

Reasoning About Approximate Match Query Results

Sudipto Guha
University of Pennsylvania
sudipto@cis.upenn.edu

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

Divesh Srivastava
AT&T Labs Research
divesh@research.att.com

Xiaohui Yu
University of Toronto
xhyu@cs.toronto.edu

Abstract

Join techniques deploying approximate match predicates are fundamental data cleaning operations. A variety of predicates have been utilized to quantify approximate match in such operations and some have been embedded in a declarative data cleaning framework. These techniques return pairs of tuples from both relations, tagged with a score, signifying the degree of similarity between the tuples in the pair according to the specific approximate match predicate.

In this paper we consider the problem of estimating various parameters on the output of declarative approximate join algorithms for planning purposes. Such algorithms are highly time consuming, so precise knowledge of the result size as well as its score distribution is a pressing concern. This knowledge aids decisions as to which operations are more promising for identifying highly similar tuples which is a key operation for data cleaning. We propose solution strategies that fully comply with a declarative framework and analytically reason about the quality of the estimates we obtain as well as the performance of our strategies.

We present the results of a detailed performance evaluation of all strategies proposed. Our experimental results, validate our analytical expectations and shed additional light to the quality and performance of our estimation framework. Our study offers a set of simple, fully declarative techniques for this problem, which are readily deployed in the SPIDER declarative data cleaning system.

1 Introduction

Data quality is a serious concern in every large production database. Poor quality data is the result of a variety of causes including incorrectly entered data (e.g., typing mistakes), lack of standards for recording database fields (e.g., addresses), poor integrity constraints (e.g., foreign key dependencies that are not maintained, and degrade over time), incorrect use of the database (e.g., additionally storing contact names in an address

field), and unanticipated evolution of the database over time (e.g., data for new applications and customer types stored in existing fields). Database administrators and users typically do not have sophisticated tools to carefully monitor and address data quality issues, as these large production databases evolve over time. Unfortunately, even simple data quality problems can severely degrade common business practices, e.g., inability to retrieve customer records during service calls, wasted inventory when unprovisioned circuits are not reflected in the database, etc.

In order to understand the structure and contents of a database (or a federation of related databases), it is necessary to understand how fields/attributes relate to one another. When these fields have poor quality data, join techniques using approximate match predicates are fundamental. These techniques return pairs of tuples from the tables, each pair tagged with a score, signifying the degree of similarity between the tuples in the pair according to the specific approximate match predicate. Such approximate join operations have received much research attention in recent years, due to their significance and practical importance (see, e.g., [16, 11, 9]).

Before such techniques can be effectively used against large production databases, containing hundreds of tables and thousands of columns, it is essential to determine which pairs of fields are worth joining, i.e., which joins using the approximate match predicates are likely to return many similar tuple pairs. Poor data quality, along with sparse and out-of-date documentation, make the available meta-data largely inadequate for this purpose. This observation has led to the development of sophisticated tools like Bellman [5], which uses database profiling to collect concise summaries of the values of the database fields. These summaries (set and multiset signatures based on min hash sampling and min hash counts) allow Bellman to determine whether two fields can be equi-joined, and if so the direction of the equi-join (e.g., one to many, many to many) and the size of the equi-join result. While Bellman's summaries and estimation techniques are very useful for equi-joins (such as between approximate keys/foreign keys, in the presence of

default values), they do not address our problem of determining which joins using the approximate match predicates are likely to return many similar tuple pairs.¹

The problem is challenging not only because of the large number of approximate join operations, but also because each approximate join operation is expensive (much more than traditional equi-joins). In this paper, we consider this problem as one of estimating (with quality guarantees) various parameters (such as result size and its score distribution) of the output of approximate join techniques, and propose solution strategies that can be declaratively expressed in SQL. Consider a relation R to be joined with two relations S_1 and S_2 . We adopt the notation $R \tilde{\bowtie}_t S$ to denote the approximate join of R and S for all pairs with scores above some threshold t . Assume that a pair of tuples in an approximate join result is further considered for examination if the pair has a score (for a specific approximate match predicate the join employs) above a threshold t . Let C_1 and C_2 be the sizes of $R \tilde{\bowtie}_t S_1$ and $R \tilde{\bowtie}_t S_2$ respectively. If $C_1 \gg C_2$ arguably $R \tilde{\bowtie}_t S_1$ is more important than $R \tilde{\bowtie}_t S_2$ and we should invest the time and resources to compute it. Choosing the threshold t however, will not always be an easy task. Its choice is commonly heavily dependent on the relations involved in the join. For example very few pairs might exist for a specific choice of t , but decreasing the value of t just a little might yield a significantly higher number of results. Thus, in such cases, it is important to track the distribution of the number of join results with specific scores, since it could aid more informed decisions. Let $D_1(t), D_2(t)$ be the distributions of the number of join results with specific scores for $R \tilde{\bowtie}_t S_1$ and $R \tilde{\bowtie}_t S_2$ respectively. If, for example, $D_1(t)$ has a higher skew towards larger scores than $D_2(t)$ and/or the specific number of join results with high scores in $D_1(t)$ is larger than those in $D_2(t)$ arguably $R \tilde{\bowtie}_t S_1$ should be computed first. The specific properties (e.g., skew) to use in order to compare $D_1(t)$ against $D_2(t)$ is an orthogonal choice (e.g., standard techniques to compare distributions can also be deployed). Notice that in this case specifying a threshold t is a much easier task. As the score under consideration decreases, the number of pairs at or close to that score increases significantly [11, 9]. For all practical purposes, the number of reported pairs is very large for further consideration at low scores. What’s more important is that the number of distinct pairs, reported as candidate duplicates at low scores, increases significantly dominating the total number. Specifying the threshold t to be a small value (e.g., 0.5) assures that we can focus on the relevant part of the distribution since pairs below t can hardly be considered as duplicates.

There exist many ways to track the result size distribution above a threshold. One example is to create a histogram on the number of pairs in the join result with scores within specific score ranges. Assuming that scores are normalized in $[0, 1]$, for a threshold t , a histogram on the number of pairs in the join

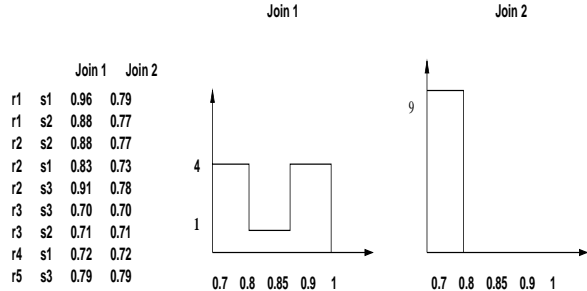


Figure 1. Example Histograms

result with scores in $[t, 1]$ will provide insight on the number of pairs with specific scores above the threshold t . Typically histograms have a specific number of buckets; the width of each bucket corresponds to a range of scores in this case. The size of the histogram (in terms of number of buckets) as well as the bucket allocation (e.g., equiwidth) is totally user dependent and completely tunable on demand. Figure 1 presents an example. A join result with two different score distributions is depicted (Figure 1). Constructing histograms as a representation of these distributions in the range $[0.7, 1]$ we get the distributions in Figure 1. The first distribution of Figure 1 is skewed towards larger scores and of higher interest as pairs with larger scores are more likely to be duplicates.

