

Brute force determinization of NFAs by means of state covers

J.-M. Champarnaud, F. Coulon, and T. Paranthoën
{jean-marc.champarnaud,fabien.coulon,thomas.paranthoen}@univ-rouen.fr

LIFAR, University of Rouen

Abstract. Finite automata determinization is a critical operation for numerous practical applications such as regular expression search. Algorithms have to deal with the possible blow up of determinization. There exist solutions to control the space and time complexity like the so called “*on the fly*” determinization. Another solution consists in performing brute force determinization, which is robust and technically fast, although *a priori* its space complexity constitutes a weakness. However, one can reduce this complexity by performing a partial brute force determinization. This paper provides optimizations that consist in detecting classes of unreachable states and transitions of the subset automaton, which leads in average to an exponential reduction of the complexity of brute force and partial brute force determinization.

1 Introduction and notation

Finite automata determinization is a critical operation for numerous practical applications such as regular expression search, *e.g.* Prosite and Swiss-prot patterns [1] in biology, or the family of grep commands in text manipulation systems [7].

Let $\mathcal{A} = \langle Q, \Sigma, \delta, I, F \rangle$ be a nondeterministic automaton, where Q is the set of states, I (resp. F) is the set of initial (resp. final) states, δ is the *transition function* defined from $Q \times \Sigma$ into 2^Q . A *computation* of the word $a_1 \cdots a_k$ in \mathcal{A} is a sequence P_1, \dots, P_{k+1} of subsets of Q such that $P_{l+1} = \delta(P_l, a_l)$ for all l . The determinization of \mathcal{A} consists in precomputing $\delta(P, a)$ for all $P \subseteq Q$ and $a \in \Sigma$, which enables to parse a text of length n within time $\mathcal{O}(n)$. Determinization *by reachability* consists in computing $\delta(P, a)$ only for *reachable* subsets P , that is, subsets that can be reached at the end of some computation starting in I . On the contrary, *brute force* determinization consists in precomputing $\delta(P, a)$ for all subsets P and letters a .

In most cases, one can proceed to a simple incremental determinization by reachability. Unfortunately, the number of reachable subsets is potentially exponential, which can make determinization practically untractable. Moreover, the size of the structure used to store subsets that have been already parsed constitutes an obstacle. Indeed, this structure has to provide a fast testing procedure for membership and hence must be stored in memory. It is usually a list, a binary search tree, a trie, combined or not with a hashing table [10, 3, 8].

There are several known solutions to deal with the exponential blow up of determinization, including the so-called *on the fly* determinization used in *grep*, and the partial brute force determinization. The latter is a trade off between parsing time and determinization complexity. This trade off consists in splitting the set of states into several blocks Q_1, \dots, Q_r . Then one precomputes $\delta(P_j, a)$ for all $P_j \subseteq Q_j$, $1 \leq j \leq r$. Yet, given a subset P of Q , one can recover $\delta(P, a)$ as $\cup_j \delta(P \cap Q_j)$ during the parse of the text. This is described by Wu and Manber in [13]. Moreover, brute force determinization has an efficient bitwise implementation [12, 8]. Indeed, subsets of Q are coded bitwise (one bit codes for the presence of one element of Q), and the deterministic transition table is an array indexed on subsets of Q . Each union is performed bitwise.

However, brute force determinization carries on a huge amount of useless precalculus. Navarro and Raffinot recently proposed a first improvement in [11] by dividing the complexity by the size of the alphabet using *position automata* [9] (or *Glushkov automata*). The *position automaton* obtained from an n -long regular expression is an $(n + 1)$ -state automaton without epsilon transitions. A *position automaton* is homogeneous, in the sense that, all the transitions entering a given state are labelled by the same letter, so that one can represent such an automaton by labelling states instead of transitions. Hence we have a unique transition function $\delta : Q \mapsto 2^Q$ and a state partitioning with respect to their labels: we note Q_a the set of states labelled by a . The technique of Navarro and Raffinot consists in precomputing $\delta(P)$ for all $P \subseteq Q$. Then $\delta(P, a)$ can be recovered during the parse as $\delta(P) \cap Q_a$. So the precalculus needs a space $2^{|Q|}$ instead of $|\Sigma|2^{|Q|}$ for the “naive” brute force determinization.

