Online Stochastic Pattern Matching

Marco Cognetta and Yo-Sub Han

Department of Computer Science, Yonsei University, Seoul, Republic of Korea,
{mcognetta, emmous}@yonsei.ac.kr

Abstract. The pattern matching problem is to find all occurrences of a given pattern in an input text. In particular, we consider the case when the pattern is a stochastic regular language where each pattern string has its own probability. Our problem is to find all matching patterns—(start, end) indices in the text—whose probability is larger than a given threshold probability. A pattern matching procedure is frequently used on streaming data in several applications, and often it is very challenging to find the start index of a matching in streaming data. We design an efficient algorithm for the stochastic pattern matching problem over streaming data based on the transformation of the pattern PFA into a weighted automaton and a constant bound on the number of backtracks required to find a start index while reading the streaming input. We also employ heuristics that enable us to reduce the number of backtracks, which improves the practical runtime of our algorithm. We establish the tight theoretical runtime of the proposed algorithm and experimentally demonstrate its practical performance. Finally, we show a possible application of our algorithm to another stochastic pattern matching problem where we search for the maximum probability substring of a text that is a superstring of a specified string.

Keywords: stochastic pattern matching, probabilistic automata, weighted automata, online matching

1 Introduction

Given a text T and a pattern P, the basic pattern matching problem is to determine whether or not P appears in T [11]. There are several variants for the pattern matching problem; for example, if P is a finite set of strings, we have a keyword pattern matching problem [1]. If P is described by a regular expression, then the problem becomes the regular expression pattern matching problem [14]. There are also several types of problems depending on what we want to report; if we only look for an end index of a matching pattern in T, which is common in `grep`-like applications, then often the problem is easier compared with the case when we want to find the exact start and end indices of each matching pattern in T.

We consider the case when the pattern set is represented by a probabilistic finite automaton, in which each pattern has a weight and the pattern set forms a probability distribution. We search for all matching substrings of T that have

probability greater than a given threshold and report their `(start, end)` indices. Additionally, since it is possible for the text to be very large, we consider this problem in a streaming setting in which we do not hold the entire text in memory throughout the computation. We call this problem *Online Stochastic Pattern Matching*.

Online stochastic pattern matching has several practical applications in fields such as natural language processing (NLP) and bioinformatics. Here is a possible example: suppose we have a PFA representing the distribution of RNA subsequences that encode some property we wish to study. In such a distribution, the probability of a subsequence corresponds to how likely it is that the property manifests if that subsequence is present. Since minor perturbations in the primary structure of an RNA subsequence or the configuration of the neighboring regions of the subsequence often have no effect on its function, the number of subsequences with non-zero probability could be very large. However, we are only interested in subsequences with probability exceeding some minimum threshold for the manifestation of the property. In this setting, given a long RNA sequence to analyze for our property, we want to detect subsequences that are likely to express the property (which are distinguished by their high probability) without having to extract all high probability subsequences from the distribution. Such problems, where PFAs (generally in an equivalent form of a hidden Markov model [6]) are used to model biological settings, is an active area of research [2, 18]. As illustrated in this example, the online stochastic pattern matching problem often involves a large input data stream and requires finding high probability substrings without explicitly extracting all of the high probability strings from the distribution ahead of time. We describe an algorithm that efficiently, both in theory and in practice, solves such problems. Our algorithm makes use of a weighted automaton construction that can effectively filter out substrings with lower probability than a given threshold probability. We also employ several properties of stochastic languages that enable us to bound the maximum length of substrings needed to be checked independent from the length of T—this is crucial to design an online algorithm for the problem without storing the whole T.

Researchers have studied several problems related to high probability strings in a stochastic language extensively. For the related class of Rabin automata, it is undecidable to determine whether or not there exists a string with probability greater than some given threshold [3]. For general PFAs, it is NP-hard to determine the highest probability string in the distribution described by the PFA [4]. Recently, de la Higuera and Oncina [9, 10] gave randomized and deterministic approaches for solving the consensus string problem. Parsing with weighted finite-state transducers is a common method in speech processing and recognition [12]. However, to the best of our knowledge, there is no prior research on detecting high probability strings from a distribution in streaming text.

2 Preliminaries

Let Σ be a finite alphabet and Σ^* be the set of all strings over Σ. The set of all strings of length k over Σ is written Σ^k. Given some string $w = w_1 w_2 \cdots w_n \in \Sigma^*$, we say that the length $|w|$ of w is n. The symbol λ denotes the null string. Given two strings w and x, wx denotes their concatenation. Then, $w\Sigma^*$ and $\Sigma^* w$ are the set of all strings in Σ^* containing w as a prefix and a suffix, respectively.

A semiring is a set $\mathbb{S}$ with two binary operations, $\oplus$ and $\otimes$, and two special elements, 0 and 1, where $(\mathbb{S}, \oplus)$ is a commutative monoid with identity 0 and $(\mathbb{S}, \otimes)$ is a monoid with identity 1. Additionally, $\otimes$ distributes over $\oplus$ and $\forall x \in \mathbb{S},\ 0 \otimes x = x \otimes 0 = 0$. Such a structure is denoted $(\mathbb{S}, \oplus, \otimes, 0, 1)$. In this paper, we utilize the *probability semiring* or *real semiring* $(\mathbb{R}_{\geq 0}, +, \times, 0, 1)$ and the *Viterbi semiring* $([0,1], \max, \times, 0, 1)$.

2.1 Weighted Automata

Weighted automata are a generalization of finite automata that compute a function $\mathcal{W} : \Sigma^* \to \mathbb{S}$, where $\mathbb{S}$ is the set of elements of some semiring. For a given string $w \in \Sigma^*$, we call the value $\mathcal{W}(w)$ the *weight* of w.

Given a semiring $(\mathbb{S}, \oplus, \otimes, 0, 1)$, a weighted automaton $\mathcal{W}$ is specified by a tuple $(Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \times Q \to \mathbb{S}$ is the transition function, $I : Q \to \mathbb{S}$ is the initial weight function, and $F : Q \to \mathbb{S}$ is the final weight function. An alternative representation of a weighted automaton is in the form of transition matrices where $\mathcal{W} = (Q, \Sigma, \{\mathbb{M}(c)\}_{c \in \Sigma}, \mathbb{I}, \mathbb{F})$. $\{\mathbb{M}(c)\}_{c \in \Sigma}$ is a set of $|Q| \times |Q|$ transition matrices where $\mathbb{M}(c)_{i,j} = \delta(q_i, c, q_j)$. $\mathbb{I}$ and $\mathbb{F}$ are $1 \times |Q|$ and $|Q| \times 1$ vectors corresponding to the initial and final weights; that is, $\mathbb{I}_i = I(q_i)$ and $\mathbb{F}_j = F(q_j)$. For brevity, we denote $\mathbb{M}(\Sigma) = \sum_{c \in \Sigma} \mathbb{M}(c)$.