The core problem we address in this paper, is to produce a random sample of the approximate join result of two relations for tuple pairs with scores above a user defined threshold t . Doing so, will allow us to estimate the number of tuples in the approximate join result with scores above the threshold t as well as to reason about the number of join results with scores $t', t' \geq t$. This will allow us to obtain estimates for the number of approximate join results at specific score ranges and effectively approximate the distribution of the number of join results at specific scores above t . In this paper we introduce sampling based techniques for this purpose. Our emphasis is on approximate match operations (and corresponding predicates) which are *fully expressible* in a declarative way (SQL). Thus, all of our solutions are fully expressible in a declarative way as well and we provide the SQL statements to efficiently realize them. This deems our approach immediately useful in a practical setting. In fact, the approach described herein is fully utilized to identify promising joins to execute in the SPIDER declarative data cleaning system [16]. We introduce the COMPLETE JOIN problem that estimates the number of results for $R_1 \tilde{\bowtie}_t R_2$ (number of pairs above the threshold t for the approximate join of two relations R_1, R_2). Estimation is performed obtaining a random sample of $R_1 \tilde{\bowtie}_t R_2$ by sampling directly from R_1, R_2 and without assessing the full $R_1 \tilde{\bowtie}_t R_2$. This paper is organized as follows: In section 2 we review related work. Section 3 reviews background material necessary for the remainder of the paper and formally defines the problems of interest to this study. In section 4 we present two sampling strategies, namely

¹Bellman’s substring resemblance techniques determine textual similarity between entire columns, which is a different problem.

TUPLES and TOKENS for problem COMPLETE JOIN; we also detail a biased sampling strategy for the same problem. In Section 5 we present results of an experimental evaluation comparing all proposals across various parameters of interest. In section 6 we highlight the applicability of our solutions to general approximate match predicates. Section 7 concludes this paper.

2 Related Work

Non declarative techniques for approximate joins have a long research history (see [17] for general references and a survey). In their seminal work, Fellegi and Sunter introduced the record linkage problem [6]. Several techniques based on statistics followed (see [4] for a survey). More recent techniques include the merge/purge technique [14] and algorithms inspired by information retrieval [3]. Approximate join techniques based on statistics make use of well established inference techniques to quantify approximate match. Other approximate join techniques use well established algorithms to reason about approximate match. Such techniques include, cosine similarity, edit distance and variants thereof [12, 10, 11, 9, 1]. Declarative techniques for approximate joins have been proposed for a variety of approximate match predicates including cosine similarity and edit distance [12, 11, 7, 15, 1].

Estimating the result size of join operations for the case of equality predicates (i.e., equijoin) were studied in the context of query optimization. Ganguly et. al., [8] introduced sample based techniques for equi-join result size estimation and Chaudhuri et. al., [2] generalized this problem by proposing techniques to obtain a sample of the distribution of the equijoin result. The work presented herein is fundamentally different. The algorithms in [8, 2] treat equijoin predicates and make use of attribute value distributions to guide the sampling procedure. In our case, the predicates are far more complex and diverse (approximate match as opposed to equality) and the attribute value distributions themselves are not directly useful. Our work however is related to the works in [8, 2] since we are also interested in estimating the join result size and track the distribution of the join results, in a sufficiently more general setting however.

3 Background and Problem Definition

In this work we focus on declarative techniques for approximate joins. Several techniques have been proposed in the literature, including techniques to embed cosine similarity and edit distance in a declarative approximate match framework. We choose to drive the presentation of our techniques focusing our attention on cosine similarity as an approximate match predicate. In section 6 we highlight the applicability of our techniques in conjunction with other approximate match predicates.

We briefly review the declarative technique for approximate match introduced in [12, 15] which will form the basis for the presentation of our techniques. Given two relations

```

SELECT      r1w.tid AS tid1, r2w.tid AS tid2
            SUM(r1w.weight*r2w.weight) AS score
FROM        R1Weights r1w, R2Weights r2w
WHERE       r1w.token = r2w.token
GROUP BY   r1w.tid, r2w.tid
HAVING      SUM(r1w.weight*r2w.weight) >= t

```

Figure 2. Approximate Join in SQL (Generating F)

$R_1(tid, A_1, \dots)$ and $R_2(tid, B_1 \dots)$ (where tid is the tuple identifier) the basic idea behind this technique is to decompose each tuple in each relation into a number of *tokens*. What constitutes a token is an orthogonal choice; a token can be any subsequence of q characters (a qgram) in an attribute (or subset of attributes) of R_i , a word, etc. Each token is assigned a score based on the well known and widely adopted measure of *term frequency inverse document frequency* ($tf.idf$) from Information Retrieval. Let D denote the set of all tokens in R_1, R_2 . Consider the j -th token w in D and a tuple t from relation R_i . Then, tf_w is the number of times w appears in t . Also, idf_w is $\frac{|R_i|}{n_w}$, where n_w is the total number of tuples in R_i that contain w . The $tf.idf$ weight for the j -th token w in tuple $t \in R_i$ is $u_t(j) = tf_w \log(idf_w)$. Thus, each tuple t corresponds to a (sparse) vector of dimensionality D with each coordinate corresponding to a token in t equal to the $tf.idf$ weight of the token. These vectors are normalized to unit length to ease our computations. As described in [12, 15] token generation and $tf.idf$ score computation can be performed in a declarative way. Under this measure, the score of a pair of tuples $t_1 \in R_1$ and $t_2 \in R_2$ is defined as follows: let u_{t_1}, u_{t_2} be the corresponding normalized weight vectors; the *cosine similarity* score of u_{t_1}, u_{t_2} is defined as $sim(u_{t_1}, u_{t_2}) = \sum_{j=1}^{|D|} u_{t_1}(j)u_{t_2}(j)$. The values of $sim(\cdot)$ are in $[0, 1]$. The approximate join of R_1, R_2 is defined as all pairs of tuples $(t_1, t_2) \in R_1 \times R_2$ such that $sim(u_{t_1}, u_{t_2}) \geq t$ for a user specified threshold t . Assume that tables $R_iWeights(tid, token, weight)$ for $i = 1, 2$ have been computed, where tid is a tuple identifier in R_i , $token$ is a specific token in the tuple of R_i with identifier tid and $weight$ the numeric value of the token for the $tf.idf$ measure. The approximate join operation can be easily expressed in SQL as shown in Figure 2.

Our interest is to construct approximations of this join result, without computing the entire result, but only a small fraction of it. We will employ sampling as our main tool to achieve our goal.

Problem 3.1 (Complete Join (CJ)) Let F denote the number of tuples in the join result of base relations R_1, R_2 with scores above $t \in [0, 1]$. Return an approximation F' to F by obtaining a uniform random sample of $R_1 \bowtie_t R_2$.

In order to solve this basic problem, we will obtain a random sample of the approximate join result tuples, with scores above

the threshold t and we will quantify the accuracy of our estimations analytically. Using this technique, will be able to obtain estimates for the number of pairs in the approximate join result, with scores t' greater than or equal to any t . As a result, we will effectively construct an image of the distribution of the number of approximate join results with specific scores in $[t, 1]$. Such an image can be easily constructed. As an example, assume we adopt histograms as a non parametric way to represent the desired distribution. Histograms consist of a number of buckets B ; buckets span specific (non-overlapping) ranges of scores and collectively cover the range $[t, 1]$. We refer to the specific ranges spanned by buckets as a *bucketization*. Both the number of buckets and the bucketization are entirely user specified and tunable. Let \mathcal{F} be the distribution of the number of pairs in the join result for a specific bucketization and number of buckets. For example, for $t = 0.7$ with three buckets and a equi-width bucketization, \mathcal{F} will consist of the exact number of tuple pairs in the join result with scores in $([0.7, 0.8), [0.8, 0.9), [0.9, 1])$. A solution to problem 3.1 will allow us to approximate \mathcal{F} . The exact image of \mathcal{F} can be obtained in a variety of ways, for example by executing a number of `count` queries on the result of the join (obtained in Figure 2), counting the number of pairs within each bucket range of interest. This image can serve as a basis to compare our approximate solutions against.

We detail our solutions in the sequel. All proofs are omitted due to space limitations.

4 Algorithms for COMPLETE JOIN

In this section we present our proposed techniques for the solution of problem 3.1. Given two relations R_1, R_2 we wish to obtain a random sample of $R_1 \tilde{\bowtie}_t R_2$ in order to derive an estimate of the number of pairs in $R_1 \tilde{\bowtie}_t R_2$, that is the result size of the query in Figure 2. We assume that relations $RiWeights, i = 1, 2$ are available and we will be operating directly on those. If they are not, they can be constructed on demand; this is a constant overhead, which is the same for all techniques under consideration. We present two candidate strategies: one is based on sampling tuples (TUPLES) and the other on sampling tokens (TOKENS). We analyze and compare them in terms of accuracy and performance identifying their relative strengths. In case statistics are available for relations $RiWeights$ (we detail the type of statistics in the sequel) we show that better estimates can be obtained for problem 3.1 under some conditions. We complement our treatment of problem 3.1 in the case of available statistics, by describing biased sampling techniques for the problem at hand.

