Online Infix Probability Computation for Probabilistic Finite Automata

Marco Cognetta, Yo-Sub Han, and **Soon Chan Kwon***
Department of Computer Science
Yonsei University, Seoul, Republic of Korea
cognetta.marco@gmail.com, {emmous, soon--chan}@yonsei.ac.kr

Abstract

Probabilistic finite automata (PFAs) are common statistical language model in natural language and speech processing. A typical task for PFAs is to compute the probability of all strings that match a query pattern. An important special case of this problem is computing the probability of a string appearing as a prefix, suffix, or infix. These problems find use in many natural language processing tasks such word prediction and text error correction.

Recently, we gave the first incremental algorithm to efficiently compute the infix probabilities of each prefix of a string (Cognetta et al., 2018). We develop an asymptotic improvement of that algorithm and solve the open problem of computing the infix probabilities of PFAs from streaming data, which is crucial when processing queries online and is the ultimate goal of the incremental approach.

1 Introduction

Weighted automata are a popular weighted language model in natural language processing. They have found use across the discipline both alone (Mohri et al., 2002) and in conjunction with more complicated language models (Ghazvininejad et al., 2016; Velikovich et al., 2018). As such, finding efficient algorithms for weighted automata has become an intensely studied topic (Allauzen and Mohri, 2009; Argueta and Chiang, 2018).

An important subclass of weighted automata are PFAs. Given a PFA, one important task is to calculate the probability of a phrase or pattern. Efficient algorithms exist for this problem when given a PFA or a probabilistic context-free grammar (PCFG) and a pattern that forms a regular language (Vidal et al., 2005a; Nederhof and Satta, 2011). One important special case of this problem is to compute the probability of all strings containing a given infix, which was first studied by Corazza et al. (1991). The problem was motivated by applications to phrase prediction and error correction. Several partial results were established with various restrictions on the statistical model or infix (Corazza et al., 1991; Fred, 2000; Nederhof and Satta, 2011). Later, Nederhof and Satta (2011) gave a general solution for PCFGs and proposed the problem of computing the infix probabilities of each prefix of a string *incrementally*—using the infix probability of $w_1 w_2 \dots w_k$ to speed up the calculation for $w_1 w_2 \dots w_k w_{k+1}$.

Recently, we gave an algorithm for this problem when the language model is a PFA, and suggested an open problem of *online* incremental infix probability calculation—where one is given a stream of characters instead of knowing the entire input string ahead of time (Cognetta et al., 2018). The online problem is of special practical importance as it is a more realistic setting than the offline problem. Not only do many speech processing tasks need to be performed "on the fly", but also many parsing algorithms can be improved by utilizing an online algorithm. For example, suppose one has calculated the infix probability of all prefixes of the phrase "...be or...", and later wishes to extend that phrase to "...be or not to be..." and retrieve all of the new infix probabilities. Instead of restarting the computation from the beginning, which would lead to redundant computation, an online method can be used to simply start from where the initial algorithm left off. As another example, suppose we have the phrase "...United States of...", and wish to extend it by a word while maximizing the resulting infix probability. An online algorithm can be used to try all extensions in the vocabulary before settling on "America", whereas naively applying an offline algorithm would require repeatedly computing already known values.

*Now at Google Korea.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5332–5337
Florence, Italy, July 28 - August 2, 2019.

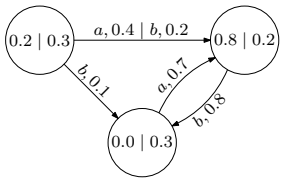

Figure 1: An example PFA. Each state has an initial and final probability, and each transition has a label and transition probability.

We first revisit our original incremental infix probability algorithm from (Cognetta et al., 2018) and improve the algorithm based on a careful reanalysis of the dynamic programming recurrence. Then, we develop an algorithm for the online incremental infix problem and demonstrate the practical effectiveness of the two new algorithms on series of benchmark PFAs.

2 Preliminaries

We assume that the reader is familiar with the definition and basic properties of automata theory. For a thorough overview of PFAs, we suggest (Vidal et al., 2005a,b).

