General Suffix Automaton Construction Algorithm and Space Bounds

Mehryar Mohri[a,b], Pedro Moreno[b], Eugene Weinstein[a,b]

[a] *Courant Institute of Mathematical Sciences*
251 Mercer Street, New York, NY 10012.
[b] *Google Research*
76 Ninth Avenue, New York, NY 10011.

Abstract

Suffix automata and factor automata are efficient data structures for representing the full index of a set of strings. They are minimal deterministic automata representing the set of all suffixes or substrings of a set of strings. This paper presents a novel analysis of the size of the suffix automaton or factor automaton of a set of strings. It shows that the suffix automaton or factor automaton of a set of strings U has at most $2Q - 2$ states, where Q is the number of nodes of a prefix-tree representing the strings in U. This bound significantly improves over $2\|U\|-1$, the bound given by Blumer et al. (1987), where $\|U\|$ is the sum of the lengths of all strings in U. More generally, we give novel and general bounds for the size of the suffix or factor automaton of an automaton as a function of the size of the original automaton and the maximal length of a suffix shared by the strings it accepts. We also describe in detail a linear-time algorithm for constructing the suffix automaton S or factor automaton F of U in time $O(|S|)$. Our algorithm applies in fact to any input suffix-unique automaton and strictly generalizes the standard on-line construction of a suffix automaton for a single input string. Our algorithm can also be used straightforwardly to generate the *suffix oracle* or *factor oracle* of a set of strings, which has been shown to have various useful properties in string-matching. Our analysis suggests that the use of factor automata of automata can be practical for large-scale applications, a fact that is further supported by the results of our experiments applying factor automata to a music identification task with more than 15,000 songs.

Key words: string-matching, pattern-matching, indexing, inverted text, finite automata, suffix trees, suffix automata, factor automata, music identification.

Email addresses: `mohri@cs.nyu.edu` (Mehryar Mohri), `pedro@google.com` (Pedro Moreno), `eugenew@cs.nyu.edu` (Eugene Weinstein)

1. Introduction

Searching for patterns in massive quantities of natural language texts, biological sequences, and other widely accessible digitized sequences is a problem of central importance in computer science. The problem has a variety of applications and has been extensively studied in the past [1, 2].

This paper considers the problem of constructing a full index, or inverted file, for a set of strings represented by a finite automaton. When the number of strings is large, such as thousands or even millions, the collection of strings can be compactly stored as an automaton, which also enables efficient implementations of search and other algorithms [3, 4]. In fact, in many contexts such as speech recognition or information extraction tasks, the entire set of strings is often directly given as an automaton.

An efficient and compact data structure for representing a full index of a set of strings is a *suffix automaton*, a minimal deterministic automaton representing the set of all suffixes of a set of strings. Since a substring is a prefix of a suffix, a suffix automaton can be used to determine if a string x is a substring in time linear in its length $O(|x|)$, which is optimal. Additionally, as with suffix trees, suffix automata have other interesting properties in string-matching problems which make their use and construction attractive [1, 2]. Another similar data structure for representing a full index of a set of strings is a *factor automaton*, a minimal deterministic automaton representing the set of all factors or substrings of a set of strings. Factor automata offer the same optimal linear-time search property as suffix automata, and are never larger.

The construction and the size of a factor automaton have been specifically analyzed in the case of a single string [5, 6]. These authors demonstrated the remarkable result that the size of the factor automaton of a string x is linear, and that, more precisely, for strings x of length more than three, it has at most $2|x| - 2$ states and $3|x| - 4$ transitions. They also gave on-line linear-time algorithms for constructing a factor automaton from x. Similar results were given for suffix automata, the minimal deterministic automata accepting exactly the set of suffixes of a string.

The construction and the size of the factor automata of a finite set of strings $U = \{x_1, \ldots, x_m\}$ has also been previously studied [7]. These authors showed that an automaton accepting all factors of U can be constructed that has at most $2\|U\| - 1$ states and $3\|U\| - 3$ transitions, where $\|U\|$ is the sum of the lengths of all strings in U, that is $\|U\| = \sum_{i=1}^{m} |x_i|$.

This paper proves a significantly better bound on the size of the suffix automaton or factor automaton of a set of strings. It shows that the suffix automaton or factor automaton of a set of strings U has at most $2Q - 2$ states, where Q is the number of nodes of a prefix-tree representation of the strings in U. The number of nodes Q can be dramatically smaller than $\|U\|$, the sum of the lengths of all strings. Thus, our space bound clearly improves on previous work [7]. More generally, we give novel bounds for the size of the suffix automaton or factor automaton of an acyclic finite automaton as a function of the size of the original automaton and the maximal length of a suffix shared by the strings accepted by the original automaton. This result can be compared to that of Inenaga et al. for compact directed acyclic word graphs whose complexity, $O(|\Sigma|Q)$, depends on the size of the alphabet [8].

Using our space bound analysis, we also give a simple algorithm for constructing the suffix automaton S or factor automaton F of U in time $O(|S|)$ from a prefix tree representing U. Our algorithm applies in fact to any input *suffix-unique automaton* and strictly generalizes the standard on-line construction of a suffix automaton for a single input string.

The original motivation for this work was the design of a large-scale music identification system [4, 9], where we represented our song database by a compact finite automaton, as we shall briefly describe later in this paper. To facilitate an efficient search of song snippets, we constructed the minimal deterministic factor automaton of the song automaton. Empirically, the size of the factor automaton was not prohibitive. But, to ensure the scalability of our approach to a larger set of songs, e.g., several million songs, we wished to derive a bound on the size of the factor automata of automata. One characteristic of the strings considered in this application as in many others is that the original strings do not share long suffixes. This motivated our analysis of the size of the factor automata with respect to the length of the common suffixes in the original automaton.

The remainder of the paper is organized as follows. Section 2 introduces the string and automata definitions and terminology used throughout the paper. In Section 3, we describe a novel analysis of factor automata and present new bounds on the size of the suffix automaton and factor automaton of an automaton. Section 4 gives a detailed description of a linear-time algorithm for the construction of the suffix automaton and factor automaton of a finite set of strings, or of any suffix-unique automaton, including a pseudocode of the algorithm. Our algorithm can also be used straightforwardly to generate the *suffix oracle* or *factor oracle* of a set of strings, which has been shown to have various useful properties [10]. Section 5 briefly describes the use of factor automata in music identification and reports several empirical results related to their size.

