Probabilistic String Similarity Joins

Jeffrey Jestes Feifei Li
Computer Science Department, FSU
Tallahassee, FL, USA
{jestes, lifeifei}@cs.fsu.edu

Zhepeng Yan Ke Yi
Dept of Computer Science & Engineering
HKUST, Hong Kong, China
cs_yzx@stu.ust.hk, yike@cse.ust.hk

ABSTRACT

Edit distance based string similarity join is a fundamental operator in string databases. Increasingly, many applications in data cleaning, data integration, and scientific computing have to deal with fuzzy information in string attributes. Despite the intensive efforts devoted in processing (deterministic) string joins and managing probabilistic data respectively, modeling and processing probabilistic strings is still a largely unexplored territory. This work studies the string join problem in probabilistic string databases, using the *expected edit distance* (EED) as the similarity measure. We first discuss two probabilistic string models to capture the fuzziness in string values in real-world applications. The *string-level model* is complete, but may be expensive to represent and process. The *character-level model* has a much more succinct representation when uncertainty in strings only exists at certain positions. Since computing the EED between two probabilistic strings is prohibitively expensive, we have designed efficient and effective pruning techniques that can be easily implemented in existing relational database engines for both models. Extensive experiments on real data have demonstrated order-of-magnitude improvements of our approaches over the baseline.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management—*Systems. Subject: Query processing*

General Terms

Algorithms

Keywords

Probabilistic strings, approximate string queries, string joins

1. INTRODUCTION

Similarity join on string-valued attributes in relational databases is an important operator with a large number of applications. Examples include data cleaning [8], data integration [10], fuzzy keyword search [15] and many others [5, 13, 28]. The *edit distance* is the most commonly used similarity measure between two strings [7,8,13,17,19,28], and edit distance based string similarity join has been extensively studied in the literature, with a suite of techniques developed ranging from simple SQL statements [8, 13] to more complicated ones that need to be implemented within the database engine [28]. However, the increasing presence of probabilistic data in many applications, represented by the recent development in the TRIO [1], MayBMS [4], MystiQ [6], PrDB [25], MCDB [14] and Orion [26] projects, introduces new and interesting challenges for string joins in databases with probabilistic string attributes, which is the focus of this paper.

Note that a probabilistic model is particularly useful for string attributes in the aforementioned applications. In data cleaning, uncertainty often arises in string values, due to data entry errors or simply typing mistakes [7]. Similar scenarios also appear in the automatic recognition of scanned images for bank checks [23]. In a scientific computing project involving strings, such as genome-sequencing and DNA-sequencing, it is common that scientists have identified the values (one of the letters in {'A', 'C', 'G', 'T'}) for 90% of the positions in a genome sequence, while there is some degree of uncertainty for the other 10%. Finally, in a data integration scenario [5, 10], string values for describing the same object from different data sources could be different. In all these applications, the "uncertainty" in strings is not "unknown", and certain knowledge, prior or posterior, is often available [11, 12, 22]. Such knowledge is best modeled by *probabilistic strings*.

Probabilistic string models. A natural way of modeling a probabilistic string S is the *string-level uncertainty model*, in which all possible instances for S are explicitly listed and they form a probability distribution function (pdf). Formally,

Definition 1 (string-level model) Let Σ be an alphabet. A *string-level probabilistic string* is $S = \{(\sigma_1, p_1), \ldots, (\sigma_m, p_m)\}$ where $\sigma_i \in \Sigma^*$, $p_i \in (0, 1]$ and $\sum_{i=1}^{m} p_i = 1$. S instantiates into σ_i with probability p_i. We also use $S(i)$ to denote its i-th choice, i.e., $S(i) = (\sigma_i, p_i)$ for $i = 1, \ldots, m$.

This model is *complete*, since it can represent any pdf of a probabilistic string. However, the uncertainty in a probabilistic string in many of the aforementioned applications often exists only at a few positions. Representing such probabilistic strings in the string-level model would be highly redundant, especially when each uncertain position has many choices or the string has a few uncertain positions, leading to an exponential blowup in the string-level model representation. Indeed, this phenomenon (uncertain positions) has been respected in existing relational databases by their explicit support for wildcards. For example, '_' is a wildcard in SQL that can match any character, so "`advis_r`" may represent either "`adviser`" or "`advisor`". Using wildcards assumes no

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

knowledge for the uncertain positions. However, in most cases, some degree of knowledge is available for these positions [21, 22]. For example, "advisur" is considered as a possible instance of "advis_r", but most likely this is not what the user intends. Based on statistics (and common sense) we can reasonably postulate that this '_' is 'o' with 80% probability and 'e' with 20% probability. With merely a wildcard '_', we have essentially assumed a uniform distribution over the entire alphabet, which is far less accurate than the pdf constructed based on statistics.

Generalizing this observation, a more concise model for many applications is the following *character-level uncertainty model*.

Definition 2 (character-level model) Let Σ be an alphabet. A *character level probabilistic string* is $S = S[1] \dots S[n]$. For $i = 1, \dots, n$, $S[i] = \{(c_{i,1}, p_{i,1}), \dots, (c_{i,\eta_i}, p_{i,\eta_i})\}$, where $c_{i,j} \in \Sigma$, $p_{i,j} \in (0, 1]$ and $\sum_{j=1}^{\eta_i} p_{i,j} = 1$. Each $S[i]$ instantiates into $c_{i,j}$ with probability $p_{i,j}$ independently.

We define a few notations. We use $|X|$ to denote the absolute value of X if X is an integer, the number of elements in X if X is a set, and the length of X if X is either a deterministic or a probabilistic string. Note that if S is a probabilistic string in the string-level model, $|S|$ is a random variable; if S is a probabilistic string in the character-level model, $|S|$ is a constant. We use $\text{size}(X)$ to denote the number of choices in X if X is a pdf. Hence, a string-level probabilistic string S has $\text{size}(S)$ choices, and a character-level probabilistic string S has $\prod_{i=1}^{n} \text{size}(S[i])$ choices.

The independence assumption in Definition 2 allows us to succinctly represent exponentially many possible instances of a random string using the character-level model, which would otherwise have to be stored explicitly in the string-level model. This is also a reasonably good approximation of reality for many of the aforementioned applications, since in many applications the number of uncertain positions in a string is usually a small and isolated percent. In fact, similar models with independent uncertain positions have also been used for molecular biology, pattern matching, and similarity search in random sequences (see [21] and the references therein). If there is non-negligible correlation between adjacent locations, it is possible to consider a hybrid model that combines the string-level model and the character-level model, by making each $S[i]$ a string-level probabilistic string instead of one single character. To keep the presentation clean we will focus on the string-level model and the character-level model in this paper, but the techniques could be extended to the hybrid model as well.

With the wide presence of uncertainty in string data, probabilistic string similarity joins have become an important problem. Its role in probabilistic databases should be comparable to that of deterministic string joins in traditional databases, which have been a focus of study in past years [8, 13, 17–19, 27, 28].

Problem formulation. We extend the edit distance, the standard similarity measure for deterministic strings, to the probabilistic setting, by considering the *expected edit distance* (EED) between two probabilistic strings. Recall that for two deterministic strings σ_1 and σ_2, the edit distance $d(\sigma_1, \sigma_2)$ is the minimum number of *edit operations* required to transform σ_1 to σ_2, where an *edit operation* is an insertion, a deletion, or a substitution of a single character. It is well-known that $d(\sigma_1, \sigma_2)$ can be computed by a simple dynamic program (the proof of its correctness is not trivial though):

$$d[i,j] = \min \begin{cases} d[i,j-1] + 1, & \text{insertion} \\ d[i-1,j] + 1, & \text{deletion} \\ d[i-1,j-1] + c(\sigma_1[i], \sigma_2[j]), & \text{substitution} \end{cases} \quad (1)$$

in which

$$c(\sigma_1[i], \sigma_2[j]) = \begin{cases} 1, \sigma_1[i] \neq \sigma_2[j], \\ 0, \sigma_1[i] = \sigma_2[j], \end{cases}$$

where $\sigma[i]$ is the ith character in a string σ, and the base case is $d[i,j] = i + j$ when $i \cdot j = 0$. In this DP, $d[i,j]$ stores the edit distance between substrings $\sigma_1[1..i]$ and $\sigma_2[1..j]$, thus $d(\sigma_1, \sigma_2) = d[|\sigma_1|, |\sigma_2|]$. This DP's running time is clearly $O(|\sigma_1| \cdot |\sigma_2|)$.

The edit distance of two probabilistic strings is a random variable, so its expectation is a natural measure of similarity between two probabilistic strings. Its formal definition uses the following *possible worlds* semantics. Let S_1 and S_2 be two probabilistic strings in either the string-level or the character-level model.

Definition 3 (possible worlds) The *possible worlds* Ω of S_1 and S_2 is the set of all possible combinations of instances of S_1 and S_2. A *possible world* $s \in \Omega$ is a pair $((\sigma_1, p_1), (\sigma_2, p_2))$, where σ_1 (σ_2) is an instance from S_1 (S_2) with the probability p_1 (p_2). The probability of s is $w(s) = p_1 \cdot p_2$, and the edit distance between S_1 and S_2 in s is $d(s) = d(\sigma_1, \sigma_2)$.

Note that for both probabilistic string models, the possible worlds Ω defines a probability space, namely $\sum_{s \in \Omega} w(s) = 1$.

Definition 4 (expected edit distance) The *expected edit distance* (EED) between S_1 and S_2 is

$$\widehat{d}(S_1, S_2) = \sum_{s \in \Omega} w(s) \cdot d(s). \quad (2)$$

Finally, the join problem is formally defined as:

Definition 5 (probabilistic string join) Given tables R and T, each with a probabilistic string attribute S, the *string join* between R and T on S is to return all pairs of records (r_i, t_j) such that $r_i \in$ R, $t_j \in$ T, and $\widehat{d}(r_i.S, t_j.S) \leq \tau$, where τ is some user-specified threshold.

Although being a distance naturally defined between two probabilistic strings, the EED is very difficult to compute. Let us consider the character-level model. One may be tempted to adapt the dynamic program (1) to compute the EED, by changing $c(\sigma_1[i], \sigma_2[j])$ to $\Pr(S_1[i] \neq S_2[j])$, which is the EED between $S_1[i]$ and $S_2[j]$. This is, unfortunately, wrong. Consider the following simple example: S_1 is a single-character string "a" or "b", each with probability 0.5, and S_2 is "ab" with probability 1, namely, a deterministic string. There are two possible worlds ("a", "ab") and ("b", "ab"), each with probability 0.5. It is clear that in both worlds, the edit distance is 1, thus the EED is also 1. However, the modified DP would give an incorrect answer 1.5. One may try other possible DP formulations, but any attempt turns out to be flawed. The fundamental difficulty with any DP-based approach is that it tries to relate the EED of a subproblem to the EED of a bigger problem. But the EED does not follow any optimal substructure property: the expectation of the minimum of two (or more) random variables is *not* equal to the minimum of their expectations. In fact, such DPs will only give upper or lower bounds on the EED (c.f. Section 4.1.2 and 4.2.2). The only immediate method available to compute the EED exactly is to do so by enumerating all the possible worlds, which is clearly very expensive, especially in the character-level model (we conjecture that this problem is #P-complete). Thus we have an even more urgent need for effective and efficient pruning techniques for string joins in the probabilistic setting, which are the main contribution of this paper.

