An Efficient Linear Pseudo-Minimization Algorithm for Aho-Corasick Automata *

Omar AitMous[1], Frédérique Bassino[1], and Cyril Nicaud[2].

[1] LIPN, UMR 7030, Université Paris 13 - CNRS, 93430 Villetaneuse, France.
[2] LIGM, UMR 8049, Université Paris-Est - CNRS, 77454 Marne-la-Vallée, France.
aitmous,bassino@lipn.univ-paris13.fr, nicaud@univ-mlv.fr

Abstract. A classical construction of Aho and Corasick solves the pattern matching problem for a finite set of words X in linear time, where the size of the input X is the sum of the lengths of its elements. It produces an automaton that recognizes A^*X, where A is a finite alphabet, but which is generally not minimal. As an alternative to classical minimization algorithms, which yields a $\mathcal{O}(n \log n)$ solution to the problem, we propose a linear pseudo-minimization algorithm specific to Aho-Corasick automata, which produces an automaton whose size is between the size of the input automaton and the one of its associated minimal automaton. Moreover this algorithm generically computes the minimal automaton: for a large variety of natural distributions the probability that the output is the minimal automaton of A^*X tends to one as the size of X tends to infinity.

1 Introduction

Pattern matching issues arise naturally in various fields of computer science, motivating an ever renewed search for efficient algorithms that answer the following question: given a set of words X, the set of patterns, what is the best way to compute the number of occurrences of words of X in a large text?

The literature mostly focuses on finite sets of patterns, and a first efficient solution to the pattern matching problem was given by Aho and Corasick [1]: they described an elegant construction for an automaton that recognizes A^*X, where A is a finite alphabet, which can thereafter be used to sequentially parse the text. The algorithm proceeds in two steps. The prefix tree of X is built first (we call its transitions *tree transitions*), and is then cleverly completed with *failure transitions*, using properties on borders of words. Defining the size of X as the sum of the lengths of its words, their solution runs in linear time. This algorithm is still widely used as a primitive brick in many situations (see for instance Baker [3] and Bird [5] algorithms for 2D pattern recognition).

Aho-Corasick automaton is always deterministic and complete, but unfortunately not necessarily minimal, as depicted on Figure 1. However, unless for specific patterns, Aho-Corasick automaton and the corresponding minimal one seem to differ only in a few number of states. This asks for some simpler process than the usual

* This work was completed with the support of the ANR project MAGNUM number 2010-BLAN-0204.

minimization algorithms [8], whose worst-case complexity is at least $\mathcal{O}(n \log n)$. It is however still not known whether Aho-Corasick automata can be minimized in linear time. The purpose of this article is to give a natural probabilistic framework on sets of patterns, under which we explain the observations above. Based on this study, we also propose a linear algorithm which generically[3] computes the minimal automaton. When this algorithm fails to output the minimal automaton, it still produces an automaton that recognizes A^*X and whose size is intermediate between the size of Aho-Corasick automaton and the one of the corresponding minimal automaton. For this reason, we call it a *pseudo-minimization*[4] *algorithm*.

Contrary to what we stated, the algorithm proposed in [2] does not always build the minimal automaton; we can nonetheless prove that it generically does. The present article can be seen as a natural continuation of this framework of analysis, which results in a much simpler algorithm, which works for more general probabilistic models. Especially, the number of words do not need to be fixed anymore in our new settings.

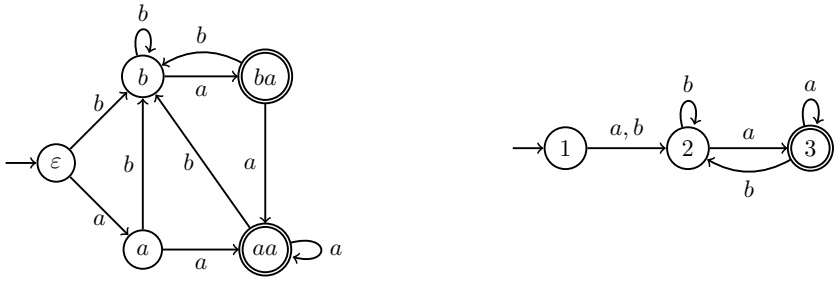

Fig. 1. On the left, an Aho-Corasick automaton that is not minimal, for $X = \{aa, ba\}$, and on the right, the corresponding minimal automaton with three states.

Our probabilistic framework can be described informally as follows. A set of patterns is generically of size $\mathcal{O}(n)$, n being the asymptotic parameter[5], with patterns of length at least $\Omega(\log n)$ and with low correlation. Low correlation means that a factor of length ℓ in a pattern of X has probability exponentially small in ℓ to appear elsewhere in X. These conditions are natural, since they are satisfied by classical probabilistic models on sets of words, as we shall see in Section 3.2. In particular they hold for the uniform distribution on sets of m words whose sum of lengths is n, with $m = \mathcal{O}(n^\gamma)$, for $0 \leq \gamma < \frac{1}{2}$.

