Building the Minimal Automaton of A*X in Linear Time, When X Is of Bounded Cardinality

Omar Aitmous, Frédérique Bassino, Cyril Nicaud

▶ **To cite this version:**

Omar Aitmous, Frédérique Bassino, Cyril Nicaud. Building the Minimal Automaton of A*X in Linear Time, When X Is of Bounded Cardinality. 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010), Jun 2010, United States. pp.275-287. hal-00450674v1

HAL Id: hal-00450674

https://hal.science/hal-00450674v1

Submitted on 26 Jan 2010 (v1), last revised 20 May 2010 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Building the Minimal Automaton of A^*X in Linear Time, When X Is of Bounded Cardinality

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

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

Abstract. We present an algorithm for constructing the minimal automaton recognizing A^*X, where the pattern X is a set of m (that is a fixed integer) non-empty words over a finite alphabet A whose sum of lengths is n. This algorithm, based on Brzozowski's minimization algorithm, uses sparse lists to achieve a linear time complexity.

1 Introduction

This paper addresses the following issue: given a *pattern* X (that is a non-empty language which does not contain the empty word ε) and a text $T \in A^+$, assumed to be very long, how to efficiently find occurrences of words of X in the text T?

A usual approach is to precompute a deterministic automaton recognizing the language A^*X and use it to sequentially treat the text T. To find the occurrences of words of X, we simply read the text and move through the automaton. An occurrence of the pattern is found every time a final state is reached. Once built, this automaton can of course be used for other texts.

The pattern X can be of different natures, and we can reasonably consider three main categories: a single word, a finite set of words and a regular language. Depending on the nature of the pattern, the usual algorithms [6] construct a deterministic automaton that is not necessary minimal.

In the case of a single word u, very efficient algorithms such as the ones of Knuth, Morris and Pratt [10, 6] or Boyer and Moore [4, 6] are used. Knuth-Morris-Pratt algorithm builds the minimal automaton recognizing $A * u$. Aho-Corasick algorithm [1] treats the case of a finite set of words by constructing a deterministic yet non-minimal automaton. An algorithm for the case of regular languages based on a deterministic automaton was proposed by Mohri in [11].

In this article, we consider the case of a set of m non-empty words whose sum of lengths is n, where m is fixed and n tends toward infinity. Aho-Corasick algorithm [1] builds a deterministic automaton that recognizes A^*X with linear time and space complexities. Experimentally we remark, by generating uniformly at random patterns of m words whose sum of lengths is n, that the probability for Aho-Corasick automaton to be minimal is very small for large n.

One can apply a minimization algorithm (such as Hopcroft's algorithm [8]) to Aho-Corasick automaton, but this operation costs an extra $\mathcal{O}(n \log n)$ time.

We propose another approach to directly build the minimal automaton of A^*X. It is based on Brzozowski's minimization algorithm described in [5]. This algorithm considers a non-deterministic automaton $\mathcal{A}$ recognizing a language $\mathcal{L}$, and computes the minimal automaton in two steps. First the automaton $\mathcal{A}$ is reversed and determinized. Second the resulting automaton is reversed and determinized too. Though the complexity of Brzozowski's algorithm is exponential in the worst case, our adaptation is linear in time and quadratic in space, using both automata constructions and an efficient implantation of sparse lists. The fact that the space complexity is greater than the time complexity is typical for that kind of sparse list implantation (see [3] for another such example, used to minimize local automata in linear time).

Outline of the paper: Our algorithm consists in replacing the first step of Brzozowski's algorithm by a direct construction of a co-deterministic automaton recognizing A^*X, and in changing the basic determinization algorithm into an *ad hoc* one using the specificities of the problem in the second step. With appropriate data structures, the overall time complexity is linear.

In Section 2 basic definitions and algorithms for words and automata are recalled. A construction of a co-deterministic automaton recognizing A^*X is described in Section 3. The specific determinization algorithm that achieves the construction of the minimal automaton is presented in Section 4. Section 5 addresses the global complexity of the construction. Note that for a lack of space, only sketches of the proofs are given.