Our contributions. In this paper we present a comprehensive investigation on the problem of efficient string joins in probabilistic string databases, using the EED as the similarity measure, in both

the string-level and character-level models. In particular, we aim at performing such joins on top of a relational database engine. There are many benefits in realizing this goal, and in some cases it is critical, as it would be more cost-effective to store and process large probabilistic data sets using existing mature relational databases [3]. Indeed, for deterministic string joins, there have been a lot of efforts in trying to perform joins using mainly SQL and UDF in a relational database [8, 13]. More specifically,

- We introduce the string-level and the character-level probabilistic string models, and define the string join problem using the ED in probabilistic string databases.
- We present efficient *lower bound filters*, based on what we call *probabilistic q-grams*, to effectively prune string pairs that cannot possibly join under both models.
- We also present efficient *upper bound filters* to effectively report string pairs that must join in the character-level model.
- We integrate the lower bound filters and the upper bound filters, and give their implementation in relational databases.
- We conduct a comprehensive experimental evaluation for our algorithms in both models on a large number of real data sets. The results show that our efforts have lead to several orders of magnitude of performance improvement compared to the baseline approach.

In the following we first provide some necessary background on *q-grams* used in deterministic string joins in Section 2. Then we develop our techniques in Section 3 and 4. The experimental evaluation is given in Section 5. We survey the related work in Section 6 and conclude in Section 7.

2. BACKGROUND ON Q-GRAMS

A deterministic string join, with an edit distance threshold τ, on two string attributes `A` and `B` from two tables `R` and `T` in a relational database may be implemented as

```
SELECT R.id, T.id FROM R, T WHERE d(A,B)≤τ,   (Q1)
```

where `d(A,B)` is a user defined function (UDF) that implements the dynamic program (1). Existing studies demonstrated that considerable improvements can be achieved using *q-grams*.

For a string σ, its q-grams are produced by sliding a window of length q over the characters of σ. To deal with the special cases at the beginning and the end of σ where there are fewer than q characters, we extend σ by prefixing it with $(q-1)$ '#' and suffixing it with $(q-1)$ '$', where '#' and '$' are not in Σ. Hence, each q-gram for the string σ has exactly q characters. A q-gram also contains its positional information. Formally,

Definition 6 (q-gram) A *q-gram* for a string σ is a pair (ℓ, g) where ℓ is the beginning position of the q-gram (in the extended string σ), and g is the substring of length q in σ beginning from ℓ.

Example 1 The 2-grams for the string *advisor* include $\{(1, \#a), (2, ad), (3, dv), (4, vi), (5, is), (6, so), (7, or), (8, r\$)\}$.

Given $q_1 = (\ell_1, g_1)$ and $q_2 = (\ell_2, g_2)$, $q_1 = q_2$ if $g_1 = g_2$, i.e., positions are *ignored* when comparing two q-grams. Let G_σ be the set of all q-grams in σ. For strings σ_1 and σ_2, define $G_{\sigma_1} \cap G_{\sigma_2} = \{(q_1, q_2) \mid q_1 = q_2, q_1 \in G_{\sigma_1}, q_2 \in G_{\sigma_2}\}$, namely all the matching pairs of q-grams from G_{σ_1} and G_{σ_2} respectively.

Clearly, a string σ has $(|\sigma| + q - 1)$ q-grams. It has been observed that strings with a small edit distance share a large number of common q-grams. This intuition has been formalized by [13,27], among others. Essentially, if we substitute a single character in σ_1 to obtain σ_2, then their q-gram sets differ by at most q q-grams. Similar arguments hold for insertions and deletions. Formally,

Lemma 1 ([13, 27]) *For two strings σ_1 and σ_2, we have*

$$|G_{\sigma_1} \cap G_{\sigma_2}| \geq \max(|\sigma_1|, |\sigma_2|) - 1 - q(d(\sigma_1, \sigma_2) - 1). \quad (3)$$

Since $|G_{\sigma_1} \cap G_{\sigma_2}|$ can be computed efficiently, Lemma 1 gives us a lower bound on $d(\sigma_1, \sigma_2)$; if this lower bound is greater than τ, we can prune this pair from further consideration.

Lemma 1 can be further improved with the positional information of the q-grams. Intuitively, suppose $d(\sigma_1, \sigma_2) = \tau$, then only those q-grams within distance τ could possibly match. So for $q_1 = (\ell_1, g_1)$ and $q_2 = (\ell_2, g_2)$ we define $q_1 \overset{k}{=} q_2$ if $g_1 = g_2$ and $|\ell_1 - \ell_2| \leq k$, and also define $\cap_k$ to be the set of matching pairs of q-grams where the matching is done w.r.t. $\overset{k}{=}$.

Lemma 2 ([13, 27]) *For strings σ_1 and σ_2, we have*

$$|G_{\sigma_1} \cap_{d(\sigma_1,\sigma_2)} G_{\sigma_2}| \geq \max(|\sigma_1|, |\sigma_2|) - 1 - q(d(\sigma_1, \sigma_2) - 1). \quad (4)$$

Assuming $d(\sigma_1, \sigma_2) \leq \tau$, by (4) we have

$$\begin{aligned} d(\sigma_1, \sigma_2) \geq & 1 + \frac{1}{q}(\max(|\sigma_1|, |\sigma_2|) - 1 - |G_{\sigma_1} \cap_{d(\sigma_1,\sigma_2)} G_{\sigma_2}|) \\ \geq & 1 + \frac{1}{q}(\max(|\sigma_1|, |\sigma_2|) - 1 - |G_{\sigma_1} \cap_{\tau} G_{\sigma_2}|). \quad (5) \end{aligned}$$

So if RHS of (5) is larger than τ, this will contradict the assumption that $d(\sigma_1, \sigma_2) \leq \tau$, thus it must be $d(\sigma_1, \sigma_2) > \tau$. Since $|G_{\sigma_1} \cap_\tau G_{\sigma_2}| \leq |G_{\sigma_1} \cap G_{\sigma_2}|$, this is a tighter lower bound than (3).

Finally, there is another straightforward observation:

Lemma 3 ([13, 27]) $d(\sigma_1, \sigma_2) \geq ||\sigma_1| - |\sigma_2||$.

The implementation of these lemmas in SQL is possible and has been explored in [13], which shows a significant performance improvement over the query `(Q1)`.

3. THE STRING-LEVEL MODEL

We represent a string-level probabilistic string attribute in a relational database using five columns: `id, cid, A, p, len`. Consider a probabilistic string $S = \{(\sigma_1, p_1), \ldots, (\sigma_m, p_m)\}$. We convert it into m rows with each row representing one choice of S. All m rows share the same `id`, which uniquely identifies S. The `cid` column will take values $1, \ldots, m$, denoting the index of these m choices. The `A` column stores the m strings for these m choices, and `p` stores the corresponding probabilities. Finally, all m rows store the same `len`, which is the expected length $\mathbf{E}[|S|] = \sum_{i=1}^{m} p_i |\sigma_i|$. Please refer to Figure 1 for an example.

The straightforward implementation of the string join over two tables `R` and `T` is to apply (2) in Definition 4 directly, with the help of the UDF `d(A,B)` (the DP for two deterministic strings in the query `(Q1)`), which leads to the following SQL query:

probabilistic strings

id	*S*
1	{(add, 0.8), (plus, 0.2)}
2	{(up, 0.9), (op, 0.1)}

relational representation

id	cid	A	p	len
1	1	add	0.8	3.2
1	2	plus	0.2	3.2
2	1	up	0.9	2
2	2	op	0.1	2

q-grams

id	cid	ℓ	g
1	1	1	#a
1	1	2	ad
1	1	3	dd
1	1	4	d$
1	2	1	#p
1	2	2	pl
1	2	3	lu
⋮	⋮	⋮	⋮

Figure 1: String-level probabilistic strings, their probabilistic q-grams ($q = 2$), and the relational representation.

```
SELECT R.id,T.id FROM R,T GROUP BY R.id,T.id
HAVING SUM(R.p*T.p*d(R.A,T.A)) ≤ τ                (Q2)
```

Clearly, this approach is very expensive, since it requires running a DP for every possible world on the attributes R.A and T.A for each pair of records in R and T. Our goal is to design efficient filters that can prune away most pairs of records that are not possible to match and execute as few DPs as possible. Note that one cannot improve this basic approach by directly applying the deterministic q-gram based join in each possible world, since a threshold on the edit distance of each world would need to be given, which cannot be determined.

Consider two string-level probabilistic strings $S_1 = \{(\sigma_{1,1}, p_{1,1}), \ldots, (\sigma_{1,m_1}, p_{1,m_1})\}$ and $S_2 = \{(\sigma_{2,1}, p_{2,1}), \ldots, (\sigma_{2,m_2}, p_{2,m_2})\}$. The possible worlds Ω is the set that contains $s = (S_1(i), S_2(j)) = ((\sigma_{1,i}, p_{1,i}), (\sigma_{2,j}, p_{2,j}))$ for $i = 1, \ldots, m_1$ and $j = 1, \ldots, m_2$.

The first filter is a direct extension of the length filter (Lemma 3), for which the proof is trivial.

Lemma 4 *For any string-level probabilistic strings S_1 and S_2:*

$$\widehat{d}(S_1, S_2) \geq \sum_{s \in \Omega} w(s) ||\sigma_{1,i}| - |\sigma_{2,j}||. \tag{6}$$

Inspired by the q-gram based lower bounds on the edit distance of two deterministic strings, we propose effective lower bounds on the EED of S_1 and S_2 based on their *probabilistic q-grams*.

Definition 7 (string-level probabilistic q-grams) Given a string-level probabilistic string $S = \{(\sigma_1, p_1), \ldots, (\sigma_m, p_m)\}$, a *probabilistic q-gram* is a quadruple (i, p, ℓ, g), where i is the choice index (cid), p is the choice probability, while ℓ and g have the same meanings as those in deterministic q-grams. The set of probabilistic q-grams for $S(i)$, denoted as $G_{S(i)}$, is:

$$G_{S(i)} = \{(i, p_i, \ell_j, g_j)\}, \text{ for all } j \text{ s.t. } (\ell_j, g_j) \in G_{\sigma_i}.$$

G_S is the set of string-level probabilistic q-grams for S:

$$G_S = G_{S(1)} \cup \cdots \cup G_{S(m)}.$$

These probabilistic q-grams can also be easily stored in a relational table, as illustrated in Figure 1.