2. Factors of a Finite Automaton

This section reviews some key properties of factors of a fixed finite automaton, generalizing similar observations made by Blumer et al. for a single string [7].

We denote by Σ a finite alphabet. The length of a string $x \in \Sigma^*$ over that alphabet is denoted by $|x|$. A *factor*, or *substring*, of a string $x \in \Sigma^*$ is a sequence of symbols appearing consecutively in x. Thus, y is a factor of x iff there exist $u, v \in \Sigma^*$ such that $x = uyv$. A *suffix* of a string $x \in \Sigma^*$ is a factor that appears at the end of x. Put otherwise, y is a suffix of x iff there exists $u \in \Sigma^*$ such that $x = uy$. Analogously, y is a *prefix* of x iff there exists $u \in \Sigma^*$ such that $x = yu$. More generally, a factor, suffix, or prefix of a set of strings U or an automaton A, is a factor, suffix, or prefix of a string in U or a string

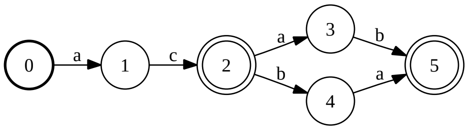


Figure 1: Finite automaton A accepting the strings $ac, acab, acba$.

accepted by A, respectively. The symbol ϵ represents the empty string. For any string $x \in \Sigma^*$, ϵ is always a prefix, suffix, and factor of x.

In some applications such as music identification the strings considered may be long, e.g., sequences of music sounds, but with relatively short common suffixes. This motivates the following definition.

Definition 1. *Let k be a non-negative integer. We will say that a finite automaton A is* k-suffix-unique *if no two strings accepted by A share a suffix of length k. A is said to be* suffix-unique *when it is k-suffix-unique with $k = 1$.*

Figure 1 gives an example of a simple automaton A accepting three strings ending in distinct symbols. Note that A is suffix-unique.

The main results of this paper hold for suffix-unique automata, but we also present some results for the general case of arbitrary acyclic automata. We denote by $F(A)$ the minimal deterministic automaton accepting the set of factors of a finite automaton A, that is the set of factors of the strings accepted by A. Similarly, we denote by $S(A)$ the minimal deterministic automaton accepting the set of suffixes of an automaton A.

Definition 2. *Let A be a finite automaton. For any string $x \in \Sigma^*$, we define end-set(x) as the set of states of A reached by the paths in A that begin with x. We say that two strings x and y in Σ^* are equivalent and denote this by $x \equiv y$, when end-set$(x) =$ end-set(y). This defines a right-invariant equivalence relation on Σ^*. We denote by $[x]$ the equivalence class of $x \in \Sigma^*$.*

Lemma 1. *Assume that A is suffix-unique. Then, a non-suffix factor x of the automaton A is the longest member of $[x]$ iff it is either a prefix of A, or both ax and bx are factors of A for distinct $a, b \in \Sigma$.*

Proof. Let x be a non-suffix factor of A. Clearly, if x is not a prefix, then there must be distinct a and b such that ax and bx are factors of A, otherwise $[x]$ would admit a longer member. Conversely, assume that ax and bx are both factors of A with $a \neq b$. Let y be the longest member of $[x]$. Let q be a state in *end-set*$(x) =$ *end-set*(y). Since x is not a suffix, q is not a final state, and there exists a non-empty string z labeling a path from q to a final state. Since A is suffix-unique, both xz and yz are suffixes of the same string. Since y is the longest member of $[x]$, x must be a suffix of y. Since ax and bx are both factors of A with $a \neq b$, we must have $y = x$. Finally, if x is a prefix, then clearly it is the longest member of $[x]$. $\square$

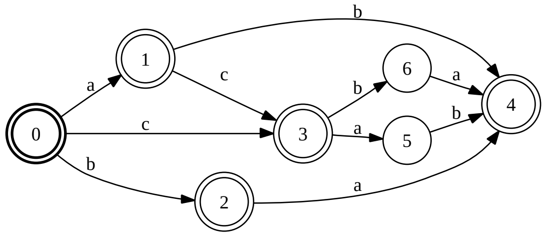


Figure 2: Suffix automaton $S(A)$ of the automaton A of Figure 1.

Proposition 1. *Assume that A is suffix-unique. Let $S_A = (Q_S, I_S, F_S, E_S)$ be the deterministic automaton whose states are the equivalence classes $Q_S = \{[x] \neq \emptyset : x \in \Sigma^*\}$, its initial state $I_S = \{[\epsilon]\}$, its final states $F_S = \{[x] : \text{end-set}(x) \cap F_A \neq \emptyset\}$ where F_A is the set of final states of A, and its transition set $E_S = \{([x], a, [xa]) : [x], [xa] \in Q_S\}$. Then, S_A is the minimal deterministic suffix automaton of A: $S_A = S(A)$.*

Proof. By construction, S_A is deterministic and accepts exactly the set of suffixes of A. Let $[x]$ and $[y]$ be two equivalent states of S_A. Then, for all $z \in \Sigma^*$, $[xz] \in F_A$ iff $[yz] \in F_A$, that is z is a suffix of A iff yz is a suffix of A. Since A is suffix-unique, this implies that either x is a suffix of y or vice versa, and thus that $[x] = [y]$. Thus, S_A is minimal. □

In what follows, we will be interested in the case where the automaton A is acyclic. We denote by $|A|_Q$ the number of states of A, by $|A|_E$ the number of transitions of A, and by $|A|$ the *size of* A defined as the sum of the number of states and transitions of A.

3. Space Bounds for Factor Automata

The objective of this section is to derive new bounds on the size of $S(A)$ and $F(A)$ in the case of interest for our applications where A is an acyclic automaton, typically deterministic and minimal, representing a set of strings.