We now describe how to compute the weight of w in a weighted automaton $\mathcal{W}$. Consider a string $w = w_1 w_2 \cdots w_n$ and its corresponding labeled path $\pi = (q_0, w_1, q_1), (q_1, w_2, q_2), \ldots, (q_{n-1}, w_n, q_n)$. Let the set of all such labeled paths be Φ_w. The weight of a path occurring is given by

$$\mathcal{W}(\pi) = I(q_0) \otimes \left(\bigotimes_{i=1}^{n} \delta(q_{i-1}, w_i, q_i) \right) \otimes F(q_n).$$

Accordingly, the weight of w is $\mathcal{W}(w) = \bigoplus_{\pi \in \Phi_w} \mathcal{W}(\pi)$.

There exists two equivalent dynamic programming approaches—the forward and backward algorithms—for computing the weight of a string in $\mathcal{W}$. We can also use the matrix formulation of $\mathcal{W}$ to compute the weight by replacing regular matrix operations with the operations defined by the semiring and evaluating $\mathcal{W}(w) = \mathbb{I} \prod_{i=1}^{|w|} \mathbb{M}(w_i) \mathbb{F}$.

Using either the dynamic programming approach or the matrix approach, we can compute the weight of a word w in $O(|w||Q|^2)$ time. For an introduction to weighted automata, we direct the reader to Droste et al. [5].

2.2 Probabilistic Finite Automata

A probabilistic finite automaton (PFA) $\mathcal{P} = (Q, \Sigma, \{\mathbb{M}(c)\}_{c\in\Sigma}, \mathbb{I}, \mathbb{F})$ is a weighted automaton over the *probabilistic semiring* with some additional constraints—$\sum_{q\in Q} I(q) = 1$ and $\forall q \in Q,\ F(q) + \sum_{q'\in Q, c\in\Sigma} \delta(q, c, q') = 1$. Then, $\mathcal{P}$ describes a probability distribution over Σ^*; in other words, $\forall w \in \Sigma^*,\ 0 \leq \mathcal{P}(w) \leq 1$ and $\sum_{w\in\Sigma^*} \mathcal{P}(w) = 1$. For PFAs, we use *probability* instead of *weight* to refer to the value $\mathcal{P}(w)$.

We consider only λ-free PFAs, as they are equivalent to regular PFAs in expressive power [5]. A PFA is deterministic (DPFA) if $\mathcal{P}$ has exactly one state with non-zero initial probability, and, for each state and character, at most one out-transition labeled with this character. While DPFAs are strictly weaker than general PFAs in expressive power, we can compute the probability of a string w in a DPFA in only $O(|w|)$ time.

Given a PFA $\mathcal{P}$, we can efficiently compute $\mathcal{P}(w\Sigma^*) = \sum_{x\in\Sigma^*} \mathcal{P}(wx)$ and $\mathcal{P}(\Sigma^* w) = \sum_{x\in\Sigma^*} \mathcal{P}(xw)$ which correspond to the probability of a word appearing as a prefix and as a suffix, respectively [7]. Let $\mathbb{M}(\Sigma^*) = \sum_{i=0}^{\infty} \mathbb{M}(\Sigma)^i = (\mathbb{1} - \mathbb{M}(\Sigma))^{-1}$ where $\mathbb{1}$ is the identity matrix (we use this notation when the dimension is clear). We now have

$$\mathcal{P}(w\Sigma^*) = \mathbb{I}(\prod_{i=1}^{|w|} \mathbb{M}(w_i))\mathbb{M}(\Sigma^*)\mathbb{F} \quad \text{and} \quad \mathcal{P}(\Sigma^* w) = \mathbb{I}\mathbb{M}(\Sigma^*)(\prod_{i=1}^{|w|} \mathbb{M}(w_i))\mathbb{F}.$$

One natural question related to PFAs is that of calculating the maximum probability parse of a string w; namely, to compute $\arg\max_{\pi\in\Phi_w} \mathcal{P}(\pi)$. We can solve this by simply coercing the PFA from the probabilistic semiring into the Viterbi semiring and calculating $\mathcal{P}(w)$. For a more detailed survey of PFAs, see Vidal et al. [16, 17].

A stochastic language over Σ is the set of strings from Σ^*, each of which has its own probability. That is, given $w \in \Sigma^*$, $0 \leq Pr(w) \leq 1$ is the probability of w and $\sum_{w\in\Sigma^*} Pr(w) = 1$. A regular stochastic language is a stochastic language that can be represented by a PFA—there exists a PFA $\mathcal{P}$ such that $\forall w \in \Sigma^*,\ \mathcal{P}(w) = Pr(w)$. A deterministic regular stochastic language is a stochastic language that can be represented by a DPFA.

We omit all proofs due to space constraints. Omitted proofs can be found in the appendix.

3 Online Stochastic Pattern Matching

The main problem is to identify all locations `(start, end)` of matching substrings in a large text that have high probability in some given distribution modeled by a PFA.

Online Stochastic Pattern Matching (OSPM): Given a streaming text T, a PFA $\mathcal{P}$, and a threshold probability $0 \leq p \leq 1$, report all pairs (i, j) such that $\mathcal{P}(T_i T_{i+1} \cdots T_j) \geq p$.

3.1 Weighted Automaton Construction

A naive approach would be to simply backtrack at each index of T and find high probability substrings with respect to $\mathcal{P}$. We notice that this method often leads to unnecessary backtracking. Therefore, we transform $\mathcal{P}$ into a weighted automaton that enables us to compute the sum of the probabilities of all suffixes of the current input without backtracking. This helps to skip unnecessary backtracking procedures where we can guarantee no high probability string ends.

Algorithm 1 Weighted Transform

```
1: procedure WEIGHTED_TRANSFORM(PFA P = (Q_P, Σ, {M_P(c)}_{c∈Σ}, I_P, F_P))
2:     Q_W ← Q_P ∪ {q'}
3:     for c ∈ Σ ∪ {λ} do
4:         M_W(c) ← (|Q_P| + 1) × (|Q_P| + 1)-zero matrix
5:         [M_W(c)]_{1,1} ← 1
6:         for i ∈ 1 ... |Q_P| do
7:             for j ∈ 1 ... |Q_P| do
8:                 [M_W(c)]_{i+1,j+1} ← [M_P(c)]_{i,j}
9:     I_W ← 1 × (|Q_P| + 1)-zero vector; F_W ← (|Q_W| + 1) × 1-zero vector
10:    [I_W]_1 ← 1
11:    for i ∈ 1 ... |Q| do
12:        [I_W]_{i+1} ← 0; [F_W]_{i+1} ← [F_P]_i; [M_W(λ)]_{1,i+1} ← [I_P]_i
13:    return W = (Q_W, Σ, {M_W(c)}_{c∈Σ}, I_W, F_W)
```

Algorithm 1 adds a new state q' that serves as the only initial state, while all other states have initial probability 0. From q' we add λ-transitions to the initial states of $\mathcal{P}$ weighted with their original initial weights and add a weight 1 self-loop for all characters in Σ. Figure 1 shows an example of the construction.