For two probabilistic q-grams $\rho_1 = (i, p_i, \ell_x, g_x)$ and $\rho_2 = (j, p_j, \ell_y, g_y)$ from G_{S_1} and G_{S_2}, respectively, we define $\rho_1 = \rho_2$ if $g_x = g_y$; $\rho_1 \stackrel{k}{=} \rho_2$ if $g_x = g_y$ and $|\ell_x - \ell_y| \leq k$. The operators $\cap$ and $\cap_k$ between G_{S_1} and G_{S_2} are defined similarly as in the deterministic case w.r.t. the $=$ and $\stackrel{k}{=}$ operators, respectively. We also let $p(\rho)$ represent the choice probability of ρ, e.g., $p(\rho_1) = p_i$ and $p(\rho_2) = p_j$ in the above example. Applying Lemma 2 in the possible world $s = (S_1(i), S_2(j))$, we have:

$$\begin{aligned} |G_{S_1(i)} \cap G_{S_2(j)}| &\geq |G_{S_1(i)} \cap_{d(s)} G_{S_2(j)}| \\ &\geq \max(|\sigma_{1,i}|, |\sigma_{2,j}|) - 1 - q(d(s) - 1). \end{aligned} \tag{7}$$

The next lemma will be useful in deriving EED lower bounds.

Lemma 5 *For two non-negative random variables X, Y:*

$$\mathbf{E}[\max(X, Y)] \geq \max(\mathbf{E}[X], \mathbf{E}[Y]).$$

PROOF.

$$\begin{aligned} \mathbf{E}[\max(X, Y)] &= \mathbf{E}\left(\frac{X + Y + |X - Y|}{2}\right) \\ &= \frac{\mathbf{E}[X] + \mathbf{E}[Y]}{2} + \frac{\mathbf{E}[|X - Y|]}{2} \\ &\geq \frac{\mathbf{E}[X] + \mathbf{E}[Y]}{2} + \frac{\mathbf{E}[X - Y]}{2} = \mathbf{E}[X]. \end{aligned}$$

Similarly, $\mathbf{E}[\max(X, Y)] \geq \mathbf{E}[Y]$. □

The following theorem gives a lower bound on $\widehat{d}(S_1, S_2)$.

Theorem 1 *For any string-level probabilistic strings S_1 and S_2:*

$$\begin{aligned} \widehat{d}(S_1, S_2) \quad \geq \quad & 1 + \frac{\max(\mathbf{E}[|S_1|], \mathbf{E}[|S_2|])}{q} \\ & - \frac{\sum_{(\rho_1, \rho_2) \in G_{S_1} \cap G_{S_2}} p(\rho_1) p(\rho_2) + 1}{q}. \end{aligned} \tag{8}$$

PROOF. Let s denote a possible world $(S_1(i), S_2(j))$ from Ω. From (7) we have:

$$\begin{aligned} &\sum_{s \in \Omega} w(s) |G_{S_1(i)} \cap G_{S_2(j)}| \geq \\ &\sum_{s \in \Omega} w(s) (\max(|\sigma_{1,i}|, |\sigma_{2,j}|) - 1 - q(d(s) - 1)). \end{aligned} \tag{9}$$

Since $\sum_{s \in \Omega} w(s) = 1$, the RHS of (9) equals

$$\begin{aligned} \textstyle\sum_{s \in \Omega} \quad & w(s) \max(|\sigma_{1,i}|, |\sigma_{2,j}|) - 1 - q \sum_{s \in \Omega} w(s) d(s) + q \\ = \quad & \mathbf{E}[\max(|S_1|, |S_2|)] - 1 - q \cdot \widehat{d}(S_1, S_2) + q \\ \geq \quad & \max(\mathbf{E}(|S_1|), \mathbf{E}(|S_2|)) - 1 - q(\widehat{d}(S_1, S_2) - 1), \end{aligned}$$

where the last step is by Lemma 5. The LHS of (9) is

$$\begin{aligned} \textstyle\sum_{s \in \Omega} w(s) \quad & |G_{S_1(i)} \cap G_{S_2(j)}| \\ = \quad & \sum_{s \in \Omega} \sum_{(\rho_1, \rho_2) \in G_{S_1(i)} \cap G_{S_2(j)}} w(s) \\ = \quad & \sum_{s \in \Omega} \sum_{(\rho_1, \rho_2) \in G_{S_1(i)} \cap G_{S_2(j)}} p(\rho_1) p(\rho_2) \\ = \quad & \sum_{(\rho_1, \rho_2) \in G_{S_1} \cap G_{S_2}} p(\rho_1) p(\rho_2). \end{aligned} \tag{10}$$

Rearranging the inequality, we obtain (8). □

Theorem 1 provides an efficient and effective way of pruning a pair of probabilistic strings. A pair will be pruned if the RHS of (8) is larger than τ. Given tables R and T, suppose their probabilistic q-grams are stored in the format of Figure 1 in auxiliary tables R_q and T_q. Theorem 1 and Lemma 4 lead to the following query for the string join between tables R and T:

```
1 SELECT R.id, T.id FROM R, T,
2 (SELECT R.id AS rid, T.id AS tid FROM R,T,Rq,Tq
3  WHERE Rq.g=Tq.g AND Rq.id=R.id AND Tq.id=T.id
4     AND Rq.cid=R.cid AND Tq.cid=T.cid
5  GROUP BY R.id, T.id, R.len, T.len
6  HAVING 1+(max(R.len,T.len)-SUM(R.p*T.p)-1)/q≤τ
7  EXCEPT
8  SELECT R.id AS rid, T.id AS tid FROM R, T
9  GROUP BY R.id, T.id
10 HAVING SUM(R.p*T.p*ABS(|R.A|-|T.A|)) >τ
11 ) AS L
12 WHERE L.rid=R.id AND L.tid=T.id
13 GROUP BY R.id, T.id
14 HAVING SUM(R.p*T.p*d(R.A,T.A)) ≤ τ          (Q3)
```

In query (Q3), the pruning conditions in Theorem 1 and Lemma 4 (as seen in the relation L) are first applied. The expensive calculation of the exact EED based on (2) is only applied in the outer query block for those pairs of strings that cannot be pruned. As a result, (Q3) is much more efficient than (Q2) in practice. In some database engines, $\max(a, b)$ (a, b are two real values) is not a built-in function. In this case we can replace $\max(a, b)$ with $(a + b + |a - b|)/2$. In addition, almost all engines evaluate the

HAVING clause in a row-by-row fashion within each group, and multiple conditions in the HAVING clause will not be evaluated using *short-circuit evaluation*, which explains our choice to place the exact ED query condition in an outer query block. The same reason also explains the forthcoming SQL statements in this paper.

Theorem 1 can be seen as the counterpart of Lemma 1 in the probabilistic setting, which ignores the position information of the q-grams. Considering the position information as in Lemma 2 potentially will give a tighter lower bound, but is unfortunately much more difficult in the probabilistic case than in the deterministic case. We can follow the proof of Theorem 1 by replacing $\cap$ with $\cap_{d(s)}$ on the LHS of (9), but the difficulty is that we cannot combine the two "$\sum$" in the last step as in (10), since $d(s)$ could be different for different possible worlds. Making things even worse, we do not know $d(s)$ as this is the very computation that we aim to avoid. We only know that its expectation, namely, $\widehat{d}(S_1, S_2) = \sum_{s\in\Omega} w(s)d(s)$ should be at most τ. More precisely, from the second inequality in (7) and following the proof of Theorem 1 (the RHS derivation stays the same), we obtain

$$\begin{aligned} & \sum_{s\in\Omega} w(s)|G_{S_1(i)} \cap_{d(s)} G_{S_2(j)}| \\ \geq\ & \max(\mathbf{E}(|S_1|), \mathbf{E}(|S_2|)) - 1 - q(\widehat{d}(S_1, S_2) - 1). \quad (11) \end{aligned}$$

Next, we will assume $\widehat{d}(S_1, S_2) \leq \tau$ and under this assumption, relax the LHS of (11) by enlarging it so that it is easy to compute while (11) still holds. The relaxed (11) will give a lower bound on $\widehat{d}(S_1, S_2)$: if this lower bound is higher than τ, that will contradict the assumption that $\widehat{d}(S_1, S_2) \leq \tau$, thus (S_1, S_2) can be pruned.

Let $\Omega' \subseteq \Omega$ be the subset of possible worlds s in which $d(s) \geq 2\tau$. Since $\widehat{d}(S_1, S_2) \leq \tau$, by the Markov inequality the probability mass of Ω' is at most $1/2$. We relax the LHS of (11) as

$$\begin{aligned} \text{LHS of (11)} \leq & \sum_{s\in\Omega'} w(s)|G_{S_1(i)} \cap G_{S_2(j)}| + \\ & \sum_{s\in\Omega\setminus\Omega'} w(s)|G_{S_1(i)} \cap_{2\tau-1} G_{S_2(j)}| \\ = & \sum_{s\in\Omega} w(s)|G_{S_1(i)} \cap_{2\tau-1} G_{S_2(j)}| \\ & + \sum_{s\in\Omega'} w(s)(|G_{S_1(i)} \cap G_{S_2(j)}| - |G_{S_1(i)} \cap_{2\tau-1} G_{S_2(j)}|). \end{aligned}$$

Since we do not know $d(s)$, hence Ω', we have to pessimistically assume that Ω' is chosen so that the LHS of (11) is maximized. Denoting $x(s) = |G_{S_1(i)} \cap G_{S_2(j)}| - |G_{S_1(i)} \cap_{2\tau-1} G_{S_2(j)}|$, the problem becomes choosing Ω' with $\sum_{s\in\Omega'} w(s) \leq 1/2$ to maximize $\sum_{s\in\Omega'} w(s)x(s)$. This is exactly a $0/1$-knapsack problem, which is NP-hard. However, since we are happy with just an upper bound, we can consider this as a fractional knapsack problem, which always yields an (actually quite tight) upper bound on the optimal solution of the $0/1$ version.

Specifically, we initialize $\Omega' = \emptyset$ and sort all the possible worlds in Ω by the "value/weight" ratio $w(s)x(s)/w(s) = x(s)$. We take possible worlds from Ω into Ω' in the decreasing order of $x(s)$ until $\sum_{s\in\Omega'} w(s) > 1/2$. Suppose s' is the last world taken. Then an upper bound on the LHS of (11) is

$$\begin{aligned} \mathrm{UB}_\tau = & \sum_{s\in\Omega} w(s)|G_{S_1(i)} \cap_{2\tau-1} G_{S_2(j)}| \\ & + \sum_{s\in\Omega'} w(s)x(s) - w(s')x(s')\frac{\sum_{s\in\Omega'} w(s) - 1/2}{w(s')}. \end{aligned}$$

This leads to a tighter lower bound on $\widehat{d}(S_1, S_2)$:

Theorem 2 *For any string-level probabilistic strings S_1 and S_2,*

$$\widehat{d}(S_1, S_2) \geq 1 + \frac{\max(\mathbf{E}(|S_1|), \mathbf{E}(|S_2|)) - \mathrm{UB}_\tau - 1}{q}.$$

Computing UB_τ requires an *aggregate UDF* that takes all probabilistic q-grams for a pair of probabilistic strings and τ as input. We omit the details on its implementation and let ub(R$_q$.cid, R.p, R$_q$.ℓ, R$_q$.g, T$_q$.cid, T.p, T$_q$.ℓ, T$_q$.g, τ) denote this UDF. In some engines, an aggregate UDF may take only one input parameter, which could be a user-defined data type. In these cases, we declare a user-defined data type with nine elements as shown above. Since the aggregate UDF could be expensive to evaluate, our idea is to only invoke this pruning after the pruning by Theorem 1 and Lemma 4. The next query implements the string join with the pruning by Theorem 1 and Lemma 4 first, then Theorem 2.

```
SELECT R.id, T.id FROM R, T,
( SELECT R.id AS rid, T.id AS tid FROM R,T,Rq,Tq,
    (···) AS L (same as lines 2-11 in Q3)
   WHERE L.rid=R.id AND L.tid=T.id AND Rq.g=Tq.g
     AND Rq.cid=R.cid AND Tq.cid=T.cid
     AND Rq.id=R.id AND Tq.id=T.id
  GROUP BY R.id, T.id, R.len, T.len
  HAVING 1 + 1/q*( max(R.len,T.len) - 1 -
 ub(Rq.cid,R.p,Rq.ℓ,Rq.g,Tq.cid,T.p,Tq.ℓ,Tq.g,τ))≤τ
) AS L2
··· (same as lines 12-14 in Q3, but with L2)     (Q4)
```

4. THE CHARACTER-LEVEL MODEL

We can certainly represent a character-level probabilistic string in the string-level model, i.e., explicitly store all possible strings it might instantiate into. But this approach incurs a huge space overhead, as there are exponentially many possible instances for a character-level probabilistic string. The large size also implies a large processing cost when performing joins.

