\tau_2^{(2)} \times \tau_3^{(2)}$	-26.8	

B.3 Solution of (10)

We assume w.l.o.g. that $\mathbf{q} = \mathbf{0}$ or, equivalently, that all the feature vectors are referred to a coordinate system that has the origin in $\mathbf{q}$, i.e., $\mathbf{x}_i \leftarrow \mathbf{x}_i - \mathbf{q}$, for $i = 1, \ldots, m$, and we adopt the shorthand notation $\mathbf{x}_i = \mathbf{x}(\tau_i)$. The centroid $\boldsymbol{\mu}$ of the full combination can be written

$$\boldsymbol{\mu} = \frac{m}{n}\boldsymbol{\nu} + \frac{n-m}{n}\mathbf{y} \tag{21}$$

where $\mathbf{y} = \mathbf{y}_{m+1} = \cdots = \mathbf{y}_n$ due to the symmetry of the problem, and $\boldsymbol{\nu} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i$ is the centroid of the partial combination.

The objective function of (10) rewrites into

$$\begin{aligned}\text{min.}\quad & w_q(n-m)\mathbf{y}^T\mathbf{y} + w_\mu(n-m)\tfrac{m^2}{n^2}\mathbf{y}^T\mathbf{y} + \\ & w_\mu(n-m)\tfrac{m^2}{n^2}\boldsymbol{\nu}^T\boldsymbol{\nu} - 2w_\mu(n-m)\tfrac{m^2}{n^2}\boldsymbol{\nu}^T\mathbf{y} + \\ & w_\mu \textstyle\sum_{i=1}^{m}\big[(\mathbf{x}_i - \tfrac{m}{n}\boldsymbol{\nu})^T(\mathbf{x}_i - \tfrac{m}{n}\boldsymbol{\nu}) + \\ & \big(\tfrac{m-n}{n}\big)^2\mathbf{y}^T\mathbf{y} - 2\tfrac{n-m}{n}(\mathbf{x}_i - \tfrac{m}{n}\boldsymbol{\nu})^T\mathbf{y}\big] + s.\end{aligned} \tag{22}$$

where s is a scalar that does not depend on $\mathbf{y}$. The problem can thus be compactly represented as

$$\begin{aligned}\text{minimize}\quad & a\mathbf{y}^T\mathbf{y} + 2\mathbf{b}^T\mathbf{y} + c \\ \text{subject to}\quad & \|\mathbf{y}\| \geq \delta\end{aligned} \tag{23}$$

where

$$a = \left[w_q(n-m) + w_\mu\frac{m}{n}(n-m)\right] \tag{24}$$

$$\mathbf{b} = -w_\mu(n-m)\frac{m}{n}\boldsymbol{\nu} \tag{25}$$

$$c = w_\mu(n-m)\frac{m^2}{n^2}\boldsymbol{\nu}^T\boldsymbol{\nu} + w_\mu\sum_{i=1}^{m}\left(\mathbf{x}_i - \frac{m}{n}\boldsymbol{\nu}\right)^T\left(\mathbf{x}_i - \frac{m}{n}\boldsymbol{\nu}\right) + s \tag{26}$$

The solution of problem (23) is obtained by imposing the

Karush-Kuhn-Tucker conditions and it is given by

$$\mathbf{y}^* = \begin{cases} -\mathbf{b}/a & \text{if } \|\mathbf{b}/a\| \geq \delta \\ \delta \frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|} & \text{otherwise} \end{cases} \tag{27}$$

or, equivalently,

$$\mathbf{y}^* = \begin{cases} \boldsymbol{\nu} \frac{m w_\mu}{m w_\mu + n w_q} & \text{if } \|\boldsymbol{\nu} \frac{m w_\mu}{m w_\mu + n w_q}\| \geq \delta \\ \boldsymbol{\nu} \frac{\delta}{\|\boldsymbol{\nu}\|} & \text{otherwise} \end{cases} \tag{28}$$