When A represents a single string, there are standard algorithms for constructing $S(A)$ and $F(A)$ from A in linear time [5, 6]. In the general case, $S(A)$ can be constructed from A as follows: add an ϵ-transition from the initial state of A to each state of A, then apply an ϵ-removal algorithm, followed by determinization and minimization. $F(A)$ can be obtained similarly by further making all states final before applying ϵ-removal, determinization, and minimization. It can also be obtained from $S(A)$ by making all states of $S(A)$ final and applying minimization. For example, if A is the simple automaton of Figure 1, then Figure 2 is its suffix automaton $S(A)$.

When A represents a single string x, the size of the automata $S(A)$ and $F(A)$ can be proved to be linear in $|x|$. More precisely, the following bounds hold for $|S(A)|$ and $|F(A)|$ [6, 5]:

$$\begin{array}{rclrcl} |S(A)|_Q & \leq & 2|x|-1 & |S(A)|_E & \leq & 3|x|-4 \\ |F(A)|_Q & \leq & 2|x|-2 & |F(A)|_E & \leq & 3|x|-4. \end{array} \tag{1}$$

These bounds are tight for strings of length more than three. [7] gave similar results for the case of a set of strings U by showing that the size of the factor automaton $F(U)$ representing this set is bounded as follows

$$|F(U)|_Q \leq 2\|U\|-1 \quad |F(U)|_E \leq 3\|U\|_E - 3, \tag{2}$$

where $\|U\|$ denotes the sum of the lengths of all strings in U.

In general, the size of an acyclic automaton A representing a finite set of strings U can be substantially smaller than $\|U\|$. In fact, $|A|$ can be exponentially smaller than $\|U\|$. Thus, we are interested in bounding the size of $S(A)$ or $F(A)$ in terms of the size of A, rather than the sum of the lengths of all strings accepted by A.

For any state q of $S(A)$, we denote by $\mathrm{suff}(q)$ the set of strings labeling the paths from q to a final state. We also denote by $N(q)$ the set of states in A from which a path labeled with a non-empty string in $\mathrm{suff}(q)$ reaches a final state.

Lemma 2. *Let A be a suffix-unique automaton and let q and q' be two states of $S(A)$ such that $N(q) \cap N(q') \neq \emptyset$, then*

$$\big(\mathrm{suff}(q) \subseteq \mathrm{suff}(q') \text{ and } N(q) \subseteq N(q')\big) \text{ or } \big(\mathrm{suff}(q') \subseteq \mathrm{suff}(q) \text{ and } N(q') \subseteq N(q)\big). \tag{3}$$

Proof. Since $S(A)$ is a minimal automaton, its states are accessible from the initial state. Let u be the label of a path from the initial I of $S(A)$ to q and similarly u' the label of a path from I to q'.

By assumption, there exists $p \in N(q) \cap N(q')$. Thus, there exist non-empty strings $v \in \mathrm{suff}(q)$ and $v' \in \mathrm{suff}(q')$ such that both v and v' label paths from p to a final state.

By definition of u and u', both uv and $u'v'$ are suffixes of A. Since A is suffix-unique and v is non-empty, there exists a unique string accepted by A and ending with v. There exists also a unique string accepted by A and ending with uv. Thus, these two strings must coincide.

This implies that any string accepted by A and admitting v as suffix also admits uv as suffix. In particular, the label of any path from an initial state to p must admit u as suffix. Reasoning in the same way for v' let us conclude that the label of any path from an initial state to p must also admit u' as suffix. Thus, u and u' are suffixes of the same string. Thus, u is a suffix of u' or vice-versa. Figure 3 illustrates this situation.

Assume without loss of generality that u is a suffix of u'. Then, for any string w, if $u'w$ is a suffix of A so is uw. Thus, $\mathrm{suff}(q') \subseteq \mathrm{suff}(q)$, which implies $N(q') \subseteq N(q)$. When u' is a suffix of u, we obtain similarly the other case of the statement of the lemma. □

Note that Lemma 2 holds even when A is a non-deterministic automaton.

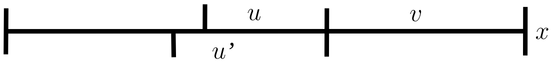


Figure 3: Illustration of the situation described in Lemma 2. uv and $u'v$ are suffixes of the same string x. Thus, u and u' are also suffixes of the same string. Thus, u is a suffix of u' or vice-versa.

Lemma 3. *Let A be a suffix-unique deterministic automaton and let q and q' be two distinct states of $S(A)$ such that $N(q) = N(q')$, then either q is a final state and q' is not, or q' is a final state and q is not.*

Proof. Assume that $N(q) = N(q')$. By Lemma 2, this implies $\mathrm{suff}(q) = \mathrm{suff}(q')$. Thus, the same non-empty strings label the paths from q to a final state or the paths from q' to a final state. Since $S(A)$ is a minimal automaton, the distinct states q and q' are not equivalent. Thus, one must admit an empty path to a final state and not the other. □

The following proposition extends the results of [7] which hold for a set of strings, to the case where A is an automaton.

Proposition 2. *Let A be a suffix-unique deterministic and minimal automaton accepting strings of length more than three. Then, the number of states of the suffix automaton of A is bounded as follows*

$$|S(A)|_Q \leq 2|A|_Q - 3. \tag{4}$$

Proof. If the strings accepted by A are all of the form a^n, $S(A)$ can be derived from A simply by making all its states final and the bound is trivially achieved. In the remainder of the proof, we can thus assume that not all strings accepted by A are of this form.

Let F be the unique final state of $S(A)$ with no outgoing transitions. Lemmas 2-3 help define a tree T associated to all states of $S(A)$ other than F by using the ordering:

$$N(q) \sqsubseteq N(q') \text{ iff} \begin{cases} N(q) \subset N(q') \text{ or} \\ N(q) = N(q') \text{ and } q' \text{ final}, q \text{ non-final.} \end{cases} \tag{5}$$

We will identify each node of T with its corresponding state in $S(A)$. By Proposition 1, each state q of $S(A)$ can also be identified with an equivalence class $[x]$. Let q be a state of $S(A)$ distinct from F, and let $[x]$ be its corresponding equivalence class. Observe that since A is suffix-unique, *end-set*(x) coincides with $N(q)$.