To remedy this problem, we store a character-level probabilistic string S as *one string* in a relational table, by representing each probabilistic character $S[i]$'s pdf as follows. We separate different choices in $S[i]$ with "|", and enclose each $S[i]$ with two "$\star$", assuming that "|" and "$\star$" do not belong to Σ. For each choice in $S[i]$, we simply write down the character followed by its probability. If $S[i]$ is deterministic, i.e., it has only one choice with probability 1, we just write it down by itself. If S is deterministic (i.e., all characters are deterministic), it is thus stored as a normal string. We also store the length $|S|$ of S. Hence, the table has three columns: id, A, len. Please refer to Figure 2 for an example.

probabilistic strings

id	*S*
1	{(A, 0.8), (C,0.2)}, {(G, 0.7), (T, 0.3)}
2	{(A, 1)}, {(G, 0.6), (T,0.4)}
3	{(C, 1)}, {(A, 1)}, {(G, 1)}

relational representation

id	A	len
1	⋆A0.8|C0.2⋆⋆G0.7|T0.3⋆	2
2	A⋆G0.6|T0.4⋆	2
3	CAG	3

q-grams

id	p	ℓ	g
1	0.80	1	#A
1	0.20	1	#C
1	0.56	2	AG
1	0.24	2	AT
1	0.14	2	CG
1	0.06	2	CT
1	0.70	3	G$
1	0.30	3	T$
⋮	⋮	⋮	⋮

Figure 2: Character-level probabilistic strings, their probabilistic q-grams ($q = 2$), and the relational representation.

The straightforward implementation of the character-level string join is to define a UDF `ed(A,B)` that computes the EED of two probabilistic strings by applying (2) in Definition 4 directly. It enumerates all possible pairs of instances (possible worlds) from the two inputs, computes the edit distance for each such pair (σ_1, σ_2) by calling the UDF $\mathtt{d}(\sigma_1, \sigma_2)$ and sums up the edit distances for all pairs weighted by their probabilities. Given `ed(A,B)`, a join with threshold τ can be expressed in SQL as follows:

```
SELECT R.id,T.id FROM R,T WHERE ed(R.A,T.A)≤τ  (Q5)
```

Clearly, (Q5) is very expensive, since there are exponentially many possible worlds, hence exponentially many calls to $\mathtt{d}(\sigma_1, \sigma_2)$, when the EED is to be computed between two probabilistic strings, as opposed to quadratic in the string-level model. Thus we have an even more urgent need for efficient and effective filters to avoid calling `ed`. Below we present two kinds of filters. Section 4.1 presents lower bound filters that prune away pairs that cannot possibly join, while Section 4.2 presents upper bound filters that ascertain that a pair must join. The expensive UDF `ed` is invoked only on those pairs for which neither type of filter works. For both kinds of filters we first present a fast, purely relational one based on *probabilistic q-grams* which are defined below, followed by a more expensive one using dynamic programming (but formulated on the probabilistic strings directly). Note that for deterministic string joins a filter cannot use DP as that is the exact computation to save; but in this case, our goal is to avoid running exponentially many DPs, so it is still beneficial if running just *one* DP can avoid so many of them. We first define the *character-level* probabilistic q-grams:

Definition 8 (character-level probabilistic q-grams) For a character level probabilistic string $S = S[1] \dots S[n]$, we first prefix and suffix S with $(q-1)$ (deterministic) characters {(#,1)} and {($,1)}. A *character-level probabilistic q-gram* is a pair $(\ell, S[\ell..\ell+q-1])$, where ℓ is the beginning position of the q-gram and $S[\ell..\ell+q-1]$ is the *probabilistic substring* $S[\ell] \dots S[\ell+q-1]$ in S. The set of all probabilistic q-grams for S is denoted as G_S.

A probabilistic q-gram in the character-level model could be viewed as *a random variable*: the probability space of a probabilistic q-gram $\gamma = (\ell, S[\ell..\ell+q-1])$ is the possible worlds for the probabilistic string $S[\ell..\ell+q-1]$. In contrast, a probabilistic q-gram in the string-level model can be actually considered as a *weighted q-gram*, where the weight is the probability of the corresponding deterministic string that generates this q-gram. In the special case when $S[\ell..\ell+q-1]$ contains no probabilistic characters, γ becomes deterministic. We denote all such γ's from G_S as G_S^d. We also introduce the notation $G^d_{S_1,S_2} = G^d_{S_1} \times G^d_{S_2}$ and $G_{S_1,S_2} = G_{S_1} \times G_{S_2}$ for conciseness.

We represent G_S in a relational table by explicitly listing all the instances for each $\gamma \in G_S$, together with their corresponding probabilities. We apply this technique for each probabilistic string S, leading to a q-gram table, denoted $\mathtt{R}_q$, with four columns `id, p, ℓ, g` as shown in Figure 2. One may be concerned with the potential exponential blowup of this representation, but we argue that this will not happen in practice. First, q is very small in practice ($q = 2$ is usually recommended [13]), so we expect at most a quadratic space overhead in size$(S[i])$. Second, a character-level probabilistic string S for most real-life applications does not have many uncertain positions. And third, we only have a quadratic blowup when these uncertain positions are consecutive.

Given any two probabilistic strings $S_1 = S_1[1] \dots S_1[n_1]$ and $S_2 = S_2[1] \dots S_2[n_2]$, the possible worlds Ω of S_1 and S_2 contains all possible combinations of deterministic strings instantiated from S_1 and S_2. As before we let s denote any possible world from Ω; s is a pair $((\sigma_1, p_1), (\sigma_2, p_2))$, where σ_1 is instantiated from S_1 with probability p_1 and σ_2 is instantiated from S_2 with probability p_2. Define $d(s) = d(\sigma_1, \sigma_2)$, $w(s) = p_1 \cdot p_2$. It is clear that $\sum_{s\in\Omega} w(s) = 1$. We also define:

$$\Pr(S_1 = S_2) = \sum_{s\in\Omega, \sigma_1=\sigma_2} w(s).$$

For any two probabilistic q-grams $\gamma_1 = (i, S_1[i..i+q-1])$ and $\gamma_2 = (j, S_2[j..j+q-1])$ from G_{S_1} and G_{S_2} respectively, the difference in positions between γ_1 and γ_2 is denoted by $\mathrm{off}(\gamma_1, \gamma_2) = |i-j|$. Note that this is not a random variable. We define:

$$\Pr(\gamma_1 = \gamma_2) = \Pr(S_1[i..i+q-1] = S_2[j..j+q-1]).$$

In particular, for $(\gamma_1, \gamma_2) \in G^d_{S_1,S_2}$, $\Pr(\gamma_1 = \gamma_2) = 1$ if $S_1[i..i+q-1] = S_2[j..j+q-1]$, 0 otherwise. In this case, we simply write $\gamma_1 = \gamma_2$ if and only if $\Pr(\gamma_1 = \gamma_2) = 1$.

Example 2 Let S_1 and S_2 be the two probabilistic strings for records 1 and 2 in Figure 2. The first probabilistic q-gram of S_1 is $\gamma_1 = (1, \{(\#, 1)\}\{(A, 0.8), (C, 0.2)\})$, and the first probabilistic q-gram of S_2 is $\gamma_2 = (1, \{(\#, 1)\}\{(A, 1)\})$. $\Pr(\gamma_1 = \gamma_2) = 0.8$, $\mathrm{off}(\gamma_1, \gamma_2) = 0$, and $\gamma_2 \in G^d_{S_2}$ (but $\gamma_1 \notin G^d_{S_1}$).

4.1 Lower bounds on the EED of S_1 and S_2

The first filter is to lower bound $\widehat{d}(S_1, S_2)$ by their length difference immediately by Lemma 3, since the length of a probabilistic string is fixed in the character-level model.

Lemma 6 *For any character-level probabilistic strings S_1, S_2,*

$$\widehat{d}(S_1, S_2) \geq ||S_1| - |S_2||.$$

4.1.1 A probabilistic q-gram based lower bound

The lower bound below is a generalization of Lemma 1 to the probabilistic setting.

Theorem 3 *For any character-level probabilistic strings S_1, S_2:*

$$\widehat{d}(S_1, S_2) \geq 1 + \frac{\max(|S_1|, |S_2|) - 1 - \sum_{\substack{\gamma_1\in G_{S_1}\\ \gamma_2\in G_{S_2}}} \Pr(\gamma_1 = \gamma_2)}{q} \tag{12}$$

PROOF. Let $s = ((\sigma_1, p_1), (\sigma_2, p_2))$ be any possible world in the possible worlds Ω for S_1 and S_2. Applying Lemma 1 to all $s \in \Omega$, we have

$$\begin{aligned} & \sum_{s\in\Omega} w(s)|G_{\sigma_1} \cap G_{\sigma_2}| \\ \geq\ & \sum_{s\in\Omega} w(s)(\max(|\sigma_1|, |\sigma_2|) - 1 - q(d(s) - 1)) \\ =\ & \max(|S_1|, |S_2|) - 1 - q(\widehat{d}(S_1, S_2) - 1). \end{aligned}$$

For any two probabilistic q-grams $\gamma_1 = (i, S_1[i..i+q-1]) \in G_{S_1}$, $\gamma_2 = (j, S_2[j..j+q-1]) \in G_{S_2}$, and a random possible world $s = ((\sigma_1, p_1), (\sigma_2, p_2))$, we define $\gamma_1 =_s \gamma_2$ as the event that $S_1[i..i+q-1] = S_2[j..j+q-1]$ in s. We have

$$\Pr(\gamma_1 =_s \gamma_2) = \begin{cases} w(s), & \text{if } \sigma_1[i..i+q-1] = \sigma_2[j..j+q-1]; \\ 0, & \text{otherwise.} \end{cases}$$

Recall that $|G_{\sigma_1} \cap G_{\sigma_2}|$ is the number of matching pairs of q-grams in G_{σ_1} and G_{σ_2}, so

$$\sum_{s\in\Omega} w(s)|G_{\sigma_1} \cap G_{\sigma_2}| = \sum_{s\in\Omega} \sum_{\substack{\gamma_1\in G_{S_1}\\ \gamma_2\in G_{S_2}}} \Pr(\gamma_1 =_s \gamma_2)$$
$$= \sum_{\substack{\gamma_1\in G_{S_1}\\ \gamma_2\in G_{S_2}}} \sum_{s\in\Omega} \Pr(\gamma_1 =_s \gamma_2) = \sum_{\substack{\gamma_1\in G_{S_1}\\ \gamma_2\in G_{S_2}}} \Pr(\gamma_1 = \gamma_2),$$

rearranging the inequality completes the proof. □

Incorporating the q-gram's position information in each possible world using Lemma 2 to derive a tighter lower bound for $\widehat{d}(S_1, S_2)$ is more involved. We can follow the proof of Theorem 3, but applying Lemma 2 instead, obtaining

$$\sum_{s\in\Omega} w(s)|G_{\sigma_1} \cap_{d(s)} G_{\sigma_2}| \geq \max(|S_1|, |S_2|) - 1 - q(\widehat{d}(S_1, S_2) - 1). \tag{13}$$

The difficulty is that $d(s)$ changes for different worlds and it is exactly the value that we avoid to compute for each possible world in a pruning method. Next, we show how to obtain an upper bound on the LHS of (13) that is efficiently computable.

