Probabilistic Threshold Indexing for Uncertain Strings

Sharma Thankachan
Georgia Institute of Technology
Georgia, USA
thanks@csc.lsu.edu

Manish Patil
Louisiana State University
Louisiana, USA
manish.m.patil@gmail.com

Rahul Shah
Louisiana State University
Louisiana, USA
rahul@csc.lsu.edu

Sudip Biswas
Louisiana State University
Louisiana, USA
sudipid@gmail.com

ABSTRACT

Strings form a fundamental data type in computer systems. String searching has been extensively studied since the inception of computer science. Increasingly many applications have to deal with imprecise strings or strings with fuzzy information in them. String matching becomes a probabilistic event when a string contains uncertainty, i.e. each position of the string can have different probable characters with associated probability of occurrence for each character. Such uncertain strings are prevalent in various applications such as biological sequence data, event monitoring and automatic ECG annotations. We explore the problem of indexing uncertain strings to support efficient string searching. In this paper we consider two basic problems of string searching, namely substring searching and string listing. In substring searching, the task is to find the occurrences of a deterministic string in an uncertain string. We formulate the string listing problem for uncertain strings, where the objective is to output all the strings from a collection of strings, that contain probable occurrence of a deterministic query string. Indexing solution for both these problems are significantly more challenging for uncertain strings than for deterministic strings. Given a construction time probability value τ, our indexes can be constructed in linear space and supports queries in near optimal time for arbitrary values of probability threshold parameter greater than τ. To the best of our knowledge, this is the first indexing solution for searching in uncertain strings that achieves strong theoretical bound and supports arbitrary values of probability threshold parameter. We also propose an approximate substring search index that can answer substring search queries with an additive error in optimal time. We conduct experiments to evaluate the performance of our indexes.

1. INTRODUCTION

String indexing has been one of the key areas of computer science. Algorithms and data structures of string searching finds application in web searching, computational biology, natural language processing, cyber security, etc. The classical problem of string indexing is to preprocess a string such that query substring can be searched efficiently. Linear space data structures are known for this problem which can answer such queries in optimal $O(m + occ)$ time, where m is the substring length and occ is the number of occurrences reported.

Growth of the internet, digital libraries, large genomic projects have contributed to enormous growth of data. As a consequence, noisy and uncertain data has become more prevalent. Uncertain data naturally arises in almost all applications due to unreliability of source, imprecise measurement, data loss, and artificial noise. For example sequence data in bioinformatics is often uncertain and probabilistic. Sensor networks and satellites inherently gather noisy information.

Existing research has focused mainly on the study of regular or deterministic string indexing. In this paper we explore the problem of indexing uncertain strings. We begin by describing the uncertain string model, possible world semantics and challenges of searching in uncertain strings.

Current literature models uncertain strings in two different ways: the string level model and the character level model. In string level model, we look at the probabilities and enumerate at whole string level, whereas character level model represents each position as a set of characters with associated probabilities. We focus on the character level model which arises more frequently in applications. Let S be an uncertain string of length n. Each character c at position i of S has an associated probability $pr(c^i)$. Probabilities at different positions may or may not contain correlation among them. Figure 1(a) shows an uncertain string S of length 5. Note that, the length of an uncertain string is the total number of positions in the string, which can be less than the total number of possible characters in the string. For example, in Figure 1(a), total number of characters in string s with nonzero probability is 9, but the total number of positions or string length is only 5.

"Possible world semantics" is a way to enumerate all the possible deterministic strings from an uncertain string. Based on possible world semantics, an uncertain string S of length n can generate a deterministic string w by choosing one possible character from each position and concatenating them in order. We call w as one of the possible world for S. Probability of occurrence of

$w = w_1 w_2 \dots w_n$ is the partial product $pr(w_1^1) \times pr(w_2^2) \times \dots \times pr(w_n^n)$. The number of possible worlds for S increases exponentially with n. Figure 1(b) illustrates all the possible worlds for the uncertain string S with their associated probability of occurrence.

A meaningful way of considering only a fraction of the possible worlds is based on a probability threshold value τ. We consider a generated deterministic string $w = w_1 w_2 \dots w_n$ as a valid occurrence with respect to τ, only if it has probability of occurrence more than τ. The probability threshold τ effectively removes lower probability strings from consideration. Thus τ plays an important role to avoid the exponential blowup of the number of generated deterministic strings under consideration.

Character	S[1]	S[2]	S[3]	S[4]	S[5]
a	.3	.6	0	.5	1
b	.4	0	0	0	0
c	0	.4	0	.5	0
d	.3	0	1	0	0

(a) Uncertain string S

w	Prob(w)	w	Prob(w)	w	Prob(w)
w[1] aadaa	.09	w[5] badaa	.12	w[9] dadaa	.09
w[2] aadca	.09	w[6] badca	.12	w[10] dadca	.09
w[3] acdaa	.06	w[7] badca	.08	w[11] dcdaa	.06
w[4] acdca	.06	w[8] badca	.08	w[12] dcdca	.06

(b) Possible worlds of S

Figure 1: An uncertain string S of length 5 and its all possible worlds with probabilities.

Given an uncertain string S and a deterministic query substring $p = p_1 \dots p_m$, we say that p matched at position i of S with respect to threshold τ if $pr(p_1^i) \times \dots \times pr(p_m^{i+m-1}) \geq \tau$. Note that, $O(m + occ)$ is the theoretical lower bound for substring searching where m is the substring length and occ is the number of occurrence reported.

1.1 Formal problem definition

Our goal is to develop efficient indexing solution for searching in uncertain strings. In this paper, we discuss two basic uncertain string searching problems which are formally defined below.

PROBLEM 1. *Substring Searching: Given an uncertain string S of length n, our task is to index S so that when a deterministic substring p and a probability threshold τ come as a query, report all the starting positions of S where p is matched with probability of occurrence greater than τ.*

PROBLEM 2. *Uncertain String Listing: Let $\mathcal{D}$=$\{d_1,\dots,d_D\}$ be a collection of D uncertain strings of n positions in total. Our task is to index $\mathcal{D}$ so that when a deterministic substring p and a probability threshold τ come as a query, report all the strings d_j such that d_j contains atleast one occurrence of p with probability of occurrence greater than τ.*

Note that the string listing problem can be naively solved by running substring searching query in each of the uncertain string from the collection. However, this naive approach will take $O(\sum_{d_i \in \mathcal{D}}$ search time on $d_i)$ time which can be very inefficient if the actual number of documents containing the substring is small. Figure 2 illustrates an example for string listing. In this example, only the string d_1 contains query substring "BF" with probability of occurrence greater than query threshold 0.1. Ideally, the query time should be proportionate to the actual number of documents reported as output. Uncertain strings considered in both these problems can contain correlation among string positions.

String collection $\mathcal{D} = \{d_1, d_2, d_3\}$:

$d_1[1]$	$d_1[2]$	$d_1[3]$	$d_2[1]$	$d_2[2]$	$d_2[3]$	$d_3[1]$	$d_3[2]$	$d_3[3]$
A .4	B .3	F .5	A .6	B .5	B .4	A .4	I .3	A 1
B .3	L .3	J .5	C .4	F .3	C .3	F .4	L .3	
F .3	F .3			J .2	E .2	P .2	P .3	
	J .1				F .1		T .3	

Output of string listing query $(\text{"}BF\text{"}, 0.1)$ on $\mathcal{D}$ is: d_1

Figure 2: String listing from an uncertain string collection $\mathcal{D} = \{d_1, d_2, d_3\}$.

1.2 Challenges in uncertain string searching

We summarize some challenges of searching in uncertain strings.

- An uncertain string of length n can have multiple characters at each position. As the length of an uncertain string increases, the number of possible worlds grows exponentially. This makes a naive technique that exhaustively enumerates all possible worlds infeasible.
- Since multiple substrings can be enumerated from the same starting position, care should be taken in substring searching to avoid possible duplication of reported positions.
- Enumerating all the possible sequences for arbitrary probability threshold τ and indexing them requires massive space for large strings. Also note that, for a specific starting position in the string, the probability of occurrence of a substring can change arbitrarily (non-decreasing order) with increasing length, depending on the probability of the concatenated character This makes it difficult to construct index that can support arbitrary probability threshold τ.
- Correlated uncertainty among the string positions is not uncommon in applications. An index that handles correlation appeals to a wider range of applications. However, handling the correlation can be a bottleneck on space and time.