2 Preliminary

In this section, the basic definition and constructions used throughout this article are recalled. For more details, the reader is referred to [9] for automata and to [6, 7] for algorithms on strings.

Automata. A *finite automaton* $\mathcal{A}$ over a finite alphabet A is a quintuple $\mathcal{A} = (A, Q, I, F, \delta)$, where Q is a finite set of *states*, $I \subset Q$ is the set of *initial states*, $F \subset Q$ is the set of *final states* and δ is a transition function from $Q \times A$ to $\mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the *power set* of Q. The automaton $\mathcal{A}$ is *deterministic* if it has only one initial state and if for any $(p, a) \in Q \times A$, $|\delta(q, a)| \leq 1$. It is *complete* if for any $(p, a) \in Q \times A$, $|\delta(q, a)| \geq 1$. A deterministic finite automaton $\mathcal{A}$ is *accessible* when for each state $q \in Q$, there exists a path from the initial state to q. The size of an automaton $\mathcal{A}$ is its number of states. The *minimal automaton* of a regular language is the unique smallest accessible and deterministic automaton recognizing this language.

The transition function δ is first extended to $\mathcal{P}(Q) \times A$ by $\delta(P, a) = \cup_{p \in P} \delta(p, a)$, then inductively to $\mathcal{P}(Q) \times A^*$ by $\delta(P, \varepsilon) = P$ and $\delta(P, w \cdot a) = \delta(\delta(P, w), a)$. A word u is recognized by $\mathcal{A}$ if there exists an initial state $i \in I$ such that $\delta(i, u) \cap T \neq \emptyset$. The set of words recognized by $\mathcal{A}$ is the language $\mathcal{L}(\mathcal{A})$.

The *reverse* of an automaton $\mathcal{A} = (A, Q, I, F, \delta)$ is the automaton ${}^t\mathcal{A} = (A, Q, F, I, {}^t\delta)$. For every $(p, q, a) \in Q \times Q \times A$, $p \in {}^t\delta(q, a)$ if and only if $q \in \delta(p, a)$.

We denote by $\widetilde{w}$ the mirror word of w. The automaton ${}^t\mathcal{A}$ recognizes the language $\widetilde{L(\mathcal{A})} = \{\widetilde{w} \mid w \in \mathcal{L}(\mathcal{A})\}$. An automaton $\mathcal{A}$ is *co-deterministic* if its reverse automaton is deterministic.

Any finite automaton $\mathcal{A} = (A, Q, I, F, \delta)$ can be transformed by the *subset construction* into a deterministic automaton $\mathcal{B} = (A, \mathcal{P}(Q), \{I\}, F_{\mathcal{B}}, \delta_{\mathcal{B}})$ recognizing the same language and in which $F_{\mathcal{B}} = \{P \in \mathcal{P}(Q) \mid P \cap F \neq \emptyset\}$ and $\delta_{\mathcal{B}}$ is a function from $\mathcal{P}(Q) \times A$ to $\mathcal{P}(Q)$ defined by $\delta_{\mathcal{B}}(P, a) = \{q \in Q \mid \exists p \in P \text{ such that } q \in \delta(p, a)\}$. In the following we consider that the determinization of $\mathcal{A}$ only produces the accessible and complete part of $\mathcal{B}$.

Two complete deterministic finite automata $\mathcal{A} = (A, Q, i_0, F, \delta)$ and $\mathcal{A}' = (A, Q', i'_0, F', \delta')$ on the same alphabet are *isomorphic* when there exists a bijection ϕ from Q to Q' such that $\phi(i_0) = i'_0$, $\phi(F) = F'$ and for all $(q, a) \in Q \times A$, $\phi(\delta(q, a)) = \delta'(\phi(q), a)$. Two isomorphic automata only differ by the labels of their states.

Combinatorics on 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* of x if $u = \varepsilon$; it is a *suffix* of x if $v = \varepsilon$. We say that y is a *proper* prefix (resp. suffix) of x if y is a prefix (resp. suffix) such that $y \neq \varepsilon$ and $y \neq x$.