Note that $\mathbf{y}^*$ has the same direction as the vector representing the centroid $\boldsymbol{\nu}$. If $\mathbf{q} \neq \mathbf{0}$, then $\mathbf{y}^* \leftarrow \mathbf{q} + \mathbf{y}^*$, thus:

$$\mathbf{y}^* = \begin{cases} \mathbf{q} + (\boldsymbol{\nu} - \mathbf{q}) \frac{m w_\mu}{m w_\mu + n w_q} & \text{if } \|(\boldsymbol{\nu} - \mathbf{q}) \frac{m w_\mu}{m w_\mu + n w_q}\| \geq \delta \\ \mathbf{q} + (\boldsymbol{\nu} - \mathbf{q}) \frac{\delta}{\|(\boldsymbol{\nu} - \mathbf{q})\|} & \text{otherwise} \end{cases} \tag{29}$$

B.4 Solution of (14)

Let $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]^T$ denote a vector representing the distances from $\mathbf{q}$ of the n vectors of the tuples that form a combination. Problem (14) can be written as a quadratic program (QP) in canonical form, as follows

$$\begin{array}{ll} \text{minimize} & \boldsymbol{\theta}^T \mathbf{H} \boldsymbol{\theta} \\ \text{subject to} & \mathbf{A}_{eq} \boldsymbol{\theta} = \mathbf{b}_{eq} \\ & \boldsymbol{\theta} \geq \boldsymbol{\ell} \end{array} \tag{30}$$

To see this, let $\mathbf{1} = [1, 1, \ldots, 1]^T \in \mathbb{R}^n$ and $\mathbf{I}_n$ be the $n \times n$ identity matrix. The objective function of (14) can be written in matrix form:

$$\begin{aligned} &\sum_{i=1}^{m} w_s \ln(\sigma(\tau_i)) + \sum_{i=m+1}^{n} w_s \ln(\sigma_i^{\max}) \\ &- \sum_{i=1}^{n} w_q \theta_i^2 - \sum_{i=1}^{n} w_\mu (\theta_i - \frac{1}{n} \sum_{j=1}^{n} \theta_j)^2 = \\ &= r - w_q \boldsymbol{\theta}^T \boldsymbol{\theta} - w_\mu (\boldsymbol{\theta} - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T \boldsymbol{\theta})^T (\boldsymbol{\theta} - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T \boldsymbol{\theta}) \\ &= r - w_q \boldsymbol{\theta}^T \boldsymbol{\theta} - w_\mu \boldsymbol{\theta}^T \left(\mathbf{I}_n - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T\right)^T \left(\mathbf{I}_n - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T\right) \boldsymbol{\theta} \end{aligned}$$

where r is an expression not containing $\boldsymbol{\theta}$. Therefore:

$$\mathbf{H} = w_q \mathbf{I} + w_\mu \left(\mathbf{I}_n - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T\right)^T \left(\mathbf{I}_n - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T\right) \tag{31}$$

From the equality constraints in (14), $\theta_i = \mathcal{P}(\mathbf{x}(\tau_i)), i = 1, \ldots, m$, it follows

$$\mathbf{A}_{eq} = \begin{bmatrix} \mathbf{I}_m & \mathbf{0}_{m \times (n-m)} \\ \mathbf{0}_{(n-m) \times m} & \mathbf{0}_{(n-m) \times (n-m)} \end{bmatrix} \tag{32}$$

$$\mathbf{b}_{eq} = [\mathcal{P}(\mathbf{x}(\tau_1)), \ldots, \mathcal{P}(\mathbf{x}(\tau_m)), \mathbf{0}_{1 \times (n-m)}]^T \tag{33}$$

Finally, from the inequality constraints in (14), $\theta_i \geq \delta_i$, $i = m+1, \ldots, n$, it follows

$$\boldsymbol{\ell} = [\mathbf{0}_{m \times 1}, \delta_{m+1}, \ldots, \delta_n]^T \tag{34}$$