Lemma 7 *For any character-level probabilistic strings S_1, S_2, $\forall \tau \geq \widehat{d}(S_1, S_2)$:*

$$\sum_{s\in\Omega} w(s)|G_{\sigma_1} \cap_{d(s)} G_{\sigma_2}| \leq \sum_{(\gamma_1,\gamma_2)\in(G_{S_1,S_2} - G^d_{S_1,S_2})} \Pr(\gamma_1 = \gamma_2)$$
$$+ \sum_{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2}, \gamma_1=\gamma_2} \min\left(1, \frac{\tau}{\mathrm{off}(\gamma_1,\gamma_2)}\right)$$

PROOF. Let γ_1 and γ_2 be any probabilistic q-gram from G_{S_1} and G_{S_2} respectively. For a random s, by the Markov inequality,

$$\Pr(d(s) \geq \mathrm{off}(\gamma_1,\gamma_2)) \leq \min\left(1, \frac{\mathbf{E}(d(s))}{\mathrm{off}(\gamma_1,\gamma_2)}\right)$$
$$= \min\left(1, \frac{\widehat{d}(S_1, S_2)}{\mathrm{off}(\gamma_1,\gamma_2)}\right) \leq \min\left(1, \frac{\tau}{\mathrm{off}(\gamma_1,\gamma_2)}\right), \tag{14}$$

where we define $\frac{1}{\mathrm{off}(\gamma_1,\gamma_2)} = \infty$ if $\mathrm{off}(\gamma_1,\gamma_2) = 0$. Next,

$$\sum_{s\in\Omega} w(s)|G_{\sigma_1} \cap_{d(s)} G_{\sigma_2}|$$
$$= \sum_{(\gamma_1,\gamma_2)\in G_{S_1,S_2}} \sum_{s\in\Omega} \Pr(\gamma_1 =_s \gamma_2 \text{ and } \mathrm{off}(\gamma_1,\gamma_2) \leq d(s))$$
$$= \sum_{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2}} \sum_{s\in\Omega} \Pr(\gamma_1 =_s \gamma_2 \text{ and } \mathrm{off}(\gamma_1,\gamma_2) \leq d(s))$$
$$+ \sum_{\substack{(\gamma_1,\gamma_2)\in\\ G_{S_1,S_2} - G^d_{S_1,S_2}}} \sum_{s\in\Omega} \Pr(\gamma_1 =_s \gamma_2 \text{ and } \mathrm{off}(\gamma_1,\gamma_2) \leq d(s))$$
$$\leq \sum_{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2}, \gamma_1=\gamma_2} \min\left(1, \frac{\tau}{\mathrm{off}(\gamma_1,\gamma_2)}\right) \quad \text{(by (14))}$$
$$+ \sum_{(\gamma_1,\gamma_2)\in(G_{S_1,S_2} - G^d_{S_1,S_2})} \sum_{s\in\Omega} \Pr(\gamma_1 =_s \gamma_2).$$

In the last step, the second term is relaxed by removing the position constraint. This completes the proof. □

Lemma 7 and (13) lead to a tighter lower bound for $\widehat{d}(S_1, S_2)$:

Theorem 4 *For any character-level probabilistic strings S_1, S_2, $\forall \tau \geq \widehat{d}(S_1, S_2)$:*

$$\widehat{d}(S_1, S_2) \geq 1 + \frac{\max(|S_1|, |S_2|) - 1}{q}$$
$$- \frac{\sum_{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2}, \gamma_1=\gamma_2} \min\left(1, \frac{\tau}{\mathrm{off}(\gamma_1,\gamma_2)}\right)}{q}$$
$$- \frac{\sum_{(\gamma_1,\gamma_2)\in G_{S_1,S_2} - G^d_{S_1,S_2}} \Pr(\gamma_1 = \gamma_2)}{q}. \tag{15}$$

In (15), $\sum_{(\gamma_1,\gamma_2)\in G_{S_1,S_2} - G^d_{S_1,S_2}} \Pr(\gamma_1 = \gamma_2)$ is the sum of the probabilities that two probabilistic q-grams from G_{S_1} and G_{S_2} equal if any one of them contains at least one probabilistic character, and $\sum_{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2}, \gamma_1=\gamma_2} \min(1, \frac{\tau}{\mathrm{off}(\gamma_1,\gamma_2)})$ is the sum of either 1, for two q-grams from G_{S_1} and G_{S_2} if they are identical and neither contains any probabilistic character and $\tau \geq \mathrm{off}(\gamma_1,\gamma_2)$, or τ divided by their position difference if $\tau < \mathrm{off}(\gamma_1,\gamma_2)$. Clearly,

$$\sum_{\substack{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2},\\ \gamma_1=\gamma_2}} \min\left(1, \frac{\tau}{\mathrm{off}(\gamma_1,\gamma_2)}\right) \leq \sum_{\substack{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2},\\ \gamma_1=\gamma_2}} 1.$$

Hence, if we let $C = 1 + \frac{\max(|S_1|,|S_2|)-1}{q}$, then:

$$(\text{RHS of (15)} - C) \cdot q$$
$$\geq - \sum_{\substack{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2},\\ \gamma_1=\gamma_2}} 1 - \sum_{\substack{(\gamma_1,\gamma_2)\in\\ G_{S_1,S_2} - G^d_{S_1,S_2}}} \Pr(\gamma_1 = \gamma_2)$$
$$= - \sum_{(\gamma_1,\gamma_2)\in G^d_{S_1,S_2}} \Pr(\gamma_1 = \gamma_2) - \sum_{\substack{(\gamma_1,\gamma_2)\in\\ G_{S_1,S_2} - G^d_{S_1,S_2}}} \Pr(\gamma_1 = \gamma_2)$$
$$= - \sum_{(\gamma_1,\gamma_2)\in G_{S_1,S_2}} \Pr(\gamma_1 = \gamma_2) = (\text{RHS of (12)} - C) \cdot q,$$

which means that Theorem 4 always provides a tighter lower bound.

4.1.2 A DP based lower bound

The succinct representation of the probabilistic string in the character level model makes it possible to extend the DP formulation for the deterministic edit distance to the probabilistic setting to derive lower and upper bounds (but unfortunately not the exact EED as argued in the introduction). The lower bound DP formulation for the EED is very similar to the classic, deterministic edit distance DP as shown in (1). For two probabilistic strings $S_1 = S_1[1] \dots S_1[n_1]$ and $S_2 = S_2[1] \dots S_2[n_2]$, we will compute lower bounds on the EED between $S_1[1..i]$ and $S_2[1..j]$ for $i = 1, \dots, n_1$ and $j = 1, \dots, n_2$ inductively. The base cases are $\widehat{ld}[i, j] = i + j$ for $i \cdot j = 0$. For other i, j, we set

$$\widehat{ld}[i, j] = \min \begin{cases} \widehat{ld}[i, j-1] + 1; \\ \widehat{ld}[i-1, j] + 1; \\ \widehat{ld}[i-1, j-1] + lc(S_1[i], S_2[j]), \end{cases} \tag{16}$$

where

$$lc(S_1[i], S_2[j]) = \begin{cases} 1, \Pr(S_1[i] = S_2[j]) = 0; \\ 0, \Pr(S_1[i] = S_2[j]) > 0. \end{cases}$$

We will show that $\widehat{ld}[i, j]$ is a lower bound on the EED between $S_1[1..i]$ and $S_2[1..j]$ for all i, j. In fact, we will prove the following stronger result:

Theorem 5 *For any two character-level probabilistic strings $S_1 = S_1[1]\dots S_1[n_1]$ and $S_2 = S_2[1]\dots S_2[n_2]$,*

$$\widehat{ld}[n_1, n_2] = \min_{s\in\Omega}(d(s)) \le \widehat{d}(S_1, S_2).$$

PROOF. Let $d_s[i, j]$ be the deterministic edit distance DP's output in the possible world $s = \{(\sigma_1, p_1), (\sigma_2, p_2)\} \in \Omega$ for $\sigma_1[1..i]$ and $\sigma_2[1..j]$. We first show that $\widehat{ld}[n_1, n_2] \le d_s[n_1, n_2]$ for $\forall s \in \Omega$ and we prove this by induction.

The base cases $\widehat{ld}[i, j] = d_s[i, j]$ for $i \cdot j = 0$ are trivial. Assume that this claim holds for $\forall i' \le i$, $\forall j' \le j$ where the two equalities do not hold at the same time. Since $lc(S_1[i], S_2[j]) \le c(\sigma_1[i], \sigma_2[j])$ for any s and by the induction hypothesis, the RHS of (16) is always at most the RHS of (1), thus we have $\widehat{ld}[i, j] \le d_s[i, j]$. In particular $\widehat{ld}[n_1, n_2] \le d_s[n_1, n_2]$ for $\forall s \in \Omega$, so $\widehat{ld}[n_1, n_2] \le \min_{s\in\Omega}(d(s))$.

Next, we can show that it is possible to construct a possible world s such that $d(s) = \widehat{ld}[n_1, n_2]$, by just following the choices of the $\widehat{ld}[i, j]$ in (16). So $\widehat{ld}[n_1, n_2] = \min_{s\in\Omega}(d(s))$. □

4.2 Upper bounds on the EED of S_1 and S_2

The lower bounds on the EED are useful to prune away many pairs that have expected edit distances larger than the join threshold. However, for string pairs which have their EED smaller than the threshold, lower bounds are not useful. To further reduce the cost of a string join, we introduce two upper bounds in this section. These upper bounds are particularly useful in efficiently identifying many string pairs that do satisfy the join condition, which helps avoid incurring the high cost of invoking `ed(·,·)` on these pairs.