1.3 Related work

Although, searching over clean data has been widely researched, indexing uncertain data is relatively new. Below we briefly mention some of the previous works related to uncertain strings.

Algorithmic approach: Li et al. [19] tackled the substring searching problem where both the query substring and uncertain sequence comes as online query. They proposed a linear time and linear space dynamic programming approach to calculate the probability that a substring is contained in the uncertain string.

Approximate substring matching: Given as input a string p, a set of strings $\{x_i | 1 \leq i \leq r\}$, and an edit distance threshold k, the substring matching problem is to find all substrings s of x_i such that $d(p, s) \leq k$, where $d(p, s)$ is the edit distance between p and s. This problem has been well studied on clean texts (see [22] for a survey). Most of the ideas to solve this problem is based on partitioning p. Tiangjian et al. [12] extended this problem for uncertain strings. Their index can handle strings of arbitrary lengths.

Frequent itemset mining: Some articles discuss the problem of frequent itemset mining in uncertain databases [6, 7, 3], where an itemset is called frequent if the probability of occurrence of the itemset is above a given threshold.

Probabilistic database: Several works [5, 26, 25] have developed indexing techniques for probabilistic databases, based on R-trees and inverted indices, for efficient execution of nearest neigh-

bor queries and probabilistic threshold queries. Dalvi et al. [8] proposed efficient evaluation method for arbitrary complex SQL queries in probabilistic database. Later on efficient index for ranked top-k SQL query answering on a probabilistic database was proposed ([24, 18]). Kanagal et al. [16] developed efficient data structures and indexes for supporting inference and decision support queries over probabilistic databases containing correlation. They use a tree data structure named junction tree to represent the correlations in the probabilistic database over the tuple-existence or attribute-value random variables.

Similarity joins: A string similarity join finds all similar string pairs between two input string collections. Given two collections of uncertain strings R, S, and input (k, τ), the task is to find string pairs (r, s) between these collections such that $Pr(ed(R, S) \leq k) > \tau$ i.e., probability of edit distance between R and S being at most k is more than probability threshold τ. There are some works on string joins, e.g., [4, 13, 17], involving approximation, data cleaning, and noisy keyword search, which has been discussed in the probabilistic setting [15]. Patil et al. [23] introduced filtering techniques to give upper and (or) lower bound on $Pr(ed(R, S) \leq k)$ and incorporate such techniques into an indexing scheme with reduced filtering overhead.

1.4 Our approach

Since uncertain string indexing is more complex than deterministic string indexing, a general solution for substring searching is challenging. However efficiency can be achieved by tailoring the data structure based on some key parameters, and use the data structure best suited for the purposed application. We consider the following parameters for our index design.

Threshold parameter τ_{min}: The task of substring matching in uncertain string is to find all the probable occurrences, where the probable occurrence is determined by a query threshold parameter τ. Although τ can have any value between 0 to 1 at query time, real life applications usually prohibits arbitrary small value of τ. For example, a monitoring system does not consider a sequence of events as a real threat if the associated probability is too low. We consider a threshold parameter τ_{min}, which is a constant known at construction time, such that query τ does not fall below τ_{min}. Our index can be tailored based on τ_{min} at construction time to suit specific application needs.

Query substring length: The query substring searched in the uncertain string can be of arbitrary length ranging from 1 to n. However, most often the query substrings are smaller than the indexed string. An example is a sensor system, collecting and indexing big amount of data to facilitate searching for interesting query patterns, which are smaller compared to the data stream. We show that more efficient indexing solution can be achieved based on query substring length.

Correlation among string positions: Probabilities at different positions in the uncertain string can possibly contain correlation among them. In this paper we consider character level uncertainly model, where a probability of occurrence of a character at a position can be correlated with occurrence of a character at a different position. We formally define the correlation model and show how correlation is handled in our indexes.

Our approach involves the use of suffix trees, suffix arrays and range maximum query data structure, which to the best of our knowledge, is the first use for uncertain string indexing. Succinct and compressed versions of these data structures are well known to have good practical performance. Previous efforts to index uncertain strings mostly involved dynamic programming and lacked theoretical bound on query time. We also formulate the uncertain string listing problem. Practical motivation for this problem is given in Section 6. As mentioned before, for a specific starting position of an uncertain string, the probability of occurrence of a substring can change arbitrarily with increasing length depending on the probability of the concatenated character. We propose an approximate solution by discretizing the arbitrary probability changes with conjunction of a linking structure in the suffix tree.

1.5 Our contribution:

In this paper, we propose indexing solutions for substring searching in a single uncertain string, searching in a uncertain string collection, and approximate index for searching in uncertain strings. More specifically, we make the following contributions:

1. For the substring searching problem, we propose a linear space solution for indexing a given uncertain string S of length n, such that all the occurrences of a deterministic query string p with probability of occurrence greater than a query threshold τ can be reported. We show that for frequent cases our index achieves optimal query time proportional to the substring length and output size. Our index can be designed to support arbitrary probability threshold $\tau \geq \tau_{min}$, where τ_{min} is a constant known at index construction time.

2. For the uncertain string listing problem, given a collection of uncertain strings $\mathcal{D} = \{d_1, \ldots, d_D\}$ of total size n, we propose a linear space and near optimal time index for retrieving all uncertain strings that contain a deterministic query string p with probability of occurrence greater than a query threshold τ. Our index supports queries for arbitrary $\tau \geq \tau_{min}$, where τ_{min} is a constant known at construction time.

3. We propose an index for approximate substring searching, which can answer substring searching queries in uncertain strings for arbitrary $\tau \geq \tau_{min}$ in optimal $O(m + occ)$ time, where τ_{min} is a constant known at construction time and ϵ is the bound on desired additive error in the probability of a matched string, i.e. outputs can have probability of occurrence $\geq \tau - \epsilon$.

1.6 Outline

The rest of the paper is organized as follows. In section 2 we show some practical motivations for our indexes. In section 3 we give a formal definition of the problem, discuss some definitions related to uncertain strings and supporting tools used in our index. In section 4 we build a linear space index for answering a special form of uncertain strings where each position of the string has only one probabilistic character. In section 5 we introduce a linear space index to answer substring matching queries in general uncertain strings for variable threshold. Section 6 discusses searching in an uncertain string collection. In section 7, we discuss approximate string matching in uncertain strings. In section 8, we show the experimental evaluation of our indexes. Finally in section 9, we conclude the paper with a summary and future work direction.

2. MOTIVATION

Various domains such as bioinformatics, knowledge discovery for moving object database trajectories, web log analysis, text mining, sensor networks, data integration and activity recognition generates large amount of uncertain data. Below we show some practical motivation for our indexes.

Biological sequence data: Sequence data in bioinformatics is often uncertain and probabilistic. For instance, reads in shotgun sequencing are annotated with quality scores for each base. These quality scores can be understood as how certain a sequencing machine is about a base. Probabilities over long strings are also used to represent the distribution of SNPs or InDels (insertions and deletions) in the population of a species. Uncertainty can arise due to a number of factors in the high-throughput sequencing technologies. NC-IUB committee standardized incompletely specified bases in DNA to address this common presence of uncertainty [20]. Analyzing these uncertain sequences is important and more complicated than the traditional string matching problem.

We show an example uncertain string generated by aligning genomic sequence of Tree of At4g15440 from OrthologID and deterministic substring searching in the sequence. Figure 3 illustrates the example.

S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	S[9]	S[10]	S[11]
P 1	S .7	F 1	P 1	Q .5	P 1	A .4	I .3	A 1	S .5	A 1
	F .3			T .5		F .4	L .3		T .5	
						P .2	P .3			
							T .3			

Figure 3: Example of an uncertain string S generated by aligning genomic sequence of the tree of At4g15440 from OrthologID.

Consider the uncertain string S of Figure 3. A sample query can be $\{p = "AT", \tau = 0.4\}$, which asks to find all the occurrences of string AT in S having probability of occurrence more than $\tau = .4$. AT can be matched starting at position 7 and starting at position 9. Probability of occurrence for starting position 7 is $0.4 \times 0.3 = 0.12$ and for starting position 9 is $1 \times 0.5 = 0.5$. Thus position 9 should be reported to answer this query.

