

Compact and Fast Algorithms for Safe Regular Expression Search

J.-M. Champarnaud*, F. Coulon, and T. Paranthoën

LIFAR, University of Rouen, 76821 Mont-Saint-Aignan Cedex, France
{fcoulon,paranth}@dir.univ-rouen.fr

Abstract. This paper describes an improvement of the brute force determinization algorithm in the case of homogeneous NFAs, as well as its application to pattern matching. Brute force determinization with limited memory may provide a partially determinized automaton, but its bounded complexity makes it be a safe procedure contrary to the classical subset construction. Actually, our algorithm is inspired both by recent results of Champarnaud concerning the subset automaton of a homogeneous NFA and by the algorithm recently designed by Navarro and Raffinot to implement the brute force determinization of the Glushkov NFA of a regular pattern. Our algorithm significantly improves Navarro-Raffinot's one since it has an average exponentially smaller memory requirement for a given level of determinization, which, considering a bounded memory, implies a quadratically smaller parsing time. This algorithm has been implemented in CCP software (<http://www.univ-rouen.fr/LIFAR/aia/ccp.html>). Tests have been carried out in the field of text processing and biology. Experimental results are reported.

Keywords: NFA determinization; homogeneous NFAs; bitwise techniques; regular expression search

C. R. Categories: F.1.1, F4.3, I.2.7, I.2.3, I5, B7.1

1 Introduction

The aim of this article is to investigate the search of regular expressions in a text by means of homogeneous NFAs. Classical algorithms which transform a regular expression into an automaton give rise to an NFA. Then the parse of the text can be carried out either by a straightforward simulation of the NFA, which is expensive in time, or by first determinizing the NFA. However, the classical subset determinization algorithm (based on reachability) may cause a memory overflow or a long preprocessing time in some cases.

Compared with the reachability-based algorithm, the brute force determinization algorithm has two main advantages. First, it has an $\mathcal{O}(n2^n)$ time complexity

* Corresponding author. e-mail: Jean-Marc.Champarnaud@univ-rouen.fr

where n is the number of states of the NFA, while the classical subset algorithm has an $\mathcal{O}(n^2 2^n)$ time complexity [3]. Secondly, the space and time complexity of the brute force determinization can be bounded simply by allowing determinization to be partial, which makes such an implementation be a safe one. Indeed, a good brute force algorithm should deal between a partial and a full determinization, depending on the size of the NFA and of the memory space allowed. Of course, a partial determinization leads to a parse which is slower than the classical deterministic one.

In this paper, we investigate the particular case of homogeneous NFAs, and especially the Glushkov NFA that is devoted to regular expression recognition. Glushkov NFAs have been characterized in terms of graphs in [2]. These automata have some specific properties, such as being homogeneous, which can be exploited for memory storage and to efficiently implement some operations, as word recognition and determinization. These aspects have been detailed in [9, 3].

Homogeneous NFAs may be determinized by a modified brute force algorithm that needs a quite lower amount of memory. A first algorithm is given by Navarro and Raffinot [7]; it is based on the state partitioning associated with any homogeneous NFA [3], and it allows to divide the memory space by the size of the alphabet. The algorithm we describe in this paper is based on the additional property saying that a state of the subset automaton of a homogeneous NFA is necessarily a homogeneous subset [3, 4]. It leads to an expected exponential reduction of the space, and thus it enables us to process a given expression with a higher degree of determinization, which implies a lower parsing time.

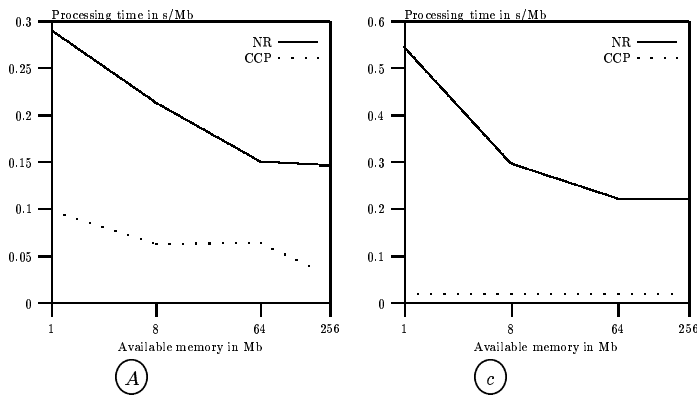


Fig. 1. Comparison of algorithms NR and CCP.

In order to illustrate the significant improvement our algorithm leads to, we have selected two tests (Figure 1) among the set of runs which have been performed using CCP software [6] (see Section 6 for a detailed description of

experimental results). They compare the processing time respectively of the algorithm of Navarro and Raffinot (NR) and of our algorithm (CCP) depending on the amount of available memory. In the graph A the expression is " $A^{14} \cdot C^{14} \cdot G^{14} \cdot T^{14}$ " and the text is a DNA sequence of size 100MB. In the graph c the expression is " $\sqcup.([Aa]lice+[Ww]hich+[Cc]aterpillar+[Ss]ize+[Ss]peak+[Mm]ushroom+[Hh]owever+[Uu]nderneath+[Jj]ust+[Pp]igeon+[Ll]isten)^*$ " and the text is a string of size 100MB built from "Alice's adventures in wonderland" book.

Section 2 gives the definitions and notation used in this paper. Section 3 is devoted to the brute force determinization. The bit-automaton of an unrestricted NFA is defined as the bitwise representation of its subset automaton. For a homogeneous NFA we introduce the notion of homogeneous subset automaton and we compare the complexity of its bitwise representation (called homogeneous bit-automaton) to the one used by Navarro and Raffinot. Section 4 presents the concept of partial determinization which leads to the definition of the block automaton of an unrestricted NFA w.r.t. a partitioning of its state set. A block bit-automaton is defined as the bitwise representation of a block automaton. The bitwise technique of horizontal table splitting turns out to be a specific case of partial determinization in the sense that it constructs a block bit-automaton. For a homogeneous NFA we introduce the notion of homogeneous block automaton and we compare the complexity of its bitwise representation (called homogeneous block bit-automaton) to the one used by Navarro and Raffinot. Section 5 enlightens the interest of these different bit-automaton constructions when applied to the case of Glushkov NFAs. It provides a discussion about the complexity of Navarro-Raffinot pattern matching method and of the new one we present here. Finally, Section 6 provides a detailed description of experimental results obtained from tests we have carried out in the fields of text processing and biology.

2 Definitions and notation

We first recall some definitions concerning finite automata and bitwise set representations. For further details about these topics we refer to [12] and [8].

2.1 Automata

An automaton is a 5-tuple $A = \langle Q, \Sigma, \delta, I, F \rangle$ where Q is a finite set of *states*, Σ is the *alphabet*, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the *transition function*, I (resp. F) is a subset of Q whose elements are the *initial states* (resp. *final states*). A *path* in the automaton A is a sequence $(q_k, a_k, q_{k+1})_{k \in [0, \dots, l-1]}$ where $(q_k, a_k, q_{k+1}) \in Q \times \Sigma \times Q$ and $q_{k+1} \in \delta(q_k, a_k)$ for all $0 \leq k < l$. Such a path is said to be *successful* if $q_0 \in I$ and $q_l \in F$. The word $a_0 a_1 \dots a_{l-1}$ is the *label* of the path. The language *recognized* by A is the set of labels of successful paths. An automaton A is *deterministic* if it has a unique initial state and if for each state $s \in Q$ and for each symbol $x \in \Sigma$, there is at most one transition labeled by x and outgoing from state s . A DFA is a deterministic automaton; an NFA is

a nondeterministic automaton. An NFA is said to be *non-returning* if it has a unique initial state without incoming transition. In the following, NFA stands for non-returning NFA. By convention a set I restricted to a unique state q_0 is denoted by q_0 instead of $\{q_0\}$.