We will show that the number of nodes of T is at most $2|A|_Q - 4$, which will yield the desired bound on the number of states of $S(A)$. To do so, we bound separately the number of non-branching and branching nodes of T.

Let q be a node of T and let $[x]$ be the corresponding equivalence class, with x its longest member. The children of q are the nodes corresponding to the equivalence classes $[ax]$ where $a \in \Sigma$ and ax is a factor of A.

By Lemma 1, if x is a non-suffix and non-prefix factor, then there exist factors ax and bx with $a \neq b$. Thus, q admits at least two children corresponding to $[ax]$ and $[bx]$ and is thus a branching node. Thus non-branching nodes can only be either nodes q where x is a prefix, or those where x is a suffix, that is when q is a final state of $S(A)$.

Since the strings accepted by A are not all of the form a^n for some $a \in \Sigma$, the empty prefix ϵ occurs at least in two distinct left contexts a and b with $a \neq b$. Thus, the prefix ϵ, which corresponds to the root of T, is necessarily branching. Also, let f be the unique final state of A with no outgoing transitions. The equivalence class of the longest factor ending in f, that is the longest string accepted by A corresponds to the state F in $S(A)$ which is not included in the tree T. Thus, there are at most $|A|_Q - 2$ non-branching prefixes.

There can be at most one non-branching node for each string accepted by A. Let N_{str} denote the number of strings accepted by A, then, the number of non-branching nodes N_{nb} of T is at most $N_{nb} \leq |A|_Q - 2 + N_{str}$.

To bound the number of branching nodes N_b of T, observe that since A is suffix-unique, each string accepted by A must end with a distinct symbol a_i, $i = 1, \ldots, N_{str}$. Each a_i represents a distinct left context for the empty factor ϵ, thus the root node $[\epsilon]$ admits all $[a_i]$s, $i = 1, \ldots, N_{str}$, as children. Let T_{a_i} represent the sub-tree rooted at $[a_i]$ and let n_{a_i} represent the number of leaves of T_{a_i}. Let a_j, $j = N_{str} + 1, \ldots, N_{str} + k$ denote the other children of the root and let T_{a_j} denote each of the corresponding sub-tree. A tree with n_{a_i} leaves has less than n_{a_i} branching nodes. Thus, the number of branching nodes of T_{a_i} is at most $n_{a_i} - 1$. The total number of leaves of T is at most the number of disjoint subsets of Q excluding the initial state and f.

Note however that when the root node $[\epsilon]$ admits only $[a_i]$s, $i = 1, \ldots, N_{str}$, as children, that is when $k = 0$, then there is at least one a_i, say a_1, that is also a prefix of A since any other symbol would have been the root node's child. The node a_1 will then have also a child since it corresponds to a suffix or final state of $S(A)$. Thus, a_1 cannot be a leaf in that case. Thus, there are at most as many as $\sum_{i=1}^{N_{str}+k} n_{a_i} \leq |A|_Q - 2 - 1_{k=0}$ leaves and the total number of branching nodes of T, including the root is at most $N_b \leq \sum_{i=1}^{N_{str}+k}(n_{a_i} - 1) + 1 \leq |A|_Q - 2 - 1_{k=0} - (N_{str} + k) + 1 \leq |A|_Q - 2 - N_{str}$. The total number of nodes of the tree T is thus at most $N_{nb} + N_b \leq 2|A|_Q - 4$. □

In the specific case where A represents a single string x, the bound of Proposition 2 matches that of [6] or [5] since $|A|_Q = |x| + 1$. The bound of Proposition 2 is tight for strings of length more than three and thus is also tight for automata accepting strings of length more than three. Note that the automaton of Figure 1 is suffix-unique, deterministic, and minimal and has $|A|_Q = 6$ states. The number of states of the minimal suffix automaton of A is $|S(A)|_Q = 7 < 2|A|_Q - 3$.

Corollary 1. *Let A be a suffix-unique deterministic and minimal automaton accepting strings of length more than three. Then, the number of states of the factor automaton of A is bounded as follows*

$$|F(A)|_Q \leq 2|A|_Q - 3. \tag{6}$$

Proof. As mentioned earlier, a factor automaton $F(A)$ can be obtained from a suffix automaton $S(A)$ by making all states final and applying minimization. Thus, $|F(A)| \leq |S(A)|$. The result follows Proposition 2. □

Blumer et al. (1987) showed that an automaton accepting all factors of a set of strings U has at most $2\|U\|-1$ states, where $\|U\|$ is the sum of the lengths of all strings in U [7]. The following gives a significantly better bound on the size of the factor automaton of a set of strings U as a function of the number of nodes of a prefix-tree representing U, which is typically substantially smaller than $\|U\|$.

Corollary 2. *Let $U = \{x_1, \ldots, x_m\}$ be a set of strings of length more than three and let A be a prefix-tree representing U. Then, the number of states of the factor automaton $F(U)$ and that of the suffix automaton $S(U)$ of the strings of U are bounded as follows*

$$|F(U)|_Q \leq 2|A|_Q - 2 \quad |S(U)|_Q \leq 2|A|_Q - 2. \tag{7}$$

Proof. Let B be a prefix-tree representing the set $U' = \{x_1\$_1, \ldots, x_m\$_m\}$, obtained by appending to each string of U a new symbol $\$_i$, $i = 1, \ldots, m$, to make their suffixes distinct and let B' be the automaton obtained by minimization of B. By construction, B has m more states than A, but since all final states of B are equivalent and merged after minimization, B' has at most one more state than A.

By construction, B' is a suffix-unique automaton and by Proposition 2, $|S(B')|_Q \leq 2|B'|_Q - 3$. Removing from $S(B')$ the transitions labeled with the extra symbols $\$_i$ and connecting the resulting automaton yields the minimal suffix automaton $S(U)$. In $S(B')$, there must be a final state reachable by the transitions labeled with $\$_i$ and only such transitions, which becomes non-accessible after removal of the extra symbols. Thus, $S(U)$ has at least one state less than $S(B')$, which gives:

$$|S(U)|_Q \leq |S(B')|_Q - 1 \leq 2|B'|_Q - 4 = 2|A|_Q - 2. \tag{8}$$

A similar bound holds for the factor automaton $F(U)$ following the argument given in the proof of Corollary 1. □

When A is k-suffix-unique with a relatively small k as in our applications of interest, the following proposition provides a convenient bound on the size of the suffix automaton.