Automatic ECG annotations: In the Holter monitor application, for example, sensors attached to heart-disease patients send out ECG signals continuously to a computer through a wireless network([9]). For each heartbeat, the annotation software gives a symbol such as N (Normal beat), L (Left bundle branch block beat), and R, etc. However, quite often, the ECG signal of each beat may have ambiguity, and a probability distribution on a few possibilities can be given. A doctor might be interested in locating a pattern such as âĂIJNNAVâĂİ indicating two normal beats followed by an atrial premature beat and then a premature ventricular contraction, in order to verify a specific diagnosis. The ECG signal sequence forms an uncertain string, which can be indexed to facilitate deterministic substrings searching.

Event monitoring: Substring matching over event streams is important in paradigm where continuously arriving events are matched. For example a RFID-based security monitoring system produces stream of events. Unfortunately RFID devices are error prone and associate probability with the gathered events. A sequence of events can represent security threat. The stream of probabilistic events can be modeled with uncertain string and can be indexed so that deterministic substring can be queried to detect security threats.

3. PRELIMINARIES

3.1 Uncertain string and deterministic string

An uncertain string $S = s_1 \dots s_n$ over alphabet Σ is a sequence of sets $s_i, i = 1, \dots, n$. Every s_i is a set of pairs of the form $(c_j, pr(c_j^i))$, where every c_j is a character in Σ and $0 \leq pr(c_j^i) \leq 1$ is the probability of occurrence of c_j at position i in the string. Uncertain string length is the total number of positions in the string, which can be less than the total number of characters in the string. Note that, summation of probability for all the characters at each position should be 1, i.e. $\sum_j pr(c_j^i) = 1$. Figure 3 shows an example of an uncertain string of length 11. A deterministic string has only one character at each position with probability 1. We can exclude the probability information for deterministic strings.

3.2 Probability of occurrence of a substring in an uncertain string

Since each character in the uncertain string has an associated probability, a deterministic substring occurs in the uncertain string with a probability. Let $S = s_1 \dots s_n$ is an uncertain string and p is a deterministic string. If the length of p is 1, then probability of occurrence of p at position i of S is the associated probability $pr(p^i)$. Probability of occurrence of a deterministic substring $p = p_1 \dots p_k$, starting at a position i in S is defined as the partial product $pr(p_1^i) \times \dots \times pr(p_k^{i+k-1})$. For example in Figure 3, $SFPQ$ has probability of occurrence $0.7 \times 1 \times 1 \times 0.5 = 0.35$ at position 2.

3.3 Correlation among string positions

We say that character c_k at position i is correlated with character c_l at position j, if the probability of occurrence of c_k at position i is dependent on the probability of occurrence of c_l at position j. We use $pr(c_k^i)^+$ to denote the probability of c_k^i when the correlated character is present, and $pr(c_k^i)^-$ to denote the probability of c_k^i when the correlated character is absent. Let $x_g \dots x_h$ be a the substring generated from an uncertain string. c_k^i,$g \leq i \leq h$ is a character within the substring which is correlated with c_l^j. Depending on the position j, we have 2 cases:

Case 1, $g \leq j \leq h$ **:** The correlated probability of (c_k^i) is expressed by $(c_l^j \implies a\ ,\ \neg c_l^j \implies b)$, i.e. if c_l^j is taken as an occurrence, then $pr(c_k^i) = pr(c_k^i)^+$, otherwise $pr(c_k^i) = pr(c_k^i)^-$. We consider a simple example in Figure 4 to illustrate this. In this string, z^3 is correlated with e^1. For the substring eqz, $pr(z^3) = .3$, and for the substring fqz, $pr(z^3) = .4$.

Case 2, $j < g$ **or** $j > h$ **:** c_l^j is not within the substring. In this case, $pr(c_k^i)$=$pr(c_l^j)$*$pr(c_k^i)^+$+$(1 - pr(c_l^j))$*$pr(c_k^i)^+$. In Figure 4, for substring qz, $pr(z^3) = .6 * .3 + .4 * .4$.

S[1]	S[2]	S[3]
e: .6 f: .4	q: 1	z: $e^1 \implies .3, \neg e^1 \implies .4$

Figure 4: Uncertain string S with correlated characters.

3.4 Suffix tree and generalized suffix tree

The suffix tree [28, 21] of a deterministic string $t[1 \dots n]$ is a lexicographic arrangement of all these n suffixes in a compact trie structure of $O(n)$ words space, where the i-th leftmost leaf represents the i-th lexicographically smallest suffix of t. For a node i (i.e., node with pre-order rank i), $path(i)$ represents the text obtained by concatenating all edge labels on the path from root to node i in a suffix tree. The locus node i_p of a string p is the node closest to the root such that the p is a prefix of $path(i_P)$. The suffix range of a string p is given by the maximal range $[sp, ep]$ such that for $sp \leq j \leq ep$, p is a prefix of (lexicographically) j-th

smallest suffix of t. Therefore, i_p is the lowest common ancestor of sp-th and ep-th leaves. Using suffix tree, the locus node as well as the suffix range of p can be computed in $O(p)$ time, where p denotes the length of p. The suffix array A of t is defined to be an array of integers providing the starting positions of suffixes of S in lexicographical order. This means, an entry $A[i]$ contains the position of i-th leaf of the suffix tree in t. For a collection of strings $\mathcal{D} = \{d_1, \ldots, d_D\}$, let $t = d_1 d_2 \ldots d_D$ be the text obtained by concatenating all the strings in $\mathcal{D}$. Each string is assumed to end with a special character \$. The suffix tree of t is called the generalized suffix tree (GST) of $\mathcal{D}$.

4. STRING MATCHING IN SPECIAL UNCERTAIN STRINGS

In this section, we construct index for a special form of uncertain string which is extended later. Special uncertain string is an uncertain string where each position has only one probable character with associated non-zero probability of occurrence. Special-uncertain string is defined more formally below.

DEFINITION 1. *A special uncertain string $X = x_1 \ldots x_n$ over alphabet Σ is a sequence of pairs. Every x_i is a pair of the form $(c_i, pr(c_i^i))$, where every c_i is a character in Σ and $0 < pr(c_i^i) \leq 1$ is the probability of occurrence of c_i at position i in the string.*

Before we present an efficient index, we discuss a naive solution similar to deterministic substring searching.

4.1 Simple index

Given a special uncertain string $X = x_1 \ldots x_n$, construct the deterministic string $t = c_1 \ldots c_n$ where c_i is the character in x_i. We build a suffix tree over t. We build a suffix array A which maps each leaf of the suffix tree to its original position in t. We also build a successive multiplicative probability array C, where $C[j] = \prod_{i=1}^{j} Pr(c_i^i)$, for $j = 1, \ldots, n$. For a substring $x_i \ldots x_{i+j}$, probability of occurrence can be easily computed by $C[i+j]/C[i-1]$. Given an input (p, τ), we traverse the suffix tree for p and find the locus node and suffix range of p in $O(m)$ time, where m is the length of p. Let the suffix range be $[sp, ep]$. According to the property of suffix tree, each leaf within the range $[sp, ep]$ contains an occurrence of p in t. Original positions of the occurrence in t can be found using suffix array, i.e., $A[sp], \ldots, A[ep]$. However, each of these occurrence has an associated probability. We traverse each of the occurrence in the range $A[sp], \ldots, A[ep]$. For an occurrence $A[i]$, we find the probability of occurrence by $C[A[i] + m - 1]/C[A[i] - 1]$. If the probability of occurrence is greater than τ, we report the position $A[i]$ as an output. Figure 5 illustrates this approach.