A PFA is specified by a tuple $\mathcal{P} = (Q, \Sigma, \{\mathbb{M}(c)\}_{c\in\Sigma}, \mathbb{I}, \mathbb{F})$, where Q is a set of states and Σ is an alphabet. The set $\{\mathbb{M}(c)\}_{c\in\Sigma}$ is a set of labeled $|Q| \times |Q|$ transition matrices—the element $\mathbb{M}(c)_{i,j}$ is the probability of transitioning from state q_i to q_j reading character c. Likewise, $\mathbb{I}$ is a $1 \times |Q|$ initial probability vector and $\mathbb{F}$ is a $|Q| \times 1$ final probability vector. PFAs have some conditions on their structure. Specifically, $\sum_{i=1}^{|Q|} \mathbb{I}_i = 1$ and for each state q_i, $\mathbb{F}_i + \sum_{c\in\Sigma,\ j\in[1,|Q|]} \mathbb{M}(c)_{i,j} = 1$. Finally, each state must be accessible and co-accessible. When these are met, a PFA describes a probability distribution over Σ^*. The probability of a string is given as $\mathcal{P}(w) = \mathbb{I}\left(\prod_{i=1}^{|w|} \mathbb{M}(w_i)\right)\mathbb{F}$. Let $\mathbb{M}(\Sigma) = \sum_{c\in\Sigma} \mathbb{M}(c)$. Then, we can find the infinite sum $\sum_{i=0}^{\infty} \mathbb{M}(\Sigma)^i = (\mathbf{1} - \mathbb{M}(\Sigma))^{-1}$, where $\mathbf{1}$ is the identity matrix. We denote this matrix $\mathbb{M}(\Sigma^*)$ and note that $\mathbb{I}\mathbb{M}(\Sigma^*)\mathbb{F} = 1$.

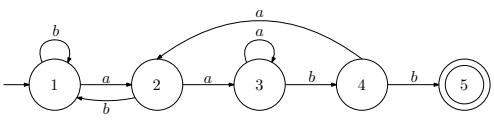

Figure 2: The KMP DFA for $w = aabb$.

The KMP automaton of w is a DFA with $|w|+1$ states that accepts the language of strings ending with the first occurrence of w, and can be built in $O(|w|)$ time (Knuth et al., 1977). By convention, the states of a KMP DFA are labeled from q_1 to $q_{|w|+1}$, with the transition between q_i and q_{i+1} corresponding to w_i. Figure 2 gives an example.

3 Incremental Infix Algorithm

We now review the method described in (Cognetta et al., 2018). The algorithm is based on state elimination for DFAs (Book et al., 1971). Given a DFA, we add two new states q_0 and q_{n+1}, where q_0 is connected by λ-transitions (λ is the empty string) to all initial states and all final states are connected to q_{n+1} by λ-transitions. We then perform a dynamic state elimination procedure to produce regular expressions $\alpha_{i,j}^k$ that describe the set of strings that, when read starting at state i, end at state j and never pass through a state with label higher than k. We use the recurrence $\alpha_{i,j}^k = \alpha_{i,j}^{k-1} + \alpha_{i,k}^{k-1}(\alpha_{k,k}^{k-1})^*\alpha_{k,j}^{k-1}$, with the base case $\alpha_{i,j}^0$ being the transitions from q_i to q_j. This method forms a regular expression stored in $\alpha_{0,n+1}^n$ that describes the same language as the input DFA. Furthermore, this regular expression is *unambiguous* in that there is at most one way to match a string in the language to the regular expression (Book et al., 1971). We then described a mapping from regular expressions to expressions of transition matrices of a PFA (Table 1) and proved that evaluating the matrix formed by the mapping gives the probability of all strings matching the regular expression (Cognetta et al., 2018).