Recall that most minimization algorithms proceed by merging Nerode-equivalent states, which leads to the minimal automaton. We first prove that under our

[3] A property holds "generically" when the probability it holds tends to one as the size tends to infinity.

[4] The notion of pseudo-minimal automaton we define in this article is not to be confused with the one introduced by Dominique Revuz in [11] for acyclic automata.

[5] It is sometime convenient to allow a size $\mathcal{O}(n)$ instead of strictly equal to n.

probabilistic models, the Aho-Corasick automata generically have the two following properties:

- The failure transitions all end at states "near the root" (at distance at most $\mathcal{O}(\log n)$ from the initial state).
- Any state that is near the root is not Nerode-equivalent to another state.

This leads to consider another equivalence relation on states defined by: p and q are $\equiv$-equivalent when p and q are either both final or both not final, and for any letter a, either both $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$ are failure transitions and $p' = q'$, or both transitions are tree transitions and $p' \equiv q'$. Notice that it differs from Nerode-equivalence in the fact that failure transitions must end at the same state instead of at equivalent states. Therefore merging $\equiv$-equivalent states results in a partially minimized automaton. However, generically, the two properties above ensure that $\equiv$-equivalence and Nerode-equivalence are equal, and that the partial minimization is in fact a full minimization. Moreover, $\equiv$-equivalence is easier to calculate than Nerode-equivalence, and we provide an algorithm that computes it in linear time, by adapting Revuz famous algorithm for minimizing acyclic automata [12].

The article is organized as follows. After recalling basic facts about words and automata in Section 2, we formally define and illustrate the considered probabilistic models in Section 3. In this section are also stated results on typical properties on associated Aho-Corasick automata. Our algorithms and the analysis of their complexities are then given in Section 4, along with experimental results.

2 Definitions and notations

In the sequel, we always work on a fixed finite alphabet A with $k \geq 2$ letters.

Words. A word y is a *factor* of a word x if there exist two words u and v such that $x = u \cdot y \cdot v$. The word y is a *prefix* (resp. a *suffix*) of x if $u = \varepsilon$ (resp. $v = \varepsilon$), ε being the empty word. We say that y is a *proper* prefix (resp. suffix) of x if y is a prefix (resp. suffix) such that $y \neq x$. Given a set of words X, we denote by $\texttt{Pref}(X)$ the set of all prefixes of words in X and by $\|X\|$ its *size*, defined as the sum of the lengths of its words. If u is a word of length n, its letters have indices in $\{0, \ldots, n-1\}$ and we denote by $u[i, j]$, for $0 \leq i \leq j \leq n-1$, the factor of u starting at index i and ending at index j.

Deterministic automata. A *deterministic and complete finite automaton* (or just *automaton* for short since we do not consider other kinds of automata in this article) over a finite alphabet A is a quintuple $\mathcal{A} = (A, Q, q_0, F, \delta)$, where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the *set of final states* and δ, the transition function, is a complete mapping from $Q \times A$ to Q. The transition function δ is extended inductively to $Q \times A^*$ by setting, for any $p \in Q$, $\delta(p, \varepsilon) = p$ and for any word $u \in A^*$ and any letter $a \in A$, $\delta(p, ua) = \delta(\delta(p, u), a)$. A word u is *recognized* by the automaton when $\delta(q_0, u) \in F$. The *size* of an automaton is its number of states. A classical result states that to each regular language L (a set of the words recognized by an automaton) one can associate its smallest automaton

which recognizes it. This automaton is unique and is called *the minimal automaton* of L. See [9] for more results about automata.

Aho-Corasick Algorithm. Let $X = \{u_1, u_2, \ldots, u_m\}$ be a set of m non-empty words, whose sum of lengths $\|X\| = \sum_{i=1}^{m} |u_i|$ is n. Aho-Corasick algorithm [1] builds an automaton that recognizes A^*X but which is not always minimal (see Figure 1). It has linear time and space complexities and proceeds in two main steps: The first step consists in constructing a tree-automaton $\mathcal{A}_X = (A, \mathtt{Pref}(X), \varepsilon, X, \delta)$, whose states are labelled by the prefixes of the words of X. The initial state is the empty word ε and the set of final states is made of the words of X. The transition function δ is defined by $\delta(u, a) = ua$ for any word u and any letter a such that $ua \in \mathtt{Pref}(X)$. Transitions of this kind are referred to as *tree transitions*. The second step consists in completing the transition function of the automaton $\mathcal{A}_X$, and possibly in adding some final states, to obtain the automaton $\mathcal{AC}_X$: For any state u and any letter a, $\delta(u, a)$ is the longest suffix of ua that is also in $\mathtt{Pref}(X)$, and the set of final states is $\mathtt{Pref}(X) \cap A^*X$; this can be done in linear time since the tree-automaton has already been calculated (more details on the complexity of the construction can be found in [6], proposition 2.9). The new transitions, which are not tree transitions, are called *failure transitions*. An example of Aho-Corasick's construction is depicted on Figure 2. For more information on Aho-Corasick automata and related algorithms, the reader is referred to [7].