2.2 Bitwise representation of integer sets

Let n be a non-zero integer and $Q = \llbracket 0, n - 1 \rrbracket$ be a finite integer set. We introduce the classical bitwise representation of integer subsets. Let \vee and \wedge respectively denote the “or” and “and” bitwise operators on integers. A subset P of Q can be represented by the integer \dot{P} , called the *bit-mask* associated with P . For any $0 \leq k < n$, \dot{P}_k denotes the bit of rank k of \dot{P} , and \dot{P} is defined by $\dot{P}_k = 1 \Leftrightarrow k \in P$. For any $q \in Q$, the bit-mask associated with $\{q\}$ is denoted by \dot{q} . Notice that q and \dot{q} are two different integers. Indeed, $\dot{q} = 2^q$.

Definition 1 Let $Q = \llbracket 0, n - 1 \rrbracket$. Let Q' be a subset of Q and let p be an element of Q . The projection of p onto Q' is the set $\pi(p, Q')$ which satisfies:

1. If $p \notin Q'$, $\pi(p, Q') = \emptyset$,
2. If $p \in Q'$, $\pi(p, Q')$ is the set whose unique element is the rank of p in Q' w.r.t. the natural integer ordering, starting from the rank 0.

We extend π such that for all $P \subseteq Q$, $\pi(P, Q') = \bigcup_{p \in P} \pi(p, Q')$.

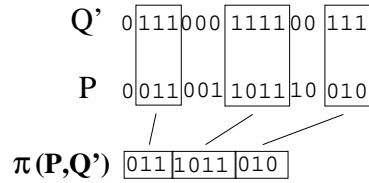


Fig. 2. Let $Q = \llbracket 0, 15 \rrbracket$, $Q' = \{1, 2, 3, 7, 8, 9, 10, 13, 14, 15\}$, and $P = \{2, 3, 6, 7, 9, 10, 11, 14\}$. We have: $P \cap Q' = \{2, 3, 7, 9, 10, 14\}$ and $\pi(P, Q') = \{1, 2, 3, 5, 6, 8\}$. Hence the bit-masks \dot{Q}' , \dot{P} and $\dot{\pi(P, Q')}$ where the least significant bit is the leftmost one.

Let Q' be a subset of Q . The function $P \mapsto \pi(P, Q')$ defined for all $P \subseteq Q'$ is injective, hence we introduce its reverse function $P \mapsto \pi^{-1}(P, Q')$. Notice that for all subsets P and Q' , we have $\pi^{-1}(\pi(P, Q'), Q') = P \cap Q'$. Let P and Q' be subsets of Q . We have an associated bitwise projection $\dot{\pi}$ such that $\dot{\pi}(\dot{P}, \dot{Q}')$ is the binary representation of $\pi(P, Q')$. In the following, we will note π_a instead of $\pi(\cdot, Q_a)$, where a is in some set X . Similarly we will note $\dot{\pi}_a$ instead of $\dot{\pi}_a(\cdot, \dot{Q}_a)$.

In this paper, the *parse* of a text via an automaton A is the computation of the set of last positions of occurrences of words recognized by A . It can be carried

out by adding loop-back arrows on initial states: (q, a, q) for all $q \in I$ and $a \in \Sigma$. A position is detected when the current state of A is final. Notice that such a parse does not really make sense if A recognizes the empty word. Let us recall that parsing a text of length l with a DFA has an $\Omega(l)$ time complexity, while parsing it with an unrestricted n -state NFA has an $\mathcal{O}(ln^2)$ time complexity.

3 Full determinization

This section deals with the classical process of determinization, that is the subset construction, and more particularly with its brute force implementation. The case of homogeneous NFAs is examined and it is shown how to reduce the space complexity of the basic version of Navarro-Raffinot's algorithm. In the whole of this section, we let $Q = \llbracket 0, n - 1 \rrbracket$. Hence $\dot{Q} = 2^n - 1$.

3.1 The subset automaton and the bit-automaton

Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be a non-returning NFA and $\mathcal{A} = \langle 2^Q, \Sigma, \Delta, 0_A, \mathcal{F} \rangle$ be the subset automaton of A . We define the bit-automaton of an NFA A as a bitwise encoding of the subset automaton of A . Such a representation is also currently referred to as bit-parallelism technique.

Definition 2 *Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be an NFA. The bit-automaton associated with A is the DFA $\dot{A} = \langle \llbracket 0, \dot{Q} \rrbracket, \Sigma, \dot{\Delta}, \dot{0}_A, \dot{F} \rangle$, where $\dot{\Delta}$ is the function from $\llbracket 0, \dot{Q} \rrbracket \times \Sigma$ onto $\llbracket 0, \dot{Q} \rrbracket$ such that $\dot{\Delta}(\dot{P}, a)$ is the bit-mask associated with $\Delta(P, a)$ for all $P \subseteq Q$ and $a \in \Sigma$.*

The function $\dot{\Delta}$ can be stored in a set of tables $(T[a])_{a \in \Sigma}$ such that $T[a][\dot{P}] = \dot{\Delta}(\dot{P}, a)$ for all $a \in \Sigma$ and $P \subseteq Q$. This is the elementary bitwise representation of an unrestricted NFA. Some variants will be introduced in the following, such as homogeneous bit-automata. For each one of them, we will express the space complexity as the memory needed to store the transition function, neglecting the space for initial and final states. Thus, the space complexity of the bit-automaton of an unrestricted n -state NFA is $|\Sigma|n2^n$ bits.

3.2 Case of homogeneous automata

We follow the definitions given in [3] about homogeneous automata.

Definition 3 *An automaton $A = \langle Q, \Sigma, \delta, 0, F \rangle$ is homogeneous if and only if, for all states q in Q , all transitions entering q are labeled by the same symbol:*

$$(\forall p, q \in Q) (\forall a, b \in \Sigma) \quad a \neq b \Rightarrow \delta(p, a) \cap \delta(q, b) = \emptyset$$

A state q is said to be labeled by a if every transition entering q is labeled by a . The initial state 0 which has no incoming transition is labeled by an arbitrary symbol α which does not belong to Σ .

Definition 4 Let A be a homogeneous NFA. For all $a \in \Sigma \cup \{\alpha\}$ we denote by Q_a the subset of Q that contains all the states labeled by a .

We now gather some properties of homogeneous automata that can be found in [3]:

Proposition 1 Let A be a homogeneous NFA and \mathcal{A} its subset automaton.

1. The set $(Q_a)_{a \in \Sigma \cup \{\alpha\}}$ is a partitioning of Q , which is called the symbol-partitioning of A in the following.
2. We have: $\forall P \in 2^Q, \forall a \in \Sigma, \Delta(P, a) \subseteq Q_a$.
3. For every reachable state $P \in 2^Q$ of the subset automaton \mathcal{A} , there exists $a \in \Sigma \cup \{\alpha\}$ such that $P \subseteq Q_a$.
4. The subset automaton of a homogeneous NFA is a homogeneous DFA.

A first improvement on the bit-automaton The Navarro-Raffinot method [7] reduces the space complexity of the bit-automaton when A is homogeneous. Instead of the set of tables $(T[a])_{a \in \Sigma}$ described in Section 3.1, a unique table N is used, defined for all $P \subseteq Q$ by:

$$N[\dot{P}] = \bigvee_{b \in \Sigma} \dot{\Delta}(\dot{P}, b)$$

Proposition 2 Let $\dot{A} = \langle [0, \dot{Q}], \Sigma, \dot{\Delta}, \dot{0}_A, \dot{F} \rangle$ be the bit-automaton associated with A . The table N defines the transition function $\dot{\Delta}$ in the sense that for all $P \subseteq Q$ and for all $a \in \Sigma$,

$$\dot{\Delta}(\dot{P}, a) = N[\dot{P}] \wedge \dot{Q}_a$$

Proof — By Proposition 1.1 and Proposition 1.2, the image of a set by a symbol a belongs to the subset Q_a , and the sets $(Q_a)_{a \in \Sigma}$ are pairwise disjoint. Hence:

$$\begin{aligned} & (\forall a, b \in \Sigma \mid a \neq b) (\forall P \in 2^Q) \quad \Delta(P, b) \cap Q_a = \emptyset \\ & (\forall a \in \Sigma) (\forall P \in 2^Q) \quad \left(\bigcup_{b \in \Sigma} \Delta(P, b) \right) \cap Q_a = \Delta(P, a) \end{aligned}$$

Hence the result. ■

The `nr` determinization algorithm consists in building the bit-automaton of a homogeneous NFA, using the table N . For an n -state NFA it needs $n2^n$ bits.