Regex	Matrix	Regex	Matrix
$\emptyset$	$\mathbf{0}$	$\mathcal{R} + \mathcal{S}$	$\mathbb{M}(\mathcal{R}) + \mathbb{M}(\mathcal{S})$
λ	$\mathbf{1}$	$\mathcal{R}\mathcal{S}$	$\mathbb{M}(\mathcal{R})\mathbb{M}(\mathcal{S})$
c	$\mathbb{M}(c)$	$\mathcal{R}^*$	$(\mathbf{1} - \mathbb{M}(\mathcal{R}))^{-1}$

Table 1: A mapping from regular expressions to expressions of transition matrices.

The basic idea behind the incremental algorithm is the following: the KMP DFA describes the infix language of the input string w. When performing the state elimination procedure, the term $a_{0,k+1}^k$ is the regular expression for the infix language of $w_1 w_2 \dots w_k$. Further, the term $a_{0,k+2}^{k+1} = \alpha_{0,k+1}^k(\alpha_{k+1,k+1}^k)^*\alpha_{k+1,k+2}^k$ includes the term $\alpha_{0,k+1}^k$ and so the result from each iteration can be used in the next. The algorithm then performs state elimination while interpret-

Algorithm 1 Incremental Infix

```
1:  procedure INFIX(w = w_1 ... w_n, PFA P)
2:      D ← KMP DFA for w
3:      T ← (n+3) × (n+3) table
4:      T_{0,1}, T_{n+1,n+2} ← 1
5:      for (q_i, c) ∈ δ do
6:          T_{i,δ(q_i,c)} ← T_{i,δ(q_i,c)} + M(c)
7:      X ← 1                                  ▷ X holds α^k_{0,k+1}.
8:      for k ∈ [1, n+1] do
9:          X ← X(1 − T_{k,k})^{-1} T_{k,k+1}
10:         yield IXM(Σ*)F                     ▷ P(Σ* w_1 ... w_k Σ*)
11:         T' ← (n+3) × (n+3) table
12:         for i ∈ [0, n+2]; j ∈ [0, n+2] do
13:             T'_{i,j} ← T_{i,j} + T_{i,k}(1 − T_{k,k})^{-1} T_{k,j}
14:         T ← T'
```

ing the terms $\alpha^k_{i,j}$ as matrices and outputs $\alpha^k_{0,k+1}$ at each step to retrieve the infix probability of $w_1 w_2 \dots w_k$. The algorithm based on this idea is given in Algorithm 1 and has a runtime of $O(|w|^3 |Q_\mathcal{P}|^m)$. We note that this analysis is considering the alphabet to be constant sized. For the remainder of the paper, we deal with variable sized (but finite) alphabet sizes. Accounting for this, the true runtime is $O(|\Sigma||w||Q_\mathcal{P}|^2 + |w|^3 |Q_\mathcal{P}|^m)$[†], with the $O(|\Sigma||w||Q_\mathcal{P}|^2)$ term coming from the initial table setup in Lines 5 to 6.

4 Asymptotic Speedup

We now describe an asymptotic speedup for Algorithm 1 based on the following two lemmas.

Lemma 1. *Computing $\alpha^n_{0,n+1}$ only requires knowledge of the terms of the form $\alpha^k_{i,j}$, where $i, j \geq k+1$, or of the form $a^k_{0,k+1}$.*

In other words, only the term $\alpha^k_{0,k+1}$ and the terms in the bottom right $k \times k$ sub-table of α^k need to be considered at step $k+1$.

Lemma 2. *Consider $\alpha^k_{i,j}$ where $k+1 \leq i < j$. Then $\alpha^k_{i,j} = \alpha^{k-1}_{i,j}$.*

Lemmas 1 and 2 imply that only $O(|w| - k) = O(|w|)$ matrix multiplications/inversions need to be performed per iteration of Algorithm 1, leading to Theorem 3.

