Most Probable String within Distance k
- interim 2 -

Yumin Kim Myeongjang Pyeon Haksu Lim

November 2018

1 Probabilistic Finite-State Automata

There is DFA/NFA that represents an accepting model which determines whether it accepts the input strings or not. If probabilities are added to the transitions of DFA/NFA, this is called PFA, and the notation is like $A=\langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$. It represents a generative model [7, 8]. The PFA satisfies the following conditions.

- Q is a finite set of states; these will be labelled $q_1, \ldots, q_{|Q|}$ unless otherwise stated
- Σ is the alphabet
- $\mathbb{I}_{\mathbb{P}} : \mathrm{Q} \to \mathbb{Q}^+ \cap [0, 1]$ *(initial-state probabilities)*
- $\mathbb{F}_{\mathbb{P}} : \mathrm{Q} \to \mathbb{Q}^+ \cap [0, 1]$ *(final-state probabilities)*
- $\delta_{\mathbb{P}} : \mathrm{Q} \times (\Sigma \cup \{\lambda\}) \times \mathrm{Q} \to \mathbb{Q}^+$ *is a transition function; the function is complete: $\delta_{\mathbb{P}}$ (q,a,q') = 0 can be interpreted as "no transition from q to q' labelled with a". We will also denote (q, a, q', P) instead of $\delta_{\mathbb{P}}(q, a, q') = P$ where P is a probability.*

$\mathbb{I}_{\mathbb{P}}$, $\delta_{\mathbb{P}}$ and $\mathbb{F}_{\mathbb{P}}$ are functions such that:

$$\sum_{q \in Q} \mathbb{I}_{\mathbb{P}}(q) = 1,$$

$$\textit{and } \forall q \in Q, \mathbb{F}_{\mathbb{P}}(q) + \sum_{a \in \sum \cup \{\lambda\}, q' \in Q} \delta_{\mathbb{P}}(q, a, q') = 1.$$

Each string in Σ^* has a probability of being generated by the PFA. We can also find the distribution in which the strings are generated through string sampling.

The distribution satisfies the following conditions.

-$Pr_D(x)$ is the probability of a string $x \in \Sigma^*$ under the distribution D
-$\sum_{x \in \Sigma^*} Pr_D(x) = 1$.
-$\forall q \in Q$, $\mathbb{F}_{\mathbb{P}}(q) + \sum_{a \in \Sigma \cup \{\lambda\}, q' \in Q} \delta_{\mathbb{P}}(q, a, q') = 1$.

If λ-transitions are added to the PFA it is called a λ-PFA. The PFA can be represnted with graph. Figure 1 is an example of the graph of the PFA [1]. The numbers on the left of the state name are initial probabilities and the numbers on the right of the state name are final probabilities. The number on the transitions are transition probabilities. For example, the probability of parsing string "ab" with this PFA is $Pr_A(ab)$ that

$$\begin{aligned} Pr_A(ab) &= \mathbb{I}_{\mathbb{P}}(q_1) \cdot \delta_{\mathbb{P}}(q_1, a, q_2) \cdot \delta_{\mathbb{P}}(q_2, b, q_4) \cdot \mathbb{F}_{\mathbb{P}}(q_4) \\ &\quad + \mathbb{I}_{\mathbb{P}}(q_2) \cdot \delta_{\mathbb{P}}(q_2, a, q_3) \cdot \delta_{\mathbb{P}}(q_3, b, q_3) \cdot \mathbb{F}_{\mathbb{P}}(q_3) \\ &\quad + \mathbb{I}_{\mathbb{P}}(q_2) \cdot \delta_{\mathbb{P}}(q_2, a, q_3) \cdot \delta_{\mathbb{P}}(q_3, b, q_4) \cdot \mathbb{F}_{\mathbb{P}}(q_4) \\ &= 0.4 \cdot 0.5 \cdot 0.4 \cdot 0.3 \\ &\quad + 0.6 \cdot 0.5 \cdot 0.2 \cdot 0.4 \\ &\quad + 0.6 \cdot 0.5 \cdot 0.4 \cdot 0.3 \\ &= 0.084 \end{aligned}$$