4.1 The case for no Available Statistics

4.1.1 Strategy TUPLES

Our first strategy obtains a uniform random sample, $RiWeights$ ($i = 1, 2$), from each of the relations. Assume, without loss of generality that $|RiWeights| = N$. For a specific value s each tuple in each relation is sampled with probability $\frac{s}{N}$ and two new relations $SRiWeights, i = 1, 2$ are cre-

```

CREATE VIEW SR1Weights (tid, token, weight) AS
SELECT *
FROM R1Weights
WHERE rand() <  $\frac{s}{N}$ 

CREATE VIEW SR2Weights (tid, token, weight) AS
SELECT *
FROM R2Weights
WHERE rand() <  $\frac{s}{N}$ 

SELECT r1s.tid, r2w.tid
SUM(r1s.weight*r2w.weight) AS score
FROM SR1Weights AS r1s, R2Weights AS r2w
WHERE r1s.token = r2w.token
GROUP BY r1s.tid, r2w.tid
HAVING SUM(r1s.weight*r2w.weight) >= t
UNION
SELECT r1w.tid, r2s.tid
SUM(r1w.weight*r2s.weight) AS score
FROM R1Weights AS r1w, SR2Weights AS r2s
WHERE r1w.token = r2s.token
GROUP BY r1w.tid, r2s.tid
HAVING SUM(r1w.weight*r2s.weight) >= t

```

Figure 3. Strategy TUPLES: SR1Weights, SR2Weights are random samples of R1Weights, R2Weights

ated. We claim $SR1Weights \tilde{\bowtie}_t SR2Weights \cup R1Weights \tilde{\bowtie}_t SR2Weights$ as a random sample of the approximate join of Figure 2 from which an approximation to the size of $R_1 \tilde{\bowtie}_t R_2$ can be obtained. This way of obtaining the estimate can be realized in a declarative way as shown in Figure 3. Obtaining a random sample from a relation can be performed in SQL in a variety of ways. The statements shown in Figure 3 assume that $rand()$ is a function that can return a random number in $[0, 1]$ for each tuple considered. SQL extensions exist for supporting various types of random samples directly. Different vendors implement various sampling techniques. We adopt this way in the sequel and we assume that random samples of relations can be materialized via one of possible ways, without delving further into the details which are orthogonal to our discussion.

Proposition 1 *The query of Figure 3 provides a random sample of $R_1 \tilde{\bowtie}_t R_2$.*

Operating on this sample, we can execute `count` queries and obtain estimates of the size of $R_1 \tilde{\bowtie}_t R_2$. We reason about the properties of such estimates.

Let $t_i \in R2Weights$; we toss a coin and with probability $\frac{s}{N}$, for some s , we include it in the sample. Let r_i be the number of tuples of $R1Weights$ that join with t_i having score greater than or equal to t . Let x_i be the contribution of t_i to the size of $R_1 \tilde{\bowtie}_t R_2$. Thus $x_i = r_i$ with probability $\frac{s}{N}$ and is 0 otherwise. Therefore, the expectation of x_i is $E[x_i] = \frac{sr_i}{N}$.

Proposition 2 Let $X = \frac{N}{s} \sum_{i=1}^s x_i$, then $E[X]$ is the size of $R_1 \tilde{\bowtie}_t R_2$.

Thus, in expectation our estimate for the size of $R_1 \tilde{\bowtie}_t R_2$ is exact. For the variance of our estimate we have:

Theorem 1 $Var(X) = (\frac{N}{s} - 1) \sum_{i=1}^s r_i^2$

Since each r_i is at most N , it follows that $\sum_{i=1}^s r_i^2 \leq N \sum_{i=1}^s r_i$. So the variance is bounded by $\frac{N^2}{s} \sum_{i=1}^s r_i$. Following Chebyshev inequality, a reasonable value for s is $O(\frac{N^2}{\sum_{i=1}^s r_i})$. Depending on the datasets involved in the approximate join, $\sum_{i=1}^s r_i$ is as low as N and as large as N^2 . As a result, a small value of s is required when each tuple in one data set has a relatively large number of tuples in the other with which it pairs up with a score above the threshold t . If that is not true, larger values of s are required. We will quantify such tradeoffs in section 5

In practice we do not expect the value of r_i to be large; for most cases, especially for large threshold t the value of r_i will be bounded by some constant u , denoting the upper bound on the number of tuples of one relation pairing up with any tuple of the other with a score above the threshold t . This value will typically be a few hundred tuples. Thus the above use of Chebyshev inequality need not be a very crisp estimate of the sample size. However the r_i 's not being the same present a problem in further sharpening the estimate. In fact if we assume that the r_i 's were identically distributed, we can get a better bound by the following:

Proposition 3 If x_i are i.i.d. random variables which take value u with probability s/N or 0, for any $0 < f, \lambda < 1$ a sample size of $O\left(\frac{2 \log \frac{1}{\lambda}}{(1-f)^2}\right)$ will guarantee that our estimate $\frac{N}{s} \sum_{i=1}^s x_i$ is within a factor $1 - f$ of the true value with probability at least $1 - \lambda$.

The above actually says that the sample size can be much smaller, (than N/u derived by the Chebyshev inequality) in an ideal setting where each tuple contributes the same amount.

In principle, by repeating this process for various t' we can obtain precise estimates for the size of any $R_1 \tilde{\bowtie}_{t'} R_2$, $t' \geq t$. Each such estimate will have guaranteed accuracy provided that is derived in accordance to proposition 3. From these we can construct a succinct image of the distribution of the number of pairs with scores above t at the expense of applying strategy TUPLES multiple times. However, operating on the result of the query in Figure 3 we can obtain estimates for the size of $R_1 \tilde{\bowtie}_{t'} R_2$ for any $t' \geq t$ as well. Such estimates for $t' > t$ will not have the guarantees of proposition 3 but we will evaluate them experimentally in section 5.

4.1.2 Strategy TOKENS

The second strategy we consider for problem 3.1 is to sample tokens directly from relations $RiWeights$, $i = 1, 2$. Relations $RiWeights$ materialize large, sparse vectors since each

tuple has been transformed to a vector of dimension $D = \cup \pi_{token}(RiWeights)$, $i = 1, 2$. Each relation materializes the non zero coordinates (tokens with non zero weight) of the tuples in each corresponding relation R_i . Conceptually each relation $RiWeights$ corresponds to an $N \times D$ matrix, M_i . The rows of each matrix $u_{t_j}^i$, $i = 1, 2$ and $j \in [1, N]$ have $\|u_{t_j}^i\| = 1$. Under this interpretation consider the matrix $C_{i,j} = 1$ if $u_{t_i}^1 (u_{t_j}^2)^T \geq t$, $i, j \in [1, N]$ and 0 otherwise. Estimating the number of entries with value 1 in matrix C will provide a solution to problem 3.1; let this number be c . Obtain a random sample of size ℓ from D . This effectively creates matrices M'_1, M'_2 of size at most $N \times \ell$. Assume that the rows of M'_i , $i = 1, 2$ are normalized to unit length. Let v_{t_i} denote a row of M'_1 and v_{t_j} a row of M'_2 . We define $X_{i,j} = 1$ if $v_{t_i} v_{t_j}^T \geq t$ and 0 otherwise. These are the entries of matrix $M'_1 \times (M'_2)^T$. It is evident that $E[X_{i,j}] = \frac{c}{N^2}$. Therefore $\hat{X} = \sum_{i \in [1, N], j \in [1, N]} X_{i,j}$ has an expectation $E[\hat{X}] = c$.