The homogeneous subset automaton By Proposition 1.3 every state of the subset automaton of a homogeneous NFA is a subset of some block Q_a in the partitioning $(Q_a)_{a \in \Sigma \cup \{\alpha\}}$. As a consequence, each state $P \subseteq Q_a$ can be represented by the pair $(a, \pi_a(P))$. Given a pair (a, P') , the associated state of the subset automaton is $\pi_a^{-1}(P')$. Thence, the image of the pair (a, P') when reading the symbol b is the pair $(b, \pi_b(\Delta(\pi_a^{-1}(P'), b)))$. For this reason, we introduce some auxiliary functions $\Delta_{a,b}$ before defining the *homogeneous subset automaton* of a homogeneous NFA:

Definition 5 For all $a \in \Sigma \cup \{\alpha\}$ and $b \in \Sigma$, let us consider the function $\Delta_{a,b} : P \mapsto \pi_b(\Delta(\pi_a^{-1}(P), b))$, defined from $2^{\llbracket 0, |Q_a| - 1 \rrbracket}$ to $2^{\llbracket 0, |Q_b| - 1 \rrbracket}$. The homogeneous subset automaton of a homogeneous NFA A is the automaton $\mathcal{A}_H = \langle \mathcal{Q}_H, \Sigma, \Delta_H, 0_H, \mathcal{F}_H \rangle$ such that:

1. $\mathcal{Q}_H = \{(a, P) \mid a \in \Sigma \cup \{\alpha\}, P \subseteq \llbracket 0, |Q_a| - 1 \rrbracket\}$,
2. $(\forall (a, P) \in \mathcal{Q}_H) (\forall b \in \Sigma) \Delta_H((a, P), b) = (b, \Delta_{a,b}(P))$,
3. $0_H = (\alpha, \{0\})$,
4. $\mathcal{F}_H = \{(a, P) \in \mathcal{Q}_H \mid \pi_a^{-1}(P) \cap F \neq \emptyset\}$.

Proposition 3 The homogeneous subset automaton \mathcal{A}_H is deterministic and equivalent to A . The number of states of \mathcal{A}_H is equal to $2 + \sum_{a \in \Sigma} 2^{|Q_a|}$.

Proof — The number of states is indeed equal to $\sum_{a \in \Sigma \cup \{\alpha\}} 2^{|Q_a|}$. We just prove that \mathcal{A}_H is equivalent to A . The rest is trivial.

Let $\mathcal{A} = \langle 2^Q, \Sigma, \Delta, 0_A, \mathcal{F} \rangle$ be the subset automaton of A , and let $\mathcal{A}_H = \langle \mathcal{Q}_H, \Sigma, \Delta_H, 0_H, \mathcal{F}_H \rangle$ be the homogeneous subset automaton of A .

Let $\Phi : \mathcal{Q}_H \mapsto 2^Q$ be the function defined for all $(a, P) \in \mathcal{Q}_H$ by $\Phi(a, P) = \pi_a^{-1}(P)$. We shall prove that Φ is an isomorphism from \mathcal{A}_H onto \mathcal{A} .

Let $(a, P) \xrightarrow{b} (b, P')$ be a transition of \mathcal{A}_H . We have $P' = \pi_b(\Delta(\pi_a^{-1}(P), b))$, that is, $P' = \pi_b(\Delta(\Phi(a, P), b))$. Since $\Delta(\Phi(a, P), b) \subseteq Q_b$, we have $\pi_b^{-1}(P') = \Delta(\Phi(a, P), b)$, that is, $\Phi(b, P') = \Delta(\Phi(a, P), b)$. Hence we have proved that the transition $\Phi(a, P) \xrightarrow{b} \Phi(b, P')$ exists.

Now, Φ is trivially an injective homomorphism. The surjectivity is due to the fact that for each state P of \mathcal{A} which is contained in a Q_a ($a \in \Sigma \cup \{0\}$), we have, $P = \Phi(a, \pi_a(P))$.

Since \mathcal{A} and \mathcal{A}_H are deterministic and complete, we have proved that Φ is an isomorphism. Hence, \mathcal{A}_H is equivalent to A . ■

The homogeneous bit-automaton The notion of homogeneous subset automaton leads to a new bit-automaton, which is more compact than the one used in nr determinization.

Definition 6 Let $\mathcal{A}_H = \langle \mathcal{Q}_H, \Sigma, \Delta_H, 0_H, \mathcal{F}_H \rangle$ be the homogeneous subset automaton of a homogeneous NFA A . The homogeneous bit-automaton associated with A is the automaton $\dot{\mathcal{A}}_H = \langle \dot{\mathcal{Q}}_H, \Sigma, \dot{\Delta}_H, \dot{0}_H, \dot{\mathcal{F}}_H \rangle$ such that:

1. $\dot{\mathcal{Q}}_H = \{(a, \dot{P}) \mid (a, P) \in \mathcal{Q}_H\}$,
2. $\dot{\Delta}_H((a, \dot{P}), b) = (b, \dot{P}')$, where $(b, P') = \Delta_H((a, P), b)$,
3. $\dot{0}_H = (\alpha, \dot{0})$,
4. $\dot{\mathcal{F}}_H = \{(a, \dot{P}) \mid (a, P) \in \mathcal{F}_H\}$.

This definition is coherent w.r.t. the definition of $\hat{\pi}$ since we have:

$$\dot{\Delta}_H((a, \dot{P}), b) = (b, \hat{\pi}_b(\dot{\Delta}(\hat{\pi}_a^{-1}(\dot{P}), b)))$$

The function $\dot{\Delta}_H$ can be stored in $|\Sigma|(|\Sigma| + 1)$ tables $(T[a][b])_{a \in \Sigma \cup \alpha, b \in \Sigma}$, such that:

$$(\forall a \in \Sigma \cup \alpha, \forall b \in \Sigma)(\forall P \subseteq \llbracket 0, |Q_a| - 1 \rrbracket) \quad T[a][b][\dot{P}] = \dot{\Delta}_{a,b}(\dot{P})$$

The ccp determinization algorithm consists in building the homogeneous bit-automaton of a homogeneous NFA, using the tables $(T[a][b])_{a \in \Sigma \cup \alpha, b \in \Sigma}$. For an n -state NFA it needs $n(2 + \sum_{a \in \Sigma} 2^{|Q_a|})$ bits.

4 Partial determinization

The brute force determinization algorithm has an exponential space complexity. Bit-parallelism implementations usually solve possible space problems by using the horizontal table splitting technique introduced by Wu and Manber [11]. This technique turns out to be a partial determinization procedure. We describe it in this more general framework, first in the case of unrestricted automata and secondly in the case of homogeneous automata.

4.1 The block automaton and the block bit-automaton

Definition 7 Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be an NFA and $B = (Q_i)_{i=0, \dots, k-1}$ a partitioning of Q . For all $i, j \in \llbracket 0, k-1 \rrbracket$, let the function $\Delta_{i,j} : (P, a) \mapsto \pi(\Delta(\pi^{-1}(P, Q_i), a), Q_j)$, be defined from $2^{\llbracket 0, |Q_i| - 1 \rrbracket} \times \Sigma$ to $2^{\llbracket 0, |Q_j| - 1 \rrbracket}$. The block automaton of A w.r.t. B is the NFA $\mathcal{A}_B = \langle \mathcal{Q}_B, \Sigma, \Delta_B, 0_B, \mathcal{F}_B \rangle$ defined by:

1. $\mathcal{Q}_B = \{(i, P) \mid i \in \llbracket 0, k-1 \rrbracket, P \subseteq \llbracket 0, |Q_i| - 1 \rrbracket\}$,
2. $\Delta_B((i, P), a) = \{(j, \Delta_{i,j}(P, a)) \mid j \in \llbracket 0, k-1 \rrbracket\}$,
3. $0_B = (i, \pi(0_A, Q_i))$, where i is the unique integer such that $0_A \in Q_i$,
4. $\mathcal{F}_B = \{(i, P) \in \mathcal{Q}_B \mid i \in \llbracket 0, k-1 \rrbracket, \pi^{-1}(P, Q_i) \cap F \neq \emptyset\}$.