The resulting weighted automaton from Algorithm 1 helps us to compute the sum of the probabilities of all suffixes of the current streaming input. Then, if the sum is smaller than a threshold probability, we can skip the backtracking procedure since there cannot be a high probability string ending at the current index.

Lemma 1. *Given a PFA* $\mathcal{P} = (Q_\mathcal{P}, \Sigma, \{\mathbb{M}_\mathcal{P}(c)\}_{c\in\Sigma}, \mathbb{I}_\mathcal{P}, \mathbb{F}_\mathcal{P})$*, the transformed weighted automaton* $\mathcal{W} = (Q_\mathcal{W}, \Sigma, \{\mathbb{M}_\mathcal{W}(c)\}_{c\in\Sigma}, \mathbb{I}_\mathcal{W}, \mathbb{F}_\mathcal{W})$*, and a string* $w = w_1 w_2 \cdots w_n$*,*

$$\mathcal{W}(w) = \mathcal{P}(\lambda) + \sum_{i=1}^{n} \mathcal{P}(w_i w_{i+1} \cdots w_n).$$

Furthermore, we can compute $\mathcal{W}(w)$ *in* $O(|w||Q_\mathcal{P}|^2)$ *time.*

Lemma 1 guarantees that if $\mathcal{W}(w) < p$, then no suffix of $w = w_1 w_2 \cdots w_n$ has probability at least p and, thus, none can be valid matching patterns. Note that the converse is not necessarily true. Nevertheless, using Lemma 1, we can skip

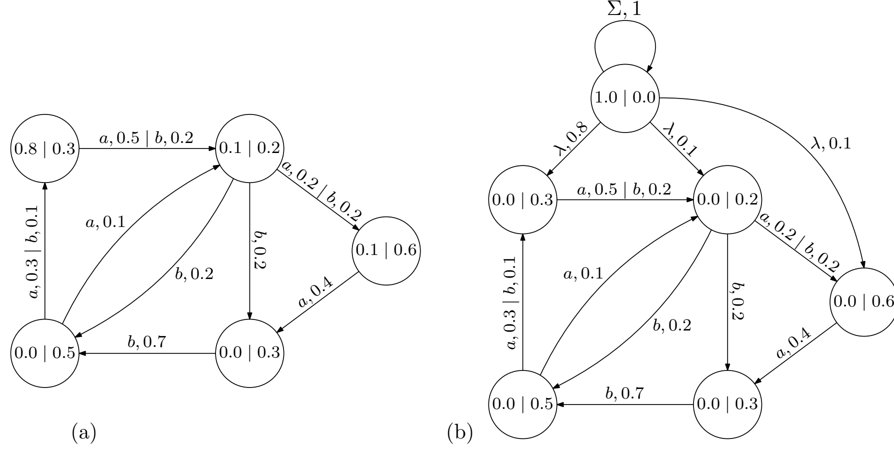

Fig. 1. (a) A PFA and (b) the resulting weighted automaton after Algorithm 1.

many backtracking steps at indices of T where no suffix can have probability greater than p.

3.2 Algorithm Structure

Although we can skip unnecessary backtracking steps when $\mathcal{W}(w) < p$, since $\mathcal{W}(w) \geq p$ does not necessarily imply that there is a suffix of w with sufficiently high probability, we still need to individually check each suffix and report a match only if its probability surpasses the threshold. This gives rise to the algorithm structure shown in Algorithm 2.

Algorithm 2 Online Stochastic Pattern Matching

```
1: procedure OSPM(Text T; PFA 𝒫; Probability p)
2:     𝒲 ← weighted_transform(𝒫)
3:     𝕍 ← 𝕀_𝒲(𝟙 + 𝕄_𝒲(λ))
4:     matches ← {}
5:     for j ∈ 1...|T| do
6:         𝕍 ← 𝕍𝕄_𝒲(T_j)(𝟙 + 𝕄_𝒲(λ))
7:         if 𝕍𝔽_𝒲 ≥ p then
8:             matches ← matches ∪ backtrack(T, 𝒫, p, j)
9:     return matches
```

In line 2, we transform a pattern PFA into a weighted automaton $\mathcal{W}$. Then using the weighted automaton, we initialize a vector $\mathbb{V}$ that allows us to iteratively compute the weight of a word in line 3. Specifically, by maintaining $\mathbb{V}$, we can quickly evaluate $\mathcal{W}(wa)$ from $\mathcal{W}(w)$ without recomputing the entire word.

Then, for each index of T, we backtrack the text only when $\mathbb{V}\mathbb{F}_{\mathcal{W}} \geq p$, and search for a matching substring whose probability is at least p using the backtrack procedure in lines 5–8. Note that if $\mathbb{V}\mathbb{F}_{\mathcal{W}} < p$, we know that there is no sufficiently high probability matching substring ending at index j and, thus, skip the backtrack procedure and move to the next streaming input.

Algorithm 3 Naive Backtracking

```
1: procedure BACKTRACK(Text T; PFA 𝒫; Probability p; Index j)
2:     output ← {}
3:     𝕏 ← 𝔽_𝒫
4:     for i ← j ... 1 do
5:         𝕏 ← 𝕄_𝒫(T_i)𝕏
6:         if 𝕀_𝒫𝕏 ≥ p then
7:             output ← output ∪ {(i, j)}
8:     return output
```

Algorithm 3 simply checks every suffix ending at a given index j of T. The runtime is $O(|T||Q_{\mathcal{P}}|^2)$ since the algorithm has to backtrack the entire text in the worst-case. Algorithm 2 runs in $O(|T|(|Q_{\mathcal{P}}|^2+|\text{Algorithm 3}|))$ time, and thus has an overall runtime of $O(|T|^2|Q_{\mathcal{P}}|^2)$. Note that, in this case, Algorithm 2 does not fit the definition of an online algorithm since the entire text must be stored in memory. We show that we only need to store a constant number of characters in the length of the text.