We proposed a second improvement in [4], simply by considering that reachable subsets of a position automaton are necessarily contained in one Q_a . Hence the parse of the text can be done as soon as one knows every $\delta(P)$ for P contained in some Q_a . The space complexity of the precalculus is then reduced to $\sum_a 2^{|Q_a|}$, which is in average exponentially smaller than the latter.

The present paper no longer considers position automata. It aims, for general automata, to extend the idea of performing brute force determinization on blocks rather than on the complete automaton. We propose different techniques, either polynomial or not, for computing partitionings or covers (Q_1, \dots, Q_r) , such that each reachable subset P is contained in one Q_j . When applied to position automata, these covers reveal to be at least as efficient as the symbol partitioning mentioned above. The tests we have carried out show that they are indeed much more efficient for reducing the space complexity of brute force determinization.

The paper is organized as follows. Section 2 describes our technique based on the notion of deterministic covers, whose practical computing is described in Section 3. Then we present experimental results in Section 4.

2 Deterministic covering automata

Many transitions outgoing from unreachable subsets are calculated in vain during a brute force determinization. In this section, we consider the problem of detecting subsets that are *a priori* known to be unreachable.

In the following, $\mathcal{A} = \langle Q, \Sigma, \delta, I, T \rangle$ stands for an NFA, and we note $n = |Q|$. Let $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$ be a collection of non empty subsets of Q . If $\bigcup_{1 \leq i \leq k} R_i = Q$, then \mathcal{R} is said to be a *cover* of Q . If \mathcal{R} is a cover and satisfies $R_i \cap R_j = \emptyset$ ($\forall i, j$), then \mathcal{R} is a *partitioning* of Q .

Definition 1 (Deterministic cover) A cover $\mathcal{R} = \{R_1, \dots, R_k\}$ of Q is a *deterministic cover* of \mathcal{A} if:

1. $(\exists R_0 \in \mathcal{R}) \quad I \subseteq R_0$,
2. $(\forall R \in \mathcal{R}) (\forall a \in \Sigma) (\exists R' \in \mathcal{R}) \quad \delta(R, a) \subseteq R'$.

A deterministic cover provides additional information about which subsets are visited during a simulation of the automaton: being in a subset of a block R , we know that one can go in some block R' by a symbol a and nowhere else. This behaviour can be modelled by a super-automaton:

Definition 2 Let \mathcal{R} be a deterministic cover of \mathcal{A} . We get a covering automaton¹ $\langle \mathcal{R}, \Sigma, \Delta, \mathcal{I} \rangle$ of \mathcal{A} by letting: $\Delta(R, a) = \{R' \mid \delta(R, a) \subseteq R'\}$, and $\mathcal{I} = \{R \mid I \subseteq R\}$.

This super-automaton is not necessarily deterministic. We call *deterministic covering automaton* any automaton obtained from the latter by keeping only one element in each $\Delta(R, a)$, and one element in \mathcal{I} .

The elements kept for getting a deterministic covering automaton can be chosen arbitrarily, since we only need that there exists one R' such that $\delta(R, a) \subseteq R'$. Covering automata do not have final states since their purpose is just to describe transitions of the original automaton.

For example, consider the automaton at left of Figure 1. The cover $\{B_1, B_2\}$ with $B_1 = \{0, 1, 2\}$ and $B_2 = \{2, 3, 4\}$ is deterministic. One associated covering automaton is drawn at right.

We easily see that every subset of the subset automaton is contained in one block of the deterministic cover. This gives an upper bound for the number of reachable subsets in the subset automaton:

Proposition 1 Let $\mathcal{A} = \langle Q, \Sigma, \delta, I, T \rangle$ be an NFA and $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$ be a deterministic cover of \mathcal{A} . The size of the subset automaton is lower than $\sum_{i=1}^k 2^{|R_i|}$.