The Figure 3 gives an example of block automaton. Notice that the block automaton of A w.r.t. the trivial partitioning with n blocks (resp. 1 block) is A itself (resp. the subset automaton of A). Furthermore, the number of states of a block automaton w.r.t. a partitioning $(Q_i)_{i=0, \dots, k-1}$ is equal to $\sum_{i=0}^{k-1} 2^{|Q_i|}$.

For all states (i, P) of \mathcal{A}_B and all symbols a of Σ , there exist at most k transitions outgoing from (i, P) and labeled with a . Thus the time complexity of parsing with a block automaton is given by the following proposition.

Proposition 4 Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be an n -state NFA and consider a partitioning $B = (Q_i)_{i=0, \dots, k-1}$ of Q . Parsing a text of length l with the block automaton \mathcal{A}_B of A w.r.t. the partitioning B has an $O(lkn)$ time complexity.

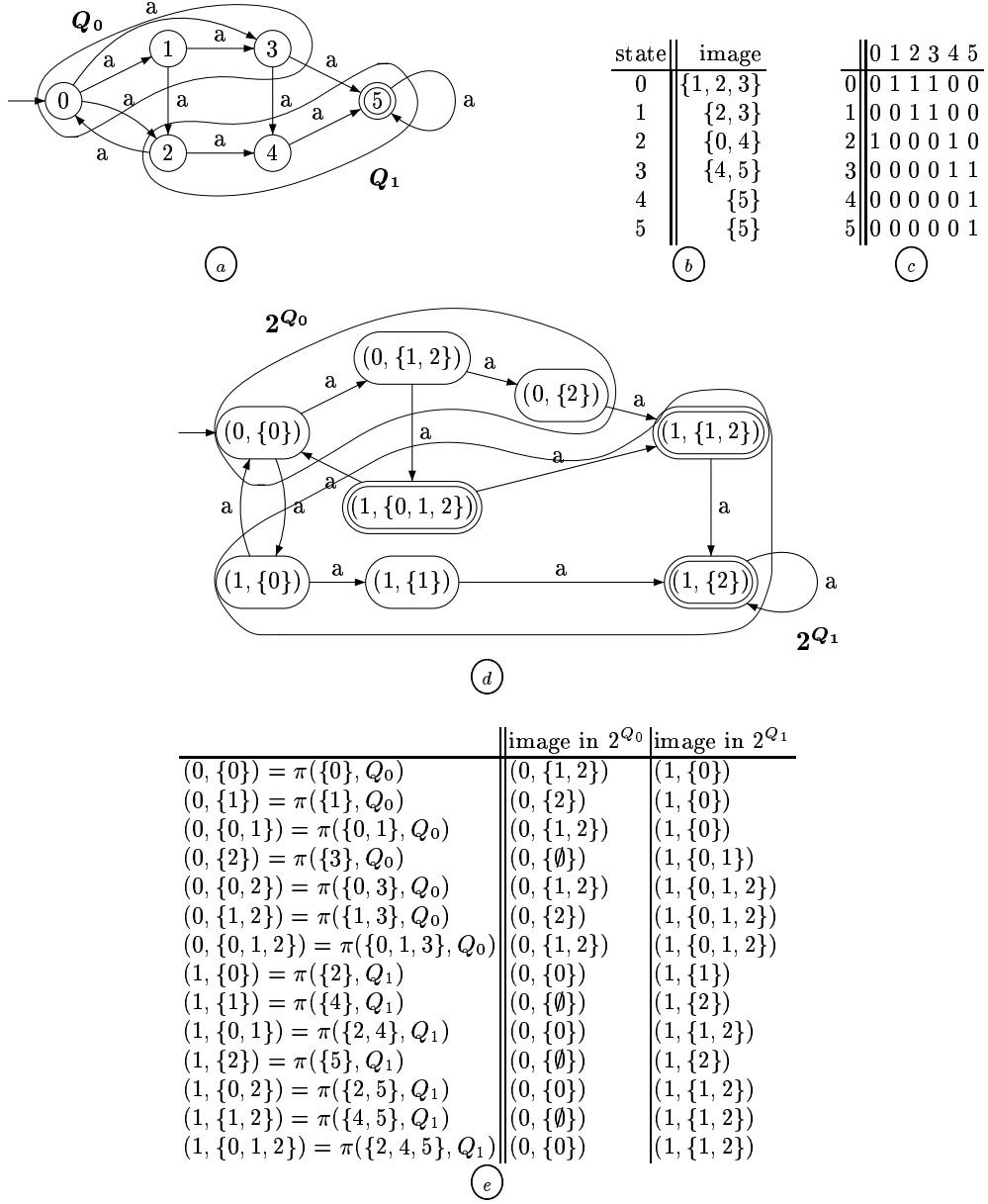


Fig. 3. Figures a, b and c respectively represent the state graph, the transition function and the bit transition function of a non-deterministic automaton A . Figure d represents the state graph of the accessible part of the block automaton \mathcal{A}_B of A w.r.t. the partitioning $B = \{\{0, 1, 3\}, \{2, 4, 5\}\} = \{Q_0, Q_1\}$. Figure e represents the transition function of \mathcal{A}_B .

Sketch of the Proof — Let $k_i = |Q_i|$ for all $i \in \llbracket 0, k-1 \rrbracket$. For all $P \subseteq Q$, we let P_i stands for $\pi(P, Q_i)$. Each subset P is bijectively associated with the sequence $(P_i)_{0 \leq i < k}$ which is considered as a representative of the subset P during the simulation of the subset automaton \mathcal{A} . Let $a \in \Sigma$, the state $\Delta(P, a) = P'$ represented by $(P'_i)_{0 \leq i < k}$, can be computed in the following way: for all i ,

$$P'_i = \bigcup_{0 \leq j < k} \Delta_{j,i}(P_j, a)$$

Since $\Delta_{j,i}(P_j, a)$ is a set whose cardinal is k_i , $\Delta(P, a)$ can be computed in time $\sum_{0 \leq i < k} \sum_{0 \leq j < k} k_i$, that is, kn . Hence, a text of length l can be processed in time $l\bar{n}k$. ■

Definition 8 Let $\mathcal{A}_B = \langle Q_B, \Sigma, \Delta_B, 0_B, \mathcal{F}_B \rangle$ be the block automaton associated with $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ w.r.t. the partitioning $B = (Q_i)_{i=0, \dots, k-1}$ of Q . The block bit-automaton of \mathcal{A}_B is the NFA $\dot{\mathcal{A}}_B = \langle \dot{Q}_B, \Sigma, \dot{\Delta}_B, \dot{0}_B, \dot{\mathcal{F}}_B \rangle$ defined by:

1. $\dot{Q}_B = \{(i, \dot{P}) \mid (i, P) \in Q_B\}$,
2. $\dot{\Delta}_B((i, \dot{P}), a) = \{(j, \dot{P}') \mid (j, P') \in \Delta_B((i, P), a)\}$,
3. $\dot{0}_B = (i, \dot{P})$, where $0_B = (i, P)$,
4. $\dot{\mathcal{F}}_B = \{(i, \dot{P}) \mid \dot{\pi}^{-1}(\dot{P}, \dot{Q}_i) \wedge \dot{P} \neq 0\}$.

The function $\dot{\Delta}_B$ can be stored in $|\Sigma|k^2$ tables $T_B[a][i][j]$ with $a \in \Sigma$ and $(i, j) \in \llbracket 0, k-1 \rrbracket^2$, such that:

$$(\forall (i, P) \in Q_B) (\forall j \in \llbracket 0, k-1 \rrbracket) (\forall a \in \Sigma) \quad T_B[a][i][j][\dot{P}] = \dot{\Delta}_{i,j}(\dot{P}, a)$$

Proposition 5 The number of entries of the table $T_B[a][i][j]$ is equal to $2^{|Q_i|}$. The set of tables T_B needs an amount of $|\Sigma| \times n \sum_{i=0}^{k-1} 2^{|Q_i|}$ bits to be stored.