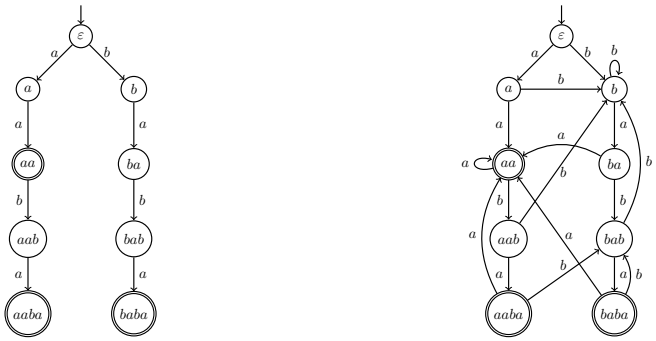

Fig. 2. The two steps construction of the Aho-Corasick automaton $\mathcal{AC}_X$ recognizing A^*X, where $X = \{aa, aaba, baba\}$. This automaton is not minimal because states aa, $aaba$ and $baba$ are equivalent, as are states a, aab and bab. Its minimal automaton has five states.

Note that since states of $\mathcal{AC}_X$ are words, we can define the size $|q|$ of a state as its length as a word of A^*.

3 Probabilistic models and generic properties

In this section we define the probabilistic models that are studied throughout this article. Then we illustrate them on natural examples and, finally, we exhibit the two main properties that hold generically for these models, and that will be used in the algorithmic section (Section 4).

3.1 Probabilistic models with low correlation

A *probabilistic model* is a sequence $(\mathbb{P}_n)_{n\geq 1}$ of probability measures on the same space. A property P is said to be *generic* for the probabilistic model $(\mathbb{P}_n)_{n\geq 1}$ when the probability that P is satisfied, tends to 1 when n tends to infinity.

Let $\mathcal{T}_m$ denote the set of all m-tuples of non-empty words and $\mathcal{T} = \cup_{m\geq 1}\mathcal{T}_m$. Under a probabilistic model $(\mathbb{P}_n)_{n\geq 1}$ over $\mathcal{T}$, a probability such as $\mathbb{P}_n\left(u_i \text{ satisfies } P\right)$ is to be understood as $\mathbb{P}_n\left(X \text{ has at least } i \text{ words and } u_i \text{ satisfies } P\right)$.

Definition 1. *A* probabilistic model with low correlation *is a probabilistic model* $(\mathbb{P}_n)_{n\geq 1}$ *on* $\mathcal{T}$ *such that there exist* $C > 0$, $\alpha > 0$ *and* $\beta > 1$, *and some* $\lambda > \frac{8}{\log \beta}$ *such that, the three following conditions hold generically:*

(1) *the tuple is of size at most* Cn*;*

(2) $\forall n \geq 1$, $\forall i, i' \geq 1$, $\forall j, j' \geq 0$, $\forall \ell \geq 0$, *if* $(i, j) \neq (i', j')$ *then*

$$\mathbb{P}_n\left(u_i[j, j+\ell-1] = u_{i'}[j', j'+\ell-1]\right) \leq \alpha\beta^{-\ell};$$

(3) *every word of the tuple is of length at least* $\lambda \log n$.

The definition above can be roughly described as follows:

• Because of Condition (1), the parameter n is, with high probability, an upper bound of the magnitude of the size of X: it is the parameter that will be used to measure the complexity. Note that it is convenient in some situations to allow this upper bound of $\mathcal{O}(n)$ instead of requiring the length to be exactly equal to n.

• Condition (2) means that generically, for two coordinates i and i' in the tuples and two starting positions j and j' in the words u_i and $u_{i'}$, the probability that the ℓ next letters coincide in u_i and $u_{i'}$ is exponentially small in ℓ.

• Condition (3) means that generically, the words in a tuple are not too small with respect to n. The value $\frac{8}{\log \beta}$ can be improved, but this is the smallest one that keeps our proof simple.

Hence, Condition (1) controls the size of the tuple, whereas Conditions (2) and (3) force the words of the tuple to have low self and mutual correlations. For instance it generically prevents that a word is a factor of another one.