Let $\tau \in D$ be a token. Without loss of generality, assume τ belongs to a set of tuples from R_1 . Let r_i denote the number of tuple pairs in $R_1 \tilde{\bowtie}_t R_2$ that a *single tuple* in that set contributes to. For the variance of this sampling scheme we can show the following

Theorem 2 The variance of the \hat{X} estimate, $Var(\hat{X})$ is lower bounded by $(\frac{D}{\ell} - 1) \sum_{i=1}^{\ell} r_i^2$ and upper bounded by $(D - \ell) (\sum_{i=1}^N r_i)^2$.

It is evident that in this case, large values of ℓ with respect to D are required to keep the variance of this estimate manageable. This is not very promising in terms of performance however. It will take $O(N^2 D)$ in the worst case to produce the product of M_1 and M_2 and compute the exact value of c . Using matrices M'_1 and M'_2 it will take time $O(N^2 \ell)$ in the worst case, which does not offer performance advantages as ℓ needs to be close to D to obtain a good variance.

We observe however, that for the case of cosine similarity of two vectors u_{t_i}, u_{t_j} , with $\|u_{t_i}\| = \|u_{t_j}\| = 1$ their cosine similarity, determines their Euclidean distance exactly: $\|u_{t_i} - u_{t_j}\|^2 = \|u_{t_i}\|^2 + \|u_{t_j}\|^2 - 2u_{t_i} u_{t_j} = 2(1 - u_{t_i} u_{t_j})$. Since, we desire $u_{t_i} u_{t_j} \geq t$, this corresponds to $\|u_{t_i} - u_{t_j}\| \leq \sqrt{2(1-t)}$. Given this relationship between cosine similarity and Euclidean distance, one could try to reduce the dimensionality of the vectors in M'_1, M'_2 before computing $M'_1 \times (M'_2)^T$. If the reduction preserves relative distances in Euclidean space, then we can still obtain a good estimate for c , hopefully in a fraction of the time required to compute $M'_1 \times (M'_2)^T$. In accordance with our declarative framework, we deploy a dimensionality reduction technique that can be efficiently expressed in a declarative way.

Lemma 4.1 (JL Lemma) Given a set of N vectors V in a space R^n , if we have a matrix $S \in R^{d \times n}$ where $d = O(\frac{1}{\epsilon^2} \log N)$ such that each element of S_{ij} is drawn from a Gaussian distribution, appropriately scaled, for any vector

$x \in V$, then $\|x\|_2 \leq \|Sx\|_2 \leq (1 + \epsilon)\|x\|_2$ holds true with vanishingly high probability, $1 - o(1/N)$.

Assume that S is an $\ell \times d$ matrix suitably populated in accordance to the JL lemma. Reducing the dimension of each of the $M'_i, i = 1, 2$ matrices amounts to computing $M'_i \times S$. This will create the $N \times d$ matrices M''_i . We can now use $M''_1 \times (M''_2)^T$ in order to obtain an estimate for c . Each row of $M''_i, i = 1, 2$ is a vector of dimension d . If $\hat{u}_{t_i}, \hat{u}_{t_j}$ are vectors in R^d , the JL lemma guarantees that:

$$\text{if } \|u_{t_i} - u_{t_j}\| \geq (1 + 2\epsilon)\sqrt{2(1-t)} \text{ then}$$

with high probability $\|\hat{u}_{t_i} - \hat{u}_{t_j}\| \geq (1 + \epsilon)\sqrt{2(1-t)}$

and similarly

$$\text{if } \|u_{t_i} - u_{t_j}\| \leq (1 + 2\epsilon)\sqrt{2(1-t)} \text{ then}$$

with high probability $\|\hat{u}_{t_i} - \hat{u}_{t_j}\| \leq (1 + \epsilon)\sqrt{2(1-t)}$

The dimensionality reduction step will provide a performance benefit, provided that the time to compute $M'_1 \times (M'_2)^T$ ($O(N^2\ell)$) is greater than the time to perform the dimensionality reduction and compute $M''_1 \times (M''_2)^T$ ($O(N\ell d) + O(N^2d)$); clearly there is range for potential performance benefit, as $d \ll \ell$, and ℓ is close to D in order to achieve estimates with small variance.

Introducing dimensionality reduction using the JL lemma, 'distorts' the distances. As a result, for a specific value of ϵ in accordance to the JL lemma, and a tuple t_i instead of estimating all tuples with score above t in the joining relation, we relax this to estimating all pairs with score above $\frac{t}{1+\epsilon}$. For a tuple t_i , let r_i denote all tuples with score above t in the joining relations. Let r'_i denote an upper bound on the number of tuples in the joining relation with score above $\frac{t}{1+\epsilon}$. We have:

Theorem 3 *The variance of the TOKENS strategy with dimensionality reduction is lower bounded by $(\frac{D}{\ell} - 1) \sum_{i=1}^N r_i^2$ and upper bounded by $D(\sum_{i=1}^N r'_i)^2 + 2\ell(\sum_{i=1}^N r'_i)^2 - 3\ell(\sum_{i=1}^N r_i)^2$*

Figure 4 presents SQL statements implementing strategy TOKENS. We assume that relations *RiWeights* have been sampled and the relations *RiTSample* have been created. Moreover the weights corresponding to each *tid* in relations *RiTSample* have been normalized to unit length. Tables *RandomProji(token, dim, rnd)* have been created and populated with samples from a Gaussian distribution in accordance to the JL lemma. *dim* is an identifier for the corresponding dimension and ranges from 1 to d . The statements in Figure 4 seem to suggest that tables *RandomProji* should be materialized at every execution of algorithm TOKENS. We choose to present the statements this way for simplicity. However, in practise, tables *RandomProji* can be materialized without explicitly materializing actual tokens in attribute *token*; instead just a generic *token identifier* is sufficient. Then, by deriving a mapping between actual tokens for each execution of

```
CREATE VIEW R1rp(tid,dim,wrp) AS
SELECT r1Ts.tid, rp.dim,
SUM(r1w.weight*rp.rnd)
FROM R1TSample AS r1Ts, RandomProj1 AS rp
R1Weights AS r1w
WHERE r1Ts.token = rp.token
AND r1Ts.tid = r1w.tid
GROUP BY r1Ts.tid, rp.dim

CREATE VIEW R2rp(tid, dim, wrp) AS
SELECT r2Ts.tid, rp.dim,
SUM(r2w.weight*rp.rnd)
FROM R2TSample AS r2Ts, RandomProj2 AS rp
R2Weights AS r2w
WHERE r2Ts.token = rp.token
AND r2Ts.tid = r2w.tid
GROUP BY r2Ts.tid, rp.dim

SELECT r1w.tid, r2w.tid,
SUM((r1w.wrp-r2w.wrp)*(r1w.wrp-r2w.wrp))
FROM R1rp AS r1w, R2rp AS r2w
WHERE r1w.dim = r2w.dim
GROUP BY r1w.tid, r2w.tid
```

Figure 4. Strategy TOKENS: Tables *RandomProji* are populated with samples from a Gaussian Distribution suitably scaled. Tables *RiTSample* contain token samples from *RiWeights*. The weights of the tokens corresponding to each *tid* in *RiTSample* are normalized to unit length.

algorithm TOKENS and those identifiers, can assure that tables *RandomProji* can be repeatedly utilized across multiple algorithm executions. Such a mapping can be easily derived in a declarative way as well. We omit the details for brevity.

4.1.3 Comparing TOKENS and TUPLES

Both strategies obtain estimates for the quantities of interest of problem 3.1. As such, they should be compared in terms of their accuracy, but also in terms of their run-time performance. We present an analytical reasoning and further validate it experimentally in section 5. Sampling in strategies TUPLES and TOKENS concerns different quantities. Let L denote the size of a tuple (in suitable units of storage) in the worst case. Similarly, N (number of tuples in the relation) is the worst case size of the information associated with a token, in suitable units of storage as well. Thus, the bounds for the variance of TOKENS, assuming that we sample the same amount of information (in terms of units of storage) in both strategies become $O((\frac{DN}{sL} - 1) \sum_{i=1}^{\ell} r_i^2)$ and $O((D - \frac{sL}{N}) \sum_{i=1}^N r_i^2)$ for the lower and upper bound respectively. It is evident that the variance of TUPLES decreases as the worst case size of the tuples increases and the other parameters remain the same. Thus, we would expect this strategy to yield reasonable accuracy for relations containing tuples of larger size. Similar observations

hold for the case of TOKENS with dimensionality reduction.