Proposition 6 Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be an n -state NFA and consider a partitioning $B = (Q_i)_{i=0, \dots, k-1}$ of Q . Parsing a text of length l with the block bit automaton $\dot{\mathcal{A}}_B$ of \mathcal{A}_B has an $\mathcal{O}(lk^2)$ time complexity.

Sketch of the Proof — As in the last proof, for all $P \subseteq Q$, we let \dot{P}_i stands for $\dot{\pi}(\dot{P}, \dot{Q}_i)$. Each subset P is bijectively associated with the sequence $(\dot{P}_i)_{0 \leq i < k}$ which is considered as a representative of the subset P during the simulation of the subset automaton \mathcal{A} . Let $a \in \Sigma$, the state $\Delta(P, a) = P'$ represented by $(P'_i)_{0 \leq i < k}$, can be computed in the following way: for all i ,

$$P'_i = \bigvee_{0 \leq j < k} \dot{\Delta}_{j,i}(\dot{P}_j, a)$$

As we consider that $\dot{\Delta}_{j,i}(\dot{P}_j, a)$ is an integer which can be stored in one machine word, and that the “or” Boolean operation is one elementary operation,

$\Delta(P, a)$ can be computed in time k^2 . Hence, a text of length l can be processed in time lk^2 . ■

The table splitting introduced by Wu and Manber [11] is a specific case of partial determinization where the k blocks of the partitioning have the same $\lceil n/k \rceil$ size. Thus an amount of $|\Sigma| nk2^{\lceil n/k \rceil}$ bits is required to store the transition function.

The advantage of partial determinization is that it allows to reduce drastically the space needed to store the transition tables without increasing too much the parse time. More precisely, given a space of size \mathcal{M} bits, we will use the smallest value of k such that $nk|\Sigma|2^{\lceil n/k \rceil} \leq \mathcal{M}$. We know that it leads to a parse time complexity of $\mathcal{O}(lk^2)$.

4.2 Case of homogeneous automata

Partial determinization can also be applied to homogeneous automata. Therefore the bit-automata used in the nr determinization algorithm (Section 3.2) and in the ccp determinization algorithm (Section 3.2) can be extended into block bit-automata.

The block bit-automaton for the nr algorithm Let $B = (Q_i)_{i=0, \dots, k-1}$ be a partitioning of Q . Let us recall that the nr algorithm is based on a single table N such that for all $P \subseteq Q$, $N[\dot{P}]$ stores the image of \dot{P} by all the symbols of Σ . This table N is replaced by k^2 tables $N_B[i][j]$ with $(i, j) \in \llbracket 0, k-1 \rrbracket^2$. These tables are defined by:

$$(\forall (i, P) \in \mathcal{Q}_B) (\forall j \in \llbracket 0, k-1 \rrbracket) \quad N_B[i][j][\dot{P}] = \bigvee_{b \in \Sigma} \dot{\Delta}_{i,j}(\dot{P}, b)$$

The transition function $\dot{\Delta}_B$ of the bit-automaton associated with B can be recovered in the following way:

$$(\forall (i, \dot{P}) \in \dot{\mathcal{Q}}_B) (\forall a \in \Sigma) \quad \dot{\Delta}_B((i, \dot{P}), a) = \left\{ N_B[i][j][\dot{P}] \wedge \dot{\pi}(\dot{Q}_a, \dot{Q}_j) \mid j \in \llbracket 0, k-1 \rrbracket \right\}$$

Proposition 7 *The number of entries in each table $N_B[i][j]$ is equal to $2^{|\mathcal{Q}_i|}$. The set of tables N_B needs an amount of $n \sum_{i=0}^{k-1} 2^{|\mathcal{Q}_i|}$ bits to be stored.*

The NR determinization algorithm consists in building the block bit-automaton using a set of tables N_B with a partitioning whose sets are of equal size. In this case, the set of tables N_B needs an amount of $\mathcal{O}(nk2^{\lceil n/k \rceil})$ bits to be stored, and the parse time complexity is $\mathcal{O}(lk^2)$.

The block bit-automaton for the ccp algorithm The two notions of homogeneous subset automaton (Section 3.2) and of block automaton are both defined w.r.t. a partitioning of the set of states. In fact, the homogeneous subset automaton \mathcal{A}_H of A is the block automaton associated with the symbol-partitioning of A . Moreover the automaton \mathcal{A}_H is deterministic. In order to reduce the space used by the tables, the blocks of the symbol-partitioning can be partitioned themselves.

In the following of this section, $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ is a homogeneous NFA and $(Q_a)_{a \in \Sigma \cup \{\alpha\}}$ is its symbol-partitioning. Moreover, for all a in $\Sigma \cup \{\alpha\}$, $B_a = (Q_{a,i})_{i=0, \dots, k_a-1}$ is a partitioning of Q_a into k_a blocks and $B = \bigcup_{a \in \Sigma \cup \{\alpha\}} B_a$ is the resulting partitioning of Q . For all $a \in \Sigma \cup \{\alpha\}$, $b \in \Sigma$, $i \in \llbracket 0, k_a - 1 \rrbracket$ and $j \in \llbracket 0, k_b - 1 \rrbracket$, we introduce the function $\Delta_{(a,i),(b,j)}$ defined from $2^{\llbracket 0, |Q_{a,i}| - 1 \rrbracket}$ to $2^{\llbracket 0, |Q_{b,j}| - 1 \rrbracket}$ by:

$$\Delta_{(a,i),(b,j)} : P \mapsto \pi(\Delta(\pi^{-1}(P, Q_{a,i}), b), Q_{b,j})$$

Definition 9 *The homogeneous block automaton of a homogeneous NFA A w.r.t. a partitioning B is the NFA $\mathcal{H}_S = \langle \mathcal{Q}_S, \Sigma, \Phi_S, 0_S, \mathcal{F}_S \rangle$ defined by:*

1. $\mathcal{Q}_S = \{(a, (i, P)) \mid a \in \Sigma \cup \{\alpha\}, i \in \llbracket 0, k_a - 1 \rrbracket, P \in \llbracket 0, |Q_{a,i}| - 1 \rrbracket\}$,
2. $\Phi_S \left((a, (i, P)), b \right) = \left\{ \left(b, \left(j, \Delta_{(a,i),(b,j)}(P) \right) \right) \mid j \in \llbracket 0, k_b - 1 \rrbracket \right\}$,
3. $0_S = (\alpha, (0, \{0\}))$,
4. $\mathcal{F}_S = \{(a, (i, P)) \in \mathcal{Q}_A \mid a \in \Sigma \cup \{\alpha\}, i \in \llbracket 0, k_a - 1 \rrbracket, \pi^{-1}(P, Q_{a,i}) \cap F \neq \emptyset\}$.

The number of states of the homogeneous block automaton \mathcal{H}_S is equal to:

$$2 + \sum_{a \in \Sigma} \sum_{i=0}^{k_a-1} 2^{|Q_{a,i}|}$$

Proposition 8 *We assume that A has n states. Parsing a text of length l with the homogeneous block automaton \mathcal{H}_S of A w.r.t. the partitioning B has a worst case time complexity equal to:*

$$\mathcal{O}(l \times \max\{k_a \mid a \in \Sigma\})$$

Sketch of the Proof — For all states of the homogeneous subset automaton (a, P) , we let $P_{a,i}$ stands for $\pi(P, Q_{a,i})$. Each state (a, P) is bijectively associated with the sequence $(a, (P_{a,i})_{0 \leq i < k_a})$ which is considered as a representative of the state (a, P) during the simulation of the homogeneous subset automaton \mathcal{A}_H . Let $b \in \Sigma$, the state $\Delta_H((a, P), b) = (b, P')$, represented by $(b, (P'_{b,j})_{0 \leq j < k_b})$ can be computed in the following way: for all j ,

$$P'_{b,j} = \bigvee_{0 \leq i < k_a} \Delta_{(a,i),(b,j)}(P_i, a)$$

As we consider that $\Delta_{(a,i),(b,j)}(P_i, a)$ is an integer which can be stored in one machine word, and that the “or” Boolean operation is one elementary operation, $\Delta_H((a, P), b)$ can be computed in time $k_a \cdot k_b$. Hence, a text of length l can be processed in time lower than $l \cdot \max\{k_a \mid a \in \Sigma\}$. ■