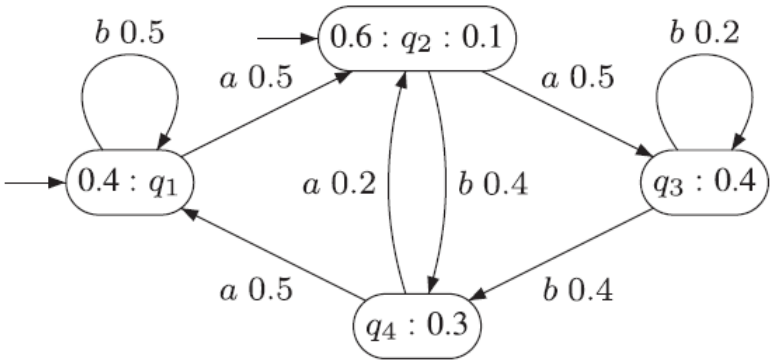

Figure 1: Graphical representation of a PFA [1]

2 Problems

2.1 Related Problems

In probabilistic automata theory, an important question is what the most probable string is. Without any constraint, the problem is called as the *consensus string* problem. The formal definition is as below.
Name: Consensus string (Cs)
Instance: A probabilistic machine $\mathcal{M}$
Question: Find in Σ^* a string x such that $\forall y \in \Sigma^*, Pr_{\mathcal{M}}(x) \geq Pr_{\mathcal{M}}(y)$.

This problem can be modified as a decision problem. It is called as the *most probable string* problem, defined as below.
Name: Most probable string (MPS)
Instance: A probabilistic machine $\mathcal{M}$ and $p \geq 0$
Question: Find in Σ^* a string x such that $Pr_{\mathcal{M}}(x) > p$.

If an extra constraint for the length of strings is added, it is called as the *bounded most probable string* problem, which is defined as below.
Name: Bounded most probable string (BMPS)
Instance: A probabilistic machine $\mathcal{M}$, $p \geq 0$, and an integer b
Question: Find in $\Sigma^{\leq b}$ a string x such that $Pr_{\mathcal{M}}(x) > p$.

2.2 Our Problem

In this project, we will solve another problem. In our setting, we are given a string so that the solution space is reduced to the set of string within $k-$distance with the given string. We call it as the *most probable string within $k-$distance* problem and the formal definition is described as below.
Name: Most probable string within $k-$distance
Instance: A probabilistic machine $\mathcal{M}$, $p \geq 0$, a string w, and $k \in \mathbb{N}$
Question: Find in Σ^* a string x such that $Pr_{\mathcal{M}}(x) > p$ and dist(x, w)$\leq k$.
Note that the solution space is a subset of $\Sigma^{\leq |w|+k}$. So, it is not necessary to add the constraint for the length of strings.

3 Previous Research

The problem of finding the most probable string (MPS) for a distribution generated by a probabilistic finite-state automaton is NP-hard.[4, 2] Therefore the decision problem of it, which determines whether there exist a string whose probability on this machine is bigger than a given probability p, is also NP-hard. The one main reason is that there is no bound on the string. The most probable string can be of exponential length.[4, 2]

We focus on the bounded decision problem of MPS(BMPS). In that case, there is an additional input integer b that bounds the length of the string lower than or equal to b. There is a pseudo-polynomial exact algorithm to solve BMPS.[4, 2]

The problem we're interested in is MPS where distance within k (k-MPS). Note that k-MPS is inherently a kind of BMPS because the length of string is bounded by edit(or hamming) distance k. There is an exact algorithm suggested by Marco Cognetta that solve this problem in pseudo-polynomial time when k is lower than or equal to 1. However this algorithm cannot solve the problem in pseudo-polynomial time when k is bigger than 1. We'll brainstorm how to efficiently solve k-MPS when k is bigger than 2.

4 Algorithms

Firstly, we have to define some useful notations to calculate probabilities of strings generated by PFA.

Definition 1. Let $\mathcal{M}$ be a probabilistic automaton. Then, we define $\mathbb{I}$, $\mathbb{F}$, and $\mathbb{M}$ as matrices containing initial probabilities, final probabilities, and transition probabilities.

Definition 2. Consider transition probabilities of $\mathbb{M}$. Then, it contains transition matrices per each alphabet $a \in$. For each alphabet $a \in \Sigma$, we define $\mathbb{M}_a$ as the transition matrix given a.