A word y is called *a border* of x if $y \neq x$ and y is both a prefix and a suffix of x. We denote by $Border(x)$ the longest border of a non-empty word x. We say that $Border(x)$ is *the* border of x. Note that any other border of x is a border of $Border(x)$. The set of all borders of x is $\{Border(x), Border(Border(x)), \ldots\}$.

In the following we note $x[i]$ the i-th letter of x, starting from position 0; the factor of x from position i to j is denoted by $x[i \cdot\cdot j]$. If $i > j$, $x[i \cdot\cdot j] = \varepsilon$.

To compute all borders of a word x of length ℓ, we construct the *border table* of x defined from $\{0, 1, \cdots, \ell\text{-}1\}$ to $\{0, 1, \cdots, \ell\text{-}1\}$ by $border[i] = \mid Border(x[0 \cdot\cdot i]) \mid$. An efficient algorithm that constructs the border table is given in [6, 7]. Its time and space complexities are $\Theta(|x|)$. It is based on the following formula, for any $x \in A^+$ and $a \in A$.

$$Border(x \cdot a) = \begin{cases} Border(x) \cdot a & \text{if } Border(x) \cdot a \text{ is a prefix of } x, \\ Border(Border(x) \cdot a) & \text{otherwise.} \end{cases} \quad (1)$$

3 A Co-Deterministic Automaton Recognizing A^*X

In this section we give a direct construction of a co-deterministic automaton recognizing A^*X.

Remark that if there exist two words $u, v \in X$ such that u is a suffix of v, one can remove the word v without changing the language, since $A^*v \subset A^*u$ and thus $A^*X = A^*(X \setminus \{v\})$. Hence, in the following we only consider finite suffix sets X, i.e. there are not two distinct words $u, v \in X$ such that u is a suffix of v.

Proposition 1. *Let X be a set of m non-empty words whose sum of lengths is n. There exists a deterministic automaton recognizing the language $\widetilde{X}A^*$ whose number of states is at most $n - m + 2$.*

Proof. (By construction) Let $\mathcal{A}$ be the automaton that recognized $\widetilde{X}$, built directly from the trie of $\widetilde{X}$ by adding an initial state to the root and final states to the leaves. The states are labelled by the prefixes of $\widetilde{X}$. Next as we are basically interested in X change every state label by its mirror, so that the states of the automaton are labelled by the suffixes of X. Finally merge all the final states into one new state labelled i, and add a loop on i for every letter in A. This automaton is deterministic and recognizes the language $\widetilde{X}A^*$. □

The space and time complexities of this construction are linear in the length of X. This automaton is then reversed to obtain a co-deterministic automaton recognizing A^*X. For a given finite set of words X, we denote by $\mathcal{C}_X$ this co-deterministic automaton.

Example 1. Let us illustrate this first step on the following example.
Let $A = \{a, b\}$ be the alphabet and $X = \{aa, aaab, bb\}$ be a set of $m = 3$ words whose sum of lengths is $n = 8$. The steps of the process are given in Figure 1.

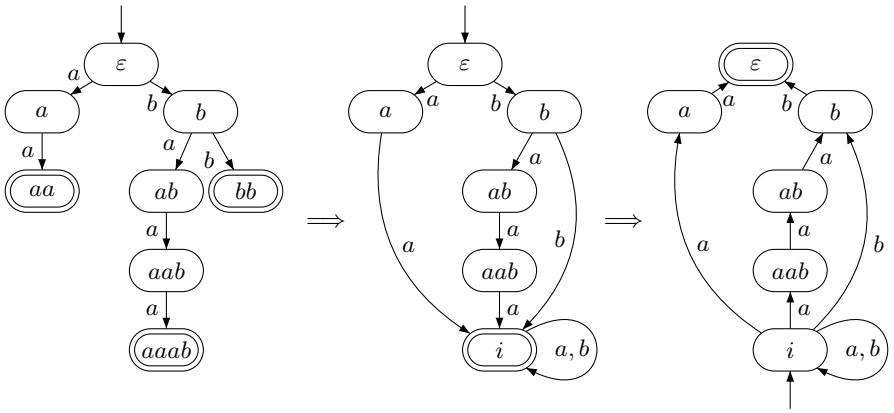

Fig. 1. Co-deterministic automaton $\mathcal{C}_X$ recognizing A^*X, where $X = \{aa, aaab, bb\}$.