▶ **Example:** Section 3.2 is devoted to examples of such distributions, but let us mention a first simple one to illustrate the definition. Let $A = \{a, b\}$ be a binary alphabet and for any $n \geq 1$, let $\mathbb{P}_n$ be the uniform distribution on words of length n, which are seen as 1-tuples of words. Condition (1) and Condition (3) hold, since the size of the tuple is the length of its only word, which is equal to n. Condition (2) also holds, since with probability 1 there is only one word: we only have to consider the case $i = i'$ with two different starting positions $j < j'$ in a random word u of length n. Using classical arguments from combinatorics on words [10], the number of words such that $u[j, j+\ell-1]$ and $u[j', j'+\ell-1]$ exist and are equal is at most $2^{n-\ell}$. Hence, Condition (2) holds with $\alpha = 1$ and $\beta = 2$. This simple probabilistic model is therefore of low correlation.

Lemma 1 below will be useful in the sequel and illustrates our terminology of "low correlation". If $X = (u_1, \ldots, u_k)$ is a tuple of words, a word w appears more than once as a factor in X if there exist $i, i' \geq 1$ and $j, j' \geq 0$, with $(i, j) \neq (i', j')$ such that $u_i[j, j+|w|-1]$ and $u_{i'}[j', j'+|w|-1]$ both exist and are equal to w.

Lemma 1. *Under a probabilistic model with low correlation, generically, no word of length $\ell \geq \frac{\lambda}{2}\log n - 1$ appears more than once as a factor in a tuple.*

Proof. The statement is similar to the one of Condition (2), except that it is not for fixed i, i', j, j' and ℓ. We bound the probability from above by summing the probabilities for each choice of the five parameters that satisfies our hypothesis.

Note that since Condition (1), Condition (2) and Condition (3) hold generically, it is sufficient to assume that they hold to prove the lemma, as we are looking for a generic result. We refer to such a tuple as a *generic tuple* in the rest of the proof.

By Condition (1), the size of a generic tuple is at most Cn, hence each of its words is also of length at most Cn. And since each word is non-empty, its length is at least 1, so that the number of components in a generic tuple is at most Cn. The length of a factor that can appear in a generic tuple is also at most Cn.

Hence, for $n \geq 1$, the probability that a word of length $\ell \geq \frac{\lambda}{2}\log n - 1$ appears more than once as a factor in a generic tuple is bounded from above by (we use Iverson bracket notation: $[\![P]\!]$ equals to 1 is P is true and 0 if P is false)

$$\begin{aligned}
&\sum_{\ell=\frac{\lambda}{2}\log n-1}^{Cn} \sum_{i,i'=1}^{Cn} \sum_{j,j'=0}^{Cn-\ell-1} \mathbb{P}_n\left(u_i[j, j+\ell-1] = u_{i'}[j', j'+\ell-1]\right) [\![(i,j) \neq (i',j')]\!] \\
&\qquad \leq C^4 n^4 \sum_{\ell=\frac{\lambda}{2}\log n-1}^{Cn} \alpha\beta^{-\ell} \leq C^4 n^4 \alpha \beta^{1-\frac{\lambda}{2}\log n} \sum_{m=0}^{\infty} \beta^{-m} \\
&\qquad \leq \frac{C^4\alpha\beta^2}{\beta-1} n^{4-\frac{\lambda}{2}\log\beta},
\end{aligned}$$

and this quantity tends to zero when n tends to infinity, since $\frac{\lambda}{2}\log\beta > 4$. □

3.2 Examples of probabilistic models with low correlation

► **Tuples of at most $\mathcal{O}(n^\gamma)$ words of total length n, $0 < \gamma < \frac{1}{2}$:** In this model, let $(m_n)_{n\geq 1}$ be a sequence of positive integers that is $\mathcal{O}(n^\gamma)$, for some $\gamma \in (0, \frac{1}{2})$. For any $n \geq 1$, we consider the uniform distribution on tuples $X = (u_1, \ldots, u_{m_n})$ of m_n words of total length n. A typical instance of this model is when m_n is a fixed integer (which is a distribution of tuple studied in [4]).

Obviously, Condition (1) always holds, with $C = 1$. Moreover, as mentioned in [4], such a tuple can be seen as a word u drawn uniformly at random in A^n and a composition of n into m_n parts also chosen uniformly at random and independently. For any integers i, i', j, j' such that $(i, j) \neq (i', j')$, if $u_i[j, j+\ell-1] = u_{i'}[j', j'+\ell-1]$, this factor appears at two different positions in u (the concatenation of the u_i's).The probability for a word of length n to contain a repetition of length ℓ is classically equal to $k^{-\ell}$. Therefore Condition (2) is satisfied for $\alpha = 1$ and $\beta = k$.