Proposition 3. *Let A be a k-suffix-unique deterministic automaton accepting strings of length more than three and let n be the number of strings accepted by A. Then, the following bound holds for the number of states of the suffix automaton of A:*

$$|S(A)|_Q \leq 2|A_k|_Q + 2kn - 3, \tag{9}$$

where A_k is the part of the automaton of A obtained by removing the states and transitions of all suffixes of length k.

Proof. Let A be a k-suffix-unique deterministic automaton accepting strings of length more than three and let the alphabet Σ be augmented with n temporary symbols $\$_1, \ldots, \$_n$. By marking each string accepted by A with a distinct symbol $\$_i$, we can turn A into a suffix-unique deterministic automaton A'.

To do that, we first unfold all k-length suffixes of A. In the worst case, all these (distinct) suffixes were sharing the same $(k-1)$-length suffix. Unfolding can thus increase the number of states of A by as many as $kn - n$ states in the worst case. Marking the end of each suffix with a distinct \$-sign further increases the size by n. The resulting automaton A' is deterministic and $|A'|_Q \leq |A_k|_Q + kn$. By Proposition 2, the size of the suffix automaton of A' is bounded as follows: $|S(A')| \leq 2|A'| - 3$. Since transitions labeled with a \$-sign can only appear at the end of successful paths in $S(A')$, we can remove these transitions and make their origin state final, and minimize the resulting automaton to derive a deterministic automaton A'' accepting the set of suffixes of A. The statement of the proposition follows the fact that $|A''| \leq |S(A')|$. □

Since the size of $F(A)$ is always less than or equal to that of $S(A)$, we obtain directly the following result.

Corollary 3. *Let A be a k-suffix-unique automaton accepting strings of length more than three. Then, the following bound holds for the factor automaton of A:*

$$|F(A)|_Q \leq 2|A_k|_Q + 2kn - 3. \tag{10}$$

The bound given by the corollary is not tight for relatively small values of k in the sense that in practice, the size of the factor automaton does not depend on kn, the sum of the lengths of suffixes of length k, but rather on the number of states of A used for their representation, which for a minimal automaton can be substantially less. However, for large k, e.g., when all strings are of the same length and k is as long as the length of the strings accepted by A, our bound coincides with that of [7].

4. Suffix Automaton Construction Algorithm

This section describes a linear-time algorithm for the construction of the suffix automaton $S(A)$ of an input suffix-unique automaton A, or similarly the factor automaton $F(A)$ of A. Since a factor automaton can be obtained from $S(A)$ by making all states of $S(A)$ final and applying a linear-time acyclic minimization algorithm [11], it suffices to describe a linear-time algorithm for the construction of $S(A)$. It is possible however to give a similar direct linear-time algorithm for the construction of F_A.

Figures 4-6 give the pseudocode for the algorithm for constructing the suffix automaton $S(A) = (Q_S, I, F_S, \delta_S)$ of an automaton $A = (Q_A, I, F_A, \delta_A)$, where A is suffix-unique and where $\delta_S : Q_S \times \Sigma \mapsto Q_S$ denotes the partial transition function of $S(A)$ and likewise $\delta_A : Q_A \times \Sigma \mapsto Q_A$ that of A. As in the previous section, f denotes the final state of A with no outgoing transitions. Additionally,

```
Create-Suffix-Automaton($A, f$)
 1  $S \leftarrow Q_S \leftarrow \{I\}$  $\triangleright$ initial state
 2  $s[I] \leftarrow$ undefined;  $l[I] \leftarrow 0$
 3  while $S \neq \emptyset$ do
 4      $p \leftarrow$ Head($S$)
 5      for each $a$ such that $\delta_A(p, a) \neq$ undefined do
 6          if $\delta_A(p, a) \neq f$ then
 7              $Q_S \leftarrow Q_S \cup \{p\}$
 8              $l[q] \leftarrow l[p] + 1$
 9              Suffix-Next($p, a, q$)
10              Enqueue($S, q$)
11  $Q_S \leftarrow Q_S \cup \{f\}$
12  for each state $p \in Q_A$ and $a \in \Sigma$ such that $\delta_A(p, a) = f$ do
13      Suffix-Next($p, a, f$)
14      Suffix-Final($f$)
15  for each $p \in F_A$ do
16      Suffix-Final($q$)
17  return $S(A) = (Q_S, I, F_S, \delta_S)$
```

Figure 4: Algorithm for the construction of the suffix automaton of a suffix-unique automaton A.

we use the term *suffix pointer* to refer to the destination state of the suffix link transition.

The algorithm is a generalization to an input suffix-unique automaton of the standard construction for an input string. Our presentation is similar to that of [6]. The algorithm maintains two values $s[q]$ and $l[q]$ for each state q of S_q. $s[q]$ denotes the suffix pointer or failure state of q. $l[q]$ denotes the length of the longest path from the initial state to q in $S(A)$. l is used to determine the so-called *solid edges* or *transitions* in the construction of the suffix automaton. A transition (p, a, q) is solid if $l[p] + 1 = l[q]$, that is it is on a longest path from the initial state to q, otherwise, it is a short-cut transition.

S is a queue storing the set of states to be examined. The particular queue discipline of S does not affect the correctness of the algorithm, but we can assume it to be a FIFO order, which corresponds to a breadth-first search and admits of course a linear-time implementation. In each iteration of the loop of lines 3-10 in Figure 4, a new state p is extracted from S. The processing of the transitions (p, a, f) with destination state f is delayed to a later stage (lines 12-14). This is because of the particular properties of f which, as discussed in the previous section, can be viewed as the child of different nodes of the tree T, and thus can admit different suffix links. Other transitions (p, a, q) are processed one at a time by creating, if necessary, the destination state q and adding it to Q_S, defining $l[q]$ and calling Suffix-Next(p, a, q).