4 Computing the Minimal Automaton

Once $\mathcal{C}_X$ is built, its determinization produces the minimal automaton recognizing the same language. It comes from the property used by Brzozowski's algorithm, namely that the determinization of a co-deterministic automaton gives the minimal automaton.

According to Aho-Corasick algorithm this minimal automaton has at most $n + 1$ states. It remains to efficiently manipulate sets of states in the determinization process. The subset construction produces the accessible part $\mathcal{B}$ of the automaton $(A, \mathcal{P}(Q), \{I\}, F_{\mathcal{B}}, \delta_{\mathcal{B}})$ from an automaton $\mathcal{A} = (A, Q, I, F, \delta)$. The states of $\mathcal{B}$ are labelled by subsets of Q. For the example, this procedure produces from the last automaton of Figure 1 the minimal automaton depicted in Figure 2.

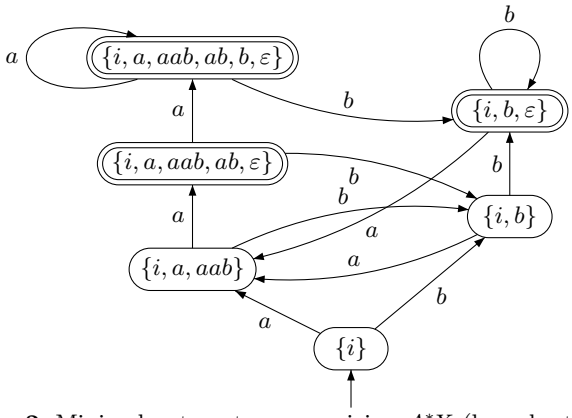

Fig. 2. Minimal automaton recognizing A^*X (by subset construction).

4.1 Cost of the Naive Subset Construction

When computing the subset construction, one has to handle sets of states: starting from the set of initial states, all the accessible states are built from the fly, using a depth-first traversal (for instance) of the result. At each step, given a set of states P and a letter a, one has to compute the set $\delta(P, a)$ and then check whether this state has already been built.

The co-deterministic automaton $\mathcal{C}_X$, is deterministic but for the initial state i. And for every letter a, the image of i by a is of size at most $m+1$, where m is the number of words in X. Hence, $\delta(P, a)$ is of cardinality at most $m+1+|P|$ and is computed in time $\Theta(|P|)$, assuming that it is done using a loop through the elements of P. So even without taking into account the cost of checking whether $\delta(P, a)$ has already been built, the time complexity is $\Omega(\sum_P |P|)$, where the sum ranges over all P in the accessible part of the subset construction.

Consider $X = \{a^{n-1}, b\}$, since the states of the result would be $\{i\}$, $\{i, a^{n-2}\}$, $\{i, a^{n-2}, a^{n-3}\}$, $\cdots$, $\{i, a^{n-2}, a^{n-3}, \cdots, a\}$, $\{i, a^{n-2}, a^{n-3}, \cdots, a, \varepsilon\}$ and $\{i, \varepsilon\}$, so that $\sum_P |P| = \Omega(n^2)$ and the time complexity of the naive subset construction is at least quadratic.

In the sequel, we present an alternative way to compute the determinization of $\mathcal{C}_X$ whose time complexity is linear.

4.2 Outline of the Construction

We make use of the following observations on $\mathcal{C}_X$. In the last automaton of Figure 1 when the state labelled ab is reached, a word $u = v \cdot aa$ has been read, and the state aab has also been reached. This information only depends on borders of prefixes of words in X: aa is a prefix of the word $x = aaab \in X$, and $Border(aa) = a$. That way, we reach the suffix aab of x since we read a word $u = v_1 \cdot a$, with $v_1 = v \cdot a$.