In order to prove that Condition (3) also holds generically, we use the following technical lemma:

Lemma 2. *For any γ and δ in $(0, 1)$ such that $2\gamma + \delta < 1$, generically, a composition of n into $m_n = \mathcal{O}(n^\gamma)$ parts contains no part of length smaller than n^δ.*

Since $\gamma < \frac{1}{2}$ in the model, there exists some $\delta > 0$ such that $2\gamma + \delta < 1$, and thus Lemma 2 applies. This proves Condition (3).

▶ **Tuples of n^γ words of length at most $n^{1-\gamma}$, $0 < \gamma < 1$:** Conditions (1) and (2) are direct consequences of the definition. Condition (3) is still easily checked since generically a word of length at most $n^{1-\gamma}$ is of length greater than $\frac{1}{2}n^{1-\gamma}$.

For an example of such a distribution, choose $\gamma = \frac{1}{2}$ and consider the set of tuples containing $m = \lfloor \sqrt{n} \rfloor$ words, each word being chosen uniformly at random (and independently) in the set $A^{\leq m}$ of all words of length at most m.

▶ **More advanced models for words:** For the same choices of distributions on sizes as above, if we enrich the model by generating the words according to a Bernoulli model (which associates a positive probability to each letter) or by an ergodic finite Markov chain, it is not difficult to check that the conditions for having low correlation still hold. It is mainly a consequence of the exponentially fast forgetfulness property of finite Markov chains (Bernoulli models are simpler instances of finite Markov chains).

3.3 Generic properties of the associated Aho-Corasick automaton

Lemma 3. *Under a probabilistic model with low correlation over the set of tuples X from which the automaton is built, any state q of the Aho-Corasick automaton such that $|q| \geq \frac{\lambda}{2} \log n$ has generically only one incoming transition.*

Proof. For any $n \geq 1$, let X be a tuple that satisfies all conditions of low correlation and therefore the generic properties of Lemma 1.

If the property of Lemma 3 does not hold, there exists a state q, with $|q| \geq \frac{\lambda}{2} \log n$ and a failure transition $p \xrightarrow{a} q$ ending on q. By construction of the transition function δ in an Aho-Corasick automaton, q is the longest proper suffix of the word $p \cdot a$ that is also a prefix of a word of X. Therefore, $q = w \cdot a$ for some word w of length at least $\frac{\lambda}{2} \log n - 1$, since $|w| = |q| - 1$. The word w is a proper suffix of p, and a prefix of q; it is therefore a factor that appears at least twice in the tuple. By Lemma 1, this event does not happen generically, concluding the proof. □

Lemma 4. *Under a probabilistic model with low correlation over the set of tuples X from which the automaton is built, generically, every state q of the Aho-Corasick automaton such that $|q| < \frac{\lambda}{2} \log n$ is not Nerode-equivalent to another state of the automaton.*

Proof. For any $n \geq 1$, let X be a tuple that satisfies all conditions of low correlation and therefore the generic properties of Lemma 1 and Lemma 3. Let $\mathcal{AC}_X$ be the Aho-Corasick automaton associated with X and q be a state such that $|q| < \frac{\lambda}{2} \log n$.

We claim that q is not Nerode-equivalent to another state of $\mathcal{AC}_X$. By contradiction, assume that there exists a state $p \neq q$ that is Nerode-equivalent to q, with $|p| \geq |q|$ (if $|p| < |q|$ then $|p| < \frac{\lambda}{2} \log n$ and one can switch the roles of p and q). Let u be a word such that there exists a path from p to a final state labelled by u that uses tree transitions only.

Consider the path labelled by u starting from q; since p and q are equivalent, this path ends at a final state f. Hence, by Condition (3) and since $|q| < \frac{\lambda}{2}\log n$, we have $|u| \geq \frac{\lambda}{2}\log n$. This path is made of tree and failure transitions, and there is at least one failure transition by Lemma 1, otherwise X would contain two occurrences of u, but $|u| \geq \frac{\lambda}{2}\log n$. Let r be the last state reached by a failure transition along this path, and let w be the suffix of $u = vw$ that labels this path from r to f. By construction, the path starting from r and labelled by w uses tree transitions only. Hence w labels paths using tree transitions only, from both states r and $p \cdot v$. These two states are different: since the path starting at state q and labelled by v uses at least one failure transition, $|r| = |\delta(q, v)| < |q| + |v|$ and since $|q| \leq |p|$, this implies that $|r| < |p \cdot v|$.

At this point, we have proved that w appears at least twice as a factor in X. It only remains to prove that w is long enough to make this situation impossible in our settings: since X satisfies the property of Lemma 3 and r is the target of a failure transition, it has at least two incoming transitions and then $|r| < \frac{\lambda}{2}\log n$. Moreover, since $r \cdot w \in X$ and X satisfies Condition (3), then $|r| + |w| \geq \lambda \log n$. Hence $|w| \geq \frac{\lambda}{2}\log n$, yielding the contradiction for a generic X, by Lemma 1. □

4 A pseudo-minimization algorithm for Aho-Corasick automata

In this section we present a simple algorithm that, under a low correlation probabilistic model for the inputs, generically minimizes the Aho-Corasick automaton in linear time and space.