Theorem 3. *Algorithm 1 can be made to run in $O(|\Sigma||w||Q_\mathcal{P}|^2 + |w|(|w||Q|^m)) = O(|\Sigma||w||Q_\mathcal{P}|^2 + |w|^2 |Q|^m)$ time when accounting for the preprocessing step.*

The new algorithm is faster than the previous known runtime of $O(|\Sigma||w||Q_\mathcal{P}|^2 + |w|^3 |Q|^m)$. To implement this speed-up, we change the iteration range in Line 11 to of Algorithm 1 to be **for** $i \in [k+1, n+2]; j \in [k+1, n+2]$ and set $T'_{i,j} = T_{i,j}$ when $j \geq k+2$. For the remaining $O(k)$ values, we compute the term $T'_{i,j} = T_{i,j} + T_{i,k}(\mathbf{1} - T_{k,k})^{-1} T_{k,j}$ as normal.

[†]The constant m is such that $n \times n$ matrices can be multiplied or inverted in $O(n^m)$ time. In practice, m is often ≈ 2.807 (Strassen, 1969).

5 Online Incremental Infix Calculation

We now consider the problem of determining the infix probabilities of strings given as a stream of characters. This is in contrast to the setting from Algorithm 1 and (Cognetta et al., 2018) in which the entire string was known ahead of time.

In this setting, we build the KMP automaton step by step (instead of all at once at the beginning), and then eliminate the most recent state to maintain our dynamic programming table. The key difficulty in this method is that when adding a new state, $|\Sigma| - 1$ back transitions (and 1 forward transition) are added to the DFA. The label and destination of each back transition cannot be predicted until a new character is added, the back transitions can go to any state up to the current one, and different configurations can arise depending on the newly added character. Together, these make correctly accounting for the paths that are generated at each step non-trivial.

Lemma 4. *The term $\alpha^k_{k+1,k+1}$ can be computed as $\sum_{c \in \Sigma - w_k} c(\alpha^{k-1}_{\delta(q_{k+1},c),k+1} + \alpha^{k-1}_{\delta(q_{k+1},c),k} (\alpha^{k-1}_{k,k})^* \alpha^{k-1}_{k,k+1})$.*

The basic intuition of Lemma 4 is to concatenate the character from the backwards transition to the front of every string that brings state $\delta(q_i, c)$ to state q_{k+1}. When finding $\alpha^k_{i,k+1}$ where $i \leq k$, the term can be computed as normal and evaluating $\alpha^k_{k+1,k+1}$ takes $O(|\Sigma||Q_\mathcal{P}|^m)$ time.

Lemma 5. *In the online setting, at each iteration k, only the $k+1$th column of table T' needs to be evaluated.*

In contrast to Lemma 1 in the offline setting, where only the elements in the $k+1$-th column below index k need to be computed, *all* elements of the $k+1$-th column need to be evaluated in the online setting. This is due to the sum in Lemma 4 being dependent on the terms $\alpha^{k-1}_{\delta(q_{k+1},c),k}$ because $\delta(q_{k+1}, c)$ can take on any value in $[1, k]$. Nevertheless, this leads to the following result.

Theorem 6. *Given a stream of characters $w = w_1 w_2 \dots$, the infix probability of each prefix*

$\lvert Q\rvert, \lvert\Sigma\rvert$	$500, 26$			$500, 100$			$1500, 26$			$1500, 100$		
Alg / $\lvert w\rvert$	Alg 1	Faster	Online	Alg 1	Faster	Online	Alg 1	Faster	Online	Alg 1	Faster	Online
1	0.917	0.103	0.104	0.912	0.107	0.198	13.396	1.780	1.201	13.371	1.720	1.605
2	0.904	0.106	0.098	0.903	0.106	0.205	13.196	1.649	1.320	13.382	1.570	1.750
3	0.909	0.089	0.110	0.926	0.085	0.214	13.154	1.446	1.459	13.290	1.447	1.849
4	0.933	0.075	0.125	0.966	0.074	0.225	13.333	1.295	1.609	13.342	1.273	1.986
5	0.891	0.068	0.133	0.930	0.067	0.238	13.378	1.161	1.763	13.319	1.143	2.135
6	0.917	0.060	0.145	0.931	0.055	0.241	14.352	1.002	1.898	13.282	0.994	2.254
7	0.964	0.051	0.156	0.942	0.053	0.251	14.287	0.869	2.056	13.571	0.832	2.368
8	0.929	0.042	0.192	0.950	0.044	0.259	14.330	0.735	2.189	13.614	0.702	2.479
9	0.912	0.035	0.207	0.954	0.035	0.269	14.673	0.591	2.367	13.661	0.568	2.679
10	0.917	0.026	0.094	0.925	0.027	0.203	13.847	0.447	1.596	13.627	0.445	1.507
Total	9.194	0.656	1.365	9.341	0.663	2.307	137.947	10.976	17.459	134.462	10.694	20.615

