Most Probable String within Distance k
- interim 1 -

Yumin Kim Myeongjang Pyeon Hak-Su Lim

September 2018

1 Probabilistic Finite-State Automata

There are DFA/NFA that represent the accepting model to verify the acceptance of the strings, but if the probabilities is added to the transitions of the automata [5, 6]. $A=\langle \Sigma, Q, \mathbb{I}_{\mathbb{P}}, \mathbb{F}_{\mathbb{P}}, \delta_{\mathbb{P}} \rangle$ this is called PFA and represents the string generating model. 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:

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

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

If the PFA generates a string, the string has a probability of being generated

by the PFA. We can also find the distributions 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

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

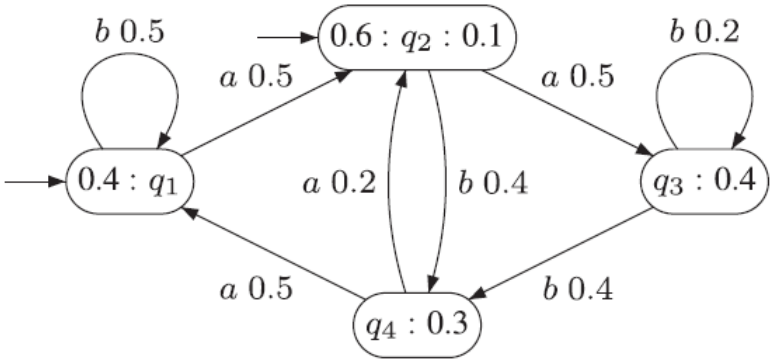

Figure 1: Graphical representation of a PFA [1]

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.[3, 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.[3, 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.[3, 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 of $\mathcal{M}$.

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 [4]. 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. □

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 [4]. 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 $O\big(n^2|Q|^{2+O(1)} + k|\Sigma|^k\binom{n}{k}|Q|^{2+O(1)}\big)$. □

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^*

5 Implementation

We implemented a PFA framework based on Python. In order to boost the performance, we used the Numpy library for parallel matrix processing. The detailed class structures are depicted in Figure 2.

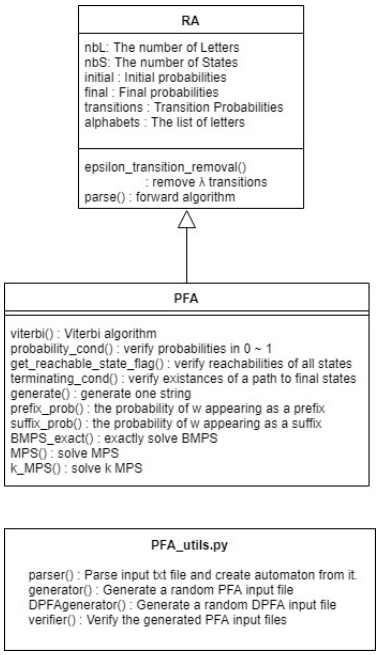

Figure 2: Class and function diagram of our implementation

6 Experiments

In order to compare our algorithm to the brute-force algorithm, we implemented both algorithms and test as adjusting n and k by sampling DPFA and strings. We sampled 10 DPFA and 100 strings per each n and k pair.

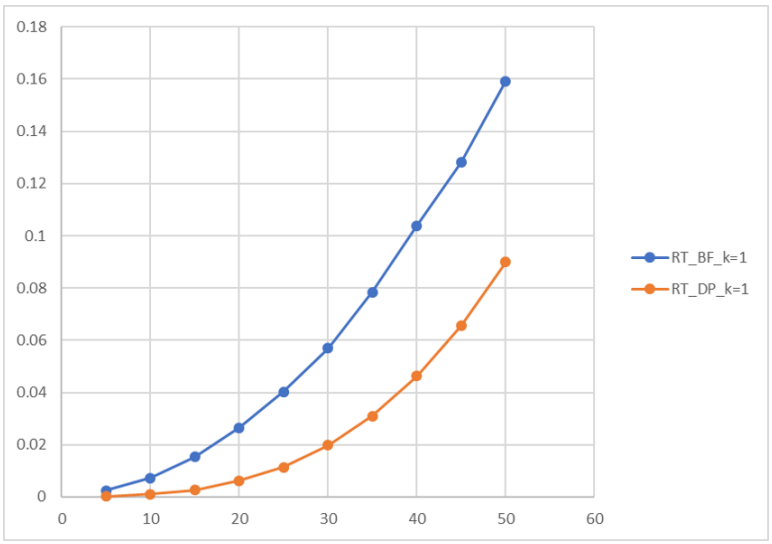

Figure 3: Running time chart when k = 1

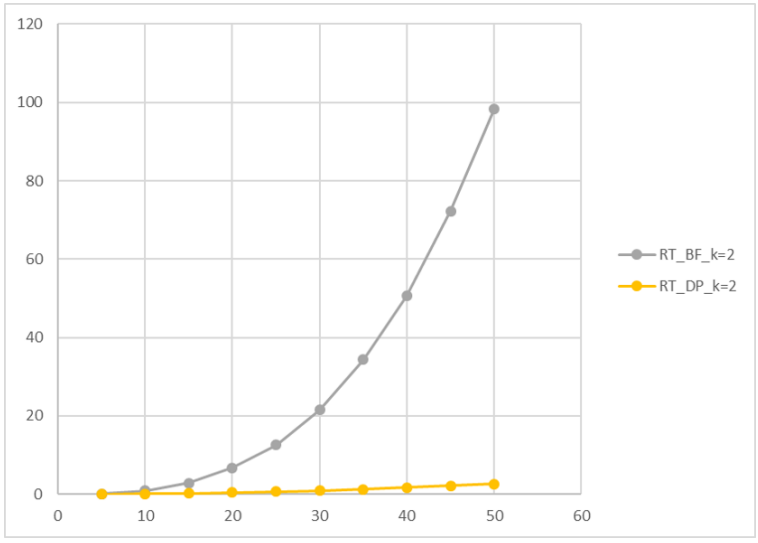

Figure 4: Running time chart when k = 2

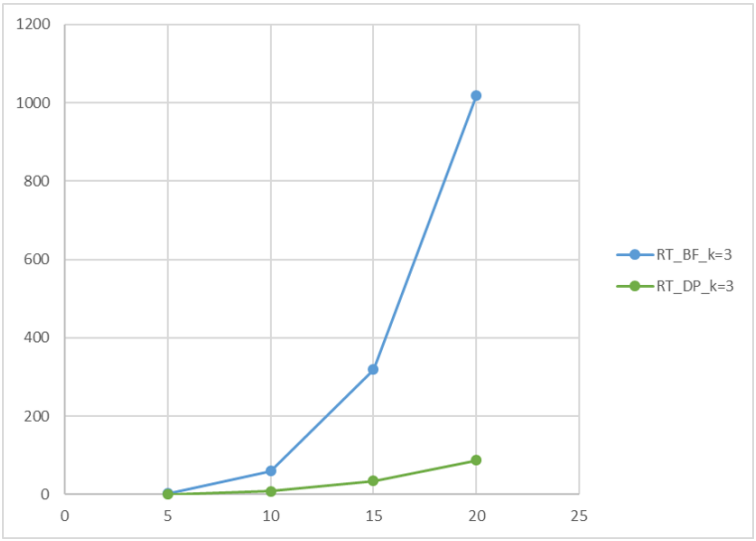

Figure 5: Running time chart when k = 3

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. "The most probable string: an algorithmic study". In: vol. 24. 2. Oxford University Press, 2013, pp. 311–330.

[4] 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.

[5] 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.

[6] 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.