Handling correlation: Correlation is handled when we check for the probability of occurrence. If $A[i]$ is a possible occurrence, then we need to consider any existing character within the substring $x_i \ldots x_{i+m-1}$, that is correlated with another character. Let c_k is a character at position j within $x_i \ldots x_{i+m-1}$, which is correlated with character $c_l^{j'}$, i.e. if $c_l^{j'}$ is included in the substring, then $pr(c_k^j)$=$pr(c_k^j)^+$, or else $pr(c_k^j)$=$pr(c_k^j)^-$. To find the correct probability of (c_k^j), if j' we check the j'-th position (j' depth character on the root to locus path in the suffix tree) of the substring. If the j'-th character is c_l, then $C[A[i] + m - 1]/C[A[i] - 1]$ is the correct probability of occurrence for $x_i \ldots x_{i+m}$. Otherwise, $C[A[i] + m - 1]/C[A[i] - 1]$ contains the incorrect probability of c_k^j. Dividing $C[A[i] + m - 1]/C[A[i] - 1]$ by $pr(c_k^j)^+$ and multiplying by $pr(c_k^j)^-$ gives the correct probability of occurrence

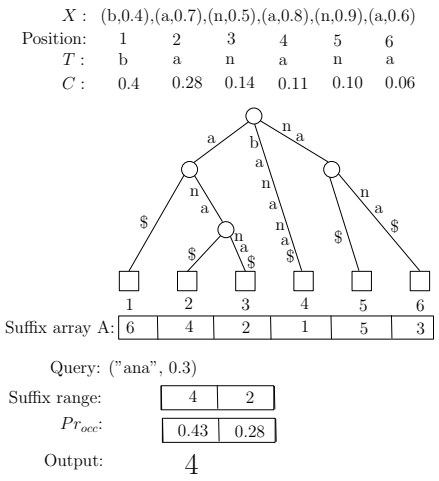

Figure 5: Simple index for special uncertain strings.

in this case. If c_l falls before or after the substring $x_i \ldots x_{i+m-1}$, $pr(c_k^j)$=$pr(c_l^{j'})$*$pr(c_k^j)^+$+$(1-pr(c_l^{j'}))$*$pr(c_k^j)^-$. Dividing $C[A[i]+m-1]/C[A[i]-1]$ by $pr(c_k^j)^+$ and multiplying by $pr(c_k^j)$ gives the correct probability of occurrence. Note that, we can identify and group all the characters with existing correlation, and search in the suffix tree in one scan for improved efficiency.

The main drawback in this approach is the query time. Within the suffix range $[sp, ep]$, possibly very few number of positions can qualify as output because of τ. So spending time on each element of the range $[sp, ep]$ is not justifiable.

4.2 Efficient index:

Bottleneck of the simple index comes from traversing each element within the suffix range. For the efficient index, we iteratively retrieve the element with maximum probability of occurrence in the range in constant time. Whenever the next maximum probability of occurrence falls below τ, we conclude our search. We use range maximum query (RMQ) data structure for our index which is briefly explained below.

Range Maximum Query: Let B be an array of integers of length n, a range maximum query(RMQ) asks for the position of the maximum value between two specified array indices $[i, j]$. i.e., the RMQ should return an index k such that $i \leq k \leq j$ and $B[k] \geq B[x]$ for all $i \leq x \leq j$. We use the result captured in following lemma for our purpose.

LEMMA 1. *[10, 11] By maintaining a $2n + o(n)$ bits structure, range maximum query(RMQ) can be answered in $O(1)$ time (without accessing the array).*

Every leaf of the suffix tree denotes a suffix position in the original text and a root to leaf path represents the suffix. For uncertain string, every character in this root to leaf path has an associated probability which is not stored in the suffix tree. Let y_j^i, for $j = 1, \ldots, n$ denote a deterministic substring which is the i-length prefix of the j-th suffix,i.e. the substring on the root to i-th leaf path. Let Y^i is the set of y_j^i, for $j = 1, \ldots, n$.

For $i = 1, \ldots, n$, we define C_i as the successive multiplicative probability array for the substrings of Y^i. j-th element of C_i is the successive multiplicative probability of the i-length prefix of the j-th suffix. More formally $C_i[j] = \prod_{k=A[j]}^{A[j]+i-1} Pr(c_k^k) = C[A[j] + i - 1]/C[A[j] - 1] (1 \leq j \leq n)$. For each $C_i (i = 1, \ldots, \log n)$ we

use range maximum query data structure RMQ_i of n bits over C_i and discard the original array C_i. We convert C_i into an integer array by multiplying each element by a sufficiently large number and then build the RMQ_i structure over it. We obtain $\log n$ number of such RMQ data structures resulting in total space of $O(n \log n)$ bits or $O(n)$ words. We also store the global successive multiplicative probability array C, where $C[j] = \prod_{i=1}^{j} Pr(c_i^i)$. Given a query (p, τ), idea is to use RMQ_i for iteratively retrieving maximum probability of occurrence elements in constant time each and validate using C. To maintain linear space, we can support query substring length of $m = 0, \ldots, \log n$ in this approach. Algorithm 1 illustrates the index construction phase for short substrings.

Query answering for short substrings ($m \leq \log n$): Given an input (p, τ), we first retrieve the suffix range $[l, r]$ in $O(m)$ time using suffix tree, where m is the length of p. We can find the maximum probability occurrence of p in $O(1)$ time by executing query $RMQ_m(l, r)$. Let max be the position of maximum probability occurrence and $max' = A[max]$ be the the original position in t. We can find the corresponding probability of occurrence by $C[max' + i - 1]/C[max' - 1]$. If the probability is less that τ, we conclude our search. If it is greater than τ, we report max' as an output. For finding rest of the outputs, we recursively search in the ranges $[l, max - 1]$ and $[max + 1, r]$. Since each call to $RMQ_m(l, r)$ takes constant time, validating the probability of occurrence takes constant time, we spend $O(1)$ time for each output. Total query time is optimal $O(m + occ)$. Algorithm 2 illustrates the query answering for short substrings. Note that, correlation is handled in similar way as described for the naive index, and we omit the details here.

Algorithm 1: Special-Short-Substring-Index-Construction

input : A special uncertain string X
output: suffix tree, suffix array A, successive multiplicative probability array C, RMQ_i $(i = 1, \ldots, \log n)$

Build deterministic string t from X
Build suffix tree over t
Build suffix array A over t
// Building successive multiplicative probability array
$C[1] = Pr(c_1^1)$
for $i = 2; i \leq n; i++$ **do**
  $C[i] = C[i-1] \times Pr(c_i^i)$
end
// Building C_i array for $i = 1, \ldots, \log n$
for $i = 1; i \leq \log n; i++$ **do**
  for $j = 1; j \leq n; j++$ **do**
    $C_i[j] = C[A[j] + i - 1]/C[A[j] - 1]$
    // Handling correlated characters
    for *all character* c_a^k *in* $t[A[j] \ldots t[A[j] + i - 1]$ *that are correlated with another character* c_b^l **do**
      if $(A[j] \leq l \leq [A[j] + i - 1]$ and c_b^l is not within $t[A[j] \ldots t[A[j] + i - 1])$
      $C_i[j] = C_i[j]/Pr(c_a^k)^+ * Pr(c_a^k)^-$
      else
      $pr(c_a^k)$=$pr(c_b^l)$*$pr(c_a^k)^+$+$(1 - pr(c_b^l))$*$pr(c_a^k)^-$
      $C_i[j] = C_i[j]/Pr(c_a^k)^+ * Pr(c_a^k)$
      end
    end
  end
  Build RMQ_i over C_i
end

Algorithm 2: Special-Short-Substring-Query-Answering

input : Query substring p, probability threshold τ)
output: Occurrence positions of p in X with probability of occurrence greater than τ

$m = length(p)$
call RecursiveRmq$(m, 1, n)$
  function RECURSIVERMQ(i, l, r) ▷ Recursive RMQ method
    $max = RMQ_m(l, r)$
    $max' = A[max]$
    if $C[max' + i - 1]/C[max' - 1] > \tau$ **then**
    Output max'
    Call $RecursiveRmq(m, l, max - 1)$
    Call $RecursiveRmq(m, max + 1, r)$
    end

Query answering for long substrings ($m > \log n$): We use a blocking scheme for answering long query substrings ($m > \log n$). Since exhaustively enumerating all possible substrings and storing the probabilities for each of them is infeasible, we only store selective probability values at construction time and compute the others at query time. We partition the entire suffix range of suffix array into different size blocks. More formally, for $i = \log n, \ldots, n$, we divide the suffix range $[1, n]$ of suffix array $A[1, n]$ into $O(n/i)$ number of blocks each of size i. Let B_i be the set of length i blocks, i.e. B_i=$\{[A[1]...A[i]], [A[i+1]...A[2i]], \ldots [A[n-i+1]...A[n]]\}$ and let B=$\{B_{\log n}, \ldots, B_n\}$. For a suffix starting at $A[j]$ and for B_i, we only consider the length i prefix of that suffix, i.e. $A[j \ldots j + i]$. The idea is to store only the maximum probability value per block. For $B_i, i = \log n, \ldots, n$, we define a probability array PB_i containing n/i elements. $PB_i[j]$ is the maximum probability of occurrence of all the substrings of length i belonging to the j-th block of B_i. We build a range maximum query structure RMQ_i for PB_i. RMQ_i takes $O(n/i)$ bits, total space is bounded by $\sum_i O(n/i) = O(n \log n)$ bits or $O(n)$ words.