¹ Let us mention that the covering automata introduced by this definition are not related to the covering automata for a finite language L due to Câmpeanu, Păun and Yu in [2].

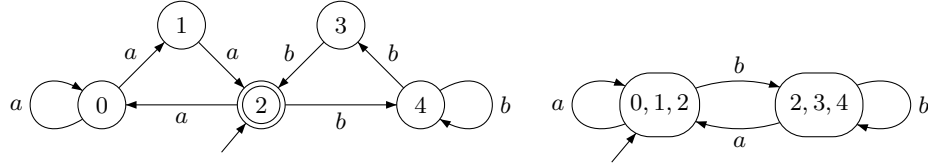


Fig. 1. An automaton recognizing $(a^3a^* + b^3b^*)^*$, and one of its covering automata.

A brute force determinization does not take this bound into account and creates the 2^n subsets in vain. On the contrary, the brute force determinization technique that we present fits the bound.

Consider a deterministic cover $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$ of \mathcal{A} and one associated deterministic covering automaton $\langle \mathcal{R}, \Sigma, \Delta, \{R_{init}\} \rangle$.

We build a deterministic automaton related to the cover \mathcal{R} by the following way. The deterministic transitions are stored in k tables $(C[i])_{i=1, \dots, k}$. Each table $C[i]$ contains $|\Sigma|$ tables $(C[i][a])_{a \in \Sigma}$. Each table $C[i][a]$ is made of $2^{|R_i|}$ entries. And we let:

$$(\forall P \in R_i)(\forall a \in \Sigma) \quad C[i][a][P] = \delta(P, a)$$

The simulation of the automaton is carried out by the following way. The current state of our system is a couple (i, P) where $P \subseteq R_i$. The initial state is $(init, I)$, then each transition is given by the transition function δ' defined by $\delta'((i, P), a) = (\Delta(i, a), C[i][a][P])$. The couple we get is a state of the system since $C[i][a][P] = \delta(P, a) \subseteq \Delta(i, a)$.

For each i , the table $C[i]$ contains the transitions of $2^{|R_i|}$ subsets. Hence,

Proposition 2 *The tables $(C[i])_{i=1, \dots, k}$ are computed and stored with a complexity lower than*

$$n|\Sigma| \sum_{i=1}^k 2^{|R_i|}$$

Using these tables, each transition of the subset automaton can be calculated in time $\mathcal{O}(n)$.

The $\mathcal{O}(n)$ time complexity is due to the manipulation of a state number. Indeed, states rank from 0 to $2^n - 1$, so that storing a state number requires n bits in the worst case. However in practice, the memory limitation implies that a table index can always be stored in a processor register.

3 Computing a deterministic cover

We focus now on the existence and the practical calculus of deterministic covers.

3.1 The maximal cover

Definition 3 Let \mathcal{D} be the set of reachable subsets of \mathcal{A} . The maximal cover of \mathcal{A} is the set of all maximal elements of \mathcal{D} with respect to inclusion.

Clearly, the maximal cover of \mathcal{A} is a deterministic cover. In practice, this cover is obtained as the fixed point of the sequence (\mathcal{R}_i) defined in the following way:

The collection \mathcal{R}_0 contains initially the only block I . The collection \mathcal{R}_{i+1} is deduced from \mathcal{R}_i as follows:

$$\mathcal{R}'_{i+1} = \mathcal{R}_i \cup \{\delta(R, a) \mid R \in \mathcal{R}_i, a \in \Sigma\}$$

Then, \mathcal{R}_{i+1} is obtained from \mathcal{R}'_{i+1} by keeping only subsets that are maximal for inclusion. The sequence (\mathcal{R}_i) converges to the maximal cover of \mathcal{A} .

The maximal cover can be exponentially greater than the automaton:

Proposition 3 If $|\Sigma| \geq 2$, then for all n , there exists an automaton \mathcal{A}_n such that its maximal cover is made up of $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ blocks.