4.1 Algorithm

The pseudo-minimization algorithm[6] we propose is an adaptation of Revuz algorithm [12] that minimizes acyclic deterministic automata.

Let us first introduce the notion of $\equiv$-*equivalence* on states of an automaton. Two states p and q are $\equiv$-*equivalent* when both p and q are terminal or both non-terminal, and for every letter a, either both $p \xrightarrow{a}$ and $q \xrightarrow{a}$ are tree transitions and $\delta(p, a)$ is $\equiv$-equivalent to $\delta(q, a)$, or both $p \xrightarrow{a}$ and $q \xrightarrow{a}$ are failure transitions and $\delta(p, a) = \delta(q, a)$. Therefore if the two transitions are not of the same nature, the two states are not $\equiv$-equivalent. Note that if two states are $\equiv$-equivalent they have to be at the same distance from terminal states. We use this observation to define the *distance function* d on the states of the Aho-Corasick automaton. For any state p, $d(p)$ is the maximal distance from p to a leaf in the prefix tree: $d(p) = \max\{|w| \mid p \cdot w \in X\}$. Denote also by $\mathcal{D}_i$ the set of states at distance i; the distance function defines a partition $\mathcal{D} = (\mathcal{D}_i)_{0 \leq i \leq d(\varepsilon)}$ of the set of states Q. The distance function relies on tree transitions only and can be computed bottom-up by a depth-first traversal of the tree.

[6] An implementation of the algorithm can be found in `http://lipn.fr/~aitmous/`.

If we have already computed $\equiv$-equivalent states for all the classes $\mathcal{D}_j$ with $j < i$, we can easily compute the $\equiv$-equivalent states in the class $\mathcal{D}_i$, using the definition. This leads to a simple pseudo-minimization method (see Algorithm 1 below).

Algorithm 1: Pseudo-Minimization Method

Inputs : Aho-Corasick Automaton $\mathcal{AC}_X$
Outputs: Pseudo-Minimal Automaton

compute $\mathcal{D}$;
for $i = 0$ **to** $d(\varepsilon)$ **do**
    detect $\equiv$-equivalent states at distance i;
    merge them;

More precisely, to detect $\equiv$-equivalent states at distance i, we proceed as follows. Let p and q be two states with $d(p) = d(q) = i$. Assuming that for $j < i$, $\equiv$-equivalent states in classes $\mathcal{D}_j$ have already been merged, p and q are $\equiv$-equivalent if and only if both p and q are terminal or non-terminal, and for every $a \in A$, $\delta(p, a) = \delta(q, a)$. The last item differs from the definition because of the dynamic merging of $\equiv$-equivalent states in the process of the method.

Algorithm 2: State-Distinguishing Algorithm

Inputs : Set of states S
Outputs: A set of lists of pseudo-equivalent states

$a \leftarrow$ first letter of A;
split S into two lists L_1 (of terminal states) and L_2 (of non-terminal states);
$R_1 \leftarrow \{L_1, L_2\}$;
$R_2 \leftarrow \emptyset$;
while $R_1 \neq \emptyset$ `and` $a \neq$ `null` **do**
    $R_2 \leftarrow R_1$;
    $R_1 \leftarrow \emptyset$;
    while $R_2 \neq \emptyset$ **do**
        $L \leftarrow$ first list of R_2;
        create an array T of $|Q|$ empty lists;
        for `all` ***states*** $p \in L$ **do**
            $q \leftarrow \delta(p, a)$;
            add p to the list $T[q]$;
        for `all` ***non-empty lists*** $\ell \in T$ **do**
            if $|\ell| \geq 2$ **then**
                add ℓ to R_1;
    $a \leftarrow$ next letter of A; `// returns null if there is no more letter in` A
return R_1;

Algorithm 2 explains how $\equiv$-equivalent states at distance i are detected. Given the list of the states of $\mathcal{D}_i$, it first splits the list in two: terminal and non-terminal states. Then these groups are, in turn, split according to each letter of the alphabet, by using a kind of bucket sort algorithm on outgoing transitions, but in which we are only interested in identifying states having identical outgoing transitions (not actually sorting them). In order to test only non-empty lists in line 14, we can use a list NE to keep track of non-empty lists. Whenever a state p is inserted in an empty list $T[q]$ in line 13, q is inserted in the list NE.

Figure 3 illustrates the pseudo-minimization of the automaton given Figure 2, showing an example where the pseudo-minimization algorithm fails to output the minimal automaton.