Our algorithm is based on limiting the length of the state labels of the minimal automaton by storing only one state per word of X, and one element to mark the state as final or not (ε or $\notin$). That way, if we read aa, we only store

ab for the word $x = aaab$. When we check for a transition labelled by $c \in A$ and $\delta(ab, c)$ is undefined, we use the borders of prefixes of words in X as mentioned above to find another state to check (aab in our example). We continue until either a transition we are looking for is found, or the unique initial state i is reached.

Let define the *failure* function f from $X \times Q \setminus \{i, \varepsilon\} \times A$ to $Q \setminus \{\varepsilon\}$ by $f(x, p, a) = q$ if $\delta(q, a)$ is defined and q is the smallest suffix of x such that $x = u \cdot p = v \cdot q$, with u and v prefixes of x, and v is a border of u. If no such state q exists, $f(x, p, a) = i$.

4.3 Precomputation of the Failure Function

Our failure function is similar to the Aho-Corasick one in [1]. The difference is that ours is not based on suffixes but on borders. To compute it the value of $Border(v \cdot a)$ for every proper prefix v of a word $u \in X$ and every letter $a \in A$ is needed.

Extended border table. Let u be a word of length ℓ. We define an *extended* border table from $\{0, 1, \cdots, \ell-1\} \times A$ to $\{0, 1, \cdots, \ell-1\}$ by $border\_ext[0][u[0]] = 0$ and $border\_ext[i][a] = |Border(u[0 \cdot\cdot i-1] \cdot a)|$ for all $i \in \{1, \cdots, \ell-1\}$. Recall that $u[0 \cdot\cdot i]$ is the prefix of u of length $i+1$. Remark that $|Border(u[0 \cdot\cdot i])| = |Border(u[0 \cdot\cdot i-1] \cdot u[i-1])| = border\_ext[i-1][u[i-1]]$.

Algorithm 1 (see Figure 3) computes the extended border table for a word u of length ℓ, considering the given alphabet A.

Table 1 depicts the border table of the word $aaab$. Values computed by a usual border table algorithm are represented in bold.

	Prefix w of u with $w \neq u$			
Letter	ε	a	aa	aaa
a	**0**	**1**	**2**	3
b	/	0	0	**0**

Table 1. Extended border table for $u = aaab$, given $A = \{a, b\}$.

Standard propositions concerning the border table algorithm given in [7] are extended to Algorithm 1.

Proposition 2. *Extended_Borders algorithm above computes the extended border table of a given word u of length ℓ considering the alphabet A. Its space and time complexities are linear in the length of the word u.*

Failure function. $f(u, p, a)$ is precomputed for every word $u \in X$, every proper suffix p of u and every letter $a \in A$ using Algorithm 2. The total time and space complexities of this operation are linear in the length of X. Let us remark that if $f(u, p, a) \neq i$ then $|\delta(f(u, p, a), a)| = 1$.

Algorithm 1 EXTENDED_BORDERS

Inputs: $u \in X$, $\ell = |u|$, alphabet A

```
1  border_ext[0][u[0]] ← 0
2  for j ← 1 to ℓ − 1 do
3     for a ∈ A do
4        i ← border_ext[j − 1][u[j − 1]]
5        while i ≥ 0 and a ≠ u[i] do
6           if i ≤ 1 then
7              i ← −1
8           else
9              i ← border_ext[i−1][u[i−1]]
10          end if
11       end while
12       i ← i + 1
13       border_ext[j][a] ← i
14    end for
15 end for
16 return border_ext
```

For every word $u \in X$ we compute its extended border table using procedure EXTENDED_BORDERS. It contains for every proper prefix x of u and every letter $a \in A$, $|Border(x \cdot a)|$.

To compute $border\_ext[j][a] = |Border(u[0 \cdot\cdot j-1]) \cdot a|$, we need the length of $Border(u[0 \cdot\cdot j-1]) = Border(u[0 \cdot\cdot j-2] \cdot u[j-1])$. Thus $|Border(u[0 \cdot\cdot j-1])| = border\_ext[j-1][u[j-1]]$.