Lemma 2. *Let $\mathcal{P} = (Q, \Sigma, \{\mathbb{M}(c)\}_{c\in\Sigma}, \mathbb{I}, \mathbb{F})$ be a PFA, p be a threshold probability, μ and σ be the mean and variance of the string length described by $\mathcal{P}$, and $c = \max\{i \mid \mathcal{P}(\Sigma^i\Sigma^*) \geq p\}$.*

At each index of T, we need only to backtrack at most

$$\ell = \min\{\mu + \frac{\sigma}{\sqrt{p}},\ \frac{(|Q|+1)^2}{p},\ c\}$$

characters, which is independent of T.

The first two bounds are purely theoretical bounds with closed-form representations based on the structure of the PFA and its distribution. The last bound, $c = \max\{i \mid \mathcal{P}(\Sigma^i\Sigma^*) \geq p\}$, must be iteratively searched for, but often provides a much tighter bound, as discussed in Section 5. As a pre-processing step, we can choose the minimum of the three as the maximum backtracking length ℓ. Now we are ready to tackle OSPM efficiently using the upper-bound of the backtracking length in Lemma 2 together with Algorithm 2.

Theorem 1. *Given a streaming text T, a PFA $\mathcal{P}$, and a probability p, we can solve the OSPM problem in $O(\ell|T||Q_{\mathcal{P}}|^2)$ time with a buffer of size ℓ.*

3.3 OSPM with DPFAs

Despite DPFAs being strictly weaker than PFAs in expressive power, they are still useful in practice [16]. This is mainly because parsing is easier for DPFAs and several difficult problems for general PFAs (most notably, inferring the distribution) can be solved efficiently for DPFAs [8, 16]. This motivates us to consider a variant OSPM problem when the distribution is deterministic regular stochastic; in other words, $\mathcal{P}$ is a DPFA. We consider the case of `grep`-like pattern matching where we only report the end position of matching substrings. We call this problem `grep`-OSPM. Recall that for PFAs, the Viterbi semiring allows us to compute the most probable parse of a given string. Furthermore, for DPFAs, the probability of the most probable parse of a string w is exactly the probability of w. We combine this result with Algorithm 1 to find the highest probability suffix of a string in a DPFA. We observe that, for `grep`-OSPM with a DPFA, backtracking is not necessary and obtain a faster algorithm.

Lemma 3. *Given a DPFA $\mathcal{P}$ and its weighted transformation $\mathcal{W}$ over the Viterbi semiring,*

$$\mathcal{W}(w_1 w_2 \cdots w_n) = \max\{\mathcal{P}(\lambda), \mathcal{P}(w_n), \mathcal{P}(w_{n-1}w_n) \ldots, \mathcal{P}(w_1 w_2 \cdots w_n)\}.$$

Moreover, we can iteratively compute $\mathcal{W}(w)$ for each prefix of $w = w_1 w_2 \cdots w_n$ in only $O(|w||Q_{\mathcal{P}}|)$ time.

Theorem 2. *Given a streaming text T, a DPFA $\mathcal{P}$, and a probability p, we can solve `grep`-OSPM in $O(|T||Q_{\mathcal{P}}|)$ time without backtracking.*

4 Heuristic Speedup

The use of the weighted construction in Algorithm 1 is technically a heuristic in the general PFA case because we could simply backtrack at every character with the same asymptotic runtime. We describe another heuristic to further speedup the practical runtime of Algorithm 2 by short-circuiting the backtracking step.

We use the following property of stochastic languages

$$\forall w, x \in \Sigma^*,\ Pr(w) \leq Pr(\Sigma^* w) \text{ and } Pr(\Sigma^* w) \geq Pr(\Sigma^* x w).$$

Lemma 2 shows that the backtracking step only has to consider a constant number of characters when a new character of T arrives in stream. However, it is possible to end the backtracking step early if we can guarantee that no longer suffix can possibly have sufficiently high probability than p. Suppose that, during the backtracking step, we have read a suffix x from the buffer, which is pre-calculated from the upper-bound in Lemma 2. If $\mathcal{P}(\Sigma^* x) \geq p$, there may still be more suffixes (including w) in the buffer with probability at least as large as our threshold p. However, if $\mathcal{P}(\Sigma^* x) < p$ (line 9 of Algorithm 4), then we can immediately terminate the backtracking procedure and move to the next streaming character of T. Thus, Algorithm 4 becomes our new backtracking algorithm. Note that Algorithm 4 performs $O(1)$ matrix-vector multiplications per character and thus runs in $O(\ell|Q_{\mathcal{P}}|^2)$ time.

Algorithm 4 Backtracking Step

```
1: procedure BACKTRACK(Text T; PFA 𝒫; Probability p; Index j)
2:     matches ← {}
3:     ℓ ← max backtracking length
4:     𝕏 ← 𝔽
5:     for i ∈ 0...ℓ do
6:         𝕏 ← 𝕄_{T_{j−i}} 𝕏
7:         if 𝕀𝕏 ≥ p then
8:             matches ← matches ∪ {j − i, j}
9:         if 𝕀𝕄_{Σ*}𝕏 < p then
10:            return matches
11:    return matches
```

5 Experimental Results

Since there is no previous algorithm to address OSPM, we simply analyze the effectiveness of the suggested heuristics, and demonstrate that non-trivial instances of the problem can be solved in reasonable time. We construct two training sets. The first is a set of 20 PFAs with up to 125 states and alphabets of size 3 to 15. This set is generated in a manner similar to the test machines from the PAutomaC PFA learning challenge [15]. We first select up to $\frac{1}{4}$ of all possible state-character-state transitions and assign weights to them before normalization to construct a valid PFA. However, the machines from the PAutomaC competition generally have very short strings with high probability. This causes OSPM to perform well in practice since there will be relatively few high probability strings and the maximum backtracking length is low. We mitigate this by modifying the structure of the PFAs by choosing a random integer n between 5 and 10 and requiring all non-zero probability strings to have length longer than n. So that all prefixes are not equiprobable in expectation, one prefix out of Σ^n is selected and weighted substantially higher than the rest.

For each PFA $\mathcal{P}$, we sample $1,000$ strings from the distribution represented by $\mathcal{P}$ and select the highest probability string as a baseline threshold. For the text, we generate strings of length 10^6 by random sampling. We then run OSPM 20 times using these parameters, halving the threshold probability each time. This gives a spread of thresholds where on one end nearly all indices in the text do not have a sufficiently high probability substring ending at the correspoding indices, whereas at the other nearly all indices do have such substrings. The second training set is drawn from real data. We first build 4, 5, and 6-grams using randomly spliced fruit fly RNA data extracted from RNACentral[1]. We use a different fruit fly RNA sequence as our input text. We then construct a 3-gram of characters from Shakespeare's *King Henry VI, Parts I & II* and use *King Henry VI, Part III* as the input text. We run the experiments on an AMD Ryzen 7 1700 (3.0 GHz) 8-Core Processor with 16GB of RAM.