The subroutine Suffix-Next processes each transition (p, a, q) in a way

```
Suffix-Next($p, a, q$)
$l[q] \leftarrow \max(l[p]+1, l[q])$
while $p \neq I$ and $\delta_S(p,a) =$ undefined do
    $\delta_S(p,a) \leftarrow q$
    $p \leftarrow s[p]$
if $\delta_S(p,a) =$ undefined then
    $\delta_S(I,a) \leftarrow q$
    $s[q] \leftarrow I$
elseif $l[p]+1 = l[\delta_S(p,a)]$ and $\delta_S(p,a) \neq q$ then
    $s[q] \leftarrow \delta_S(p,a)$
else $r \leftarrow q$
    if $\delta_S(p,a) \neq q$ then
        $r \leftarrow$ copy of $\delta_S(p,a)$  $\triangleright$ new state with same transitions
        $Q_S \leftarrow Q_S \cup \{r\}$
        $s[q] \leftarrow r$
    $s[r] \leftarrow s[\delta_S(p,a)]$
    $s[\delta_S(p,a)] \leftarrow r$
    $l[r] \leftarrow l[p]+1$
    while $p \neq$ undefined and $l[\delta_S(p,a)] \geq l[r]$ do
        $\delta_S(p,a) \leftarrow r$
        $p \leftarrow s[p]$
```

Figure 5: Subroutine of Create-Suffix-Automaton processing a transition of A from state p to state q labeled with a.

```
Suffix-Final($p$)
if $p \in F_S$ then
    $p \leftarrow s[p]$
while $p \neq$ undefined and $p \notin F_S$ do
    $F_S \leftarrow F_S \cup \{p\}$
    $p \leftarrow s[p]$
```

Figure 6: Subroutine of Create-Suffix-Automaton making all states on the suffix chain of p final.

that is very similar to the standard string suffix automaton construction. The loop of lines 2-4 inspects the iterated suffix pointers of p that do not admit an outgoing transition labeled with a. It further creates such transitions reaching q from all the iterated suffix pointers until the initial state or a state p' already admitting such a transition is reached. In the former case, the suffix pointer of q is set to be the initial state I and the transition (I, a, q) is created.

In the latter case, if the existing transition (p', a, q') is solid and $q' = q$, then the suffix pointer of q is simply set to be q' (line 9). Otherwise, if $q' \neq q$, a copy of the state q', r, with the same outgoing transitions is created (line 12) and the suffix pointer of q is set to be r. The suffix pointer of r is set to be $s[q']$ (line 15), that of q' is set to r (16), and $l[r]$ defined as $l[p] + 1$ (17). The transitions labeled with a leaving the iterated suffix pointers of p are inspected and redirected to r so long as they are non-solid transitions (lines 18-20).

The subroutine Suffix-Final sets the finality and the final weight of states in $S(A)$. For any state p that is final in A, p and all the states found by following the chain of suffix pointers starting at p are made final in $S(A)$ in the loop of lines 3-5.

We have implemented and tested the suffix-construction algorithm just described. Figure 7 illustrates the application of the algorithm to a particular suffix-unique automaton. All intermediate stages of the construction of $S(A)$ are indicated, including the information about the suffix pointers $s[q]$ for each state q.

In the construction of the so-called *suffix oracle* [10] no new state is created with respect to the input. The suffix oracle of A can thus be constructed in a similar way simply by replacing line 12 in Figure 5 by: $r \leftarrow \delta_S(p, a)$ and removing lines 15-17. This algorithm thus straightforwardly extends the construction of the suffix oracle to the case of suffix-unique input automata.

For the complexity result that follows, we will assume an efficient representation of the transition function such that an outgoing transition with a specific label can be found in constant time $O(1)$ at any state. Other authors are sometimes assuming instead an adjacency list representation and a binary search to find a transition at a given state, which costs $O(\min\{\log |\Sigma|, e_{\max}\})$ where $e_{\max}$ is the maximum outdegree [6, 2]. If one adopts that assumption, the complexity results we report as well as those of Blumer et al. [5, 7] should be multiplied with the factor $\min\{\log |\Sigma|, e_{\max}\}$.

We refer to a redirection of a transition that has already previously been redirected as a *multiple redirection*.

Proposition 4. *Let A be a minimal deterministic suffix-unique automaton. Then, the runtime complexity of algorithm* Create-Suffix-Automaton(A, f) *is $O(|S(A)|)$.*

Proof. We give a brief sketch of the proof. Suffix-Next is called at most once per transition, so the total number of calls of Suffix-Next is $O(|A|)$. Fix a transition (p, a, q) of A with $q \neq f$. The cost of the execution of the steps 1-20 by Suffix-next is proportional to the total number of iterated suffix link