Table 2: Timings from the experimental analysis of each algorithm. Alg 1 refers to Algorithm 1. "Faster" refers to the speedup described in Theorem 3. Online refers to Algorithm 2. All results are in seconds.

Algorithm 2 Online Incremental Infix

procedure INFIX(Stream $w = w_1 w_2 \ldots$, PFA $\mathcal{P}$)
  $\mathcal{D} \leftarrow$ KMP DFA for w_1
  $T \leftarrow$ re-sizable table
  $T_{0,1} \leftarrow \mathbf{1}$
  for $i \in [1,3]; j \in [1,3]; c \in \Sigma$ **do**
    if $\delta(q_i, c) = q_j$ **then**
      $T_{i,j} \leftarrow T_{i,j} + \mathbb{M}(c)$
  $\mathbb{X} \leftarrow \mathbf{1}, k \leftarrow 1$ ▷ $\mathbb{X}$ holds $\alpha_{0,k+1}^{k}$.
  while w is not exhausted **do**
    Extend $\mathcal{D}$ with new character
    $\mathbb{X} \leftarrow \mathbb{X}(\mathbf{1} - T_{k,k})^{-1} T_{k,k+1}$
    yield $\mathbb{I}\mathbb{X}\mathbb{M}(\Sigma^*)\mathbb{F}$ ▷ $\mathcal{P}(\Sigma^* w_1 \ldots w_k \Sigma^*)$
    $T' \leftarrow$ re-sizable table
    for $i \in [0, k+1]$ **do**
      $j \leftarrow k+1$
      if $i \leq k$ **then**
        $T'_{i,j} \leftarrow T_{i,j} + T_{i,k}(\mathbf{1} - T_{k,k})^{-1} T_{k,j}$
      else if $i = k+1$ **then**
        $T'_{i,j} = \sum_{c \in \Sigma - \{w_k\}} \mathbb{M}(c) T_{\delta(q_i, c), j}$
    $T \leftarrow T', k \leftarrow k+1$

of w can be computed online in $O(|w|(|w| + |\Sigma|)|Q_{\mathcal{P}}|^m)$ time.

6 Experimental Results

We now demonstrate the practical effectiveness of the improved and online algorithms. We generate a series of PFAs with varying state space and alphabet size. Because we store transition matrices as dense matrices and the algorithms depend only on $|Q|$ and $|\Sigma|$ (but not the number of transitions), the underlying structure of the PFA is unimportant. Thus, we can artificially generate the PFAs to control $|Q|$ and $|\Sigma|$ exactly. We consider PFAs with $|\Sigma| \in \{26, 100\}$ and $|Q| \in \{500, 1500\}$. For each test, we use a random string of 10 characters and measure the time to perform each iteration of Algorithm 1, the asymptotic speedup described in Section 4, and Algorithm 2. We list the median of 10 trials for each iteration. The tests were implemented using Python 3.5 and NumPy and run on an Intel i7-6700 processor with 16gb of RAM. Table 2 contains the experimental results.

Note that the asymptotic speedup and online algorithm outperform Algorithm 1 in every setting, which is in line with our theoretical analysis. Across all trials, each iteration of the improved algorithm speeds up while the online version slows down. These observations are not unexpected. The improved version only recomputes a $k \times k$ sub-table at iteration k and only requires $O(|w| - k)$ multiplications. On the other hand, the online algorithm must perform $O(k + |\Sigma|)$ multiplications at iteration k so we expect the runtime to slowly increase. Unlike the online version, the number of operations per iteration of Algorithm 1 and the improved version do not depend on $|\Sigma|$, so their runtimes do not differ as $|\Sigma|$ grows.