Proof. We build an n -state automaton $\mathcal{A}_n = \langle Q_n, \Sigma, \delta_n, I_n, F_n \rangle$ whose maximal cover contains $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ blocks. We note $Q_n = \{1, \dots, n\}$, and we let $I_n = \{1, \dots, \lfloor \frac{n}{2} \rfloor\}$. Let $a, b \in \Sigma$, we let

$$\delta_n(q, a) = \begin{cases} 2 & \text{if } q = 1 \\ 1 & \text{if } q = 2 \\ q & \text{otherwise} \end{cases} \quad \text{and } \delta_n(q, b) = \begin{cases} q + 1 & \text{if } q < n \\ 1 & \text{if } q = n \end{cases}$$

The mappings $q \mapsto \delta_n(q, a)$ and $q \mapsto \delta_n(q, b)$ are permutations of Q_n , respectively the transposition $(1, 2)$ and the cycle $(1, 2, \dots, n)$. Yet, these two permutations are generators of the symmetric group of degree n . Hence, the reachable subsets of \mathcal{A}_n are all permutations of I , that is, all subsets of cardinality $\lfloor \frac{n}{2} \rfloor$.

The algorithm for computing a maximal cover is as follows:

MAXIMAL COVER(\mathcal{A})

- 1 Let \mathcal{R} contain the only block I , and mark I .
- 2 **while** there exists a marked block in \mathcal{R} **do**
- 3 pick a marked block R in \mathcal{R} and unmark it
- 4 **for each** $a \in \Sigma$ **do**
- 5 $P \leftarrow \delta(R, a)$
- 6 **if** P is not included in any block of \mathcal{R} **then**
- 7 remove blocks of \mathcal{R} that are included in P
- 8 add P to \mathcal{R} and mark it

All the following cover algorithms have been implemented using the same schema.

3.2 Other covers

Different techniques can provide deterministic covers with a polynomial complexity. But those covers may contain larger blocks than blocks of the maximal cover.

The optimal partitioning Among deterministic covers, the class of covers that are just partitionings can be easily studied. We refer to these covers as *deterministic partitionings*. There exists a unique optimal deterministic partitioning that can be calculated by an algorithm similar to the latter one, but whose complexity is polynomial.

Consider the partitioning of Q obtained as the fixed point of the sequence (\mathcal{P}_i) defined in the following way. The collection \mathcal{P}_0 contains initially the block I and all singletons of $Q \setminus I$. The collection \mathcal{P}_{i+1} is deduced from \mathcal{P}_i as the finest partitioning that is compatible with \mathcal{P}_i and with each block $\delta(B, a)$, ($B \in \mathcal{P}_i$, $a \in \Sigma$). That is, let first $\mathcal{P}_{i+1} \leftarrow \mathcal{P}_i$. For all couples of blocks B_1, B_2 in \mathcal{P}_i that intersect a same block $\delta(B, a)$, B_1 and B_2 are removed and replaced by $B_1 \cup B_2$. This operation is repeated until all blocks $\delta(B, a)$, ($B \in \mathcal{P}_i$, $a \in \Sigma$) are included in a block $B \in \mathcal{P}_{i+1}$.

The sequence $(\mathcal{P}_i)_i$ converges toward a deterministic partitioning of \mathcal{A} . On the other hand, since the partitioning \mathcal{P}_{i+1} is obtained by merging elements of \mathcal{P}_i , the sequence reaches its fixed point at least at \mathcal{P}_n . As a result:

Proposition 4 *The fixed point of the sequence $(\mathcal{P}_i)_i$ is a deterministic partitioning finer than any other deterministic partitioning of \mathcal{A} . Hence, it is called the optimal deterministic partitioning. Moreover, it can be calculated in time $|\Sigma|n^3$.*

Proof. The partitioning \mathcal{P}_0 is clearly finer than any deterministic partitioning, and \mathcal{P}_i inherits this property from \mathcal{P}_{i+1} for all i . Indeed, \mathcal{P}_{i+1} is the finest partitioning that is compatible with \mathcal{P}_i and all $\delta(\mathcal{P}_i, a)$. On the other hand, a deterministic partitioning \mathcal{P} is compatible with itself and all $\delta(\mathcal{P}, a)$. If \mathcal{P} is coarser than \mathcal{P}_i , then \mathcal{P} is compatible with \mathcal{P}_i and all $\delta(\mathcal{P}_i, a)$, hence \mathcal{P} is coarser than \mathcal{P}_{i+1} .