B.5 Dominance

Let τ^α be the partial combination whose dominance is to be checked. We denote as $\{\tau^1, \ldots, \tau^u\} = PC(M) - \tau^\alpha$ the set of partial combinations excluding τ^α. In order to determine whether $\mathcal{D}(\tau^\alpha)$ is empty, we need to solve the following feasibility linear program (LP)

$$\begin{array}{ll} \text{minimize} & 1 \\ \text{subject to} & \begin{bmatrix} 2(\mathbf{b}^\alpha - \mathbf{b}^1)^T \\ \cdots \\ 2(\mathbf{b}^\alpha - \mathbf{b}^u)^T \end{bmatrix} \mathbf{y} \leq \begin{bmatrix} c^1 - c^\alpha \\ \cdots \\ c^u - c^\alpha \end{bmatrix} \end{array} \tag{35}$$

If (35) is unfeasible, we can conclude that $\mathcal{D}(\tau^\alpha)$ is empty, thus τ^α is dominated. We observe that solving (35) can be costly, since the number of constraints $u = \prod_{i \in M} p_i - 1$ grows quickly as more tuples are retrieved.

Yet, a speed-up can be obtained by discarding from the constraints of problem (35) those partial combinations in $PC(M)$ that have already been flagged as dominated.

C. SCORE-BASED ACCESS

Under score-based access, it is possible to compute a correct upper bound by keeping track of the scores $\sigma(R_i[1])$ and $\sigma(R_i[p_i])$ of the first and, respectively, last accessed tuples from each relation. The upper bound is given by

$$t_c^s = \max\{t_1^s, \ldots, t_n^s\}, \text{ with } t_i^s = f\left(\bar{\mathcal{S}}_1^s, \ldots, \underline{\mathcal{S}}_i^s, \ldots, \bar{\mathcal{S}}_n^s\right) \tag{36}$$

where $\bar{\mathcal{S}}_j^s$ is an upper bound on the proximity weighted score that can be attained by a tuple $\tau_j \in R_j$, i.e.,

$$\bar{\mathcal{S}}_j^s = g_j(\sigma(R_i[1]), 0, 0) \tag{37}$$

Similarly $\underline{\mathcal{S}}_i^s$ denotes an upper bound on the proximity weighted score that can be attained by an unseen tuple $\tau_i \in R_i - P_i$:

$$\underline{\mathcal{S}}_i^s = g_i(\sigma(R_i[p_i]), 0, 0) \tag{38}$$

Such a bound is a corner bound (like that of *HRJN* [9]), and it does not consider any geometric constraints of the problem at hand. As for the case of distance-based access, the bound is not tight and precludes instance optimality.

THEOREM C.1. *Let A be an algorithm complying with the ProxRJ template for score-based access using the corner bound* (36). *Then A is not instance optimal for problems with at least two relations.*

PROOF. Consider an aggregation function of form (2) on a one-dimensional space, with $w_s = w_q = w_\mu = 1$, $\mathbf{q} = \mathbf{0}$, and a problem $I = (R_1, R_2, \mathcal{S}, 1)$ with

$$\begin{array}{llll} \sigma(\tau_1^{(1)}) = 1 & \mathbf{x}(\tau_1^{(1)}) = [1] & \sigma(\tau_2^{(1)}) = 1 & \mathbf{x}(\tau_2^{(1)}) = [1] \\ \sigma(\tau_1^{(2)}) = e^{-5} & \mathbf{x}(\tau_1^{(2)}) = [0] & \sigma(\tau_2^{(2)}) = 1 & \mathbf{x}(\tau_2^{(2)}) = [1/3] \end{array}$$

The combination $\tau = \tau_1^{(1)} \times \tau_2^{(2)}$ has score $\mathcal{S}(\tau) = -4/3$, which is the highest possible for all seen and unseen combinations. However, when $p_1 = p_2 = 2$ the corner bound is $t_c^s = 0$, thus τ cannot be guaranteed to be top by t_c^s, as $t_c^s > \mathcal{S}(\tau)$. Deepening on R_1 cannot lower t_c^s. When p_2 grows, t_c^s remains above $\mathcal{S}(\tau)$ until the first tuple $\tau_2^{(j)}$ is seen with $\sigma(\tau_2^{(j)}) \leq e^{-4/3}$. The number of tuples between $\tau_2^{(2)}$ and $\tau_2^{(j)}$ in R_2 is arbitrary, hence the claim. □