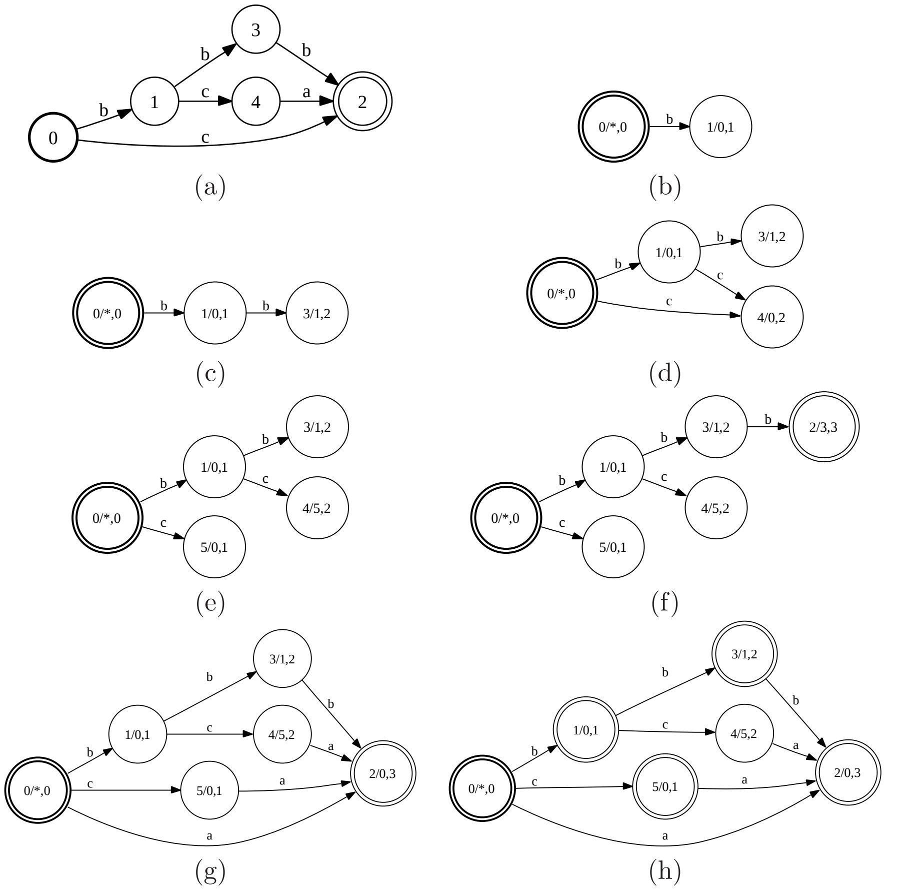


Figure 7: Construction of the suffix automaton using CREATE-SUFFIX-AUTOMATON. (a) Original automaton A. (b)-(h) Intermediate stages of the construction of $S(A)$. For each state $(n/s, l)$, n is the state number, s is the suffix pointer of n, and l is $l[n]$.

traversals in the loop of lines 2-4 and lines 18-20. Each iteration of lines 2-4 results in a new transition being created in $S(A)$, so the total number of loop iterations over all calls of SUFFIX-NEXT is $O(|S(A)|)$.

The analysis of the total number of redirection iterations of the while loop of lines 18-20 relies on an extension of the analysis for the single-string case [12, 7]. The linear bound on the total number of redirections in the single-string case is applicable to our automaton case for a linear chain of states in A. Given the required combination of substrings in A to cause a redirection, it can be shown that the total number of multiple redirections is $O(|A|)$. Thus, the total complexity is $O(|S(A)|)$. □

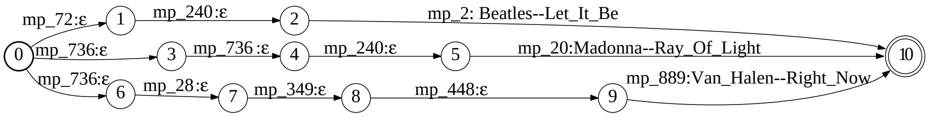


Figure 8: Finite-state transducer T_0 mapping each song to its identifier.

5. Factor Automata for Music Identification

We have verified the above insights into factor automata in the context of a music identification system [4, 9]. Music identification is the task of matching an audio stream to a particular song. In our system, we learn an inventory of music phone units similar to phonemes in speech and a unique sequence of music phones characterizing each song. We view the music phone set as our alphabet and the music phone sequences as a set of strings, transforming the task into a factor recognition problem. Our approach is to construct a compact transducer mapping music phone sequences to corresponding song identifiers.

5.1. Factor Transducer Construction

Let Σ denote the set of music phones and let the set of music phone sequences describing m songs be $U = \{x_1, \ldots, x_m\}, x_i \in \Sigma^*$ for $i \in \{1, \ldots, m\}$. In our experiments, $m = 15{,}455$, $|\Sigma| = 1{,}024$ and the average length of a transcription x_i is more than 1,700. Thus, in the worst case, there can be as many as $15{,}455 \times 1{,}700^2 \approx 45 \times 10^9$ factors. The size of a naive prefix-tree-based representation would thus be prohibitive. Hence, we represent the set of factors with a much more compact factor automaton. We construct a deterministic and minimal automaton representing the sequences in U and subsequently a deterministic and minimal finite-state transducer mapping each song to its identifier using transducer determinization and minimization algorithms [13, 14].

Let T_0 be the unoptimized transducer mapping phone sequences to song identifiers. Figure 8 shows T_0 when U is reduced to three short songs. Let A be the acceptor obtained by omitting the output labels of T_0. The compact factor automaton $F(A)$ (Figure 9(a)) is constructed as described in Section 3: by creating ϵ-transitions from the initial state of A to all other states, making all states final, and applying ϵ-removal, determinization, and minimization. Note that $F(A)$ does not output the song identifier associated with each factor.

For the purposes of the following description, we briefly review some properties of weighted automata. A weighted automaton is defined over a semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$, which specifies the weight set used and the algebraic operations for combining weights along a path and between paths. The tropical semiring $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +, +\infty, 0)$ is one used extensively in fields such as speech and text processing. In the tropical semiring, the total weight assigned by the automaton to a string s is the minimum-weight path in the automaton with the label s, where the total path along a given path is found by adding the weights of the transitions composing the path.

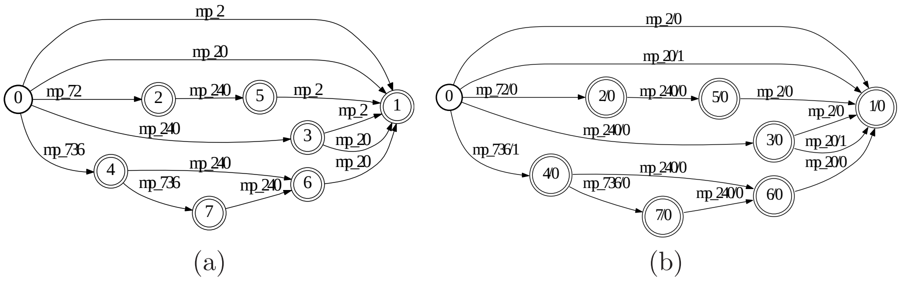


Figure 9: (a) Deterministic and minimal unweighted factor acceptor $F(A)$ for two songs. (b) Deterministic and minimal weighted factor acceptor $F_w(A)$ for two songs.