The neighbourhood cover We propose a first polynomial heuristic providing covers that are not just partitionings.

Definition 4 *Let q be a state, the left language of q is the set of labels of paths starting in an initial state and ending in q . It is denoted $\overleftarrow{\mathcal{L}}(q)$.*

Definition 5 *Consider the non oriented graph $\mathcal{G} = \langle Q, G \rangle$ whose vertices are the states of \mathcal{A} , and whose set of edges G is defined by*

$$(q, p) \in G \iff \overleftarrow{\mathcal{L}}(q) \cap \overleftarrow{\mathcal{L}}(p) \neq \emptyset$$

Proposition 5 *The graph \mathcal{G} can be calculated in time $\mathcal{O}(n^4)$. Let $q, p \in Q$, the edge (q, p) is in G if and only if there exists a reachable subset P of \mathcal{A} that contains q and p .*

Definition 6 *We denote $\mathcal{V}(p) = \{p\} \cup \{q \in Q \mid (q, p) \in G\}$ the neighbourhood of a state p .*

Proposition 6 *If the automaton \mathcal{A} is complete and has just one initial state ($I = \{i_0\}$), then the cover $\mathcal{R} = \{\mathcal{V}(p) \mid p \in Q\}$ is deterministic and can be computed in time $\mathcal{O}(n^4)$.*

Proof. Let p be a state of \mathcal{A} and $a \in \Sigma$, there exists $p' \in \delta(p, a)$ since the automaton is complete. It is sufficient to prove that $\delta(\mathcal{V}(p), a) \subseteq \mathcal{V}(p')$. So, let $q \in \delta(\mathcal{V}(p), a)$. We have $q' = \delta(q, a)$ for some $q \in \mathcal{V}(p)$. Hence, there exists a word w contained in $\overleftarrow{\mathcal{L}}(p) \cap \overleftarrow{\mathcal{L}}(q)$, which implies $wa \in \overleftarrow{\mathcal{L}}(p') \cap \overleftarrow{\mathcal{L}}(q')$, thus $(q', p') \in G$ or $q' = p'$, and finally $q' \in \mathcal{V}(p')$.

The merging cover We present a second cover calculus whose complexity is polynomial.

Here is the general description of the algorithm: initially, we have the cover \mathcal{R} made up of I and the singletons of $Q \setminus I$. As long as the cover \mathcal{R} is not deterministic, there exists a block P and a letter a such that $\delta(P, a)$ is included in no block. The algorithm picks a block $R \in \mathcal{R}$ and replaces it by $R \cup \delta(P, a)$. This operation is repeated until one gets a deterministic cover. The blocks of the cover \mathcal{R} are numbered from 1 to n . The i^{th} block is noted B_i .

MERGING COVER(\mathcal{A})

```

1 The cover  $\mathcal{R}$  is made up of  $I$  and the singletons of  $Q \setminus I$ 
2 Mark all blocks  $B_1, \dots, B_n$ 
3 while there exists a marked block in  $\mathcal{R}$  do
4   | pick a marked block  $B_i$  in  $\mathcal{R}$  and unmark it
5   | for each  $a \in \Sigma$  do
6     | pick a block  $B_j$  in  $\mathcal{R}$ 
7     | if  $\delta(B_i, a) \not\subseteq B_j$  then
8       |    $B_j \leftarrow B_j \cup \delta(B_i, a)$ 
9       |   mark  $B_j$ 
10  |   remove blocks  $B_k$  ( $k \neq j$ ) that are included in  $B_j$ 
```

During the running of the algorithm, the number of blocks is n , and each iteration raises strictly the size of a given block. Hence, such an algorithm ends within n^2 iterations. The algorithm efficiency depends on the technique used to pick the block B_j at line 6.