In terms of run-time performance, strategy TUPLES can obtain an estimate in time $O(sNL)$ in the worst case (since the number of tokens is related to the tuple length). Strategy TOKENS requires time $O(Nld) + O(N^2d)$ in the worst case (that’s including the dimensionality reduction overhead). From these, one may conclude that for a specific parameter setting, when L becomes larger, while the other parameters remain unchanged, strategy TOKENS can provide some performance benefits as well.

4.2 The case of Available Statistics

We now focus on a variant of problem 3.1 by considering a setting in which statistical information is available for relations R_i . In traditional joins, the attribute value frequency distribution captured by histograms, can be useful when considering estimations of equijoin result sizes [8, 2]. In the case of approximate joins however, attribute value frequency distributions are not very useful. The predicates are sufficiently more complex and equality of values does not offer an estimate of the number of joining pairs (as in the case of equi-joins). In such operations we need to capture different types of statistics from the underlying relations. Moreover, to assure generality, as in the case of histograms in traditional query optimization, such statistics should be captured independently per relation and utilized in subsequent approximate join operations with a variety of other relations.

We propose to record for each tuple in each relation R_i the set of tuples from R_i that join with it in an approximate way (approximate self join), with score greater than or equal to a threshold t . Thus, for each tuple in R_i we are recording the *neighborhood* in cosine similarity space, for a threshold t . This information is maintained (in principle) for each tuple, by assessing the self join of each of the relations R_i . It can be computed and materialized once (independently of any other relation) in a preprocessing step (via simple SQL expressions as in [12]) and used subsequently for *any* approximate join involving R_i . This can be perceived as the analog to materializing histograms as is the case in traditional query optimization. Similarly to *end biased histograms* [18] we may choose to materialize such information, only for the tuples with the highest such numbers, resulting in tunable space requirements. We next detail our techniques for the problem at hand, assuming the existence of such information.

4.3 Samples for Single count Queries

For each relation R_i we create a relation $NeighborR_i(tid, tidN)$ that records for each tuple of R_i with tid , $tidN$, the tuples from R_i with tid ’s $tidN$ that join with it with score above t , for some t . We use w_i to denote the number of tuples of R_i that join with a tuple t_i of R_i with score above t .

We detail the use of such information for a slight variation of problem 3.1. Provided that the specific value of t is fixed a priori, we present a technique that obtains samples with the

aid of such statistics providing an estimate for the number of pairs in the approximate join with scores above $4 * t - 3$. The main difference with the original statement of problem 3.1 is that as a result of this technique we only obtain a single count estimate without obtaining a uniform random sample of the approximate join result. Since the technique biases the samples towards a specific value of t , it has the promise of increased accuracy and it is particularly useful for obtaining precise estimates for high values of t .

Our technique deploys *bifocal* sampling [8] suitably extending it to the case of approximate match predicates. We partition tuples of a relation into two categories, namely those that are of high density (HD) and those that are of low density (LD). As before, assume relations $R_iWeights$ are of the same size N . A tuple $t_j \in R_iWeights$, $1 \leq j \leq N$ belongs to HD if $w_j \geq \delta$ (for some δ) for the tuple with the corresponding tid in R_i and is LD otherwise. The sampling strategy derives two sets of random samples. The first one of size s_1 is on the HD tuples of $R_iWeights$; each HD tuple is sampled with probability $\frac{s_1}{N_h}$, where N_h is the number of HD tuples in relations $R_iWeights$ (without loss of generality assume that both relations have the same number of HD tuples). The second one is of size s_2 and is obtained on the LD tuples of relations $R_iWeights$ where each tuple is sampled with probability $\frac{s_2}{N_l}$ (number of LD tuples in $R_iWeights$).

We obtain s_1 samples from each of relations $R_iWeights$ by sampling HD tuples from each relation, effectively creating relations $SR_iWeights$. We compute $SR1Weights \tilde{\bowtie}_t SR2Weights$ and for each tuple pair (t_i, t_j) in this join result, we would like to use w_i, w_j in our estimation. Notice that by the triangle inequality, since each tuple t_i joins with a score at least t , with all tuples that contribute to w_i and also with the tuple t_j , all tuples that contribute to w_i join with a score at least $4 * t - 3^2$ with t_j . A symmetric argument holds for t_j . Assume that for each tuple t_i in $SR_iWeights$ none of the tuples contributing to the neighborhood w_i of t_i are in $SR_iWeights$. In this case we say that the neighborhood of the samples are *well separated*. In this case, accumulating $\sum_{i,j} w_i * w_j$, for all pairs $(t_i, t_j) \in SR1Weights \tilde{\bowtie}_t SR2Weights$ is a good estimate. It is likely however that the neighborhoods of some samples are not well separated. In this case accumulating $\sum_{i,j} w_i * w_j$ will be an overestimate. Consider tuples t_k in $SR_iWeights$ belonging to the neighborhood of some tuple t_i which also belongs to $SR_iWeights$. Computing $\sum_{i,j} w_i * w_j$ for all pairs (t_i, t_j) in $SR1Weights \tilde{\bowtie}_t SR2Weights$ could account for multiple contributions of some tuples and could be a crude overestimate. Figure 5 presents an example. It is imperative to eliminate this possibility before obtaining such an estimate. Let E_1 be

²Consider t_i, t_j at distance at least t and a t_k in the neighborhood of t_i . We have $\|u_{t_k} - u_{t_j}\| \leq \|u_{t_i} - u_{t_j}\| + \|u_{t_i} - u_{t_k}\|$. Using, $\|u_{t_i} - u_{t_j}\| = \sqrt{2(1 - u_{t_i}u_{t_j})}$ and $\|u_{t_i} - u_{t_j}\| \geq t$, $\|u_{t_i} - u_{t_k}\| \geq t$ we get $u_{t_k}u_{t_j} \geq 4 * t - 3$

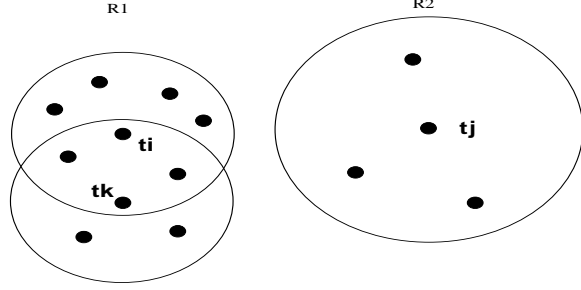


Figure 5. Neighborhoods of tuples t_i and t_k . If t_i has a score of at least t with respect to t_j then t_k is accounted for in the neighborhood of t_i . However, if t_k is also in the sample and has a score at least t with respect to t_j , it will be accounted again as well as all tuples in the overlap of t_k 's and t_i 's neighborhoods

the size of $\pi_{NeighborR1.tidN, NeighborR2.tidN} (NeighborR1 \bowtie SR1Weights \bowtie_t SR2Weights \bowtie NeighborR2)$. E_1 accounts for the true contribution of the neighborhoods of each tuple pair in $SR1Weights \bowtie_t SR2Weights$ since multiple contributions from tuples in the various neighborhoods have been eliminated. Let $Res1(tid) = \pi_{tidN}(SR1Weights \bowtie NeighborR1)$. Let s'_1 be the size of $Res1$. Let $Res2(tid) = \pi_{tidN}(SR2Weights \bowtie NeighborR2)$ and s''_1 be it's size. This result is further scaled by $\frac{N_h}{s'_1} \frac{N_h}{s''_1}$ and the estimate $\hat{E}_1 = \frac{N_h}{s'_1} \frac{N_h}{s''_1} E_1$ is obtained.