Remark 1. We can extend definition 2 to strings. For each string $w \in \Sigma^+$,

$$\mathbb{M}_w = \prod_{i=1}^{|w|} \mathbb{M}_{w_i} \tag{1}$$

4.1 DP based algorithm

If we design a brute-force algorithm for the $k-$MPS, we have to consider all k replacements of alphabets in a string and calculate its probability, which runs in $O\big(n|\Sigma|^k \binom{n}{k} |Q|^{2+O(1)}\big)$ by Le Gall algorithm [6]. However, it is intractable. We can reduce some fractions of computations by using *dynamic programming*. The proposed algorithm is described in Algorithm 1.

Theorem 1. *The string obtained by Algorithm 1 is the most probable string within distance k.*

Proof. It is trivial that the calculation of probabilities is correct. So, now we assume it and prove that the obtained string is the most probable string. Suppose that there exists w' such that $d_{hamming}(w, w') = k^{'} \leq k$ and $p(w') > p(w^*)$. Then, $p(w') > p(w_0)$ for all strings obtained by replacing k or less characters in w with others. Since $d_{hamming}(w, w') \leq k$, it is considered at line 10. Thus, $w' = w^*$, which contradicts to the assumption. □

Algorithm 1 DP based MPS within hamming distance k

Input: a probabilistic automaton $\mathcal{M}$, a string w, and a distance limit k
Output: the most probable string w^*

$n \leftarrow |w|$
$\mathbb{M}_\lambda \leftarrow \mathrm{I}_{|Q|\times|Q|}$
$prefix(i) \leftarrow \begin{cases} \mathbb{I}, & if\ i = 0 \\ \mathbb{I}\mathbb{M}_{w_1 \cdots w_i}, & if\ 1 \leq i \leq n \end{cases}$
$suffix(i) \leftarrow \begin{cases} \mathbb{F}, & if\ i = n+1 \\ \mathbb{M}_{w_i \cdots w_n}\mathbb{F}, & if\ 1 \leq i \leq n \end{cases}$
$infix(i,j) \leftarrow \begin{cases} \mathbb{M}_{w_i} \cdots \mathbb{M}_{w_j}, & if\ 1 \leq i \leq j \leq n \\ \mathrm{I}_{|Q|\times|Q|}, & if\ i > j \end{cases}$
$w^* \leftarrow \lambda$
$p^* \leftarrow 0$
for each k-combination σ of n **do**
    for each $(c_1, \cdots, c_k) \in \Sigma^k$ **do**
        $w' \leftarrow$ a string obtained by replacing $w_{\sigma(i)}$ by c_i for all $1 \leq i \leq k$
        $p(w') \leftarrow prefix(\sigma(1) - 1)$
        for $i \leftarrow 1, \cdots, k-1$ **do**
            $p(w') \leftarrow p(w')\mathbb{M}_{c_i} infix(\sigma(i)+1, \sigma(i+1)-1)$
        $p(w') \leftarrow p(w')\mathbb{M}_{c_k} suffix(\sigma(n)+1)$
        if $p^* < p(w')$ **then**
            $w^* \leftarrow w'$
            $p^* \leftarrow p(w')$
return w^*

Theorem 2. *Algorithm 1 runs in* $O\big(n^2|Q|^{2+O(1)} + k|\Sigma|^k\binom{n}{k}|Q|^{2+O(1)}\big)$.

Proof. Note that the running time of matrix multiplication of $A_{n\times m}$ and $B_{m\times p}$ is $O(nmp)$. However, if $n = m = p$, we can calculate it in $O\big(n^{2+O(1)}\big)$ by Le Gall algorithm [6]. So, we can calculate $prefix$, $suffix$, and $infix$ in $O\big(n^2|Q|^{2+O(1)}\big)$ and w^* in $O\big(k|\Sigma|^k\binom{n}{k}|Q|^{2+O(1)}\big)$. Thus, the total running time is $O\big(n^2|Q|^{2+O(1)} + k|\Sigma|^k\binom{n}{k}|Q|^{2+O(1)}\big)$. □

4.2 Intersection based algorithm

Although there is no previous researches on the most probable string within k distance, the most probable string problem is previously covered. So, it is natural to extend such researches to our work.