[1] http://rnacentral.org/

PFA	$\lvert Q \rvert$	$\lvert \Sigma \rvert$	threshold	c	avg	max	time (sec)	PFA	$\lvert Q \rvert$	$\lvert \Sigma \rvert$	threshold	c	avg	max	time (sec)
1	20	3	1.67E-05	787	6.62	13	5.97	11	54	10	1.22E-06	706	3.23	7	29.75
2	24	5	5.73E-08	2347	9.44	13	5.46	12	69	15	4.15E-05	429	4.02	5	48.42
3	14	10	1.47E-09	1923	6.36	10	3.88	13	109	3	5.44E-06	1890	9.41	14	108.75
4	10	15	3.79E-07	663	4.47	7	2.86	14	107	5	3.93E-08	1655	8.55	13	103.10
5	56	3	7.62E-07	614	9.77	17	35.28	15	104	10	1.46E-05	850	4.10	6	100.39
6	51	5	1.27E-06	1682	4.77	9	25.97	16	103	15	6.14E-06	1376	3.25	5	103.76
7	50	10	3.85E-06	3793	4.07	6	22.53	17	121	3	1.15E-06	1084	9.40	16	130.23
8	53	15	2.74E-06	468	6.20	7	25.69	18	121	5	3.98E-07	1735	10.18	12	119.01
9	74	3	5.75E-07	1528	7.12	14	72.65	19	115	10	7.86E-08	5012	4.31	8	124.29
10	71	5	3.29E-08	3579	3.72	11	88.65	20	115	15	3.57E-10	5196	4.31	8	119.63
RNA 4-gram	341	4	2.06E-04	40	2.10	6	39.19	RNA 5-gram	1145	4	2.20E-04	39	3.26	9	108.10
RNA 6-gram	2096	4	2.08E-04	42	4.45	8	99.87	King Henry	20533	46	1.50E-05	50	2.26	8	489.11

Table 1. For each PFA, we list the number of states, the alphabet size, the median threshold value, the backtracking bound calculated by $c = \max\{i \mid \mathcal{P}(\Sigma^i\Sigma^*) \geq p\}$, and the average and maximum number of characters backtracked when the backtrack procedure is executed.

We first consider the bound calculations. Of the three bounds given, our experimental results show $c = \max\{i \mid \Sigma^i\Sigma^* \geq p\}$ performs the best. Furthermore, even in the worst-case, the gap between c and $\mu + \frac{\sigma}{\sqrt{p}}$ or $\frac{(|Q|+1)^2}{p}$ is several orders of magnitude. This shows that determining c is necessary in practice, since it is impractical to maintain a buffer the size of which is required by the other two bounds. The experiments also show that the suffix probability heuristic allows us to quickly terminate the backtracking procedure when we could guarantee we would not find a string with probability greater than the threshold later in the buffer. Throughout the trials, we recorded the average and maximum number of characters backtracked during each backtracking call. Compared to even the lowest buffer bound, the average number of characters ever backtracked was very low, making the algorithm run very quickly in practice. The worst ratio of the average and maximum backtracking to buffer length is in PFA #5, with ratios $\frac{9.77}{614} \approx 0.016$ and $\frac{17}{614} \approx 0.028$, respectively. In no cases did a backtracking procedure exhaust the entire buffer. Even for very large PFAs, such as 20533-state PFA built from the *King Henry VI* data, our algorithm was able to parse large texts in reasonable time frames. In these cases, the c bound is especially useful as it can be found quickly, while $(\mathbb{1} - \mathbb{M}(\Sigma))^{-1}$ and thus $\mu + \frac{\sigma}{\sqrt{p}}$ takes an impractical amount of time to compute. Table 1 presents our experimental results.[2]

[2] More detailed experimental results including all three backtrack bounds and different threshold probability cases are summarized in Tables 2 and 3 in the appendix. We omit this due to the space limit

6 Application

Consider again the scenario where we have a PFA representing a distribution of RNA subsequences that are associated with the manifestation of some property. Suppose further that we have a long RNA sequence that exhibits the property, which implies that a non-zero probability subsequence from our distribution is present. Then it is of practical importance to determine what the highest probability subsequence present in the overall RNA sequence is, since that is the subsequence which most likely caused the property to manifest. Here we present a variant of OSPM to solve the aforementioned problem.

Problem 1. **Highest Probability Matching Substring**.
Given a streaming text T, a PFA $\mathcal{P}$, and a string $w \in \Sigma^+$, report the highest probability substring of T that is a superstring of w or λ if w never appears in T.

Let $\mathcal{D}$ be the DFA induced by the Knuth-Morris-Pratt next function of w. We construct a new weighted automaton $\mathcal{W} = \mathcal{P} \cap \mathcal{D}$ that has the property $\sum_{w \in \Sigma^*} \mathcal{W}(w) = \sum_{w \in \mathcal{L}(\mathcal{D})} \mathcal{P}(w)$ [13, 16]. We then perform our weighted construction on $\mathcal{W}$ to construct a new automaton $\mathcal{W}'$. Finally, we compute an initial threshold probability that lower bounds the maximum probability of any string containing w. There are several ways to construct such a lower bound, the simplest being to use $\mathcal{P}(w)$. Unfortunately, this can lead to the pathological case where $\mathcal{P}(w) = 0$ and T does not contain any instance of w. A better method is to perform breadth-first search from any starting state to find a non-zero probability path to any final state and use the probability of the corresponding string as our initial threshold. This has the added benefit of allowing us to immediately terminate with λ if no such path exists. We then use Algorithm 2 on T with $\mathcal{W}'$ while adaptively updating the threshold probability and saving only the maximum probability matching substring throughout the computation.

7 Conclusions

We have proposed an algorithm that efficiently solves OSPM and suggested heuristics to speed up its practical runtime. For a variant of OSPM, where only the end index of high probability substrings are reported, we have suggested a faster algorithm without backtracking when a pattern is a deterministic PFA. For the general OSPM problem, we have first introduced a weighted automaton construction that allows us to filter out indices at which no sufficiently high probability substring can end. We then proposed a new bound on the maximum possible length of a string with probability above a given threshold. This bound is often several orders of magnitude smaller than the previous known bounds. Finally, we have presented a heuristic based on a property of stochastic languages that allows us to terminate the backtracking procedure of our algorithm early when we can guarantee backtracking further would not find a sufficiently high probability string. Experimental results on both artificial and real-life data have

shown that our algorithm, combined with the suggested heuristics, solves OSPM quickly in practice.