Then, we proceed to obtain estimates of the contribution to the final result, from the joining combinations of tuples in which LD tuples of one relation join with LD or HD tuples of the other. For this reason starting from $R1Weights$ we join the sample of size s_2 obtained from it sampling only LD tuples ($S'R1Weights$), with the entire relation $R2Weights$. Similarly, we have to eliminate possible overestimates caused by tuples belonging not only to the neighborhood of a tuple t_i but also to $S'R1Weights$. Thus, we compute the size E_{12} of $\pi_{NeighborR1.tidN, R2Weights.tid} (S'R1Weights \bowtie_t R2Weights \bowtie NeighborR1)$. E_{12} accounts for the correct contribution of the neighborhoods for each tuple pair in the final estimate by eliminating multiple contributions. Let $s'_2 = \pi_{tidN}(S'R1Weights \bowtie NeighborR1)$; the value E_{12} is subsequently scaled returning $\hat{E}_{12} = (\frac{N_h}{s'_2}) E_{12}$. We repeat this procedure symmetrically for $R2Weights$ and obtain the estimate \hat{E}_{21} . Our final result for the number of pairs $R1Weights, R2Weights$ joining with a score at least $4 * t - 3$ is the sum of these three estimates, namely $\hat{E}_1 + \hat{E}_{12} + \hat{E}_{21}$. Figure 6 presents SQL statements realizing this strategy.

In a setting, where each tuple has some number $u = O(\delta)$ of neighbors within the threshold, and we have a well separated instance, we can show that a small sample size is enough.

```

CREATE VIEW V (tid1, tid2) AS
SELECT srlw.tid, sr2w.tid
FROM SR1Weights AS srlw, SR2Weights AS sr2w,
WHERE srlw.token = sr2w.token
GROUP BY srlw.tid, sr2w.tid
HAVING SUM(srlw.weight*sr2w.weight) >= t

CREATE VIEW E1(tid1, tid2) AS
SELECT tid1, tid2
FROM V
UNION
SELECT N1.tidN, N2.tidN
FROM V, NeighborN1 AS N1, NeighborN2 AS N2
WHERE V.tid1 = N1.tid, V.tid2 = N2.tid

CREATE VIEW V1 (tid1, tid2) AS
SELECT s'rlw.tid, r2w.tid
FROM S'R1Weights AS s'rlw, R2Weights AS r2w,
WHERE s'rlw.token = r2w.token
GROUP BY s'rlw.tid, r2w.tid
HAVING SUM(s'rlw.weight*r2w.weight) >= t

CREATE VIEW E12 (tid1, tid2) AS
SELECT tid1, tid2
FROM V1
UNION
SELECT N1.tidN, V1.tid2
FROM V1, NeighborN1 AS N1
WHERE V1.tid1 = N1.tid

CREATE VIEW V2 (tid1, tid2) AS
SELECT rlw.tid, s'r2w.tid
FROM R1Weights AS rlw, S'R2Weights AS s'r2w,
WHERE rlw.token = s'r2w.token
GROUP BY rlw.tid, s'r2w.tid
HAVING SUM(rlw.weight*s'r2w.weight) >= t

CREATE VIEW E21 (tid1, tid2) AS
SELECT tid1, tid2
FROM V2
UNION
SELECT V2.tid1, N2.tidN
FROM V2, NeighborN2 AS N2
WHERE V2.tid2 = N2.tid

```

Figure 6. Statements realizing biased sampling strategy. Relations $SRiWeights$, are populated with samples of size s_1 from the HD tuples of corresponding $RiWeights$. Relations $S'RiWeights$ are populated with samples of size s_2 from the LD tuples of corresponding $RiWeights$. Counting distinct tuples from $E1, E12, E21$, appropriately scaling each estimate, and summing them up, provides the final estimate on the size of $R1 \bowtie_{4*t-3} R2$

Proposition 4 *If all the neighborhoods have the same size u and are well separated and we have an uniform sample of the neighborhoods, a sample size of $O\left(\frac{2\log\frac{1}{\lambda}}{(1-f)^2}\right)$ is sufficient to ensure that our estimate is within $(1-f)$ factor of the true value with probability at least $1-\lambda$. If we sample tuples and eliminate duplicates, a sample size of $O\left(\frac{2u\log\frac{1}{\lambda}}{(1-f)^2}\right)$ samples is sufficient to ensure an uniform sample of neighborhoods of the required size mentioned above.*

Applying similar methodologies, one can derive related expressions for $\hat{E}_{12}, \hat{E}_{21}$ as well as, the sample size s_2 for the case of the other estimates when the neighborhoods of the samples are well separated.

5 Experimental Evaluation

We prototyped the strategies proposed herein and in this section we report the results of an experimental evaluation comparing them in terms of accuracy and performance. All techniques are easy to realize on top of any RDBMS. We use SQL Server Beta 2, running on a DELL Precision 4, 3 GHZ with 4GB of RAM and two 250GB SATA drives. SQL Server is used with the parameters it ships with.

We used both real and synthetic data sets for our study. Our real data sets consist of customer name data. We report results using a pair of real data in our experiments which we refer to as `Real1`. All of our experiments are repeated *thirty times* (30) each and averages are reported. For this reason, to keep running time manageable during our experiments we constraint the size of `Real1` to 30K tuples (each). Average length of a tuple in `Real1` is 25 characters. In order to be able to vary data set parameters in a flexible way, we also include synthetic data sets in our study. Our synthetic data sets have varying sizes (both in terms of number of tuples and tuple length). They consist of uniformly distributed strings.

With our experiments we wish to identify the tradeoffs between the various strategies, for data sets with varying characteristics and seek to validate our analytical expectations. We report on the performance of the techniques by measuring response time of the various queries implementing our strategies. Response time does not include pre processing (i.e., time to construct relations *RiWeights* since it is a constant overhead for all techniques and time to collect statistics in the biased strategies since it is a constant overhead as well. Both are amortized across multiple uses). For strategies TUPLESS, TOKENS response time includes the time to sample tables *RiWeights*, the time for dimensionality reduction when applicable, and the time to join and construct the final result (counting the answer and scaling it). Accuracy is quantified in terms of the *average absolute relative error* (ARE) defined as $\frac{|E-E'|}{E}$, where E and E' are the exact and the estimated values for the quantity of interest. Samples are specified as a fraction of the total relation size (total number of tokens in strategy TOKENS).

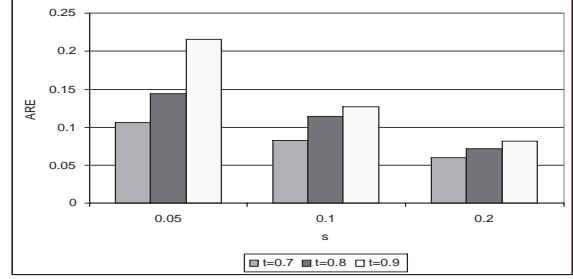


Figure 7. Accuracy of TUPLES on synthetic dataset, $N=20K$ tuples, average tuple size 50

Our first experiment aims to quantify the accuracy and the performance of strategy TUPLES. Figure 8 shows the accuracy and runtime performance of the TUPLES strategy on `Real1`. Figure 8(a) presents computation time as the size of the underlying data set increases. It is evident that TUPLES requires only a fraction of the time required to obtain the estimate exactly. Evidently, the benefits increase as the size of the underlying data set increases. These performance curves were consistent across a large collection of real and synthetic data sets; we omit additional graphs on performance due to space constraints. Figure 8(b), presents the accuracy of our estimates. For different sample sizes, we obtain estimates on the total number of pairs above a specific threshold. These estimates are obtained by deriving a uniform random sample of the join result using TUPLES for the smallest value of t (0.7 in figure 8(b)) and then using this uniform sample to obtain estimates for the remaining thresholds. It is evident that accuracy improves as the sample rate (s denotes sample rate in this context, i.e., fraction of tuples sampled) increases. Accuracy is usually better for smaller values of t and this is partly due to the fact that the uniform random sample is derived with respect to the lower threshold. As t increases, our expectation for the maximum number of strings u with score above t for any string, becomes smaller (this is verified by an analysis of the properties of data sets `Real1`). In section 4.1 we argued that a sample of roughly $O\left(\frac{N}{u}\right)$ is required to obtain good accuracy, attesting to the requirement for a larger sample size. Nevertheless, it is evident that even a single run of TUPLES obtaining one sample is enough to quantify the distribution of the join result size for various thresholds. In Figure 7 we report, the accuracy of TUPLES on uniform data. The trends are very similar to those on the real dataset. Accuracy appears better on synthetic data. For uniform data, in expectation each tuple contributes approximately the same amount to the final joins result; thus, uniform sampling appears more effective for such data.