Since our problem space is reduced to the neighborhood of the given string w, our probabilistic automaton has to generate strings over this domain, not the whole strings. In order to achieve this, we obtain the deterministic finite-state automaton (DFA) which accepts strings whose distances from the given string w is smaller than or equal to k and intersect the given deterministic probabilistic finite-state automaton (DPFA) and the DFA. We define such a DFA as the Hamming automaton.

Definition 3. Let Σ be the set of alphabets. Then, *Hamming automata* are deterministic finite-state automata accepting a string if and only if the Hamming distance from the given string w is smaller than or equal to the given distance limit k.

The construction of an Hamming automaton is done by Algorithm 2 by Theorem 3 and the algorithm runs in $O(k|w|\cdot|\Sigma|)$ by Theorem 4 (see Figure 1 as an example).

Theorem 3. *Given a string w and a distance limit k, Algorithm 2 returns the Hamming automaton over the set of alphabets Σ,*

Proof. Let d be the hamming distance and w' be an input to the obtained automaton. If $|w'| < |w|$, after $\mathcal{A}$ sees w', the result state is $q_{i,j}$ for some $0 \le i \le k$ and $i \le j < k$. In addition, the result state is q_{exit}, if the $|w'| > |w|$. So, the automaton dose not accept any string whose length is different from the given string. Now we assume that the input string length is same as the given string. Suppose that $d(w', w) = k_0 \le k$. Then, $\mathcal{A}$ accepts it at state $q_{k_0,k}$. So, $\mathcal{A}$ accepts all strings within k distance from w. On the other hand, suppose that $d(w', w) > k$. Then, after $\mathcal{A}$ sees w', the result state is q_{exit}. So, $\mathcal{A}$ does not accept any string such that $d(w', w) > k$. Thus, by the definition of the Hamming automaton, Algorithm 2 is correct. □

Theorem 4. *Algorithm 2 runs in* $O(k|w|\cdot|\Sigma|)$.

Proof. proof omitted since it is trivial. □

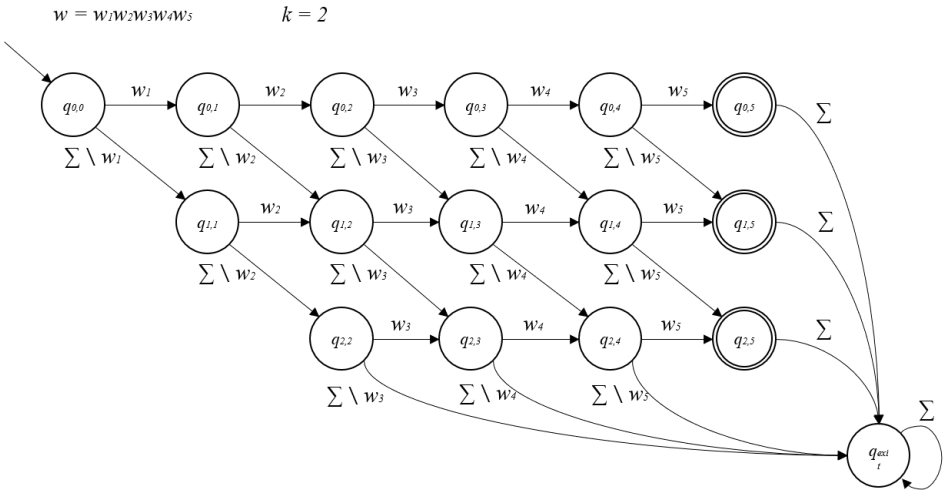

Figure 2: An example of hamming automata when $k = 2$ and $|w| = 5$

Algorithm 2 Construct an Hamming automaton

Input: a string w, a distance limit k, and the set of alphabets Σ

Output: the Hamming automaton $\mathcal{A}$, which accepts a string if and only if the distance between it and w is smaller than or equal to k