References

1. Aho, A., Corasick, M.: Efficient string matching: An aid to bibliographic search. Communications of the ACM 18, 333–340 (1975)
2. Birney, E.: Hidden Markov models in biological sequence analysis. IBM Journal of Research and Development 45, 449–454 (2001)
3. Blondel, V.D., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. Theory of Computing Systems 36, 231–245 (2003)
4. Casacuberta, F., de la Higuera, C.: Computational complexity of problems on probabilistic grammars and transducers. In: Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI. pp. 15–24 (2000)
5. Droste, M., Kuich, W., Volger, H.: Handbook of Weighted Automata. Springer Publishing Company, Incorporated (2009)
6. Dupont, P., Denis, F., Esposito, Y.: Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. Pattern Recognition 38, 1349–1371 (2005)
7. Fred, A.L.N.: Computation of substring probabilities in stochastic grammars. In: Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000. pp. 103–114 (2000)
8. Guttman, O.: Probabilistic Automata and Distributions over Sequences. Ph.D. thesis, The Australian National University (2006)
9. de la Higuera, C., Oncina, J.: Computing the most probable string with a probabilistic finite state machine. In: Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing, FSMNLP. pp. 1–8 (2013)
10. de la Higuera, C., Oncina, J.: The most probable string: an algorithmic study. Journal of Logic and Computation 24, 311–330 (2014)
11. Knuth, D.E., James H. Morris, J., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6, 323–350 (1977)
12. Mohri, M., Pereira, F., Riley, M.: Speech recognition with weighted finite–state transducers. Computer Speech & Language 16, 69–88 (2002)
13. Nederhof, M., Satta, G.: Computation of infix probabilities for probabilistic context–free grammars. In: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP. pp. 1213–1221 (2011)
14. Thompson, K.: Regular expression search algorithm. Communications of the ACM 11, 419–422 (1968)
15. Verwer, S., Eyraud, R., de la Higuera, C.: Pautomac: a probabilistic automata and hidden Markov models learning competition. Machine Learning pp. 129–154 (2014)
16. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite–state machines–part I. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 1013–1025 (2005)
17. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite–state machines–part II. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 1026–1039 (2005)
18. Yoon, B.J.: Hidden Markov models and their applications in biological sequence analysis. Current Genomics pp. 402–415 (2009)

Appendix

Proof of Lemma 1:
Given $\mathcal{P} = (Q_{\mathcal{P}}, \Sigma, \{\mathbb{M}_{\mathcal{P}}(c)\}_{c\in\Sigma}, \mathbb{I}_{\mathcal{P}}, \mathbb{F}_{\mathcal{P}})$, the weighted automaton $\mathcal{W} = (Q_{\mathcal{W}}, \Sigma \cup \{\lambda\}, \{\mathbb{M}_{\mathcal{W}}(c)\}_{c\in\Sigma\cup\{\lambda\}}, \mathbb{I}_{\mathcal{W}}, \mathbb{F}_{\mathcal{W}})$ after performing Algorithm 1 has the following matrix structure:

- $\mathbb{I}_{\mathcal{W}} = \begin{bmatrix} 1 & \underbrace{0 \cdots 0}_{|Q_{\mathcal{P}}|} \end{bmatrix}$
- $\mathbb{F}_{\mathcal{W}} = \begin{bmatrix} 0 \\ \mathbb{F}_{\mathcal{P}} \end{bmatrix}$
- $\mathbb{M}_{\mathcal{W}}(\lambda) = \left.\begin{bmatrix} 0 & \mathbb{I}_{\mathcal{P}} \\ 0 & 0 \cdots 0 \\ \vdots & \vdots \ddots \vdots \\ 0 & \underbrace{0 \cdots 0}_{|Q_{\mathcal{P}}|} \end{bmatrix}\right\} |Q_{\mathcal{P}}| + 1$, denoted $\left[\begin{array}{c|c} 0 & \mathbb{I}_{\mathcal{P}} \\ \hline 0 & 0 \end{array}\right]$
- $\mathbb{M}_{\mathcal{W}}(c) = \left[\begin{array}{c|c} 1 & 0 \;\cdots\; 0 \\ \hline 0 & \\ \vdots & \mathbb{M}_{\mathcal{P}}(c) \\ 0 & \end{array}\right]$, denoted $\left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & \mathbb{M}_{\mathcal{P}}(c) \end{array}\right]$

Let

$$\mathcal{U}(w) = \mathbb{I} \prod_{i=1}^{n} \mathbb{M}_{\mathcal{P}}(w_i),$$

that is, the $1 \times |Q_{\mathcal{P}}|$ vector produced by computing $\mathcal{P}(w)$ without multiplying by the final probability vector. In other words, $\mathcal{P}(w) = \mathcal{U}(w)\mathbb{F}_{\mathcal{P}}$. Similarly, let

$$\mathcal{U}'(w) = \mathbb{I}_{\mathcal{W}}(\mathbb{1} + \mathbb{M}_{\mathcal{W}}(\lambda)) \prod_{i=1}^{|w|} \mathbb{M}_{\mathcal{W}}(w_i)(\mathbb{1} + \mathbb{M}_{\mathcal{W}}(\lambda))\ ^{3}.$$

We also define $\mathcal{X}(w) = \mathcal{U}(\lambda) + \sum_{i=1}^{n} \mathcal{U}(w_i w_{i+1} \cdots w_n)$.

By induction, we show that $\mathcal{U}'(w) = \begin{bmatrix} 1 & \mathcal{X}(w) \end{bmatrix}$. The base case is clear as

$$\mathcal{U}'(\lambda) = \mathbb{I}_{\mathcal{W}}(\mathbb{1} + \mathbb{M}_{\mathcal{W}}(\lambda)) = \begin{bmatrix} 1 \ 0 \cdots 0 \end{bmatrix} \left[\begin{array}{c|c} 1 & \mathbb{I}_{\mathcal{P}} \\ \hline 0 & \mathbb{1} \end{array}\right] = \begin{bmatrix} 1 \ \mathbb{I}_{\mathcal{P}} \end{bmatrix}.$$

[3] Note that for $\mathcal{P}$, as there are no λ-transitions, $\mathbb{1} + \mathbb{M}_{\mathcal{P}}(\lambda) = \mathbb{1}$ and can be omitted from multiplications. $\mathbb{1} + \mathbb{M}_{\mathcal{P}}(\lambda)$ is chosen here since there are no λ-loops and $\mathbb{M}_{\mathcal{W}}(\lambda)^n$ is the zero matrix for $n > 1$.