According to Equation (1), if $Border(u[0 \cdot\cdot i-1]) \cdot a$ is not a prefix of $u[0 \cdot\cdot i-1] \cdot a$, we need to find the longest border of the prefix of u of length i.
Since $Border(u[0 \cdot\cdot i-1]) = Border(u[0 \cdot\cdot i-2] \cdot u[i-1])$, we have $|Border(u[0 \cdot\cdot i-1])| = border\_ext[i-1][u[i-1]])$.

Fig. 3. Extended border table construction algorithm.

4.4 Determinization Algorithm

Let $\mathcal{C}_X = (A, Q, \{i\}, \{\varepsilon\}, \delta)$ be the co-deterministic automaton recognizing the language A^*X as constructed in Section 3, where $X = \{u_1, \cdots, u_m\}$ is a set of m non-empty words whose sum of lengths is n. We denote by $\mathcal{B}_X$ the accessible part of the automaton $(A, I_\mathcal{B}, Q_\mathcal{B}, F_\mathcal{B}, \delta_\mathcal{B})$, where $Q_\mathcal{B} = Q \setminus \{\varepsilon\}^m \times \{\varepsilon, \notin\}$, $I_\mathcal{B} = (i, \cdots, i, \notin)$ and for all $P \in F_\mathcal{B}$, $P = (v_1, v_2, \cdots, v_m, \varepsilon)$, where $v_r \in Q \setminus \{\varepsilon\}$ for all $r \in \{1, \cdots, m\}$. Given a state $P \in Q_\mathcal{B}$ and a letter $a \in A$ we use Algorithm 3 (see Figure 5) to compute $\delta_\mathcal{B}(P, a)$. Note that the automaton $\mathcal{B}_X$ is complete.

Theorem 1. *$\mathcal{B}_X$ is the minimal automaton recognizing A^*X.*

Proof. (Sketch) The idea of the proof is to show that $\mathcal{B}_X$ and the automaton produced by the classical subset construction are isomorphic.

Denote by $\mathcal{M} = (A, Q_\mathcal{M}, I_\mathcal{M}, F_\mathcal{M}, \delta_\mathcal{M})$ the minimal automaton built by the subset construction. Knowing a state $P \in Q_\mathcal{B}$ (resp. $P \in Q_\mathcal{M}$) and the smallest word u (smallest by length, and if two words have the same length we compare them lexicographically) such that $\delta_\mathcal{B}(I_\mathcal{B}, u) = P$ (resp. $\delta_\mathcal{M}(I_\mathcal{M}, u) = P$) we construct the unique corresponding state R in $\mathcal{M}$ (resp. in $\mathcal{B}_X$).

Finally, determinizing a co-deterministic automaton produces the minimal automaton recognizing the same language (see [5]). □

Example 2. Algorithm 3 produces the automaton depicted in Figure 6 that is the minimal automaton recognizing A^*X, where $X = \{aa, bb, aaab\}$.

Algorithm 2 FAILURE_FUNCTION

Inputs: $u \in X$, p proper suffix of u, $a \in A$

```
1  if p[0] = a and |p| > 1 then
2     return p
3  end if
4  j ← border_ext[|u| − |p|][a]
5  if j ≤ 1 then
6     return i
7  end if
8  return x[j − 1··|u| − 1]
```

Let v be the prefix of u such that $u = v \cdot p$.
If $\delta(p, a)$ is defined and different than ε then $f(u, p, a) = p$.
If $|Border(v \cdot a)| = 0$ then $f(u, p, a) = i$, where i is the unique initial state of the co-deterministic automaton $\mathcal{A}$ recognizing A^*X (see Section 3).
If $|Border(v \cdot a)| > 1$ then $Border(v \cdot a) = w \cdot a$, with $w \in A^*$. If $w = \varepsilon$ then $f(u, p, a) = i$.
Otherwise, $f(u, p, a) = q$, with $Border(v \cdot a) = w_1 \cdot a$ and $u = w_1 \cdot q$.

Fig. 4. Failure function.

5 Complexity

We analyze the complexity of our construction of the minimal automaton. The co-deterministic automaton $\mathcal{C}_X$ of size at most $n - m + 2$ recognizing A^*X is built in time $\mathcal{O}(n)$, where X is a set of m words whose sum of lengths is n. As stated before, the analysis is done for a fix m, when n tends toward infinity.