4.2.1 A probabilistic q-gram based upper bound

For any two deterministic q-grams $q_1 = (\ell_1, g_1)$ and $q_2 = (\ell_2, g_2)$ from two strings σ_1 and σ_2 respectively, we define $q_1 \equiv q_2$ if and only if $g_1 = g_2$ *and* $\ell_1 = \ell_2$, and $(q_1 \equiv q_2)$ returns 1 if $q_1 \equiv q_2$, 0 otherwise.

Lemma 8 *For any two strings σ_1 and σ_2:*

$$d(\sigma_1, \sigma_2) \le \max(|\sigma_1|, |\sigma_2|) + q - 1 - \sum_{q_1\in G_{\sigma_1}, q_2\in G_{\sigma_2}} (q_1 \equiv q_2) \tag{17}$$

PROOF. Without loss of generality, assume that $|\sigma_1| \ge |\sigma_2|$. Clearly, $|G_{\sigma_1}| = |\sigma_1| + q - 1$ and there are $|\sigma_1| + q - 1$ positions (including the prefix and suffix characters '#' and '$') for possible edit operations to transform σ_1 into σ_2. Without touching any position in which the corresponding q-grams $q_1 \in \sigma_1$ and $q_2 \in \sigma_2$ satisfy $q_1 \equiv q_2$, we can always transform σ_1 into σ_2 by editing all other positions in the worst case, which is $|\sigma_1| + q - 1 - \sum_{q_1\in G_{\sigma_1}, q_2\in G_{\sigma_2}} (q_1 \equiv q_2)$ edit operations. Since $d(\sigma_1, \sigma_2)$ is the minimum number of edit operations required to make $\sigma_1 = \sigma_2$, this immediately implies (17). □

This observation directly leads to an upper bound for the EED of two probabilistic strings based on their probabilistic q-grams. For any two probabilistic q-grams $\gamma_1 = (i, S_1[i..i + q - 1])$ and $\gamma_2 = (j, S_2[j..j + q - 1])$ for two character-level probabilistic strings S_1 and S_2, we define that

$$\Pr(\gamma_1 \equiv \gamma_2) = \Pr(\gamma_1 = \gamma_2) \text{ if } i = j, 0 \text{ otherwise.}$$

Theorem 6 *For any character-level probabilistic strings S_1, S_2:*

$$\widehat{d}(S_1, S_2) \le \max(|S_1|, |S_2|) + q - 1 - \sum_{(\gamma_1,\gamma_2)\in G_{S_1,S_2}} Pr(\gamma_1 \equiv \gamma_2)$$

PROOF. This is derived by utilizing (17) in each possible world $s \in \Omega$ and summing over all worlds. □

4.2.2 A DP based upper bound

It is possible to obtain a very tight upper bound for the EED with a DP formulation on the probabilistic characters directly. For any two probabilistic strings $S_1 = S_1[1]\dots S_1[n_1]$ and $S_2 = S_2[1]\dots S_2[n_2]$, we set $\widehat{ud}[i, j]$ as

$$\widehat{ud}[i, j] = \min \begin{cases} \widehat{ud}[i, j-1] + 1, \\ \widehat{ud}[i-1, j] + 1, \\ \widehat{ud}[i-1, j-1] + \Pr(S_1[i] \ne S_2[j]), \end{cases} \tag{18}$$

for $i \in [1, n_1]$, $j \in [1, n_2]$, with the boundary condition $\widehat{ud}[i, j] = i + j$ for $i \cdot j = 0$. We will show that $\widehat{ud}[i, j]$ thus computed is an upper bound on the EED between $S_1[1..i]$ and $S_2[1..j]$. We first present two technical results.

Lemma 9 *For two non-negative random variables X, Y,*

$$\mathbf{E}[\min(X, Y)] \le \min(\mathbf{E}[X], \mathbf{E}[Y]).$$

PROOF. Observing that $\mathbf{E}[\min(X, Y)] = \mathbf{E}[\frac{X+Y-|X-Y|}{2}]$, this is proved by following a similar argument as in Lemma 5. □

Corollary 1 *For three non-negative random variables X, Y, Z,*

$$\mathbf{E}[\min(X, Y, Z)] \le \min(\mathbf{E}[X], \mathbf{E}[Y], \mathbf{E}[Z]).$$

Theorem 7 $\widehat{ud}[i, j] \ge \widehat{d}(S_1[1..i], S_2[1..j])$ *for all i, j.*

PROOF. We prove this theorem by induction. The theorem clearly holds when $i \cdot j = 0$ since $\widehat{ud}[i, j] = i + j = \widehat{d}(S_1[1..i], S_2[1..j])$.

Assume that the theorem holds for $\forall i' \le i, j' \le j$ where the two equalities do not hold at the same time. Let $s = \{(\sigma_1, p_1), (\sigma_2, p_2)\}$ be a random world from Ω, the edit distance DP formulation (1) in the world s is:

$$d_s[i, j] = \min \begin{cases} d_s[i-1, j] + 1, \\ d_s[i, j-1] + 1, \\ d_s[i-1, j-1] + c(\sigma_1[i], \sigma_2[j]). \end{cases}$$

Taking expectation on both sides over all possible worlds, we get:

$$\begin{aligned} & \widehat{d}(S_1[1..i], S_2[1..j]) \\ = \; & \mathbf{E}\left[\min \begin{cases} d_s[i-1, j] + 1 \\ d_s[i, j-1] + 1 \\ d_s[i-1, j-1] + c(\sigma_1[i], \sigma_2[j]) \end{cases}\right] \\ \le \; & \min \begin{cases} \widehat{d}(S_1[1..i-1], S_2[1..j]) + 1 \\ \widehat{d}(S_1[1..i], S_2[1..j-1]) + 1 \\ \widehat{d}(S_1[1..i-1], S_2[1..j-1]) + \Pr(S_1[i] \ne S_2[j]) \end{cases} \quad \text{(by Corollary 1)} \\ \le \; & \min \begin{cases} \widehat{ud}[i-1, j] + 1 \\ \widehat{ud}[i, j-1] + 1 \\ \widehat{ud}[i-1, j-1] + \Pr(S_1[i] \ne S_2[j]) \end{cases} \quad \text{(by induction)} \\ = \; & \widehat{ud}[i, j]. \end{aligned}$$

The theorem is therefore proved. □

4.3 Query implementation

The next query implements the character-level string join, by incorporating the lower bounds (Lemma 6, Theorem 4 and 5) and the upper bounds (Theorem 6 and 7). The first part in the relation L before the EXCEPT, lines 2-9, is the probabilistic q-gram lower bound in Theorem 4 with the length pruning in Lemma 6 (line 4). In particular, line 6 is the first two terms in the RHS of (15); lines 7-8 is the third term in the RHS of (15), since FLOOR(R_q.p*T_q.p) equals 0 for $(\gamma_1, \gamma_2) \in G_{S_1,S_2} - G^d_{S_1,S_2}$, and 1 for $(\gamma_1, \gamma_2) \in G^d_{S_1,S_2}$, note that max(1,ABS($R_q$.$\ell$-$T_q$.$\ell$)) in line 8 handles the special case when ABS(R_q.ℓ-T_q.ℓ)=0; line 9 is the fourth term in the RHS of (15), since CEILING(1-R_q.p*T_q.p) equals 1 for $(\gamma_1, \gamma_2) \in G_{S_1,S_2} - G^d_{S_1,S_2}$, and 0 for $(\gamma_1, \gamma_2) \in G^d_{S_1,S_2}$.

The second part in the relation L after the EXCEPT, lines 11-15, is simply the probabilistic q-gram upper bound in Theorem 6 with the length pruning in Lemma 6. Essentially, L contains those pairs that cannot be pruned by the length, or the probabilistic q-gram lower and upper bounds. Finally, the outer-block query joins tables R, T with L. For each pair in L, it performs the lower bound and upper bound DPs first (UDF ld(·,·) and ud(·,·) respectively), followed by the naive calculation with ed(·,·) as a last resort if a pair cannot be pruned by ld(·,·) and ud(·,·). Similarly to Q3 and Q4, $\max(a, b)$ and $\min(a, b)$ might not be defined for two real values a, b, they could be simply replaced by $(a + b + |a - b|)/2$ and $(a + b - |a - b|)/2$ if necessary.

```
1  SELECT R.id,T.id FROM R,T,
2  (SELECT R.id AS rid,T.id AS tid FROM R,T,Rq,Tq
3   WHERE Rq.g=Tq.g AND R.id=Rq.id AND T.id=Tq.id
4     AND ABS(R.len - T.len) ≤ τ
5   GROUP BY R.id, T.id, R.len, T.len
6   HAVING 1 + (max(R.len,T.len)-1)/q -
7    SUM( FLOOR(Tq.p*Rq.p)*
8      min(1, τ/max(1,ABS(Rq.ℓ - Tq.ℓ))) )/q -
9    SUM( CEILING(1-Rq.p*Tq.p)*Tq.p*Rq.p )/q ≤ τ
10  EXCEPT
11  SELECT R.id AS rid,T.id AS tid FROM R,T,Rq,Tq
12  WHERE Rq.g=Tq.g AND Rq.ℓ=Tq.ℓ AND R.id=Rq.id
13    AND T.id=Tq.id AND ABS(R.len - T.len) ≤ τ
14  GROUP BY R.id, T.id, R.len, T.len
15  HAVING max(R.len,T.len)+q-1-SUM(Tq.p*Rq.p)≤τ
16 ) AS L
17 WHERE L.rid=R.id AND L.tid=T.id AND ld(R.A,T.A)
18  ≤τ AND ud(R.A,T.A)>τ AND ed(R.A,T.A)≤τ   (Q6)
```

Lastly, to be complete, results from Q6 need to be unioned with those pruned away by either the probabilistic q-gram (lines 11-15) or the DP (ud) upper bound. It has been omitted in (Q6) for simplicity, and we use (Q6) to refer to this complete SQL query.

5. EXPERIMENTS

We implemented all methods in Microsoft SQL Server 2008 (the Enterprise edition). All experiments were executed on a machine with an Intel Xeon E5405@2.00GHz CPU running Windows XP. We have 2GB memory allocated to SQL Server.

Datasets. We created 4 datasets, 2 for each model, using 3 real data sources. The first data source is the author names in DBLP. There are two types of ambiguities/uncertainties in people's names. The first is due to spelling errors or conversion between different character sets (e.g., the letter 'ä' is often converted to 'a' in ASCII), and the second is due to the fact that a name may have different forms. The former can be represented by the character-level model, while the latter requires the string-level model. So we created two datasets, one for each model, as follows. For a name σ in the DBLP database, we find its *similar set* $A(\sigma)$ that consists of all names in DBLP within edit distance 3 to σ. Note that $A(\sigma)$ contains duplicated names. Then we create a character-level probabilistic string S based on σ as follows. We identify the non-ASCII characters in σ (we randomly pick a few if there are no such characters), and for each such position i, the pdf of $S[i]$ is generated based on the normalized frequencies of the letters in the i-th position of all the strings in $A(\sigma)$. The other positions of S are deterministic and the same as in σ. A string-level probabilistic string is created by simply taking all the distinct strings in $A(\sigma)$ as the possible choices, with the probabilities being proportional to their frequencies. To mimic the case where the same name may have drastically different forms, we also replace a few choices with randomly picked names outside $A(\sigma)$. We denote these datasets as *Author1* (string-level) and *Author2* (character-level) accordingly. Note that *Author1* and *Author2* represent *completely different* information.