Now, given $\mathcal{U}'(w)$, we have

$$\begin{aligned}\mathcal{U}'(w)\mathbb{M}_{\mathcal{W}}(a)(\mathbb{1}+\mathbb{M}_{\mathcal{W}}(\lambda)) &= \begin{bmatrix}1 & \mathcal{X}(w)\end{bmatrix}\left[\begin{array}{c|c}1 & 0\\ \hline 0 & \mathbb{M}_{\mathcal{P}}(a)\end{array}\right](\mathbb{1}+\left[\begin{array}{c|c}0 & \mathbb{I}_{\mathcal{P}}\\ \hline 0 & 0\end{array}\right])\\ &= \begin{bmatrix}1 & \mathcal{X}(w)\mathbb{M}_{\mathcal{P}}(a)\end{bmatrix}(\mathbb{1}+\left[\begin{array}{c|c}0 & \mathbb{I}_{\mathcal{P}}\\ \hline 0 & 0\end{array}\right])\\ &= \begin{bmatrix}1 & \mathcal{X}(wa)\end{bmatrix}(\mathbb{1}+\left[\begin{array}{c|c}0 & \mathbb{I}_{\mathcal{P}}\\ \hline 0 & 0\end{array}\right])\\ &= \mathcal{U}'(wa).\end{aligned}$$

Finally, noting that

$$\begin{aligned}\begin{bmatrix}1\ \mathcal{X}(w)\end{bmatrix}\mathbb{F}_{\mathcal{W}} &= \begin{bmatrix}1\ \mathcal{X}(w)\end{bmatrix}\begin{bmatrix}0\\ \mathbb{F}_{\mathcal{P}}\end{bmatrix}\\ &= \mathcal{X}(w)\mathbb{F}_{\mathcal{P}}\\ &= (\mathcal{U}(\lambda)+\sum_{i=1}^{n}\mathcal{U}(w_i w_{i+1}\cdots w_n))\mathbb{F}_{\mathcal{P}}\\ &= \mathbb{I}_{\mathcal{P}}(\sum_{i=1}^{n}\prod_{j=i}^{n}\mathbb{M}_{\mathcal{P}}(w_j))\mathbb{F}_{\mathcal{P}}\end{aligned}$$

completes the proof. □

Proof of Lemma 2:
We describe three possible bounds on the maximum backtracking length, all of which are independent of the length of T:

1. Let μ and σ be the mean and variance of the string length of the distribution described by a PFA. de la Higuera and Oncina [10] showed that the maximum length of a string with probability greater than a given p is $\mu + \frac{\sigma}{\sqrt{p}}$ using Chebychev's inequality.
2. de la Higuera and Oncina [9] also showed that for a given λ-free PFA (which is what we consider here), a string of probability greater than p cannot be longer than $\frac{(|Q|+1)^2}{p}$.
3. Finally, let
$$c = \max\{i \mid \mathcal{P}(\Sigma^i\Sigma^*) \geq p\},$$
where $\mathcal{P}(\Sigma^i\Sigma^*) = \sum_{w\in\Sigma^*}\sum_{x\in\Sigma^i}\mathcal{P}(wx) = \mathbb{I}\mathbb{M}(\Sigma)^i\mathbb{M}(\Sigma^*)\mathbb{F}$. Since no string with length greater than c can have probability greater than p, c is another upper bound for the backtracking length.

All three of these are independent of T and the statement holds. □

Proof of Theorem 1:
From Lemma 1, we can scan the entire text for candidate positions where we need to backtrack in $O(|T||Q_{\mathcal{P}}|^2)$ time. Furthermore, by Lemma 2, we know we only have to backtrack at most ℓ number of steps for each candidate position and therefore only need to make one full pass of the data, as ℓ depends only on the distribution and the input p. Our *backtrack* implementation has complexity $O(\ell|Q_{\mathcal{P}}|^2)$. Thus, the overall runtime is $O(|T|(|Q_{\mathcal{P}}|^2 + \ell|Q_{\mathcal{P}}|^2)) = O(\ell|T||Q_{\mathcal{P}}|^2)$. □

Proof of Lemma 3:
The proof of

$$\mathcal{W}(w_1w_2\cdots w_n) = \max\{\mathcal{P}(\lambda), \mathcal{P}(w_n), \mathcal{P}(w_{n-1}w_n)\ldots, \mathcal{P}(w_1w_2\cdots w_n)\}$$

is the same as Lemma 1 using the Viterbi semiring operations. We use the forward algorithm to compute the weight of a word. Since $|Q_{\mathcal{W}}| \in O(|Q_{\mathcal{P}}|)$ and each state of $\mathcal{W}$ has at most 1 outgoing transition per character (including λ), updating the forward algorithm calculation table takes $O(1)$ time per cell leading to an overall runtime of $O(|w||Q_{\mathcal{P}}|)$. □

Proof of Theorem 2:
By Lemma 3, we can iteratively calculate

$$\max\{\mathcal{P}(\lambda), \mathcal{P}(T_i), \mathcal{P}(T_{i-1}T_i), \ldots, \mathcal{P}(T_1T_2\cdots T_i)\}$$

for each index of T in $O(|T||Q_{\mathcal{P}}|)$ time. It immediately follows that if the result at index i is less than p, then there is no substring of T ending at index i with probability greater than p and vice versa. □

PFA	$\|Q\|$	$\|\Sigma\|$	threshold	c	$\mu + \frac{\sigma}{\sqrt{p}}$	$\frac{(\|Q\|+1)^2}{p}$	avg	max	time (sec)
1	20	3	5.34E-04	541	2.18E+05	8.26E+05	6.17	9	4.53
	20	3	1.67E-05	787	1.23E+06	2.64E+07	6.62	13	5.97
	20	3	5.22E-07	1033	6.97E+06	8.46E+08	7.04	16	13.89
2	24	5	1.83E-06	1861	1.45E+07	3.41E+08	9.16	11	5.45
	24	5	5.73E-08	2347	8.20E+07	1.09E+10	9.44	13	5.46
	24	5	1.79E-09	2833	4.64E+08	3.49E+11	9.66	15	5.80
3	14	10	4.69E-08	1597	4.09E+07	4.80E+09	6.28	8	3.48
	14	10	1.47E-09	1923	2.32E+08	1.54E+11	6.36	10	3.88
	14	10	4.58E-11	2249	1.31E+09	4.91E+12	6.49	11	5.88
4	10	15	1.21E-05	509	5.64E+05	9.98E+06	5.50	6	2.86
	10	15	3.79E-07	663	3.19E+06	3.19E+08	4.47	7	2.86
	10	15	1.18E-08	817	1.81E+07	1.02E+10	4.18	9	2.91
5	56	3	2.44E-05	466	3.72E+05	1.33E+08	9.40	13	33.13
	56	3	7.62E-07	614	2.11E+06	4.27E+09	9.77	17	35.28
	56	3	2.38E-08	763	1.19E+07	1.37E+11	10.27	20	48.16
6	51	5	4.05E-05	1254	2.39E+06	6.67E+07	4.29	7	23.81
	51	5	1.27E-06	1682	1.35E+07	2.13E+09	4.77	9	25.97
	51	5	3.96E-08	2110	7.66E+07	6.83E+10	5.11	11	34.86
7	50	10	1.23E-04	2740	8.32E+06	2.11E+07	4.01	5	22.39
	50	10	3.85E-06	3793	4.71E+07	6.75E+08	4.07	6	22.53
	50	10	1.20E-07	4847	2.66E+08	2.16E+10	4.14	8	22.81
8	53	15	8.78E-05	343	1.38E+05	3.32E+07	-1.00	0	25.70
	53	15	2.74E-06	468	7.80E+05	1.06E+09	6.20	7	25.69
	53	15	8.57E-08	592	4.41E+06	3.40E+10	6.15	8	25.66
9	74	3	1.84E-05	1161	2.61E+06	3.06E+08	6.55	11	55.23
	74	3	5.75E-07	1528	1.48E+07	9.78E+09	7.12	14	72.65
	74	3	1.80E-08	1895	8.36E+07	3.13E+11	7.10	18	103.13
10	71	5	1.05E-06	2860	4.20E+07	4.93E+09	3.84	8	65.18
	71	5	3.29E-08	3579	2.37E+08	1.58E+11	3.72	11	88.65
	71	5	1.03E-09	4298	1.34E+09	5.04E+12	3.00	3	83.81