Minimizing $\mathcal{C}_X$ produces an automaton $\mathcal{B}_X$ whose number of states is linear in n and our determinization process creates only states labelled with sequences of $m+1$ elements. During this process we use sparse lists to manage these states.

Sparse lists. Let $g : E \to F$ be a partial function where E is a finite set. Denote by $Dom(g)$ the domain of g. A sparse list (see [2][Exercise 2.12 p.71] or [6][Exercise 1.15 p.55]) is a data structure one can use to implement g and perform the following operations in constant time: initialize g with $Dom(g) = \emptyset$, set a value $g(x)$ for a given $x \in E$; test whether $g(x)$ is defined or not; find the value for $g(x)$ if it is defined; remove x from $Dom(g)$.

Since the states we build are labelled by sequences of size $m + 1$, and each of the first m members is either i or a proper suffix of the corresponding word in the pattern, we use a sort of tree of sparse lists to store our states. Let $X = \{u_1, \cdots, u_m\}$ be the pattern and denote by $\text{Suff}(u_r)$ the set of all proper suffixes of u_r for $1 \leq r \leq m$. We define a partial function g on $\{0, \cdots, |u_1| - 1\}$ whose values are partial functions $g(|v_1|)$ for $v_1 \in \text{Suff}(u_1) \cup \{i\}$, with i the unique initial state of the automaton $\mathcal{A}_X$. We consider that $|i| = 0$. These functions $g(v_1)$ are defined on $\{0, \cdots, |u_2 - 1|\}$ and their values are again partial functions we denote by $g(|v_1|, |v_2|)$ for $v_1 \in \text{Suff}(u_1) \cup \{i\}$ and $v_2 \in \text{Suff}(u_2) \cup \{i\}$. By extention we build functions $g(|v_1|, |v_2|, \cdots, |v_m|) : \{0, \cdots, |u_1| - 1\} \times \{0, \cdots, |u_2| - 1\} \times \cdots \times \{0, \cdots, |u_m| - 1\} \times \{\varepsilon, \not\in\} \to Q_{\mathcal{B}}$ where $v_1 \in \text{Suff}(u_1) \cup \{i\}$, $v_2 \in \text{Suff}(u_2) \cup \{i\}$, $\cdots$, $v_m \in \text{Suff}(u_m) \cup \{i\}$ and $Q_{\mathcal{B}}$ is the set of states in the automaton $\mathcal{B}_X$. That way for a given state $P = (v_1, v_2, \cdots, v_m, j) \in Q_{\mathcal{B}}$, $g(|v_1|, |v_2|, \cdots, |v_m|, j) = P$.

When inserting a state $P = (v_1, v_2, \cdots, v_m, j)$ into our data structure, the existence of $g(|v_1|)$ is tested and if it does not exist a sparse list representing

Algorithm 3 TRANSITION_FUNCTION

Inputs: $P=(v_1, v_2, \cdots v_m, j) \in Q_{\mathcal{B}}, a \in A$

```
1  j' ← ∉
2  for r ∈ {1, ··· m} do
3     v'_r ← i
4     if δ(v_r, a) = ε then
5        j' ← ε
6     end if
7  end for
8  for ℓ = 1 to m do
9     v_ℓ ← f(u_ℓ, v_ℓ, a)
10    if v_ℓ ≠ i then
11       if v'_ℓ = i or |δ(v_ℓ, a)| < |v'_ℓ| then
12          v'_ℓ ← δ(v_ℓ, a)
13       end if
14    else
15       for r = 1 to s such that x_r ≠ ε do
16          if v'_t = i or |x_r| < |v'_t| then
17             v'_t ← x_r
18          end if
19       end for
20    end if
21 end for
22 return R = (v'_1, v'_2, ···, v'_m, j')
```

We initialize the first m elements of R to the unique initial state i in $\mathcal{A}$. The value of the last term of R is calculated (marking the state as final or non-final).