Definition 10 Let $\mathcal{H}_S = \langle \mathcal{Q}_S, \Sigma, \Phi_S, 0_S, \mathcal{F}_S \rangle$ be the homogeneous block automaton of A w.r.t. B . The homogeneous block bit-automaton of \mathcal{H}_S is the NFA $\dot{\mathcal{H}}_S = \langle \dot{\mathcal{Q}}_S, \Sigma, \dot{\Phi}_S, \dot{0}_S, \dot{\mathcal{F}}_S \rangle$ defined by:

1. $\dot{\mathcal{Q}}_S = \{(a, (i, \dot{P})) \mid (a, (i, P)) \in \mathcal{Q}_S\}$,
2. $\dot{\Phi}_S((a, (i, \dot{P})), b) = \{(b, (j, \dot{P}')) \mid (b, (j, P')) \in \Phi_S((a, (i, P)), b)\}$,
3. $\dot{0}_S = (\alpha, (0, \dot{0}))$,
4. $\dot{\mathcal{F}}_S = \{(a, (i, \dot{P})) \mid \dot{\pi}^{-1}(\dot{P}, Q_{a,i}) \wedge \dot{P} \neq 0\}$.

The transition function $\dot{\Phi}_S$ can be stored in $k^2 |\Sigma| (|\Sigma| + 1)$ tables $C_S[a][b][i][j]$ defined for all $a \in \Sigma \cup \{\alpha\}$, $b \in \Sigma$ and all $i \in \llbracket 0, k_a - 1 \rrbracket$, $j \in \llbracket 0, k_b - 1 \rrbracket$ by:

$$(\forall \dot{P} \in \llbracket 0, 2^{|\mathcal{Q}_{a,i}|} - 1 \rrbracket) \quad C_S[a][b][i][j][\dot{P}] = \dot{\Delta}_{(a,i),(b,j)}(\dot{P})$$

Proposition 9 The number of entries in the table $C_S[a][b][i][j]$ is equal to $2^{|\mathcal{Q}_{a,i}|}$. The set of tables C_S needs an amount of $n(2 + \sum_{a \in \Sigma} \sum_{i=0}^{k_a-1} 2^{|\mathcal{Q}_{a,i}|})$ bits to be stored.

The CCP determinization algorithm consists in building the homogeneous block bit-automaton using the set of tables C_S . The coefficients $(k_a)_{a \in \Sigma}$ are previously computed according to the available memory.

5 Glushkov NFAs and bitwise regular expression search

We have presented some improvements on the construction of the subset automaton of a homogeneous NFA. In this part, we shall focus on their issue in the case of the Glushkov NFA of a regular expression and its use in pattern matching.

Let E be an epsilon-free¹ regular expression of size n which does not recognize the empty word. In this context, the size of a regular expression is the number of alphabetical symbols that appear in the expression. For instance the expression $(a^*.a).a^3$ is of size 5. The Glushkov automaton of E is an $(n + 1)$ -state homogeneous NFA that recognizes the language denoted by E and can be built in $\mathcal{O}(n^2)$ time [1, 5, 10]. It may be easily modified in order to recognize the language Σ^*E : just add arrows $(0, a, 0)$ for all $a \in \Sigma$, where 0 is the initial state. Hence, every state of the resulting NFA is homogeneous excepting the state 0, and every accessible state of its subset automaton contains the state 0. Then we can create the bit-automaton of the homogeneous part of the Glushkov NFA by the different ways described in Section 3.2 and in Section 4.2.

We should mention that the NR algorithm takes advantage of the fact that one position in the Glushkov NFA can be marked by several symbols. Consider an alphabet Σ and the set \mathcal{E} of epsilon-free regular expressions on Σ . Let us

¹ E is epsilon-free if the symbol ε does not appear in E .

introduce an extended class of regular expressions $[\mathcal{E}]$ whose *atoms* are the symbols of the alphabet Σ and the expressions in the form of $[a_1 a_2 \cdots a_k]$ where $a_j \in \Sigma$, $(\forall j)$. Expressions in $[\mathcal{E}]$ derives into expressions of \mathcal{E} through the following rewriting rule: $[a_1 a_2 \cdots a_k] \longrightarrow a_1 + a_2 + \cdots + a_k$. During the computation of the Glushkov NFA, the algorithm NR considers each expression $[a_1 a_2 \cdots a_k]$ as one position. Since the resulting automaton is non homogeneous, the algorithm CCP cannot be applied to an expression in $[\mathcal{E}]$ unless it is translated into \mathcal{E} .

Let E be a regular expression in $[\mathcal{E}]$. We define the size of E as the number of atoms appearing in it. Let $a \in \Sigma$, an atom $[a_1 a_2 \cdots a_k]$ is said to be an occurrence of a if and only if a is one a_j . The following proposition intends to show that CCP has a better complexity with the exception of cases where square brackets are extensively used.

Proposition 10 *Let E be a regular expression in $[\mathcal{E}]$ of size m . Consider that symbols are equally distributed, that is, symbols have all the same number of occurrences n . We let $s = |\Sigma|$ and $\rho = \frac{n}{m}$. If \mathcal{M} is the available memory in bits, the algorithm CCP has a lower time complexity than the algorithm NR as soon as we have:*

$$\rho \leq 1 - \frac{\log(s^2)}{\log(\mathcal{M}) - \log(m^2)}$$

Proof — Let t be the number of tables needed by the method NR. The memory space used by NR is then $tm2^{\frac{m}{t}}$ bits. In order to get an equivalent complexity, CCP needs $s^2tn2^{\frac{n}{t}}$ bits. We have to determine whether this memory requirement is lower than NR's. Indeed:

$$s^2tn2^{\frac{n}{t}} \leq tm2^{\frac{m}{t}} \iff n \leq m - t(\log(s^2n) - \log(m))$$

Besides, as $tm2^{\frac{m}{t}} \leq \mathcal{M}$ we have an upper bound for t :

$$t \leq \frac{m}{\log(\mathcal{M}) - \log(m^2)}$$

We deduce:

$$\rho \leq 1 - \frac{\log(s^2\rho)}{\log(\mathcal{M}) - \log(m^2)}$$

which implies the expected condition. ■

6 Performance analysis

We consider a regular expression E of size n , where the number of occurrences of each symbol $a \in \Sigma$ is denoted by n_a . Our purpose is to implement pattern matching based on the Glushkov automaton of E which we shall determinize using one of the algorithms `nr`, `NR`, `ccp` or `CCP`. We first give some considerations about the implementation of these algorithms (space and time complexity, input size limitation). Experimental results are then provided in graph or table form for tests we carried out in the fields of text processing and biology. We end with a brief analysis of these results, with regard to theoretical predictions.

6.1 Implementation considerations

Until now the size of the tables of the different algorithms has been expressed in bits. This size is denoted by `BinSpa` in the following tables. By care of efficiency, our implementation handles tables satisfying word alignment. The table size is then expressed in bytes and it is denoted by `MachSpa`. The available memory (in bytes) is denoted by `EspMax` while `smask` stands for the size (in bytes) of a machine word.

The worst case time complexity of parsing a text of length l is denoted by `CompTime`. This complexity is identical for the algorithms `nr` and `ccp`.

As far as the algorithms `nr` and `ccp` are concerned, the item `Limits` gives the conditions which ensure that the (full) determinization is possible. On the opposite, the algorithms `NR` and `CCP` are safe ones. Although the algorithms `nr` and `ccp` are specific cases of the algorithms `NR` and `CCP` they have been implemented separately by care of efficiency, so that our program chooses between `nr` and `NR` algorithms or between `ccp` and `CCP` algorithms considering the pre-computed memory requirement.