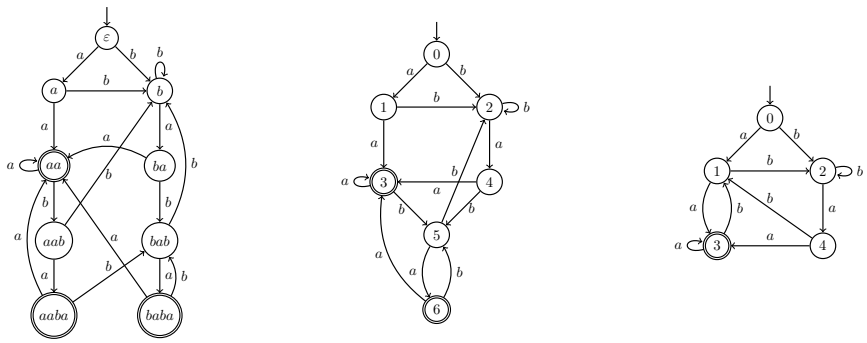

Fig. 3. The pseudo-minimization of the Aho-Corasick automaton recognizing A^*X, where $X = \{aa, aaba, baba\}$ (left), produces the automaton (center). The automaton is not minimal since states 3 and 6 are equivalent, as are states 1 and 5. On the right is depicted the minimal automaton.

Lemma 5. *With the states of $\mathcal{D}_i$, and all equivalent states at distance less than i already merged, Algorithm 2 computes the classes of $\equiv$-equivalent states of $\mathcal{D}_i$.*

Proof. The proof of the proposition uses the same arguments as in [12]. We prove the validity of the algorithm by induction on the number of executions of the outer loop (lines 5-16). After c executions, R_1 contains only lists of states that are either all terminal or all non-terminal, and have the same outgoing transitions labelled by the c first letters of the alphabet.

In lines 8-16, the list of states L is split by the next outgoing transitions. Thus the lists added to R_1 only contain states with the same $c+1$ first transitions. Since the inner loop is executed on every list in R_2, the property holds.

At the end of the outer-loop, if there are $\equiv$-equivalent states, they are in the same lists of R_1. □

Theorem 1. *Under a probabilistic model with low correlation on the inputs of the algorithm, the pseudo-minimization algorithm generically returns the minimal automaton of A^*X.*

Proof. Let $\mathcal{A}$ be an Aho-Corasick automaton satisfying the statements of Lemmas 3 and 4. Since these properties are generic, it is sufficient to prove that such an automaton is minimized by the pseudo-minimization algorithm.

Let p and q be two Nerode-equivalent states of $\mathcal{A}$ such that $p \neq q$. For any letter $a \in A$, if $\delta(p, a)$ is a failure transition, then by Lemma 3, $\delta(p, a)$ ends at some state r with $|r| < \frac{\lambda}{2} \log n$. And by Lemma 4 the state r is not Nerode-equivalent to another state. Hence $\delta(q, a) = r$ too. If $\delta(p, a)$ and $\delta(q, a)$ are both tree transitions and p and q are in $\mathcal{D}_i$, then $p \cdot a$ and $q \cdot a$ are in $\mathcal{D}_j$ for some $j < i$, and have already been merged if they were Nerode-equivalent. Hence $\delta(p, a) = \delta(q, a)$ in this case too. □

4.2 Complexity

Lemma 6. *Algorithm 2 distinguishes r states in space and time $\mathcal{O}(|Q| + r)$.*

Proof. Lines 14-16 take at most the same number of steps as the loop in lines 11-13 (in the worst case, each state of L has a different outgoing transition labeled by a). Thus the time complexity is bounded by the number of times line 11 is executed and that is kr in the worst case. The additional time and space is used for the array T of size $|Q|$. □

The space complexity of Algorithm 2 is linear in $|Q|$ since, for any letter a, lists of states are split according to their outgoing transition labelled by a, which can reach any state of $\mathcal{AC}_X$. Hence, the array T is of size $|Q|$. This bound can be lowered by a renumbering of the states, using an idea of Revuz [12]: at step i, we use a counter to associate numbers to states that are adjacent to elements of $\mathcal{D}_i$. These numbers range from 1 to the number n_i of such states. Since the automaton is deterministic, $n_i \leq |\mathcal{D}_i| \times k$. We gain in complexity using these numbers as indices of T instead of elements of Q. To avoid conflicts when going to step $i + 1$, we also store for each state at which step its number has been updated: at Line 12, when a state $q = \delta(p, a)$ is checked for some state p, either q has already been numbered at this step and we do not change it, or we associate q with the next value of the counter. In both cases, p is added at the corresponding index in T.