For a query (p, τ), we first retrieve the suffix range $[l, r]$. This suffix range can spread over multiple blocks of B_m. We use RMQ_m to proceed to next step. Note that RMQ_m consists of N/m bits, corresponding to the N/m blocks of B_m in order. Our query proceeds by executing range maximum query in $RMQ_m(l, r)$, which will give us the index of the maximum probability element of string length m in that suffix range. Let the maximum probability element position in RMQ_m is max and the block containing this element is B_{max}. Using C array, we can find out if the probability of occurrence is greater than τ. Note that, we only stored one maximum element from each block. If the maximum probability found is greater than τ, we check all the other elements in that block in $O(m)$ time. In the next step, we recursively query $RMQ_m(l, max - 1)$ and $RMQ(max + 1, r)$ to find out subsequent blocks. Whenever RMQ query for a range returns an element having probability less than τ, we stop the recursion in that range. Number of blocks visited during query answering is equal to the number of outputs and inside each of those block we check m elements, obtaining total query time of $O(m \times occ)$.

In practical applications, query substrings are rarely longer than $\log n$ length. Our index achieves optimal query time for substrings of length less than $\log n$. We show in the experimental section that on average our index achieves efficient query time proportional to substring length and number of outputs reported.

5. SUBSTRING MATCHING IN GENERAL UNCERTAIN STRING

In this section we construct index for general uncertain string based on the index of special uncertain string. The idea is to convert a general uncertain string into a special uncertain string, build the data structure similar to the previous section and carefully eliminate the duplicate answers. Below we show the steps of our solution in details.

5.1 Transforming general uncertain string

We employ the idea of Amihood et al [1] to transform general uncertain string into a special uncertain string. **Maximal factor** of an uncertain string is defined as follows.

DEFINITION 2. *A **maximal factor** of a uncertain string S starting at location i with respect to a fixed probability threshold τ_c is a string of maximal length that when aligned to location i has probability of occurrence at least τ_c.*

For example in figure 3, maximal factors of the uncertain string S at location 5 with respect to probability threshold 0.15 are "QPA", "QPF", "TPA", "TPF".

An uncertain string S can be transformed to a special uncertain string by concatenating all the maximal factors of S in order. Suffix tree built over the concatenated maximal factors can answer substring searching query for a fixed probability threshold τ_c. But this method produces a special uncertain string of $\Omega(n^2)$ length, which is practically infeasible. To reduce the special uncertain string length, Amihood et al. [1] employs further transformation to obtain a set of extended maximal factors. Total length of the extended maximal factors is bounded by $O((\frac{1}{\tau_c})^2 n)$.

LEMMA 2. *Given a fixed probability threshold value $\tau_c (0 < \tau_c \leq 1)$, an uncertain string S can be transformed into a special uncertain string X of length $O((\frac{1}{\tau_c})^2 n)$ such that any deterministic substring p of S having probability of occurrence greater than τ_c is also a substring of X.*

Simple suffix tree structure for answering query does not work for the concatenated extended maximal factors. A special form of suffix tree, namely property suffix tree is introduced by Amihood et al. [1]. Also substring searching in this method works only on a fixed probability threshold τ_c. A naive way to support arbitrary probability threshold is to construct special uncertain string and property suffix tree index for all possible value of τ_c, which is practically infeasible due to space usage.

We use the technique of lemma 2 to transform a given general uncertain string to a special uncertain string of length $O((\frac{1}{\tau_{min}})^2 n)$ based on a probability threshold τ_{min} known at construction time, and employ a different indexing scheme over it. Let X be the transformed special uncertain string. A running example is shown in Appendix B in the full version of this paper [27]. Following section elaborates the subsequent steps of the index construction.

5.2 Index construction on the transformed uncertain string

Our index construction is similar to the index of section 4. We need some additional components to eliminate duplication and position transformation.

Let $N = |X|$ be the length of the special uncertain string X. Note that $N = O((\frac{1}{\tau_{min}})^2 n) = O(n)$, since τ_{min} is a constant known in construction time. For transforming the positions of X into the original position in S, we store an array Pos of size N, where $Pos[i]$=position of the i-th character of X in the original string S. We construct the deterministic string $t = c_1 \dots c_N$ where c_i is the character in X_i. We build a suffix tree over t. We build a suffix array A which maps each leaf of the suffix tree to its position in t. We also build a successive multiplicative probability array C, where $C[j] = \prod_{i=1}^{j} Pr(c_i^i)$, for $1 \leq j \leq N$. For a substring of length j starting at position i, probability of occurrence of the substring in X can be easily computed by $C[i+j-1]/C[i-1]$. For $i = 1, \dots, n$, we define C_i as the successive multiplicative probability array for substring length i i.e. $C_i[j] = \prod_{k=A[j]}^{A[j]+i-1} Pr(c_k^k) = C[A[j]+i-1]/C[A[j]-1]$ $(1 \leq j \leq n)$. Appendix B of the full version [27] shows Pos array and C array after transformation of an uncertain string. Below we explain how duplicates may arise in outputs and how to eliminate them.

Possible duplicate positions in the output arises because of the general to special uncertain string transformation. Note that, distinct positions in X can correspond to the same position in the original uncertain string S, resulting in same position possibly reported multiple times. A key observation here is that for two different substrings of length m, if the locus nodes are different than the corresponding suffix ranges are disjoint. These disjoint suffix ranges collectively cover all the leaves of the suffix tree. For each such disjoint ranges, we need to store probability values for only the unique positions of S. Without loss of generality we store the value for leftmost unique position in each range.

For any node u in the suffix tree, $depth(u)$ is the length of the concatenated edge labels from root to u. We define by L_i as the set of nodes u_i^j such that $depth(u_i^j) \geq i$ and $depth(parent(u_i^j)) \leq i$. For $L_i = u_i^1, \dots, u_i^k$, we have a set of disjoint suffix ranges $[sp_i^1, ep_i^1], \dots, [sp_i^k, ep_i^k]$. A suffix range $[sp_i^j, ep_i^j]$ can contain duplicate positions of S. Using the Pos array we can find the unique positions for each range and store only the values corresponding to the unique positions in C_i.

We use range maximum query data structure RMQ_i of n bits over C_i and discard the original array C_i. Note that, RMQ data structure can be built over an integer array. We convert C_i into an integer array by multiplying each element by a sufficiently large number and then build the RMQ_i structure over it. We obtain $\log n$ number of such RMQ data structures resulting in total space of $O(n \log n)$ bits or $O(n)$ words. For long substrings ($m > \log n$), we use the blocking data structure similar to section 4. Detailed construction phase is shown in Algorithm 3 of Appendix A in the full version of this paper [27].

5.3 Query answering

Query answering procedure is almost similar to the query answering procedure of section 4. Only difference being the transformation of position which is done using the Pos array. Detailed query answering Algorithm for short query substrings is included in Appendix A of the full version of this paper [27]. See Appendix B for an illustrative example of query answering.

5.4 Space complexity

For analyzing the space complexity, we consider each component of our index. Length of the special uncertain string X and deterministic string t are $O(n)$, where n is the number of positions in S. Suffix tree and suffix tree each takes linear space. We store a successive probability array of size $O(n)$. We build probability array C_i for $i = 1, \dots, \log n$, where each C_i takes of $O(n)$. However we build RMQ_i of n bits over C_i and discard the original array C_i. We obtain $\log n$ number of such RMQ data structures resulting in total space of $O(n \log n)$ bits or $O(n)$ words. For the blocking scheme, we build RMQ_i data structure for

$i = \log n, \ldots, n$. RMQ_i takes n/i bits, total space is $\sum_i n/i = O(n \log n)$ bits or $O(n)$ words. Since each component of our index takes linear space, total space taken by our index is $O(n)$ words.

5.5 Proof of correctness

In this section we discuss the correctness of our indexing scheme.