Consider the second use case for the online algorithm from Section 1, where we have a 500-state PFA with $|\Sigma| = 26$ and an input string of length 9, which we wish to extend while maximizing the resulting infix probability. We extrapolate from the timings in Table 2 and anticipate that finding the appropriate extension would take $26 * 0.656 \approx 17.056$ seconds using the faster offline algorithm. On the other hand, we expect the online method to only take $1.271 + 26 * 0.094 \approx 3.715$ seconds.

7 Conclusion

Building off of our previous work, we have considered the problem of incrementally computing the infix probabilities of each prefix of a given string. We provide an improved analysis of our incremental algorithm that leads to an asymptotic speedup. Furthermore, we solve the open problem of computing the infix probabilities of each prefix of a stream of characters. The problem of adapting

this approach to higher order statistical language models (such as PCFGs) remains open.

Acknowledgments

This work was supported by the Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (2018-0-00247).

References

Cyril Allauzen and Mehryar Mohri. 2009. N-way composition of weighted finite-state transducers. *International Journal of Foundations of Computer Science*, 20(4):613–627.

Arturo Argueta and David Chiang. 2018. Composing finite state transducers on GPUs. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 2697–2705.

Ronald Book, Shimon Even, Sheila Greiback, and Gene Ott. 1971. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, 20:149–153.

Marco Cognetta, Yo-Sub Han, and Soon Chan Kwon. 2018. Incremental computation of infix probabilities for probabilistic finite automata. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2732–2741.

Anna Corazza, Renato De Mori, Roberto Gretter, and Giorgio Satta. 1991. Computation of probabilities for an island-driven parser. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):936–950.

Ana L. N. Fred. 2000. Computation of substring probabilities in stochastic grammars. In *Grammatical Inference: Algorithms and Applications*, pages 103–114.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191.

Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. 1977. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Mark-Jan Nederhof and Giorgio Satta. 2011. Computation of infix probabilities for probabilistic context-free grammars. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1213–1221.

Volker Strassen. 1969. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356.

Leonid Velikovich, Ian Williams, Justin Scheiner, Petar S. Aleksic, Pedro J. Moreno, and Michael Riley. 2018. Semantic lattice processing in contextual automatic speech recognition for google assistant. In *Interspeech*, pages 2222–2226.

Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. 2005a. Probabilistic finite-state machines–part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1013–1025.

Enrique Vidal, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. 2005b. Probabilistic finite-state machines–part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1026–1039.

A Proofs

Lemma 1. *Computing $\alpha^n_{0,n+1}$ only requires knowledge of the terms of the form $\alpha^k_{i,j}$, where $i, j \geq k+1$, or of the form $a^k_{0,k+1}$.*

Proof. This can be seen by expanding the term $\alpha^n_{0,n+1}$. As $\alpha^n_{0,n+1} = \alpha^{n-1}_{0,n+1} + \alpha^{n-1}_{0,n}(\alpha^{n-1}_{n,n})^*\alpha^{n-1}_{n,n+1}$. The term $\alpha^{n-1}_{0,n+1}$ is always the empty set as there is no path from state $n-1$ to $n+1$ that does not go through state n in the KMP DFA. Recursively applying this expansion to $\alpha^{n-1}_{n,n}$ and $\alpha^{n-1}_{n,n+1}$ proves the claim. □

Lemma 2. Consider $\alpha^k_{i,j}$ where $k+1 \leq i < j$. Then $\alpha^k_{i,j} = \alpha^{k-1}_{i,j}$.