Figure 9 presents the accuracy and performance of TOKENS on synthetic data (of average string length 200). Our large sets of experiments with TOKENS on real and synthetic data indicated that this strategy can achieve reasonable accuracy for

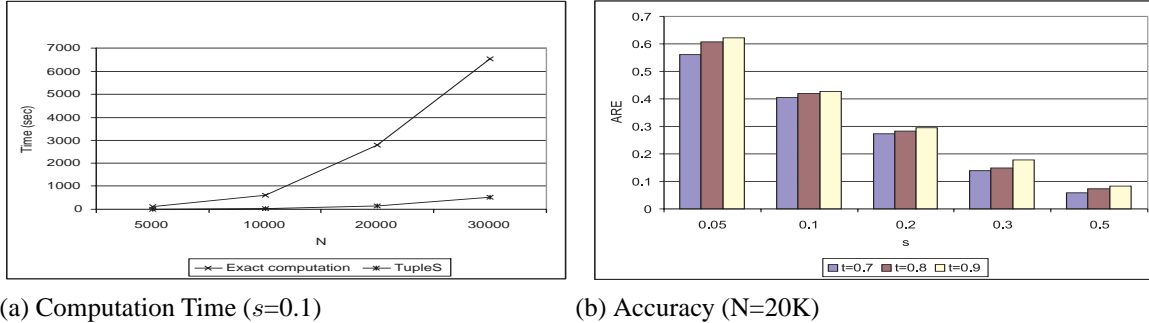


Figure 8. Accuracy and run time performance for TUPLES on Reall1

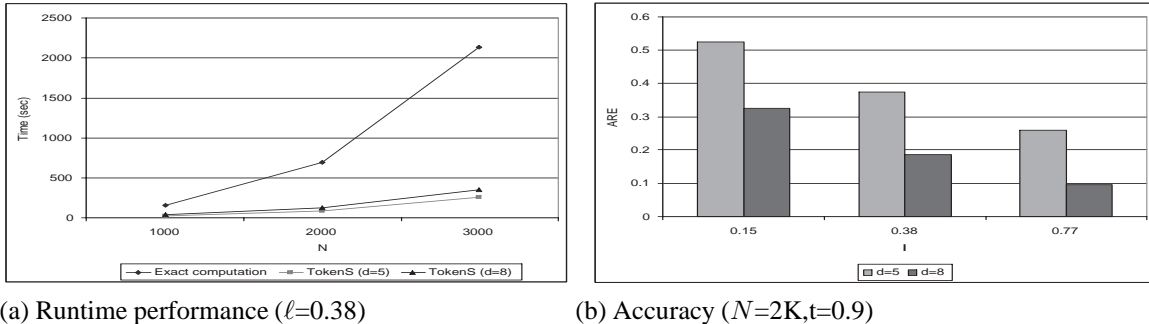


Figure 9. Accuracy and runtime performance of TOKENS on uniform data ($D=650, L=200$)

strings of size large, relative to the total number of distinct tokens D (a fact which is consistent with our analytical expectation). We thus, present sample results on TOKENS using synthetic data, in which we can vary tuple length in a flexible way. In Figure 9(b) we quantify accuracy as a function of ℓ (denoting here the fraction of D sampled) and we present results for two distinct values of d (dimension of the space after application of the JL lemma). Large values of ℓ are in accordance to our analysis in order to obtain good variance. Notice that accuracy progressively improves as a function of ℓ and it's also better as d increases; this is in accordance to the JL lemma. Figure 9(a) presents the corresponding performance graph. Performance benefits, when compared with that of the exact computation (full approximate join of underlying tables) are large and increase as the dataset size increases. For increasing values of d , TOKENS requires progressively more time, since vectors of larger size are involved in the join.

Having evaluated TOKENS and TUPLES in isolation, we now turn to reason about their comparative performance. We thus present the results of an experiment, that compares both strategies in terms of accuracy and performance varying parameters of interest. Figure 10(a) compares TUPLES and TOKENS in terms of their performance. Sample sizes for both strategies (in terms of total bytes sampled) are exactly the same. As the average length of strings (L) in the underlying data sets increases, maintaining D (number of tokens) the

same, it is evident that the performance of TOKENS improves progressively and there exists a crossover point in the performance of the two techniques. This experiment confirms our analytical expectation that TOKENS can provide performance benefits for strings of larger size. The trends of this behavior remain the same for different parameter settings (d and t). Figure 10(b) presents the results of a similar experiment, for the accuracy of the two techniques. The trends are similar, once again confirming our analytical expectations.

Figure 11 depicts the accuracy and performance of the biased sampling approach we introduce compared with TUPLES. We inject four tuples with similarity above 0.7 to 30% of the tuples of each Reall1 relation independently. We assume that statistics for neighborhoods are pre computed via a self join for each underlying Reall1 relation. If we wish to estimate the size of the final result above a threshold t , the neighborhoods are computed using threshold $\frac{t+3}{4}$. As the number of tuples in the underlying relations increases in Figure 11(a) the time required by TUPLES to derive an estimate increases faster than that of the biased counterpart. The total sample size derived is 0.1 of the size of the underlying relations. A tuple is considered HD in this experiment if the size of its neighborhood (δ) is greater than or equal to 3^3 . As we increase the fraction

³Setting δ to a value that captured the top quantiles of the largest neighborhoods was sufficient for all of our experiments

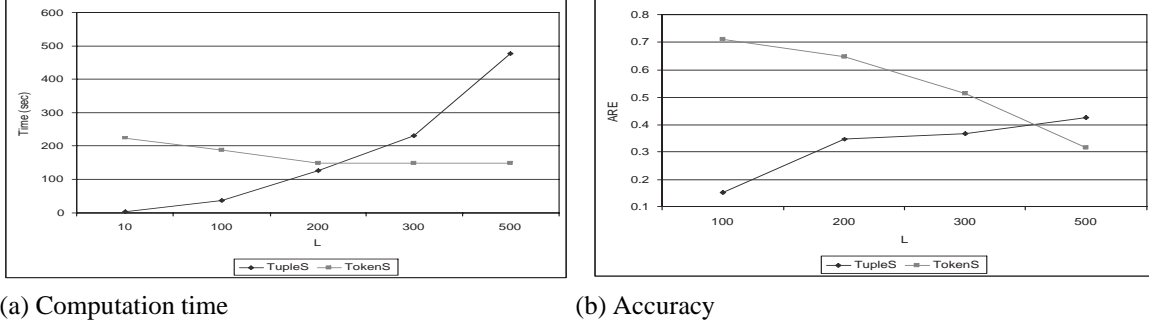


Figure 10. Comparing TUPLES and TOKENS; varying L (average string length) for synthetic data. $D = 600, N = 2K, d = 10, t = 0.9, s = \ell = 0.1$