Substring conservation property of the transformation: At first we show that any substring of S with probability of occurrence greater than query threshold τ can be found in t as well. According to lemma 2, a substring having probability of occurrence greater than τ_{min} in S is also a substring of the transformed special uncertain string X. Since query threshold value τ is greater than τ_{min}, and entire character string of X is same as the deterministic string t, a substring having probability of occurrence greater than query threshold τ in S will be present in the deterministic string t.

Complete set of ocurrences are outputted: For contradiction, we assume that an occurrence position z of substring p in S having probability of occurrence greater than τ is not included in the output. From the aforementioned property, p is a substring of t. According to the property of suffix tree, z must be present in the suffix range $[sp, ep]$ of p. Using RMQ structure, we report all the occurrence in $[sp, ep]$ in their decreasing order of probability of occurrence value in S and stop when the probability of occurrence falls below τ, which ensures inclusion of z.

No incorrect occurrence appears in output: An output z can be incorrect occurrence if it is not present in uncertain string S or its probability of occurrence is less than τ. We query only the occurrences in the suffix range $[sp, ep]$ of p, according to the property of suffix tree all of which are valid occurrences. We also validate the probability of occurrence for each of them using the successive multiplicative probability array C.

6. STRING LISTING FROM UNCERTAIN STRING COLLECTION

In this section we propose an indexing solution for problem 2. We are given a collection of D uncertain strings $\mathcal{D} = \{d_1, \ldots, d_D\}$ of n positions in total. Let i denotes the string identifier of string d_i. For a query (p, τ), we have to report all the uncertain string identifiers j such that d_j contains p with probability of occurrence more than τ. In other words, we want to list the strings from a collection of a string, that are relevant to a deterministic query string based on probability parameter.

Relevance metric: For a deterministic string t and an uncertain string S, we define a relevance metric, $Rel(S, t)$. If t does not have any occurrence in S, then $Rel(S, t)$=0. If s has only one occurrence of t, then $Rel(S, t)$ is the probability of occurrence of t in S. If s contains multiple occurrences of t, then $Rel(S, t)$ is a function of the probability of occurrences of t in S. Depending on the application, various functions can be chosen as the appropriate relevance metric. A common relevance metric is the maximum probability of occurrence, which we denote by $Rel(S, t)_{max}$. The OR value of the probability of occurrences is another useful relevance metric. More formally, if a deterministic string t has nonzero probable occurrences at positions $i_1, \ldots, i_k$ of an uncertain string S, then we define the relevance metric of t in S as $Rel(S, t)_{OR}$ = $\sum_{j=i_1}^{i_k} pr(t_j) - \prod_{j=i_1}^{i_k} pr(t_j)$, where $pr(t_j)$ is the probability of occurrence of t in S at position j. Figure 6 shows an example.

Practical motivation: Uncertain string listing finds numerous practical motivation. Consider searching for a virus pattern in a collection of text files with fuzzy information. The objective is to quarantine the files that contain the virus pattern. This problem can be modeled as a uncertain string listing problem, where the collection of text files is the uncertain string collection D, the virus pattern is the query pattern P, and τ is the confidence of matching. Similarly, searching for a gene pattern in genomic sequences of different species can be solved using uncertain string listing data structure.

Uncertain string S:

$S[1]$	$S[2]$	$S[3]$	$S[4]$	$S[5]$	$S[6]$
A .4	B .3	A .5	A .6	B .5	A .4
B .3	L .3	F .5	B .4	F .3	C .3
F .3	F .3			J .2	E .2
	J .1				F .1

$Rel(S, "BFA")_{max}$=.09
$Rel(S, "BFA")_{OR}$=$(.06 + .09 + .048) - (.06 * .09 * .048)$
=.19786

Figure 6: Relevance metric for string listing.

The index: As explained before, a naive search on each of the string will result in $O(\sum_i$ search time on $d_i)$ which can be much larger than the actual number of strings containing the string. Objective of our index is to spend only one search time and time proportional to the number of output strings. We construct a generalized suffix tree so that we have to search for the string only once. We concatenate $d_1, \ldots, d_D$ by a special symbol which is not contained in any of the document and obtain a concatenated general uncertain string $S = d_1\$ \ldots \$d_D$. Next we use the transformation method described in previous section to obtain deterministic string t, construct suffix tree and suffix array for t. According to the property of suffix tree, the leaves under the locus of a query substring t contains all the occurrence positions of t. However, these leaves can possibly contain duplicate positions and multiple occurrence of the same document. In the query answering phase, duplicate outputs can arise because of the following two reasons:

1. Distinct positions in t can correspond to the same position in the original uncertain string S
2. Distinct positions in S can correspond to the same string identifier d_j which should be reported only once

Duplicate elimination is important to keep the query time proportional to the number of output strings. At first we construct the successive multiplicative probability array C_i similar to the substring searching index, then show how to incorporate $Rel(S, t)$ value for the multiple occurrences cases in the same document and duplicate elimination.

Let y_j^i, for $j = 1, \ldots, n$ denote a deterministic substring which is the i-length prefix of the j-th suffix,i.e. the substring on the root to i-th leaf path. Note that, multiple y_j^i can belong to the same locus node in the suffix tree. Let Y^i is the set of y_j^i, for $j = 1, \ldots, n$. The i-depth locus nodes in the suffix tree constitutes disjoint partitions in Y^i. For $i = 1, \ldots, n$, we define C_i as the successive multiplicative probability array for the substrings of Y^i. j-th element of C_i is the successive multiplicative probability of the i-length prefix of the j-th suffix. More formally $C_i[j] = \prod_{k=A[j]}^{A[j]+i-1} Pr(c_k^k) = C[A[j] + i - 1]/C[A[j] - 1] (1 \leq j \leq n)$.

The i-depth locus nodes in the suffix tree constitutes disjoint partitions in C_i. Let u be a i-depth locus node having suffix range $[j \ldots k]$ and root to u substring t. Then the partition $C_i[j \ldots k]$ belongs to u. For this partitions, we store only one occurrence of

a string d_j with the relevance metric value $Rel(S,t)$, and discard the other occurrences of d_j in that range. We build RMQ structure similar to section 5.

Query answering: We explain the query answering for short substrings. Blocking scheme described in previous section can be used for longer query substrings. Given an input (p,τ), we first retrieve the suffix range $[l,r]$ in $O(m)$ time using suffix tree, where m is the length of p. We can find the maximum relevant occurrence of p in $O(1)$ time by executing query $RMQ_m(l,r)$. Let max be the position of maximum relevant occurrence and $max' = A[max]$ be the the original position in t. For relevance metric $Rel(S,t)_{max}$, we can find the corresponding probability of occurrence by $C[max'+i-1]/C[max'-1]$. In case of the other complex relevance metric, all the occurrences need to be considered to retrieve the actual value of $Rel(S,t)$. If the relevance metric is less that τ, we conclude our search. If it is greater than τ, we report max' as an output. For finding rest of the outputs, we recursively search in the ranges $[l, max-1]$ and $[max+1, r]$. Each call to $RMQ_m(l,r)$ takes constant time. For simpler relevance metrics (such as $Rel(S,t)_{max}$), validating the relevance metric takes constant time. Total query time is optimal $O(m+occ)$. However, for more complex relevance metric, all the occurrences of t might need to be considered, query time will be proportionate to the total number of occurrences.

7. APPROXIMATE SUBSTRING SEARCHING

In this section we introduce an index for approximate substring matching in an uncertain string. As discussed previously, several challenges of uncertain string matching makes it harder to achieve optimal theoretical bound with linear space. We have proposed index for exact matching which performs near optimally in practical scenarios, but achieves theoretical optimal bound only for shorter query strings. To achieve optimal theoretical bounds for any query, we propose an approximate string matching solution. Our approximate string matching data structure answers queries with an additive error ϵ, i.e. outputs can have probability of occurrence $\geq \tau - \epsilon$.

At first we begin by transforming the uncertain string S into a special uncertain string X of length $N = O((\frac{1}{\tau_{min}})^2 n)$ using the technique of lemma 2 with respect to a probability threshold value τ_{min}. We obtain a deterministic string t from X by concatenating the characters of X. We build a suffix tree for t. Note that, each leaf in the suffix tree has an associated probability of occurrence $\geq \tau_{min}$ for the corresponding suffix. Given a query p, substring matching query for threshold τ_{min} can now be answered by simply scanning the leafs in subtree of locus node i_p. We first describe the framework (based on Hon et. al. [14]) which supports a specific probability threshold τ and then extend it for arbitrary $\tau \geq \tau_{min}$.