$Q \leftarrow \{q_{i,j} : 0 \leq i \leq k, i \leq j \leq k\} \cup \{q_{\text{exit}}\}$ ▷ q_{exit} is a sink state
$s \leftarrow q_{0,0}$
$F \leftarrow \{q_{i,k} : 0 \leq i \leq k\}$
let the transition function δ be a function from $Q \times \Sigma$ to Q
for $i \leftarrow 0, \cdots, k$ **do**
    $\delta(q_{i,k}, a) \leftarrow q_{\text{exit}}$ for all $a \in \Sigma$
    for $j \leftarrow i, \cdots, k-1$ **do**
        $\delta(q_{i,j}, w_{j+1}) \leftarrow q_{i,j+1}$
        if $i \neq k$ **then**
            $\delta(q_{i,j}, a) \leftarrow q_{i+1,j+1}$ for all $a \in \Sigma \setminus \{w_{j+1}\}$
        else
            $\delta(q_{i,j}, a) \leftarrow q_{\text{exit}}$ for all $a \in \Sigma \setminus \{w_{j+1}\}$
$\delta(q_{\text{exit}}, a) \leftarrow q_{\text{exit}}$ for all $a \in \Sigma$
return $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ ▷ return the DFA

Since we have to intersect the DPFA with the DFA, we have to modify the DPFA to an equivalent λ-free DPFA. For this, we use the result of Higuera [5]. We described it in Algorithm 3.

Algorithm 3 Remove lambda transitions in a PFA [5]

Input: a PFA $\mathcal{M} = (Q, \Sigma, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}})$
Output: the lambda-free PFA $\mathcal{M}\prime$ derived from $\mathcal{M}\prime$

$Q\prime \leftarrow Q \cup \{q_{new}\}$
for $q \in Q$ **do**
    $\delta_{\mathbb{P}}(q_{new}, \lambda, q) \leftarrow \mathbb{I}_{\mathbb{P}}(q)$
    $\mathbb{I}_{\mathbb{P}}(q) \leftarrow 0$
$\mathbb{I}_{\mathbb{P}}(q_{new}) \leftarrow 1$, $\mathbb{F}_{\mathbb{P}}(q_{new}) \leftarrow 0$
$\mathcal{M}\prime = (Q\prime, \Sigma, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}})$ ▷ only one initial state q_{new}
while there still are λ-transitions in $\mathcal{M}\prime$ **do**
    if there exists a λ-loop (q, λ, q) **then**
        for each $(a, q') \in \Sigma \times Q$ **do**
            $\delta_{\mathbb{P}}(q, a, q\prime) \leftarrow \delta_{\mathbb{P}}(q, a, q\prime) \cdot \frac{1}{1-\delta_{\mathbb{P}}(q,\lambda,q)}$
        $\mathbb{F}_{\mathbb{P}}(q) \leftarrow \mathbb{F}_{\mathbb{P}}(q) \cdot \frac{1}{1-\delta_{\mathbb{P}}(q,\lambda,q)}$
        $\delta_{\mathbb{P}}(q, \lambda, q) \leftarrow 0$
    else ▷ there are no λ-loops
        let (q, λ, q_m) be a λ-transition with m maximal
        for each $(q_m, \lambda, q_n, P_\lambda)$ **do** ▷ $n < m$
            $\delta_{\mathbb{P}}(q, \lambda, q_n) \leftarrow \delta_{\mathbb{P}}(q, \lambda, q_n) + \delta_{\mathbb{P}}(q, \lambda, q_m) \cdot \delta_{\mathbb{P}}(q_m, \lambda, q_n)$
        for each (q_m, a, q_n, P_a) **do** ▷ $a \in \Sigma$
            $\delta_{\mathbb{P}}(q, \lambda, q_n) \leftarrow \delta_{\mathbb{P}}(q, a, q_n) + \delta_{\mathbb{P}}(q, \lambda, q_m) \cdot \delta_{\mathbb{P}}(q_m, a, q_n)$
        $\mathbb{F}_{\mathbb{P}}(q) \leftarrow \mathbb{F}_{\mathbb{P}}(q) + \delta_{\mathbb{P}}(q, \lambda, q_m) \cdot \mathbb{F}_{\mathbb{P}}(q_m)$
        $\delta_{\mathbb{P}}(q, \lambda, q_m) \leftarrow 0$
return $\mathcal{M}\prime$

Theorem 5. *Algorithm 3 is correct.*

Proof. proved by Higuera [5]. □

Theorem 6. *Algorithm 3 runs in* $O(|Q_{\mathcal{M}}| \cdot |\Sigma|)$.

Proof. Since the number of transitions of a DPFA is $O(|Q_{\mathcal{M}}| \cdot |\Sigma|)$, the algorithm runs in $O(|Q_{\mathcal{M}}| \cdot |\Sigma|)$. □

Then, we intersect the λ-free DPFA and the Hamming automaton over Σ for w and k. For this, we use the algorithm proposed by Vidal et al. [7], which described in Algorithm 4.