Similarly to Section 3, let $\tau \in PC(M)$ denote a partial combination that can be formed using seen tuples from R_i, $i \in M$. A tight upper bound $t^s(\tau)$ is obtained by solving the following unconstrained optimization problem in the variables $\mathbf{y}_i \in \mathbb{R}^d$, $i \in \{1, \ldots, n\} - M$

$$\text{maximize} \quad f(\mathcal{S}_1^s, \ldots, \mathcal{S}_n^s), \text{ where}$$
$$\mathcal{S}_i^s = \begin{cases} g_i(\sigma(\tau_i), \delta(\mathbf{x}(\tau_i), \mathbf{q}), \delta(\mathbf{x}(\tau_i), \boldsymbol{\mu})), & i \in M \\ g_i(\sigma(R_i[p_i]), \delta(\mathbf{y}_i, \mathbf{q}), \delta(\mathbf{y}_i, \boldsymbol{\mu})), & i \in \{1, \ldots, n\} - M \end{cases} \tag{39}$$

A tight bound for the score-based case is then given by

$$t^s = \max\{t_M^s | M \subset \{1, \ldots, n\}\}, t_M^s = \max\{t^s(\tau) | \tau \in PC(M)\} \tag{40}$$

This suffices to obtain instance optimality. The proofs of these claims are analogous to those of Theorems 3.2 and 3.3.

THEOREM C.2. *The bounding scheme* (40) *is tight.*

THEOREM C.3. *ProxRJ with the tight bounding scheme* (40) *and a round-robin pulling strategy is instance-optimal for proximity rank join with score-based access.*

Unlike the case of distance-based access, for each set M, we need to keep track of only one partial combination τ^M_{best} that dominates all the others. In other words, if, at a given state of execution, $t^s(\tau^\alpha) \geq t^s(\tau^\beta)$, for two given partial combinations τ^α and τ^β, then the same inequality holds when additional tuples are retrieved from R_i, $i \in \{1, \ldots, n\} - M$. Retrieving additional tuples from R_i, $i \in M$ creates new partial combinations. Updating the tight bound when a new tuple is retrieved can be done incrementally, as it suffices to solve (39) for each new partial combination and to keep track of the partial combination with the highest upper bound.

C.1 Updating the bound

Algorithm 3 provides the pseudocode that is executed for updating the value of the tight bound after each retrieved tuple τ_i for the case of score-based access. The algorithm proceeds as Algorithm 2, with two main difference: *i)* the upper bound of a partial combination is computed according to (39), and *ii)* partial combinations that do not exceed the current value of t^s_M can be immediately flagged as dominated, and will remain so. Note that, for each $M \subset \{1, \ldots, n\}$, we keep track of the partial combination τ^M_{best} with the currently highest upper bound, each stored as a global variable. At line 10, if the upper bound of the current partial combination τ' exceeds t^s_M, we flag the previous τ^M_{best} (if defined) as dominated (line 11), we update t^s_M (line 12), and we set τ' as the new τ^M_{best} (line 13).

Algorithm 3: *updateBound*(τ_i) score-based case

```
   Input  : last seen tuple $\tau_i = R_i[p_i]$; seen tuples $P_j$,
            $j = 1, \ldots, n$, curr. values of $t^s(\cdot)$ for all seen
            combinations, $\tau^M_{best}$ for every $M \subset \{1, \ldots, n\}$
   Output: Tight upper bound $t^s$
1  begin
2      $t^s \leftarrow -\infty$;
3      for $M \subset \{1, \ldots, n\}$ do
4          $t^s_M \leftarrow -\infty$;
5          for $\tau' \in PC(M)$ do
6              if $\tau'$ is not dominated then
7                  if $i \in M \wedge \tau'_i = \tau_i \vee i \notin M$ then
8                      Compute $t^s(\tau')$ solving (39);
9                  end
10                 if $t^s(\tau') > t^s_M$ then
11                     Flag $\tau^M_{best}$ as dominated;
12                     $t^s_M \leftarrow t^s(\tau')$;
13                     $\tau^M_{best} \leftarrow \tau'$;
14                 else
15                     Flag $\tau'$ as dominated;
16                 end
17             end
18         end
19     end
20     return $t^s$
21 end
```