The second dataset (*Category*) is from the Wikipedia database, using the category-link string field in the category-link table. Many category-links refer to the same subject with different string values, which are best represented by the string-level model. We generate its string-level representation in the same way as for *Author1*.

The third dataset (*Genome*) is based on the genome sequence database from the Rhodococcus project. Genome sequences may contain unknown positions, and they are best represented by the character-level model. We broke a very long genome sequence into many shorter strings of controlled length, then generated character-level probabilistic strings in the same way as for the *Author2* dataset.

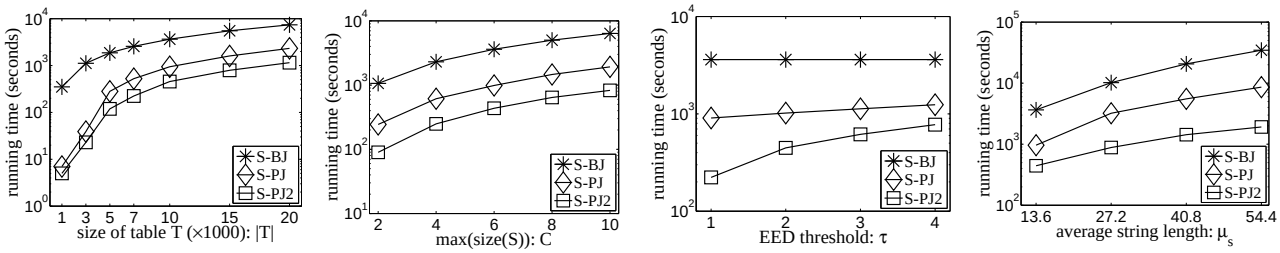
Finally, we create two tables R and T with a certain number of probabilistic strings (in either model). Each probabilistic string is created based on a random σ from the corresponding data source.

Setup. In the string-level model, an interesting factor is the number of choices, size(S), that each probabilistic string S may have. Since many strings from our datasets have a large number of choices, we specify the maximum size(S), denoted as C. If S has more than C choices, we only take its C most probable choices and normalize them. Similarly, in the character-level model, we limit the number of choices that a probabilistic character $S[i]$ may have to χ. We also control the fraction of uncertain positions in a character-level probabilistic string, θ, as we generate the datasets.

In both models, the string length obviously affects the join cost. In the string-level model, we use μ_s to denote the average length of the deterministic strings from all choices of all probabilistic strings. To study its effect, we append each deterministic string to itself for $\omega = 0, 1, 2, \ldots$ times. In the character-level model, we use μ_c to denote the average length of all probabilistic strings, and to study its effect, we append each probabilistic string to itself ω times. When $\omega = 0$ (no self-concatenation), $\mu_s = 13.6$ and $\mu_c = 14.3$ for *Author1* and *Author2*, and the string length distributions in both models follow approximately a normal distribution in the range of $[5, 36]$; for the *Category* dataset (string-level model), $\mu_s = 19.9$ and its string length distribution is (very roughly) a heavy-tail distribution in the range of $[3, 50]$; for the *Genome* dataset (character-level model), $\mu_c = 14.9$ and its string length distribution is roughly a uniform distribution in the range of $[10, 20]$.

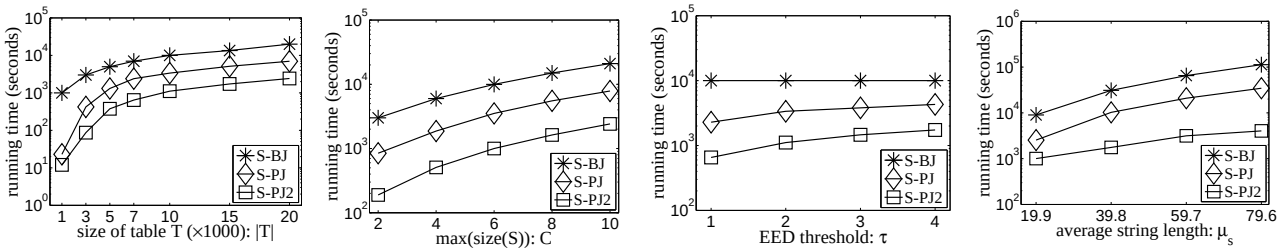
Since the basic methods are very expensive, we limit the sizes of the tables. The default sizes for R and T are $|\mathtt{R}| = 1,000$ and $|\mathtt{T}| = 10,000$. We study the scalability of all methods by varying the size of T. The default values for other parameters are: $q = 2$, $\tau = 2$, $C = 6$, $\chi = 6$, $\theta = 20\%$, and $\omega = 0$.

In what follows, we refer to the basic methods for string joins in the string-level and character-level models, (Q2) and (Q5), as *S-BJ* and *C-BJ*, respectively. Our pruning-based join methods are denoted as *S-PJ* (Q4) and *C-PJ* (Q6), respectively. In both



(a) vary the size of T: |T|. (b) vary $\max(\text{size}(S))$: C. (c) vary τ. (d) vary avg string length: μ_s.

Figure 3: String-level model, running time analysis, *Author1* dataset.



(a) vary the size of T: |T|. (b) vary $\max(\text{size}(S))$: C. (c) vary τ. (d) vary avg string length: μ_s.

Figure 4: String-level model, running time analysis, *Category* dataset.

models, we built clustered indices on the id columns in tables R, T, R_q, T_q, and secondary indices on the g columns in R_q, T_q.

An astute reader may observe an implicit underlying assumption when deriving the SQL queries for the string-level (Q4) and character-level (Q6) probabilistic string models. We have assumed that when performing a probabilistic string join over tables R and T on the probabilistic string attribute S, all pairs of records $\{(r_i, t_j) | r_i \in \mathtt{R} \wedge t_j \in \mathtt{T}\}$ which satisfy $\widehat{d}(r_i.S, t_j.S) \leq \tau$ also share at least one common q-gram. However, it is clear there may be some (r_i, t_j) pairs which satisfy $\widehat{d}(r_i.S, t_j.S) \leq \tau$ yet do not share any common q-grams. This issue could be easily addressed by adding one common "dummy" q-gram in the q-gram tables to all probabilistic strings, which is not equal to any existing q-grams in the database. We leave the details to the full version of the paper.

5.1 The string-level model

Recall that in order to derive a tighter lower bound on the EED in the string-level model (as shown in Theorem 2) by incorporating positional information of the probabilistic q-grams, we used the Markov inequality and the fractional knapsack problem to relax the LHS of (11). Such a relaxation clearly is too pessimistic. In practice, we observed that only considering those probabilistic q-grams whose positions are within 2τ is good enough, based on the hint from UB_τ. In fact, we can show that this yields a very tight lower bound on the EED except in some rare, contrived cases. This leads to a more efficient query (no need to solve the knapsack problem), as well as better pruning power. We can simply add one more condition, AND ABS($R_q.\ell - T_q.\ell$) $\leq 2\tau$, to the WHERE clause at line 3 in (Q3). We denote this algorithm as *S-PJ2* and compare its performance to *S-PJ* and *S-BJ*. In practice, we found that in all the experiments *S-PJ2* returns the correct set of results without missing any pair that should join, indicating that those contrived cases almost never happen in most applications. *S-PJ2* is a pure SQL implementation, except the edit distance UDF that has to be included.

Effects of the table size. We first study the scalability of all algorithms w.r.t. the input size, by varying |T| from $1,000$ to $20,000$, while fixing |R|. Figures 3(a) and 4(a) show the results for the *Author1* and *Category* datasets respectively. The trends are very similar on both datasets. All algorithms become more expensive when |T| increases, and they all incur higher costs in the *Category* dataset due to the longer strings it contains. In particular, on both datasets, *S-PJ2* is the best method and it is at least 1 order of magnitude faster than *S-BJ*. *S-PJ* is about 5 to 10 times better than *S-BJ*.

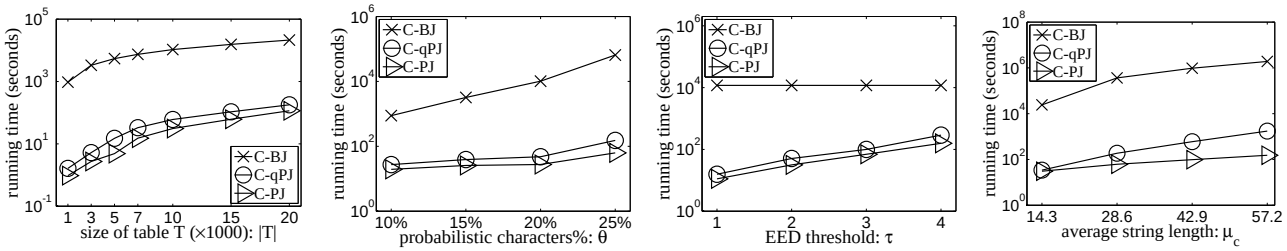
Effects of the maximum number of choices C. We next vary the maximum number of choices a probabilistic string may have, using $C = \max(\text{size}(S))$ from 2 to 10. The results from the *Author1* and *Category* datasets are shown in Figures 3(b) and 4(b). All algorithms become more costly when C increases. However, on both datasets, we observe that *S-PJ2* and *S-PJ* maintain their superior performance over *S-BJ* in all cases. They are more than 10 times or approximately 5 times better than the basic method, respectively.

Effects of the threshold τ. Figures 3(c) and 4(c) show the results on the *Author1* and *Category* datasets when τ changes from 1 to 4. Clearly, the cost of the basic method *S-BJ* stays as a constant, whereas the costs of *S-PJ2* and *S-PJ* increase when τ becomes larger. Naturally, larger τ values weakens the pruning power of our pruning techniques. Nevertheless, even in the worst case when $\tau = 4$, *S-PJ* and *S-PJ2* are still 3 or 6 times more efficient than *S-BJ* on both datasets.

Effects of the string length. Our last experiment demonstrates the impact of the string lengths. The results from the *Author1* and *Category* datasets, when ω changes from 0 to 3, are shown in Figures 3(d) and 4(d). Clearly, all algorithms require more time to finish. For the *S-BJ* method, the main driving force is the higher cost for each edit distance DP in every possible world. For the *S-PJ2* and *S-PJ* methods, an additional factor is that they have to process more probabilistic q-grams when string lengths increase. However, longer strings also imply a higher pruning power when τ is fixed, which means that fewer strings will join. Hence, the performance gap between *S-BJ* and the pruning based join methods actually has enlarged. For example, when $\mu_s = 54.4$ on the *Author1* dataset, *S-PJ2* (*S-PJ*) becomes more than 30 (10) times faster than *S-BJ*.