Table 2. The complete experimental results for each PFA. We include the 25-50-75 percentile threshold probabilities used for each PFA. The number of states, alphabet size, threshold, buffer bounds (c, $\mu + \frac{\sigma}{\sqrt{p}}$, and $\frac{(|Q|+1)^2}{p}$), and the average and maximum number of characters backtracked when the backtrack procedure are listed. We omit the second bound for PFAs where calculating $(\mathbb{1} - \mathbb{M}(\Sigma))^{-1}$ is too computationally expensive.

PFA	$\|Q\|$	$\|\Sigma\|$	threshold	c	$\mu + \frac{\sigma}{\sqrt{p}}$	$\frac{(\|Q\|+1)^2}{p}$	avg	max	time (sec)
11	54	10	3.91E-05	527	4.24E+05	7.74E+07	3.06	5	28.45
	54	10	1.22E-06	706	2.40E+06	2.48E+09	3.23	7	29.75
	54	10	3.82E-08	884	1.36E+07	7.92E+10	3.50	9	37.37
12	69	15	1.33E-03	282	4.87E+04	3.69E+06	4.00	4	48.63
	69	15	4.15E-05	429	2.76E+05	1.18E+08	4.02	5	48.42
	69	15	1.30E-06	575	1.56E+06	3.78E+09	4.03	6	48.52
13	109	3	1.74E-04	1352	1.83E+06	6.95E+07	9.19	11	106.41
	109	3	5.44E-06	1890	1.03E+07	2.22E+09	9.41	14	108.75
	109	3	1.70E-07	2429	5.84E+07	7.11E+10	9.70	17	114.52
14	107	5	1.26E-06	1320	8.30E+06	9.27E+09	8.28	11	102.32
	107	5	3.93E-08	1655	4.69E+07	2.97E+11	8.55	13	103.10
	107	5	1.23E-09	1989	2.66E+08	9.49E+12	8.77	16	106.20
15	104	10	4.66E-04	586	2.67E+05	2.37E+07	4.05	5	105.42
	104	10	1.46E-05	850	1.51E+06	7.57E+08	4.10	6	100.39
	104	10	4.55E-07	1113	8.56E+06	2.42E+10	4.18	8	107.73
16	103	15	1.96E-04	980	9.33E+05	5.51E+07	3.17	4	96.81
	103	15	6.14E-06	1376	5.28E+06	1.76E+09	3.25	5	103.76
	103	15	1.92E-07	1772	2.99E+07	5.64E+10	3.24	6	104.91
17	121	3	3.67E-05	811	1.02E+06	4.05E+08	9.13	13	126.56
	121	3	1.15E-06	1084	5.77E+06	1.30E+10	9.40	16	130.23
	121	3	3.59E-08	1356	3.26E+07	4.15E+11	9.82	19	153.91
18	121	5	1.27E-05	1330	3.84E+06	1.17E+09	10.00	10	120.28
	121	5	3.98E-07	1735	2.17E+07	3.74E+10	10.18	12	119.01
	121	5	1.24E-08	2141	1.23E+08	1.20E+12	10.37	14	120.33
19	115	10	2.52E-06	3951	5.91E+07	5.35E+09	4.25	7	121.35
	115	10	7.86E-08	5012	3.34E+08	1.71E+11	4.31	8	124.29
	115	10	2.46E-09	6073	1.89E+09	5.48E+12	4.43	10	126.77
20	115	15	1.14E-08	4369	5.32E+08	1.18E+12	4.14	7	115.94
	115	15	3.57E-10	5196	3.01E+09	3.77E+13	4.31	8	119.63
	115	15	1.12E-11	6023	1.70E+10	1.21E+15	4.58	10	140.67
RNA 4-gram	341	4	6.60E-03	25	228.01	1.17E+07	3.29	4	18.73
	341	4	2.06E-04	40	1257.82	5.67E+07	2.10	6	39.19
	341	4	6.45E-06	54	7083.35	1.81E+10	2.09	4	39.34
RNA 5-gram	1145	4	3.56E-03	28	—	3.69E+08	3.53	6	37.88
	1145	4	1.11E-04	42	—	1.18E+10	3.27	9	111.13
	1145	4	3.47E-06	57	—	3.78E+11	3.29	9	108.75
RNA 6-gram	2096	4	3.33E-03	31	—	1.32E+09	4.31	6	49.39
	2096	4	1.04E-04	45	—	4.22E+10	4.47	8	98.63
	2096	4	3.25E-06	59	—	1.35E+12	4.48	12	101.25
King Henry	20533	46	2.40E-04	38	—	1.76E+12	2.58	8	313.98
	20533	46	7.50E-06	54	—	5.69E+13	2.25	9	480.58
	20533	46	2.34E-07	69	—	1.82E+15	2.25	6	480.32

Table 3. The complete experimental results for each PFA. We include the 25-50-75 percentile threshold probabilities used for each PFA. The number of states, alphabet size, threshold, buffer bounds (c, $\mu + \frac{\sigma}{\sqrt{p}}$, and $\frac{(|Q|+1)^2}{p}$), and the average and maximum number of characters backtracked when the backtrack procedure are listed. We omit the second bound for PFAs where calculating $(\mathbb{1} - \mathbb{M}(\Sigma))^{-1}$ is too computationally expensive.