Theorem 7. *Algorithm 4 is correct.*

Proof. proved by Vidal et al. [7]. □

Algorithm 4 Intersect a DPFA and a DFA [7]

Input: a DPFA $\mathcal{M}$ and a DFA $\mathcal{A}$
Output: the intersected sub-DPFA $\mathcal{M}\prime$

1: let $\mathcal{M}\prime$ be a sub-DPFA
2: $Q_{\mathcal{M}'} \leftarrow \mathrm{Q}_{\mathcal{M}} \times Q_{\mathcal{A}}$
3: let the new transition function of $\mathcal{M}\prime$ be a function $\delta_{\mathcal{M}\prime} : Q_{\mathcal{M}'} \times \Sigma \times Q_{\mathcal{M}'} \to \mathbb{Q}^+$
4: initialize all values of $\delta_{\mathcal{M}\prime}$ as 0
5: $\mathbb{I}_{\mathcal{M}\prime}[(q_{\mathcal{M}}, q_{\mathcal{A}})] \leftarrow \begin{cases} \mathbb{I}_{\mathcal{M}}[q_{\mathcal{M}}], & \text{if } q_{\mathcal{M}} \in Q_{\mathcal{M}} \text{ and } q_{\mathcal{A}} = s_{\mathcal{A}} \\ 0, & \text{if } q_{\mathcal{M}} \in Q_{\mathcal{M}} \text{ and } q_{\mathcal{A}} \in Q_{\mathcal{A}} \setminus \{s_{\mathcal{A}}\} \end{cases}$
6: $\mathbb{F}_{\mathcal{M}\prime}[(q_{\mathcal{M}}, q_{\mathcal{A}})] \leftarrow \begin{cases} \mathbb{F}_{\mathcal{M}}[q_{\mathcal{M}}], & \text{if } q_{\mathcal{M}} \in Q_{\mathcal{M}} \text{ and } q_{\mathcal{A}} \in F_{\mathcal{A}} \\ 0, & \text{if } q_{\mathcal{M}} \in Q_{\mathcal{M}} \text{ and } q_{\mathcal{A}} \in Q_{\mathcal{A}} \setminus F_{\mathcal{A}} \end{cases}$
7: **for** each $(q_{\mathcal{A}}, a) \in Q_{\mathcal{A}} \times \Sigma$ **do**
8: $\quad q'_{\mathcal{A}} \leftarrow \delta_{\mathcal{A}}(q_{\mathcal{A}}, a)$
9: $\quad$ **for** each $(q_{\mathcal{M}}, q'_{\mathcal{M}}) \in Q_{\mathcal{M}} \times Q_{\mathcal{M}}$ **do**
10: $\quad\quad \delta_{\mathcal{M}\prime}((q_{\mathcal{M}}, q_{\mathcal{A}}), a, (q'_{\mathcal{M}}, q'_{\mathcal{A}})) \leftarrow \delta_{\mathcal{M}}(q_{\mathcal{M}}, a, q'_{\mathcal{M}})$
11: return $\mathcal{M}\prime$ ▷ return the intersected automaton

Theorem 8. *Algorithm 4 runs in* $O(|\Sigma| \cdot |Q_{\mathcal{M}}|^2 \cdot |Q_{\mathcal{A}}|)$.

Proof. proof omitted since it is trivial. □

Remark 2. Note that, if $Q_{\mathcal{A}}$ is an Hamming automaton over Σ for w and k, this algorithm runs in $O(|\Sigma| \cdot |Q_{\mathcal{M}}|^2 \cdot |Q_{\mathcal{A}}|) = O(k|w| \cdot |\Sigma|^2 \cdot |Q_{\mathcal{M}}|^2|)$, since $|Q_A| = O(k|w| \cdot |\Sigma|)$.

However, the result of intersection is not always the DPFA. It is an instance of sub-DPFA[7], which means it satisfies all structures of DPFA except for the probability condition[7]. So, we should normalize it so that it can be a DPFA. Such an algorithm is suggest by De La Higuera and Oncina [3]. We described it in Algorithm 5.