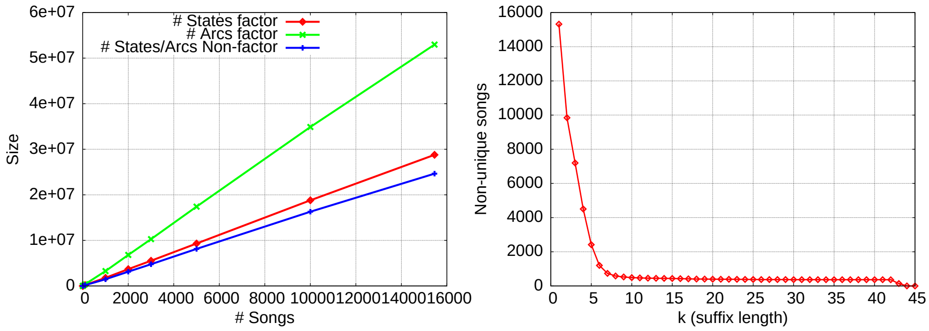
To construct a factor automaton that preserves the song identifiers, we create a compact weighted acceptor over the tropical semiring accepting the factors of U that associates the total weight s_x to each factor x. A crucial advantage of this representation is the use of weighted determinization and minimization [13] during which the song identifier is treated as a weight possibly distributed along a path. These operations preserve the property that the total weight along the path labeled with x is s_x. Let $F_w(A)$ be constructed analogously to $F(A)$, but with each added ϵ-transition weighted with the corresponding song identifier. The weighted acceptor $F_w(A)$, after determinization and minimization over the tropical semiring, is transformed into a song recognition transducer T by treating each output weight integer as an output symbol. Given a music phone sequence as input, the associated song identifier is obtained by summing the outputs yielded by T.

5.2. Automata Size

Figure 9(b) shows the weighted automaton $F_w(A)$ corresponding to the unweighted automaton $F(A)$ of Figure 9(a). Note that $F_w(A)$ is no larger than $F(A)$. Remarkably, even in the case of 15,455 songs, the total number of transitions of $F_w(A)$ was 53.0M, only about 0.004% more than $F(A)$. We also have $|F(A)|_E \approx 2.1|A|_E$. As is illustrated in Figure 10(a), this multiplicative relationship is maintained as the song set size is varied between 1 and 15,455. Furthermore, for the case of 15,455 songs, U is 45-suffix-unique. Figure 10(b) demonstrates that the number of suffix "collisions" drops rapidly as the suffix size is increased. We also have $|F_w(A)|_Q \approx 28.8M \approx 1.2|A|_Q$, meaning the bound of Corollary 3 is verified in this empirical context.

6. Conclusion

We presented a novel analysis of the size of the suffix automaton and factor automaton of a set of strings represented by an automaton in terms of the size of the original automaton. Our analysis shows that suffix automata and factor automata can be practical for constructing an index of a large number of strings.



(a) (b)

Figure 10: (a) Comparison of automaton sizes for different numbers of songs. "#States/Arcs Non-factor" is the size of the automaton A accepting the entire song transcriptions. "# States factor" and "# Arcs factor" is the number of states and transitions in the weighted factor acceptor $F_w(A)$, respectively. (b) Number of strings in U for which the suffix of length k is also a suffix of another string in U.

Additionally, our application to a large-scale music identification task further demonstrates this fact. Factor automata of automata are likely to form a useful and compact index for very large-scale tasks. We further gave a linear-time algorithm for constructing the suffix automaton or factor automaton of a set of strings in time linear in the size of a prefix tree representing them. Our algorithm applies to any input suffix-unique automaton and strictly generalizes the standard on-line construction of a suffix automaton for a single input string. Our algorithm and analysis raise the natural question of an efficient construction of the suffix automaton of an arbitrary input automaton.

Acknowledgments

We thank Cyril Allauzen for several discussions about the material presented. The research of Mehryar Mohri and Eugene Weinstein was partially supported by the New York State Office of Science Technology and Academic Research (NYSTAR). This project was also sponsored in part by the Department of the Army Award Number W81XWH-04-1-0307. The U.S. Army Medical Research Acquisition Activity, 820 Chandler Street, Fort Detrick MD 21702-5014 is the awarding and administering acquisition office. The content of this material does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

[1] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, Cambridge, UK., 1997.

[2] M. Crochemore, W. Rytter, Jewels of Stringology, World Scientific, 2002.

[3] C. Allauzen, M. Mohri, M. Saraclar, General Indexation of Weighted Automata – Application to Spoken Utterance Retrieval, in: Proceedings of the Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval (HLT/NAACL), Boston, Massachusetts, 2004, pp. 33–40.

[4] E. Weinstein, P. Moreno, Music Identification with Weighted Finite-State Transducers, in: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Honolulu, Hawaii, 2007, pp. 689–692.

[5] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, J. I. Seiferas, The smallest automaton recognizing the subwords of a text, Theoretical Computer Science 40 (1985) 31–55.

[6] M. Crochemore, Transducers and repetitions, Theoretical Computer Science 45 (1986) 63–86.

[7] A. Blumer, J. Blumer, D. Haussler, R. M. McConnell, A. Ehrenfeucht, Complete inverted files for efficient text retrieval and analysis, Journal of the ACM 34 (1987) 578–589.

[8] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, G. Pavesi, On-line construction of compact directed acyclic word graphs, Discrete Applied Mathematics 146 (2) (2005) 156–179.

[9] M. Mohri, P. Moreno, E. Weinstein, Robust music identification, detection, and analysis, in: Proceedings of the International Conference on Music Information Retrieval (ISMIR), Vienna, Austria, 2007, pp. 135–139.

[10] C. Allauzen, M. Crochemore, M. Raffinot, Efficient experimental string matching by weak factor recognition, in: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM), Springer-Verlag, London, UK, 2001, pp. 51–72.

[11] D. Revuz, Minimisation of acyclic deterministic automata in linear time, Theoretical Computer Science 92 (1992) 181–189.

[12] J. A. Blumer, Algorithms for the directed acyclic word graph and related structures, Ph.D. thesis, Denver University (1985).

[13] M. Mohri, Finite-state transducers in language and speech processing, Computational Linguistics 23 (2) (1997) 269–311.

[14] M. Mohri, Statistical Natural Language Processing, in: M. Lothaire (Ed.), Applied Combinatorics on Words, Cambridge University Press, 2005.