Proof. Let $i = k+1+x$ and $j = k+1+y$ where $x \geq 0$ and $y > 0$. Consider the expansion of the term $\alpha^k_{k+1+x,k+1+j} = \alpha^{k-1}_{k+1+x,k+1+j} + \alpha^{k-1}_{k+1+x,k}(\alpha^{k-1}_{k,k})^*\alpha^{k-1}_{k,k+1+j}$. In the KMP DFA, state q_i has exactly one transition to state q_{i+1} and $|\Sigma| - 1$ transitions to lower (or equal) states. In other words, there is no path from a state of label i to a state with label at least $i+2$ that does not go through state $i+1$. Thus, $\alpha^{k-1}_{k,k+1+y} = \emptyset$. Then, $\alpha^{k-1}_{k+1+x,k}(\alpha^{k-1}_{k,k})^*\alpha^{k-1}_{k,k+1+y} = \emptyset$, so $\alpha^k_{k+1+x,k+1+j} = \alpha^{k-1}_{k+1+x,k+1+j}$. □

Theorem 3. *In Algorithm 1, the k-th iteration requires only $O(|w|)$ matrix inversions and multiplications to update the dynamic programming table.*

Proof. We use Lemmas 1 and 2. At iteration k of Algorithm 1, Lemma 1 states that we only need to update the lower right $k \times k$ table as that is all

that is required to complete the $k+1$-th iteration. Lemma 2 tells us that all of the terms in the lower right $k \times k$ table except for the terms in the k-th column are the same as in the previous iteration. Thus, those terms can simply be copied and the $O(|w|)$ terms in the k-th column will be updated normally, with only. □

Lemma 4. *The term* $\alpha^k_{k+1,k+1}$ *can be computed as* $\sum_{c \in \Sigma - w_k} c(\alpha^{k-1}_{\delta(q_{k+1},c),k+1} + \alpha^{k-1}_{\delta(q_{k+1},c),k}(\alpha^{k-1}_{k,k})^* \alpha^{k-1}_{k,k+1})$.

Proof. For simplicity, we assume there are no self loops in the KMP DFA except on the initial state. The case where there are can be handled similarly. Note that there can only be at most one self loop not on the initial state of a KMP DFA. Such a self loop will be on the state corresponding to the last state where $w_k = w_{k-1} = \dots w_1$.

First, we expand the term $\alpha^k_{k+1,k+1} = \alpha^{k-1}_{k+1,k+1} + \alpha^{k-1}_{k+1,k}(\alpha^{k-1}_{k,k})^* \alpha^{k-1}_{k,k+1}$. Since we assume there are no self loops on states k or $k+1$, we can simplify the expression to be $\alpha^k_{k+1,k+1} = \alpha^{k-1}_{k+1,k}\alpha^{k-1}_{k,k+1}$. The term $\alpha^{k-1}_{k,k+1}$ is whatever character is on the transition from state k to $k+1$. On the other hand, $\alpha^{k-1}_{k+1,k}$ is the set of paths that take state $k+1$ to state k without passing through states higher than k.

□

Lemma 5. *In the online setting, at each iteration* k*, only the* $k+1$*th column of table* T' *needs to be evaluated.*

Proof. First, we know that $\alpha^k_{k+1,k+1}$ requires knowledge of each term in the kth column of α^{k-1}. Further, expanding the term $\alpha^k_{i,k+1}$ shows that only terms on the k-th and $k+1$-th column of α^{k-1} are required for any of them. Elements on the $k+1$th column of α^{k-1} are equal to the transitions between state q_i and q_{k+1} per Lemma 2. We then proceed by induction on k and the claim follows. □

Theorem 6. *Given a stream of characters* $w = w_1 w_2 \dots$*, the infix probability of each prefix of* w *can be computed online in* $O(|w|(|w| + |\Sigma|)|Q_{\mathcal{P}}|^m)$ *time.*

At iteration k, we need only recompute the k-th column in the table. All but the k-th element in the column are computed using the normal recurrence which each require $O(1)$ multiplications. Computing the k-th element requires $O(|\Sigma|)$ multiplications and inversions, so in total each iteration requires $O(k + |\Sigma|)$ matrix multiplications. Since $O(k) = O(|w|)$ and we perform $O(|w|)$ iterations, we find the runtime is $O(|w|(|w| + |\Sigma|)|Q_{\mathcal{P}}|^m)$.