We begin by marking nodes in the suffix tree with positional information by associating $Pos_{id} \in [1,n]$. Here, Pos_{id} indicates the starting position in the original string S. A leaf node l is marked with a $Pos_{id} = d$ if the suffix represented by l begins at position d in S. An internal node u is marked with d if it is the lowest common ancestor of two leaves marked with d. Notice that a node can be marked with multiple position ids. For each node u and each of its marked position id d, define a link to be a triplet $(origin, target, Pos_{id})$, where $origin = u$, target is the lowest proper ancestor of u marked with d, and $Pos_{id} = d$. Two crucial properties of these links are listed below.

- Given a substring p, for each position d in S where p matches with probability $\geq \tau_{min}$, there is a unique link whose origin is in the subtree of i_p and whose target is a proper ancestor of i_p, i_p being the locus node of substring p.
- The total number of links is bounded by $O(N)$.

Thus, substring matching query with probability threshold τ_{min} can now be answered by identifying/reporting the links that originate in the subtree of i_p and are targeted towards some ancestor of it. By referring to each node using its pre-order rank, we are interested in links that are stabbed by locus node i_p. Such queries can be answered in $O(m+occ)$, where $|p| = m$ and occ is the number of answers to be reported (Please refer to [14] for more details).

As a first step towards answering queries for arbitrary $\tau \geq \tau_{min}$, we associate probability information along with each link. Thus each link is now a quadruple $(origin, target, Pos_{id}, prob)$ where first three parameters remain same as described earlier and $prob$ is the probability of $prefix(u)$ matching uncertain string S at position $Pos_{id} = d$. It is evident that for substring p and arbitrary $\tau \geq \tau_{min}$, a link stabbed by locus node i_p with $prob \geq \tau$ corresponds to an occurrence of p in S at position d with probability $\geq \tau$. However, a link stabbed by i_p with $prob < \tau$ can still produce an outcome since $prefix(i_P)$ contains additional characters not included in p, which may be responsible for matching probability to drop below τ. Even though we are interested only in approximate matching this observation leads up the next step towards the solution. We partition each link $(origin = u, target = v, Pos_{id} = d, prob)$ into multiple links $(or_1 = u, tr_1, d, prob_1)$, $(or_2 = tr_1, tr_2, d, prob_2)$, ..., $(or_k = tr_{k-1}, tr_k = v, d, prob_k)$ such that $prob_j - prob_{j-1} \leq \epsilon$ for $2 \leq j \leq k$. Here $or_2, \ldots, or_k$ may not refer to the actual node in the suffix tree, rather it can be considered as a dummy node inserted in-between an edge in suffix tree. In essence, we move along the path from node $u = or_1$ towards its ancestors one character at a time till the probability difference is bounded by ϵ i.e., till we reach node tr_1. The process then repeats with tr_1 as the origin node and so on till we reach the node v. It can be see that the total number of links can now be bounded by $O(N/\epsilon)$. In order to answer a substring matching query with threshold $\tau \geq \tau_{min}$, we need to retrieve all the links stabbed by i_p with $prob \geq \tau$. Occurrence of substring p in S corresponding to each such link is then guaranteed to have its matching probability at-least $\tau - \epsilon$ due to the way links are generated (for any link with (u,v) as origin and target probability of $prefix(v)$ matching in S can be more than that of $prefix(v)$ only by ϵ at the most).

8. EXPERIMENTAL EVALUATION

In this section we evaluate the performance of our substring searching and string listing index. We use a collection of query substrings and observe the effect of varying the key parameters. Our experiments show that, for short query substrings, uncertain string length does not affect the query performance. For long query substrings, our index fails to achieve optimal query time. However this does not deteriorate the average query time by big margin, since the probability of match also decreases significantly as substring gets longer. Index construction time is proportional to uncertain string size and probability threshold parameter τ_{min}.

We have implemented the proposed indexing scheme in C++. The experiments are performed on a 64-bit machine with an Intel Core i5 CPU 3.33GHz processor and 8GB RAM running Ubuntu. We present experiments along with analysis of performance.

8.1 Dataset

We use a synthetic datasets obtained from their real counterparts. We use a concatenated protein sequence of mouse and human (alphabet size $|\Sigma| = 22$), and break it arbitrarily into shorter strings.

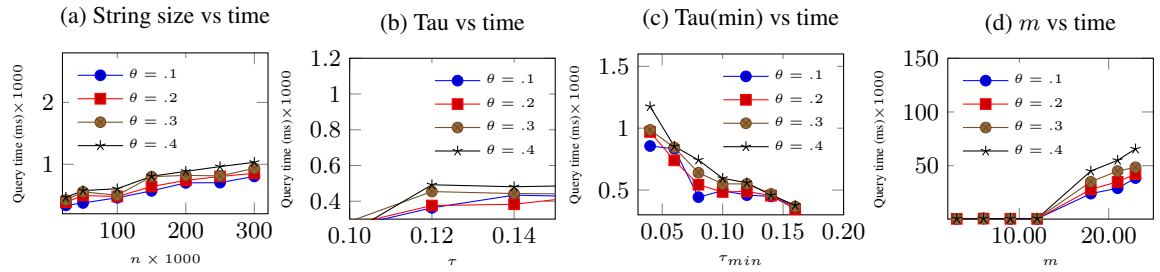

Figure 7: Substring searching query time for different string lengths (n), query threshold value τ, construction time threshold parameter τ_{min} and query substring length m.

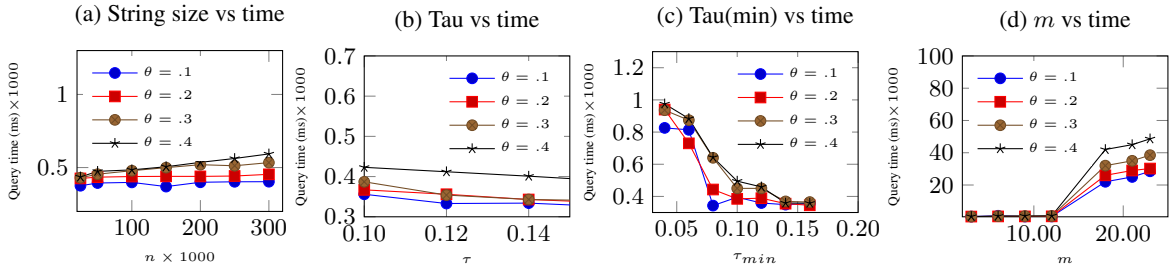

Figure 8: String listing query time for different string lengths (n), query threshold value τ, construction time threshold parameter τ_{min} and query substring length m.

For each string s in the dataset we first obtain a set $A(s)$ of strings that are within edit distance 4 to s. Then a character-level probabilistic string S for string s is generated such that, for a position i, the pdf of $S[i]$ is based on the normalized frequencies of the letters in the i-th position of all the strings in $A(s)$. We denote by θ the fraction of uncertain characters in the string. θ is varied between 0.1 to 0.5 to generate strings with different degree of uncertainty. The string length distributions in this dataset roughly follows a normal distribution in the range of $[20, 45]$. The average number of choices that each probabilistic character $S[i]$ may have is set to 5.

8.2 Query time for different string lengths (n) and fraction of uncertainty (θ)

We evaluate the query time for different string lengths n, ranging from $2K$ to $300K$ and θ ranging from 0.1 to 0.5. Figure 7(a) and Figure 8(a), shows the query times for substring searching and string listing. Note that, n is number of positions in the uncertain string where each position can have multiple characters. We take the average time for query lengths of 10,100,500,1000. We use $\tau_{min} = 0.1$ and query threshold $\tau = 0.2$. As shown in the figures, query times does not show much irregularity in performance when the length of string goes high. This is because for shorter query length, our index achieves optimal query time. Although for longer queries, our index achieves $O(m \times occ)$ time, longer query strings probability of occurrence gets low as string grows longer resulting in less number of outputs. However when fraction of uncertainty(θ) increases in the string, performance shows slight decrease as query time increases slightly. This is because longer query strings are more probable to match with strings with high level of uncertainty.