The features of the algorithms `nr` and `ccp` (resp. `NR` and `CCP`) are compared in Table 4 (resp. Table 5).

| | <code>nr</code> | <code>ccp</code> |
|------------------------------|---|--|
| <code>BinSpa</code> (bits) | $(n + 1)2^{n+1}$ | $n \sum_{a \in \Sigma} 2^{n_a}$ |
| <code>MachSpa</code> (bytes) | $smask \cdot 2^{n+1}$ | $smask \cdot \Sigma \sum_{a \in \Sigma} 2^{n_a}$ |
| <code>CompTime</code> | $\mathcal{O}(l)$ | $\mathcal{O}(l)$ |
| <code>Limits</code> | $n \leq 8 \cdot smask - 1$ $smask \cdot 2^{n+1} \leq EspMax$ | $n_a \leq 8 \cdot smask$ $smask \cdot \Sigma \cdot \sum_{a \in \Sigma} 2^{n_a} \leq EspMax$ |

Table 4. Spatial complexity of the algorithms `nr` and `ccp`.

6.2 Experimental protocol

The algorithm `nr` has been implemented in the software `nours` [8] in C language. We have developed the algorithms `ccp` and `CCP` and we have included the algorithms `nr` and `NR` in our new program, using the same structures, in order to fairly compare the performance of all these algorithms.

| | NR | CCP |
|-----------------|---------------------|--|
| BinSpa (bits) | $kn2^{n/k}$ | $n \sum_{a \in \Sigma} k_a 2^{n_a/k_a}$ |
| MachSpa (bytes) | $smask.k^2 2^{n/k}$ | $smask. \sum_{a \in \Sigma} (k_a 2^{n_a/k_a}) \sum_{b \in \Sigma} k_b$ |
| CompTime | $\mathcal{O}(lk^2)$ | $\mathcal{O}(l \times \max^2(\{k_a \mid a \in \Sigma\}))$ |

Table 5. Space and time complexity of the algorithms NR and CCP.

The machine we used has the following characteristics: 2 processors 1 GHz Pentium III, a cache memory of 256 Kb, 1GB of RAM. We have used the `gcc` compiler with its option `-O3`.

Tests have been carried out on two different types of text. The first one is a DNA sequence duplicated in order to obtain a 100 MB file. The sequence is a part of human DNA downloaded from the GenBank site ². The second one is the “Alice’s adventures in wonderland” book which is duplicated in order to obtain a 100 MB file. This text can be downloaded from the Canterbury Corpus site ³.

Our aim is to show how the horizontal splitting technique influences the behavior of the algorithms NR and CCP for a fixed memory size. Therefore we have carried out two types of test. The first one is based on random regular expressions, and the second one on fixed regular expressions.

For tests depending on the amount of available memory, results are presented with graphs whose abscissae denote the amount of memory. These tests are referred to as “graph-tests” in the following. For the other tests, the results are gathered in tables in which times are denoted by $a + bn$ as in [7], where a is the preprocessing time (for the construction of the tables) and b is the time necessary to parse 1 MB of text. These tests are referred to as “table-tests” in the following. All the times are in tenths of second.

6.3 Fixed and random regular expressions

The following table lists the fixed regular expressions used to test the algorithms on the text “Alice’s adventures in wonderland”. The symbol \sqcup stands for the symbol space, and the symbol \downarrow denotes carriage return.

² <http://www.ncbi.nlm.nih.gov/Genbank/index.html>

³ <http://corpus.canterbury.ac.nz/>

The random regular expressions on DNA are built with a specific percentage of each regular operator. The Kleene stars are positioned in order to prevent the expression from recognizing the empty word.

6.4 Experimental results

I DNA tests based on random regular expressions

The Figure 4 illustrates the variation of the parsing time and of the number of tables depending on the amount of available memory. It shows that CCP algorithm needs a smaller number of tables than NR algorithm. The degree of determinization of CCP algorithm is higher than the one of NR algorithm, and thus its parsing time is smaller.

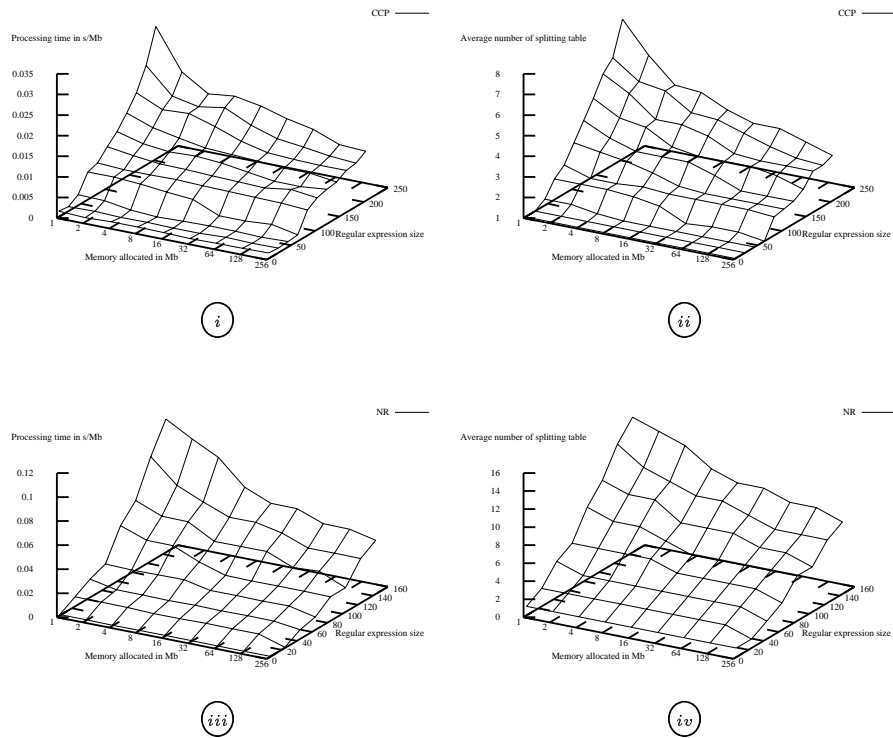


Fig. 4. Parsing Time and Number of Tables of the algorithms CCP (Figures *i*, *ii*) and NR (Figures *iii*, *iv*).

In the following, the results of table-tests have been obtained with an amount of 1 MB available memory.

II Alice tests

| | NR | | CCP | |
|---|-------------------------|----------|-------------------------|----------|
| | Preprocessing + Parsing | # Tables | Preprocessing + Parsing | # Tables |
| 1 | 0+0.073 | 1 | 0+0.16125 | 1 |
| 2 | 0.0123+0.07875 | 1 | 0+0.1673750 | 1 |
| 3 | 0.0025+0.076025 | 1 | 0.015+0.1332250 | 1 |
| 4 | 0.0025+0.109925 | 1 | 0.0225+0.180425 | 1 |
| 5 | 0.05+1.0065 | 2 | 0.0833+0.659667 | 4 |
| 6 | 0.2+21.9948 | 18 | 0.4833+1.79117 | 7 |

Table 8: Alice table-tests.

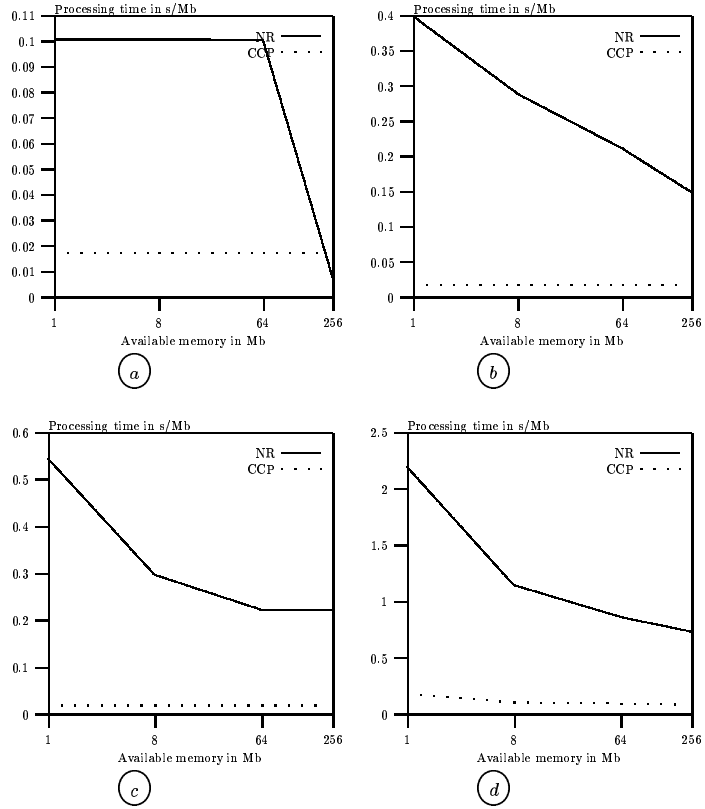


Fig. 5. Alice graph-tests.