Theorem 2. *Algorithm 1 is linear in time and space.*

Proof. The distance function can be computed in linear time using a depth-first traversal of the tree automaton. Once it is done, the calculation of $\equiv$-equivalent states is done bottom-up, using a depth-first traversal of the tree automaton too. Using the renumbering technique mentioned above, we avoid the $|Q|$ multiplier at each level, which leads to an overall linear complexity, since $\sum_{i=0}^{d(\varepsilon)} |\mathcal{D}_i|$ is equal to the number of states of the Aho-Corasick automaton. □

Remark: in practice, one can use closed hashing to assign a unique number to every $\equiv$-equivalent class or state label both viewed as strings. We see this closed hash table as an injective function from strings to positive integers. Start with adding the label of every state in the hash table. Then associate to any state q a

string signature $\sigma(q)$ of the form $\sigma(q) =$ "$T|n_a|n_b|n_c$", for $A = \{a, b, c\}$, where T stands for "q is terminal", and where n_a is the number assigned by the hash table to the class of $\delta(q, a)$ if it is a tree transition, or the number assigned to state $\delta(q, a)$ if it is a failure transition. Doing this bottom-up, two states are $\equiv$-equivalent if and only if they have the same signature, the number associated to the signature being obtained by a search in the hash table. This leads to a linear algorithm in practice.

4.3 Experimental results

We experimented our algorithm on different types of patterns. First we tested it on the genome of the *mycoplasma genitalium*, a parasitic bacterium. The genome consists of 482 protein encoding genes, for a total length of more than 580,000 base pairs. For this experiment, our algorithm did output the minimal automaton. There are no small words, as the genes' length range from around 100 to around 5500. It is therefore in the framework of our theoretical model, which explains why our algorithm behaves well, as shown in Table 1.

Aho-Corasick automaton	Pseudo-minimal automaton	Minimal automaton
528 670 states	526 347 states	526 347 states

Table 1. For the mycoplasma genitalium genome, the pseudo-minimization algorithm actually computes the minimal automaton.

We also tried our algorithm on a French dictionary, which has almost 380,000 words. This result depicted in Table 2 has been anticipated: as a consequence of having both a lot of small words and high correlations, we are clearly out of our low correlation model, and the classical minimization is more efficient in reducing the space representation (though a lot of space is already gained using our linear algorithm).

Aho-Corasick automaton	Pseudo-minimal automaton	Minimal automaton
722 074 states	113 130 states	27 362 states

Table 2. For the French dictionary, the pseudo-minimization does not compute the minimal automaton.

We also observed in several situations that though not computing the minimal automaton, the output of our algorithm differs from the minimal automaton by only a few states. This is typically the case when taking a set made of all the words that differ from a random word u by at most one letter.

Conclusion

We proposed a pseudo-minimization algorithm for Aho-Corasick automata and proved that, under natural probabilistic models, it outputs the minimal automaton with high probability. In order to obtain more general results we isolated the probabilistic properties that make the analysis work from the study of models of patterns that satisfy these properties: this way we obtained more general results, that can be reused for new models, provided one checks that the low correlation properties hold. Note that there is a trade off between Conditions (2) and (3): one can ask for a polynomial decrease in Condition (2) at the cost of a more restrictive size requirement in Condition (3). We choose this statement because most common sources for words satisfy the exponential decrease of Condition (2).

Experiments show that for probabilistic models with low correlation, the algorithm as expected outputs the minimal automaton. An interesting phenomenon can be observed for patterns that are clearly out of our probabilistic framework. Although not computing the minimal automaton, the algorithm still computes an automaton whose number of states is often close to that of the minimal automaton.

A natural continuation of this work is to quantify that kind of observations for relaxed probabilistic models.

References

1. Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.
2. Omar AitMous, Frédérique Bassino, and Cyril Nicaud. Building the minimal automaton of A^*X in linear time, when X is of bounded cardinality. In *21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010)*, volume 6129. LNCS, pages 275–287. Springer-Verlag, 2010.
3. Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM J. Comput.*, pages 533–541, 1978.
4. Frédérique Bassino, Laura Giambruno, and Cyril Nicaud. The average state complexity of rational operations on finite languages. *Int. J. Found. Comput. Sci.*, 21(4):495–516, 2010.
5. Richard S. Bird. Two dimensional pattern matching. *Inf. Process. Lett.*, 6(5):168–170, 1977.
6. Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings.* Cambridge University Press, 2007.
7. Maxime Crochemore and Wojciech Rytter. *Text Algorithms.* Oxford Univ. Press, 1994.
8. John E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.
9. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
10. M. Lothaire. *Applied Combinatorics on Words.* Cambridge University Press, 2005.
11. Dominique Revuz. *Dictionnaires et lexiques: methodes et algorithmes.* PhD thesis, Institut Blaise Pascal, 1991.
12. Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoret. Comput. Sci.*, 92(1):181–189, 1992.