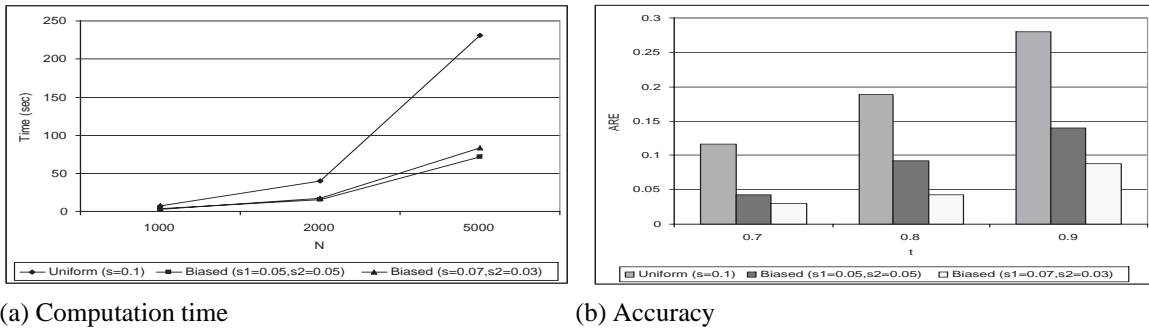


Figure 11. Run time performance and accuracy of biased sampling on data sets Real1

of the HD tuples we sample for the case of biased sampling (from $s_1 = 0.05$ to $s_1 = 0.07$), additional time is required to derive an estimate as more dense neighborhoods are sampled and have to be processed to eliminate possible duplicates. Figure 11(b) presents the corresponding accuracy trends. Biased sampling appears more accurate than TUPLES and the accuracy improves as the number of HD tuples sampled increases. This behavior is consistent for varying values of the threshold t . The error appears to increase for larger values of t since the sample size for all experiments of this graph is fixed to 0.1 of the size of each of the underlying relations. A larger sample size is required to obtain higher accuracy for larger values of t .

6 Applicability to Other Predicates

Our entire discussion up to now utilized cosine similarity as the approximate match predicate to assess a proximity score for tuple pairs. We comment on the applicability of our methods to other approximate match predicates.

Edit distance [13] is a widely used predicate for assessing the closeness of strings. It has been successfully utilized in a declarative framework [11, 7] for data cleaning purposes. Commonly, using edit distance as an approximate match predicate in a join operation, requests all pairs of tuples (strings) from the two relations, at edit distance at most k . Since a

straightforward application of edit distance in a join operation of two relations of size n , would require $O(n^2)$ time (assessing edit distance between every pair of tuples) it is imperative to reduce the number of candidate pairs considered. Facilitating use of edit distance in SQL involves decomposing a string into a set of overlapping subsequences of q characters (for some q) called q -grams and subsequently applying a set of filters manipulating the q -grams. In [11] three filters were proposed, namely the count filter (two strings σ_1, σ_2 can be at edit distance k if the number of common q -grams they share is above $\max(\text{length}(\sigma_1), \text{length}(\sigma_2)) - 1 + (k-1)q$, the length filter (two strings cannot be at edit distance k if their length difference is above k) and the position filter (corresponding q -grams of two strings cannot be at distance more than k apart). These three filters were shown to significantly reduce the amount of computation required for evaluating approximate joins using edit distance via SQL expressions.

Strategy TUPLESS can be readily utilized in conjunction with edit distance (we omit a detailed presentation due to space constraints). All three filters can be readily applied between the sampled tuples and the tuples of the joining relation. It is also possible to derive related expressions for the sample size required to obtain good accuracy. Strategy TOKENS is also applicable, with the exception of the dimensionality re-

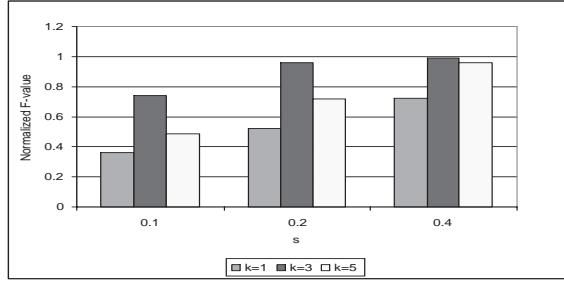


Figure 12. TUPLES using edit distance: accuracy for various values of k and sampling rates

duction step which is specific to cosine similarity. As a result this technique is not expected to offer any performance advantages when compared with TUPLES in the case of edit distance.

Overall, we have:

Proposition 5 Strategy TUPLES as well as our count estimation for the case of available statistics, will apply as is and is possible to derive related analytical guarantees for any approximate match predicate which is a metric and can be adapted into a declarative framework.

Figure 12 presents the results of an experiment of TUPLES utilizing edit distance on the technique of [11] (a technique that will generate false positives, so further post processing of the results is required to identify the true number of pairs within edit distance k). We adopt the F -measure (harmonic mean of the precision and the recall) to quantify accuracy. This is a measure typically adopted to report combined information about precision and recall. We report the normalized F -measure as the F -measure on the result derived by sampling divided by the F -measure of the full approximate join, returned by the technique of [11]. Notice that as the sample rate increases the accuracy of TUPLES gets very close to that of the technique described in [11]. Counting these results to generate estimates for the true size of the approximate join (pairs within edit distance k) provides precise estimates; moreover, the entire operation (approximate join, post processing to filter our false positives and counting) is conducted in a fraction of the time required by the technique of [11]; we omit further graphs due to space constraints.

7 Conclusions

Join techniques deploying approximate match predicates are fundamental data cleaning operations. In this paper we introduced several strategies to estimate parameters on the output of declarative approximate join algorithms. We evaluated those strategies under various parameters settings and identified their relative strengths. Unless very large tuples are involved, one of the strategies, namely TUPLES and its biased variant are the strategy of choice both in terms of accuracy and performance.

Moreover, TUPLES can be applied in conjunction to *any* approximate match metric predicate which is possible to embed in a declarative framework.

This work raises various interesting problems for further study. COMPLETE JOIN is one type of a very useful operation to estimate. However in the context of approximate joins, variants of this operation could also be of interest. For example estimating properties of a join result in which each tuple from one relation participates with its maximum score only is also useful, as it will report statistics on 'best matches' for individual tuples. In fact, variants of the techniques proposed herein can be applied to solve this problem as well (details omitted due to space limitations). Other interesting extensions are also possible.

References

- [1] R. Ananthkrishna, S. Chaudhuri, and V. Ganti. Eliminating Fuzzy Duplicates in Data Warehouses. *Proceedings of VLDB*, Aug. 2002.
- [2] S. Chaudhuri, R. Motwani, and V. Narasayya. On Random Sampling Over Joins. *SIGMOD*, 1999.
- [3] W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. *SIGMOD*, pages 201–212, June 1998.
- [4] J. Copas and F. Hilton. Record Linkage: Statistical Models for Matching Computer Records. *Journal of the Royal Statistical Society*, 1990.
- [5] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining Database Structure or How to Build a Data Quality Browser. *ACM SIGMOD*, 2002.
- [6] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328), pages 1183–1210, Dec. 1969.
- [7] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and E. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. *Proceedings of VLDB*, pages 133–145, 2001.
- [8] S. Ganguly, P. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for Skew Resistant Join size Estimation. *SIGMOD*, 1996.
- [9] V. Ganti, S. Chaudhuri, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. *SIGMOD*, June 2003.
- [10] V. Ganti, S. Chaudhuri, and R. Motwani. Robust Identification of Fuzzy Duplicates. *ICDE*, Apr. 2005.
- [11] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate strings joins in a database (almost) for free. *Proceedings of VLDB*, pages 144–156, 2001.
- [12] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text Joins in an RDBMS for Web Data Integration. *Proceedings of WWW*, 2003.
- [13] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1998.
- [14] M. Hernandez and S. Stolfo. The Merge Purge Problem for Large Databases. *SIGMOD*, pages 127–138, June 1995.
- [15] N. Koudas, A. Marathe, and D. Srivastava. Approximate String Processing Against Large Databases in Practice. *VLDB*, 2004.
- [16] N. Koudas, A. Marathe, and D. Srivastava. SPIDER: Flexible Matching in Databases. *Proceedings of ACM SIGMOD*, 2005.
- [17] L. Gu and R. Baxter and D. Vickers and C. Rainsford. Record Linkage: Current Practice and Future Directions. *CMIS Technical Report N. 03/83*, 2003.
- [18] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of ACM SIGMOD, Montreal Canada*, pages 294–305, June 1996.