8.3 Query time for different τ and fraction of uncertainty (θ)

In Figure 7(b) and Figure 8(b), we show the average query times for string matching and string listing for probability threshold $\tau = 0.04, 0.06, 0.08, 0.1, 0.12$ for fixed $\tau_{min} = 0.1$. In terms of performance, query time increases with decreasing τ. This is because more matching is probable for smaller τ. Larger τ reduces the output size, effectively reducing the query time as well.

8.4 Query time for different τ_{min} and fraction of uncertainty (θ)

In Figure 7(c) and Figure 8(c), we show the average query times for string matching and string listing for probability threshold $\tau_{min} = 0.04, 0.06, 0.08, 0.1, 0.12$ which shows slight impact of τ_{min} over query time.

8.5 Query time for different substring lengths (m) and fraction of uncertainty (θ)

In figure 7(d) and figure Figure 8(d), we show the average query times for string matching and string listing. As it can be seen long pattern length drastically increases the query time.

8.6 Construction time for different string lengths and fraction of uncertainty (θ)

Figure 9(a) shows the index construction times for uncertain string length n ranging from $2K$ to $300K$. We can see that the construction time is proportional to the string length n. Increasing uncertainty factor θ also impacts the construction time as more permutation is possible with increasing uncertain positions. Figure 9(b) shows the impact of θ on construction time.

8.7 Space usage

Theoretical bound for our index is $O(n)$. However, this bound can have hidden multiplicative constant. Here we elaborate more on the actual space used for our index.

For our indexes, we construct the regular string t of length $N = O((\frac{1}{\tau_{min}})^2 n)$ by concatenating all the extended maximal factors based on threshold τ_{min}. We do not store the string t in our index.

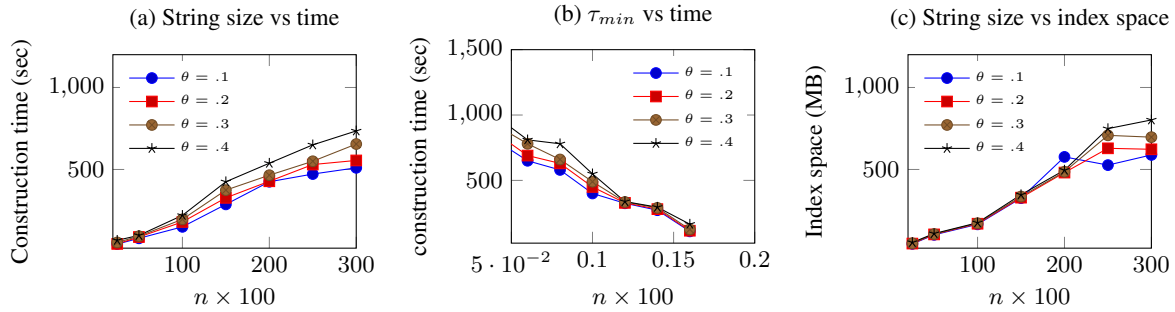

Figure 9: Construction time and index space for different string lengths (n) and probability threshold $\tau_{min} = .1$

We built RMQ structures RMQ_i for $i = 1, \ldots, \log n$ which takes $O(N \log n)$ bits. The practical space usage of RMQ is usually very small with hidden multiplicative constant of $2 - 3$. So the average space usage of our RMQ structure in total can be stated as $3N$ words. For a query string p, we find the suffix range of p in the concatenated extended maximum factor string t. For this purpose, instead of using Generalized Suffix Tree(GST), we use its space efficient version i.e., a compressed suffix array (CSA) of t. There are many versions of CSA's available in literature. For our purpose we use the one in [2] that occupies $N \log \sigma + o(N \log \sigma) + O(N)$ bits space and retrieves the suffix range of query string p in $O(p)$ time. In practice, this structure takes about $2.5N$ words space. We also store an array D of size N storing the partial probabilities, which takes approximately $4N$ bytes of space. Finally Pos array is used for position transformation, taking N words space. Summing up all the space usage, our index takes approximately $3N + 2.5N + 4N + N = 10.5N = (\frac{1}{\tau_{min}})^2 10.5n$. Figure Figure 9(c) shows the space usage for different string length(n) and θ.

9. CONCLUSIONS

In this paper we presented indexing framework for searching in uncertain strings. We tackled the problem of searching a deterministic substring in uncertain string and proposed both exact and approximate solution. We also formulated the uncertain string listing problem and proposed index for string listing from a uncertain string collection. Our indexes can support arbitrary values of probability threshold parameter. Uncertain string searching is still largely an unexplored area. Constructing more efficient index, variations of the string searching problem satisfying diverse query constraints are some interesting future work direction.

10. REFERENCES

[1] A. Amir, E. Chencinski, C. S. Iliopoulos, T. Kopelowitz, and H. Zhang. Property matching and weighted matching. *Theor. Comput. Sci.*, 395(2-3):298–310, 2008.

[2] D. Belazzougui and G. Navarro. Alphabet-independent compressed text indexing. In *ESA*, pages 748–759, 2011.

[3] T. Bernecker, H. Kriegel, M. Renz, F. Verhein, and A. Züfle. Probabilistic frequent pattern growth for itemset mining in uncertain databases. In *Scientific and Statistical Database Management - 24th International Conference, SSDBM 2012, Chania, Crete, Greece, June 25-27, 2012. Proceedings*, pages 38–55, 2012.

[4] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 5, 2006.

[5] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 876–887. VLDB Endowment, 2004.

[6] C. K. Chui and B. Kao. A decremental approach for mining frequent itemsets from uncertain data. In *Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings*, pages 64–75, 2008.

[7] C. K. Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *Advances in Knowledge Discovery and Data Mining, 11th Pacific-Asia Conference, PAKDD 2007, Nanjing, China, May 22-25, 2007, Proceedings*, pages 47–58, 2007.

[8] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544, 2007.

[9] S. Dash, K. Chon, S. Lu, and E. Raeder. Automatic real time detection of atrial fibrillation. *Annals of biomedical engineering*, 37(9):1701–1709, 2009.

[10] J. Fischer and V. Heun. A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array. In *ESCAPE*, pages 459–470, 2007.

[11] J. Fischer, V. Heun, and H. M. Stühler. Practical Entropy-Bounded Schemes for $O(1)$-Range Minimum Queries. In *IEEE DCC*, pages 272–281, 2008.

[12] T. Ge and Z. Li. Approximate substring matching over uncertain strings. *PVLDB*, 4(11):772–782, 2011.

[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, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 491–500, 2001.

[14] W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 713–722. IEEE, 2009.

[15] J. Jestes, F. Li, Z. Yan, and K. Yi. Probabilistic string similarity joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 327–338, 2010.

[16] B. Kanagal and A. Deshpande. Indexing correlated probabilistic databases. In *Proceedings of the 2009 ACM*

SIGMOD International Conference on Management of data, pages 455–468. ACM, 2009.

[17] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 257–266, 2008.

[18] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *The VLDB JournalŮThe International Journal on Very Large Data Bases*, 20(2):249–275, 2011.

[19] Y. Li, J. Bailey, L. Kulik, and J. Pei. Efficient matching of substrings in uncertain sequences. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 767–775, 2014.

[20] D. M. Lilley, R. M. Clegg, S. Diekmann, N. C. Seeman, E. Von Kitzing, and P. J. Hagerman. Nomenclature committee of the international union of biochemistry and molecular biology (nc- iubmb) a nomenclature of junctions and branchpoints in nucleic acids recommendations 1994. *European Journal of Biochemistry, s. FEBS J*, 230(1):1–2, 1996.

[21] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

[22] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[23] M. Patil and R. Shah. Similarity joins for uncertain strings. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1471–1482. ACM, 2014.

[24] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 886–895. IEEE, 2007.

[25] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. Hambrusch. Indexing uncertain categorical data. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 616–625. IEEE, 2007.

[26] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of the 31st international conference on Very large data bases*, pages 922–933. VLDB Endowment, 2005.

[27] S. Thankachan, M. Patil, R. Shah, and S. Biswas. Probabilistic threshold indexing for uncertain strings full version. `http://csc.lsu.edu/~sbiswas/papers/uncertainIndexingFullpaper.pdf`.

[28] P. Weiner. Linear pattern matching algorithms. In *SWAT (FOCS)*, pages 1–11, 1973.