C.2 Tight bound for Euclidean distance

Consider an aggregation function of the form given in (2). W.l.o.g. let us assume $M = \{1, \ldots, m\}$ and a partial combination $\tau \in PC(M)$. The (partial) upper bound is obtained by solving the following problem

$$\max. \quad \sum_{i=1}^{m} w_s\sigma(\tau_i) - w_q\|\mathbf{x}(\tau_i) - \mathbf{q}\|^2 - w_\mu\|\mathbf{x}(\tau_i) - \boldsymbol{\mu}\|^2 + \sum_{i=m+1}^{n} w_s\sigma(R[p_i]) - w_q\|\mathbf{y}_i - \mathbf{q}\|^2 - w_\mu\|\mathbf{y}_i - \boldsymbol{\mu}\|^2$$

The problem is of the same kind as (10), but without constraints. Therefore, the optimal solution is given by

$$\mathbf{y}^*_{m+1} = \cdots = \mathbf{y}^*_n = \mathbf{q} + (\boldsymbol{\nu} - \mathbf{q})\frac{mw_\mu}{mw_\mu + nw_q} \qquad (41)$$

i.e., the upper bound on the score of unseen combinations is achieved by completing the partial combination τ with tuples from R_i, $i = m+1, \ldots, n$ such that: *i)* the unseen tuples are located along the ray that originates from the query $\mathbf{q}$ and goes through the centroid $\boldsymbol{\nu}$ of the partial combination; *ii)* the score is equal to the score of the last seen tuple of R_i.

D. EXPERIMENTS SETUP

D.1 Synthetic data sets

We generated the synthetic data sets used in our experiments as follows. For each of the relations R_i, $i = 1, \ldots, n$, we generate a number of tuples. Each tuple τ_i is assigned a random score $\sigma(\tau_i)$, sampled from a uniform distribution, and a feature vector $\mathbf{x}(\tau_i)$. The feature vectors are obtained by sampling a d-dimensional uniform distribution centered in $\mathbf{0}$ a number of times so as to obtain the desired average density ρ expressed in terms of tuples per volume unit. When testing $n = 2$, we change the skewness parameter ρ_1/ρ_2 in such a way as to generate relations with different densities.

We emphasize that the size of the data sets is not a relevant parameter in our study. Indeed, solving the proximity rank join problem for a target number of results K calls for retrieving only a prefix of the relations.

D.2 Real data sets

The real data sets contain information needed to answer the type of query introduced in Section 1. Each data set is obtained by retrieving customer ratings, latitude and longitude (thus $d = 2$) of entertainment places in five different destinations (San Francisco, New York, Boston, Dallas and Honolulu) by means of the YQL console available at [1]. These data sets are used to feed $n = 3$ Web services endowed with distance-based access returning, respectively, hotels, restaurants and cinemas. The query vector $\mathbf{q}$ is represented by a specific location within these cities (e.g. Fishermans Wharf in San Francisco, Battery Park in New York, etc.). For each query, we retrieve the top-10 combinations.

D.3 Testing environment

All the algorithms have been executed on a PC with the Windows 7 operating system, an Intel® Core2-Duo processor at 2.4GHz and 4Gb RAM. We remark that the *sumDepth* metrics is completely oblivious of the testing environment. As for the CPU time, we measured the wall clock execution time needed to return the top-K combinations, assuming that tuples of the joined relations are available locally. Thus, we did not consider the time needed for fetching the tuples when data is available from remote sources.