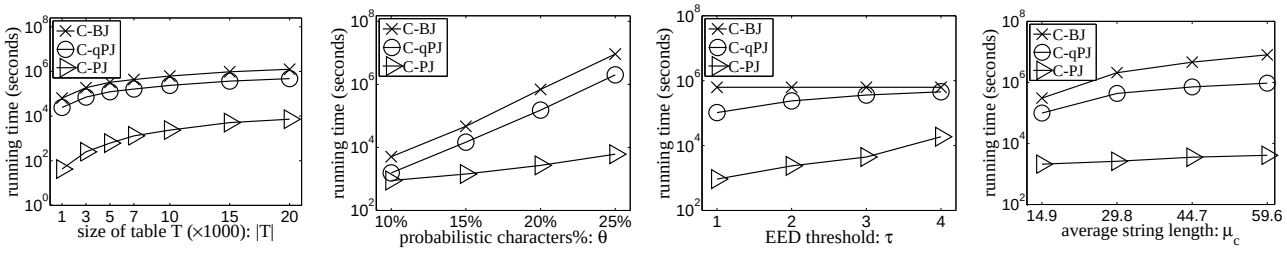
5.2 The character-level model

We performed string joins on our two character-level datasets *Author2* and *Genome*. A notable difference of these two datasets is their alphabet size. *Author2* has an alphabet size of more than 52 while *Genome* only has 4 letters ('A', 'T', 'G', 'C').



(a) vary the size of T: |T|. (b) vary θ. (c) vary τ. (d) vary avg string length: μ_c.

Figure 5: Character-level model, running time analysis, *Author2* dataset.



(a) vary the size of T: |T|. (b) vary θ. (c) vary τ. (d) vary avg string length: μ_c.

Figure 6: Character-level model, running time analysis, *Genome* dataset.

To show the usefulness of the DP based lower and upper bounds in the character-level model, in addition to *C-BJ* and *C-PJ*, we have also tested the string join method that uses only probabilistic q-gram based pruning, i.e., not calling `ld` and `ud` in lines 17 and 18 in `(Q6)`. We denote this query as *C-qPJ*.

Effects of the table size. Figures 5(a) and 6(a) show the scalability of various algorithms on the *Author2* and *Genome* datasets respectively, where we fix |R| and vary |T| from $1,000$ to $20,000$. Clearly, *C-PJ* is consistently at least 2 orders of magnitude better than the basic method *C-BJ*. Note that when *T* has $20,000$ probabilistic strings, *C-BJ* takes 11 days to finish on the *Genome* dataset. All methods are more costly on the *Genome* dataset. This is because even though the maximum size$(S[i])$ is set at $\chi = 6$, in the *Author2* dataset, most characters only have 1 or 2 choices, while in the *Genome* dataset, if a position is unknown, it almost always has 4 choices (but of course with different probabilities).

Comparing *C-qPJ* and *C-PJ*, we note that the extra cost of running the DP-based filters always pays off, although the amount of improvement is quite different on the two datasets. The DP-based filters are more effective, or one can say that the probabilistic q-gram based filters are less effective, on the *Genome* dataset. This is due to the crucial difference in their alphabet size. When the alphabet is small, many q-grams will match, which lowers their pruning power. On the contrary, with a large alphabet size, the q-grams are very selective, thus having a stronger pruning power. The same trends have been observed in the other experiments on these two datasets. Therefore, our recommendation is that the DP filters are essential when the alphabet is small; but when the alphabet is large, they can be dismissed if one wants a simpler, purely SQL implementation, except the edit distance UDF which has to be included to compute the exact EED for those pairs that cannot be pruned.

Effects of θ. Recall that θ is the fraction of probabilistic characters in a probabilistic string. Figures 5(b) and 6(b) show the results on *Author2* and *Genome* respectively, when θ changes from 10% to 25%. It is not surprising to see that with more probabilistic characters, the performance gap between *C-PJ* and *C-BJ* enlarges due to the exponential rate of increase of the number of possible worlds. When θ reaches 25%, *C-PJ* becomes 3 orders of magnitude more efficient than *C-BJ* on both datasets.

Effects of the threshold τ. Figures 5(c) and 6(c) show the results on the *Author2* and *Genome* datasets respectively, when τ changes from 1 to 4. Obviously, *C-BJ* is not affected by τ. The cost of *C-PJ* gradually increases as τ. Nevertheless, we see that *C-PJ* is still 2 orders of magnitude faster than *C-BJ* when τ is 4. An interesting fact to mention is that when τ becomes larger, our upper-bound filters become more effective, which, to some degree, offsets the decrease in the pruning power of the lower-bound filters.

Effects of the string length. We then tested all algorithms by varying the length of the probabilistic string using $\omega = 0, 1, 2$ and 3. To ensure that the basic method does not get excessively expensive, we limit the number of probabilistic characters in a probabilistic string to be $\min(20\% \cdot |S|, 3)$. The results from the *Author2* and the *Genome* datasets are shown in Figures 5(d) and 6(d) respectively. Clearly, the costs of all algorithms increase with longer strings, partly due to the fact that each DP takes more time to run. Another cause for the increasing costs of *C-qPJ* and *C-PJ* is the increasing number of probabilistic q-grams to process. On the other hand, note that the rates of the cost increase for *C-qPJ* and *C-PJ* are slower than for *C-BJ*. In fact, when the average string length increases to around 60, *C-PJ* becomes 3 orders of magnitude faster than *C-BJ*. This is because the pruning techniques also become more effective when the strings become longer while τ is fixed.

Effects of χ. Our study shows that *C-qPJ* and *C-PJ* outperform *C-BJ* by an increasingly larger margin when χ increases. The details were omitted due to the space constraint.

Size of the probabilistic q-gram table. Since the probabilistic q-gram table may be large, theoretically speaking, in the character-level model, we have examined the number of probabilistic q-grams that need to be stored. For reasons explained right after Definition 8, in practice the worst case almost never happens. In fact, in all of our experiments, the storage space (in bytes) of the probabilistic q-gram table is only a factor of approximately 5 times larger than the storage space (in bytes) of the probabilistic string table for $q = 2$.

6. RELATED WORK

We have reviewed q-gram based deterministic string joins [8, 13] in Section 2. Improvements to these methods were later proposed

using non-relational techniques [28]. Variable-length q-grams have also been proposed [19]. How to adapt these improvements to the probabilistic setting is an intriguing open problem.

The observation that a string attribute often can be represented by multiple choices has also been made in the literature [5], however, with a very different focus from our work. The main objective in [5] is to design transformation frameworks that allow users to express user-specified transformations on a set of deterministic strings, with the help of the combination of core similarity functions (such as the edit distance, Jaccard distance and etc).

Modeling fuzzy information from the underlying data source is a fundamental issue in probabilistic databases. An excellent discussion on this topic may be found in [24]. Several on-going projects, TRIO [1], MayBMS [4], MystiQ [6], PrDB [25], MCDB [14] and Orion [26], represent the active development in modeling, storing and processing probabilistic data. Nevertheless, query processing techniques specifically for probabilistic strings, especially for the string-join problem, have not been discussed.

Although this paper is the first to study probabilistic strings from a data management perspective, they have been used in other fields. In particular, Louchard and Szpankowski [21] studied the edit distance over two random strings, using a model that is a special case of the character-level model: they assumed that all characters in the two strings follow the same Bernoulli distribution on a limited alphabet, and that all characters are mutually independent. Under this model, they analyzed the statistical behavior of the edit distance when the string length goes to infinity. Their analysis does not carry to our (more general) character-level model, and also does not offer any help in computing or bounding the EED efficiently. Some other pattern matching problems on random sequences using the character-level model have also been considered (see the related work in [21]) but they are less relevant to this work.

Efficient join processing for probabilistic data for atomic data types (e.g., int, real) has been discussed in [2, 9, 16, 20], with different types of join conditions, but none of them has dealt with probabilistic strings. Sampling methods, such as the Monte Carlo approach [14], may be used to reduce the cost of the basic methods. However, this approach will only return approximate solutions. In contrast, our methods produce exact answers.

7. CONCLUSION

This work studies the important problem of efficient string joins in probabilistic string databases, using the natural definition of the expected edit distance. We present effective and efficient algorithms that could be easily implemented in existing relational database engines. Our study covers both the complete (string-level) and succinct (character-level) models of probabilistic strings, and leads to orders-of-magnitude improvements to the basic approach. This effort opens the gateway to many interesting future works, such as indexing probabilistic strings for similarity search, selectivity estimations, keyword search in probabilistic string databases, etc.

Another interesting and intriguing open problem is to study other string similarity measures in the probabilistic setting. This is especially important for some instances in the string-level model, where any probabilistic string S will have a large portion (w.r.t. the probability mass) of its choices having drastically different string values. Note that the same issue also exists for deterministic strings, i.e., there are cases where other similarity measures are more preferred than using the edit distance [5].

8. ACKNOWLEDGMENT

Feifei Li was partially supported by the NSF Grant IIS-0916488. Ke Yi was supported in part by Hong Kong Direct Allocation Grant DAG07/08. Jeffrey Jestes was supported by the GAANN Fellowship from the US Department of Education.

9. REFERENCES

[1] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.

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

[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, 2008.

[4] L. Antova, C. Koch, and D. Olteanu. Query language support for incomplete information in the MayBMS system. In *VLDB*, 2007.

[5] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *ICDE*, 2008.

[6] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*, 2005.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[8] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.

[9] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia. Efficient join processing over uncertain data. In *CIKM*, 2006.

[10] X. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, 2007.

[11] J. T. Dunnen and S. E. Antonarakis. Recommendations for the description of DNA sequence variants with uncertainties. www.genomic.unimelb.edu.au/mdi/mutnomen/uncertain.html.

[12] J. T. Dunnen and S. E. Antonarakis. Mutation nomenclature extensions and suggestions to describe complex mutations: A discussion. *Human Mutation*, 15(1):7–12, 2000.

[13] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.

[14] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[15] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, 2009.

[16] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *DASFAA*, 2006.

[17] H. Lee, R. T. Ng, and K. Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *VLDB*, 2007.

[18] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, 2008.

[19] C. Li, B. Wang, and X. Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, 2007.

[20] V. Ljosa and A. K. Singh. Top-k spatial joins of probabilistic objects. In *ICDE*, 2008.

[21] G. Louchard and W. Szpankowski. A probabilistic analysis of a string editing problem and its variations. *Combinatorics, Probability and Computing*, 4(02):143–166, 1995.

[22] A. Motro. *Uncertainty Management in Information Systems: From Needs to Solutions*. Kluwer Academic Publishers, 1997.

[23] R. Palacios and A. Gupta. A system for processing handwritten bank checks automatically. *Image and Vision Computing*, 26(10):1297–1313, 2008.

[24] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[25] P. Sen, A. Deshpande, and L. Getoor. PrDB: Managing and exploiting rich correlations in probabilistic databases. *VLDBJ*, 18(5):1065–1090, 2009.

[26] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *SIGMOD*, 2008.

[27] E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In *ESA*, 1995.

[28] C. Xiao, W. Wang, and X. Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. In *VLDB*, 2008.