Theorem 9. *Algorithm 5 is correct.*

Proof. proved by De La Higuera and Oncina [3]. □

Theorem 10. *Algorithm 5 runs in* $O\big(|Q_{\mathcal{M}]}|^2 \cdot |\Sigma|^2\big)$.

Proof. The optimal implementation for line 7 is using breadth first search (BFS). Note that BFS runs in $O(|V| + |E|)$ for a graph $G = (V, E)$. So, it runs in $O\big(|Q_{\mathcal{M}]}|^2 \cdot |\Sigma|^2\big)$, since the number of transitions in a DPFA is $O(|Q_{\mathcal{M}}| \cdot |\Sigma|)$. □

Finally, we obtain the most probable string of the normalized DPFA. We exploit the algorithm proposed by De La Higuera and Oncina [2]. We describe it in Algorithm 6 with the following definition.

Algorithm 5 Normalize a sub-DPFA to a DPFA [3]

Input: a sub-DPFA $\mathcal{M}\prime$
Output: the most probable string $\mathcal{M}$

let $\mathcal{M}$ be a DPFA
$Q_{\mathcal{M}} \leftarrow Q_{\mathcal{M}\prime}$
$\mathbb{I}_{\mathcal{M}} \leftarrow \frac{\mathbb{I}_{\mathcal{M}\prime}}{|\mathbb{I}_{\mathcal{M}\prime}|}$
let the transition function of $\mathcal{M}$ be a function $\delta_{\mathcal{M}} : Q_{\mathcal{M}} \times \Sigma \times Q_{\mathcal{M}} \to \mathbb{Q}^+$
$\delta_{\mathcal{M}}(q_{\mathcal{M}}, \alpha, q'_{\mathcal{M}'}) \leftarrow 0$ for all $(q_{\mathcal{M}'}, \alpha, q'_{\mathcal{M}'}) \in Q_{\mathcal{M}'} \times \Sigma \times Q_{\mathcal{M}'}$
for each $q_i \in Q_{\mathcal{M}}$ **do**
    $w \leftarrow$ an arbitrary string that starts from initial state and ends at q_i
    $\mathbb{F}_{\mathcal{M}\prime}[q_i] \leftarrow \frac{p(w)}{prefix(w)}$
    for each $\alpha \in \Sigma$ **do**
        $q' \leftarrow \arg\max_{q \in \Sigma} \delta(q_i, \alpha, q)$
        $\delta_{\mathcal{M}}(q_i, \alpha, q\prime) \leftarrow \frac{prefix(w\alpha)}{prefix(w)}$
return $\mathcal{M}$ ▷ return the normalized DPFA given a sub-DPFA

Definition 4. Let $\mathcal{M}$ be a PFA. The *Potential Probability* $\mathcal{PP}$ of a prefix string w is

$$\mathcal{PP}(w) \leftarrow \min(\mathcal{P}r_{\mathcal{M}}(w\Sigma^*), \frac{|Q_{\mathcal{M}}|^2}{|w|}) \tag{2}$$

Theorem 11. *Algorithm 6 is correct.*

Proof. proved by De La Higuera and Oncina [2]. □

Theorem 12. *Algorithm 6 runs in* $O\left(\log\left(\frac{|\Sigma| \cdot |Q_{\mathcal{M}}|}{p_{opt}^2}\right)\right)$.

Proof. proved by De La Higuera and Oncina [2]. □

Therefore, given a DPFA $\mathcal{M}$, a string w, and a distance limit k, the proposed algorithm to obtain the most probable string within distance k from w is as in Algorithm 7.

Theorem 13. *Algorithm 7 is correct.*

Proof. explained above. □

Theorem 14. *Algorithm 7 runs in*

$$O\left(k^2|w|^2 \cdot |\Sigma|^4 \cdot |Q_{\mathcal{M}}|^2 + \log\left(\frac{k|w| \cdot |\Sigma|^2 \cdot |Q_{\mathcal{M}}|}{p_{opt}^2}\right)\right).$$

Algorithm 6 Most probable string of a probabilistic finite-state automaton [2]

Input: a deterministic probabilistic finite-state automaton $\mathcal{M}$
Output: the most probable string w^*