For each member v_ℓ we check the value of the failure function $f(u_\ell, v_\ell, a)$. If $f(u_\ell, v_\ell, a) \neq i$ then $|\delta(f(u_\ell, v_\ell, a), a)| = 1$ and we have found a potential value for v'_ℓ that is a suffix of $u_l \in X$. It remains to compare it to the already existing one and store the smallest in length different than i.

When the initial state i is reached, we are at the beginning of all the words in X. We define variables used in lines 15-17 as follows. From the definition of the automaton $\mathcal{A}$, $\delta(i, a) = \{x_1, x_2, \cdots, x_s\}$ where $0 \leq s \leq m$ and $a \cdot x_1 \in X, \cdots, a \cdot x_s \in X$. For every couple of integers $(r_1, r_2) \in \{1, \cdots, s\}^2$ such that $r_1 \neq r_2$, $a \cdot x_{r_1} \neq a \cdot x_{r_2}$. For all $r \in \{1, \cdots, s\}$ there exists a unique $t \in \{1, \cdots, m\}$ such that $a \cdot x_r = u_t \in X$.

Fig. 5. Transition function.

this partial function is created. Then the existence of $g(|v_1|)(|v_2|)$ is checked. The same process is repeated until a function $g(|v_1|, |v_2|, \cdots, |v_m|)$ is found. We test whether $g(|v_1|, |v_2|, \cdots, |v_m|)(j)$ is defined, and if not the value for $g(|v_1|, |v_2|, \cdots, |v_m|)(j)$ is set to P.

Testing the existence of a state works in the same way, but if a partial function is not found then this state is not in the data structure.

Proposition 3. *When using sparse lists, the construction of the minimal automaton recognizing A^*X runs in time $\mathcal{O}(n)$ and requires $\mathcal{O}(n^2)$ space where n is the length of the pattern X.*

Proof. The proof is based on the fact that the determinization produces the minimal automaton whose number of states is at most $n+1$. Since each state requires $m+1$ sparse lists of size $|u_1|, |u_2|, \cdots |u_m|, 2$, the total space complexity is quadratic in n. The time complexity of the determinization is linear in n as searching and inserting a state takes $\mathcal{O}(1)$ time, as m is a fixed integer. □

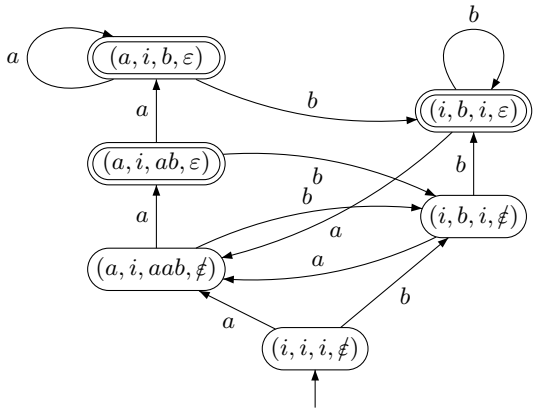

Fig. 6. Minimal automaton recognizing A^*X (by our construction).

Remark 1. In practice a hash table can be used to store these states. Under the hypothesis of a simple uniform hashing the average time and space complexities of the determinization process are $\mathcal{O}(n)$.

The natural continuation of this work is to investigate constructions based on Brzozowski's algorithm in the case where m is not fixed anymore.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
3. Marie-Pierre Béal and Maxime Crochemore. Minimizing local automata. In G. Caire and M. Fossorier, editors, *IEEE International Symposium on Information Theory (ISIT'07)*, number 07CH37924C, pages 1376–1380. IEEE Catalog, 2007.
4. R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):62–72, 1977.
5. J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*, pages 529–561. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962. Volume 12 of MRI Symposia Series.
6. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007. 392 pages.
7. M. Crochemore and W. Rytter. *Jewels of Stringology*. World Scientific Publishing Company, 2002.
8. J. 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. J. E. Hopcroft and J. D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
10. D. E. Knuth, J.H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.
11. M. Mohri. String-matching with automata. *Nordic Journal of Computing*, 4(2):217–231, Summer 1997.