III DNA tests

| | NR | | CCP | |
|-----|-------------------------|---------|-------------------------|---------|
| | Preprocessing + Parsing | #Tables | Preprocessing + Parsing | #Tables |
| I | 0+1.4771437 | 3 | 0+0.66903711 | 2 |
| II | 0.025+2.290166 | 5 | 0.025+0.98100386 | 2 |
| III | 0 +0.086448616 | 1 | 0 + 0.12403497 | 1 |
| IV | 0 +0.10900043 | 1 | 0 + 0.19544904 | 1 |

Table 9: DNA table-tests

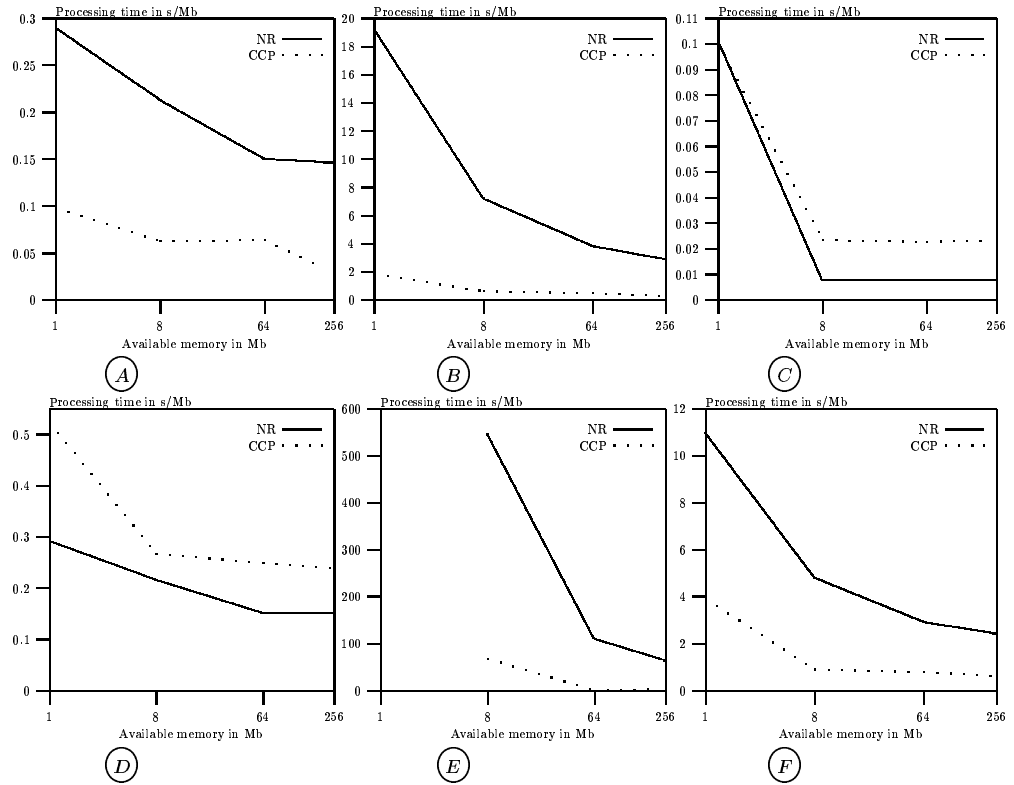


Fig. 6. DNA graph-tests.

6.5 Analysis

Let us discuss different points separately:

- **Little expressions** — The aim of CCP is to deal better with large regular expressions, but of course, the algorithm `nr` is suitable for little expressions. There is no apparent reason why `ccp` might be slower than `nr` though it is true for table-tests 1, 2, III, IV. In terms of assembler instructions and memory access, `ccp` should be a bit faster than `nr`, but `nr` manipulates a lower memory space for little expressions, which makes it more suitable for processors with cached memory.

- **Quadratic complexity** — The theoretical study pretends that the table splitting is linear w.r.t. the size of the regular expression for both NR and CCP, and that the parsing time is quadratic w.r.t. the table splitting. This quadratic increase appears in graph-tests i and iii.

- **Using square brackets in NR** — The square bracket operators have been studied in Section 5. In a way, table-tests 3 and 4 are worst cases for CCP since we have very little expressions with very extensive brackets. Table-test 5 and graph-test D show expressions where we have an extensive use of brackets that is balanced by the size of the expression, while graph-test C is an intermediate case. Table-test 6 and graph-tests c, d, E and F show expressions where brackets are moderately used, which has no real effect on the comparison.

- **Effect of the distribution of symbols** — The method CCP should give better results when symbols are well distributed, which can be seen by comparing Alice with DNA, whose alphabet is quite smaller.

7 Conclusion

Before implementing a determinization procedure such as CCP, one shall wonder if it gives better results than a classical simulation over the original nondeterministic automaton. The answer is obviously no when the original automaton happens to be already deterministic, so that the simulation is in $\mathcal{O}(l)$ for a text of length l . We can expect that for an almost deterministic automaton, a simple simulation gives also better results. Actually, it depends on the average size of accessible states of the subset automaton. In our particular domain, that is regular pattern matching, we produce an automaton with loop-back arrows on initial states, whose accessible subsets tend to grow fast. Hence, using partial determinization is justified.

References

1. A. Brüggemann-Klein. Regular expressions into finite automata. *Theoret. Comp. Sc.*, 120:197–213, 1993.
2. P. Caron and D. Ziadi. Characterization of Glushkov automata. *Theoret. Comp. Sc.*, 233:75–90, 2000.
3. J.-M. Champarnaud. Subset construction complexity for homogeneous automata, position automata and ZPC-structures. *Theoret. Comp. Sc.*, 267(1-2):17–34, 2001.
4. J.-M. Champarnaud. A propos du calcul exhaustif du déterminisé d'un automate homogène. Manuscript, 2002.

5. C.-H. Chang and R. Paige. From regular expression to DFA's using NFA's. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Lecture Notes in Computer Science, 3rd Symposium on Combinatorial Pattern Matching*, number 664, pages 90–110. Springer-Verlag, Berlin, 1992.
6. F. Coulon. CCP – a compact and fast algorithm for regular expression search. Source code, <http://www.univ-rouen.fr/LIFAR/aia/ccp.html>, <http://www.cs.ucr.edu/stelo/pattern.html>, 2003.
7. G. Navarro and M. Raffinot. Compact DFA representation for fast regular expression search. In G. S. Brodal, D. Frigioni, and A. Marchetti-Spaccamella, editors, *Lecture Notes in Computer Science, 5th Workshop of Algorithm Engineering*, number 2141, pages 1–12. Springer-Verlag, Berlin, 2001.
8. G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002. ISBN 0-521-81307-7.
9. J.-L. Ponty. An efficient null-free procedure for deciding regular language membership. *Theoret. Comp. Sc.*, 231:89–101, 2000.
10. J.-L. Ponty, D. Ziadi, and J.-M. Champarnaud. A new quadratic algorithm to convert a regular expression into an automaton. In D. Raymond and S. Yu, editors, *Lecture Notes in Computer Science, First Workshop on Implementing Automata*, number 1260, pages 109–119. Springer-Verlag, Berlin, 1997.
11. S. Wu and U. Manber. Fast text searching algorithm allowing errors. *CACM*, 35(10):83–91, October 1992.
12. S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume I, Word, Language, Grammar*, pages 41–110. Springer-Verlag, Berlin, 1997.