$p_{\text{opt}} \leftarrow 0$
let $\mathbf{Q}$ be a priority queue
$\mathbf{Q} \leftarrow [\lambda]$
$Continue \leftarrow$ True
while $\mathbf{Q}$ is not empty and $Continue$ is True **do**
    $w \leftarrow \textsc{Pop}(\mathbf{Q})$
    if $\mathcal{PP}(w) > p_{\text{opt}}$ **then**
        $p \leftarrow \mathcal{P}r_{\mathcal{M}}(w)$
        if $p > p_{\text{opt}}$ **then**
            $p_{\text{opt}} \leftarrow p$
            $w^* \leftarrow w$
        for each $a \in \sigma$ **do**
            if $\mathcal{PP}(wa) > p_{\text{opt}}$ **then**
                $\textsc{Insert}(wa, \mathcal{PP}(wa), \mathbf{Q})$
    else
        $Continue \leftarrow$ False
return w^* ▷ return the most probable string

Algorithm 7 Intersection based MPS within hamming distance k

Input: a deterministic probabilistic automaton $\mathcal{M}$, a string w, and a distance limit k
Output: the most probable string w^* of $\mathcal{M}$ such that $d(w^*, w) \leq k$

$\mathcal{A} \leftarrow \textsc{ConstructHammingAutomata}(w, k)$
$\mathcal{M}\prime \leftarrow \textsc{RemoveLambdaTransitions}\mathcal{M}'$
$\mathcal{M}' \leftarrow \textsc{Intersect}(\mathcal{M}, \mathcal{A})$
$\mathcal{M}'' \leftarrow \textsc{Normalize}(\mathcal{M}')$
$w^* \leftarrow \textsc{MostProbableString}(\mathcal{M}'')$
return w^*

Proof. Lines from 1 to 3 run in $O(k|w| \cdot |\Sigma|^2 \cdot |Q_{\mathcal{M}}|^2|)$ by Theorem 4, 6, 8, and Remark 2. In addition, according to Remark 2,

$$|Q_{\mathcal{M}\prime}| = |Q_{\mathcal{M}}| \cdot |Q_{\mathcal{A}}| = O(k|w| \cdot |\Sigma| \cdot |Q_{\mathcal{M}}|) = |Q_{\mathcal{M}\prime\prime}|.$$

So, line 4 runs in

$$O(|Q_{\mathcal{M}\prime}|^2 \cdot |\Sigma|^2) = O(k^2|w|^2 \cdot |\Sigma|^4 \cdot |Q_{\mathcal{M}}^2|)$$

by Theorem 10 and line 5 does in

$$O\bigg(\log\bigg(\frac{|\Sigma| \cdot |Q_{\mathcal{M}\prime\prime}|}{p_{\text{opt}}^2}\bigg)\bigg) = O\bigg(\log\bigg(\frac{k|w| \cdot |\Sigma|^2 \cdot |Q_{\mathcal{M}}|}{p_{\text{opt}}^2}\bigg)\bigg).$$

Thus, the algorithm runs in

$$O\bigg(k^2|w|^2 \cdot |\Sigma|^4 \cdot |Q_{\mathcal{M}}|^2 + \log\bigg(\frac{k|w| \cdot |\Sigma|^2 \cdot |Q_{\mathcal{M}}|}{p_{\text{opt}}^2}\bigg)\bigg).$$

□

References

[1] Colin De La Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010, pp. 99–132.

[2] Colin De La Higuera and José Oncina. "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*. 2013, pp. 1–8.

[3] Colin De La Higuera and José Oncina. "Learning stochastic finite automata". In: *International Colloquium on Grammatical Inference*. Springer. 2004, pp. 175–186.

[4] Colin De La Higuera and José Oncina. "The most probable string: an algorithmic study". In: vol. 24. 2. Oxford University Press, 2013, pp. 311–330.

[5] Colin de la Higuera. "Why ϵ-transitions are not necessary in probabilistic finite automata". In: *EURISE, University of Saint-Etienne, Tech. Rep* 301 (2003).

[6] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th international symposium on symbolic and algebraic computation*. ACM. 2014, pp. 296–303.

[7] Enrique Vidal et al. "Probabilistic finite-state machines-part I". In: vol. 27. 7. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, pp. 1013–1025.

[8] Enrique Vidal et al. "Probabilistic finite-state machines-part II". In: vol. 27. 7. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, pp. 1026–1039.