The objective is to generate small blocks. A technique likely to be efficient consists in picking the block B_j such that $B_j \cup \delta(B_i, a)$ is the smallest possible.

This choice can be performed in time n^2 . Hence,

Proposition 7 *The merging cover of \mathcal{A} is deterministic and can be calculated in time $\mathcal{O}(n^4)$.*

4 Experimental results

4.1 Implementation

Three algorithms have been implemented for computing a deterministic cover of an NFA: the maximal cover algorithm, the merging cover algorithm and the optimal partitioning algorithm. They are deduced from an unique scheme described in Section 3.1 for the case of maximal cover algorithm. The implementation of the algorithms must fit with the theoretical complexities given previously. Consequently, a marked block should be accessed in constant time, which is achieved by representing a cover by a double linked list.

4.2 Performance tests

Tests have been carried out from two different sets of regular expressions, and from random NFAs. In the first set (**RandExpr**), regular expressions are randomly built on an alphabet of size 4. In the second one (**RandText**), they are built from words randomly picked in the text "Alice's adventures in wonderland". The random NFAs (**RandNFAs**) are generated as described in [5]. As usually, each random regular expression is prefixed by Σ^* in order to perform pattern matching over a text.

We define the space requirement of a cover as the number of states of the deterministic automaton related to this cover. The distribution of the space requirement of the covers obtained by each algorithm applied on each random object is illustrated in the following figures. On these figures, for each algorithm, a graph gives the number of regular expressions (or random NFAs) whose cover leads to a given space requirement. Each graph have been made from 10 000 random regular expressions (or random NFAs).

The Figures 2, 3 and 4 give the logarithm of the space requirement of the covers obtained respectively from the optimal partitioning algorithm (b), the maximal cover algorithm (d) and the merging cover algorithm (c).

The following observations can be made about the space distribution:

- For all algorithms the space distribution is better in the case of **RandExpr** and **RandText** than in the case of **RandNFAs**. Moreover it is better in the case of **RandText** than in the case of **RandExpr**.
- The distribution of the maximal cover gives informations about the size of the subsets obtained during a determinization by reachability. The larger the space requirement is, the larger is the size of the subsets.
- In the case of **RandomNFAs**, since the reachable subsets are large ones (see [5]), the space requirement of all covers is important.
- In the case of **RandExpr**, since both alphabet and words used to build these expressions are small, the distribution of the optimal partitioning is similar to the distribution of the symbol partitioning.
- In the case of **RandText**, the distribution of the merging cover is similar to the distribution of the maximal cover. Both of these two distributions are closer to the space requirement of the determinization by reachability; the reason is that the size of reachable subsets is small.

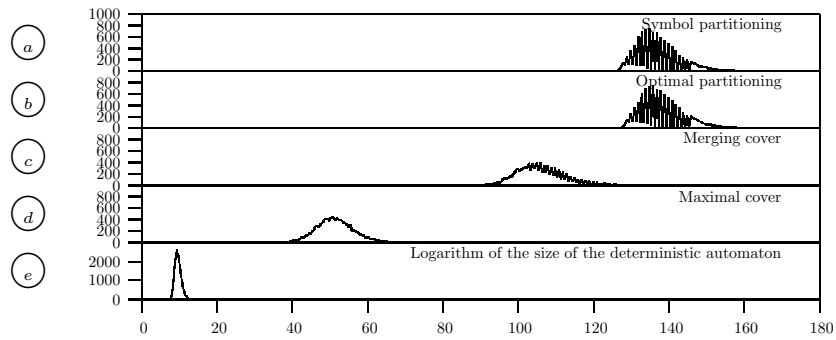


Fig. 2. Case of `RandExpr` expressions of size 500 (on an alphabet of size 4).

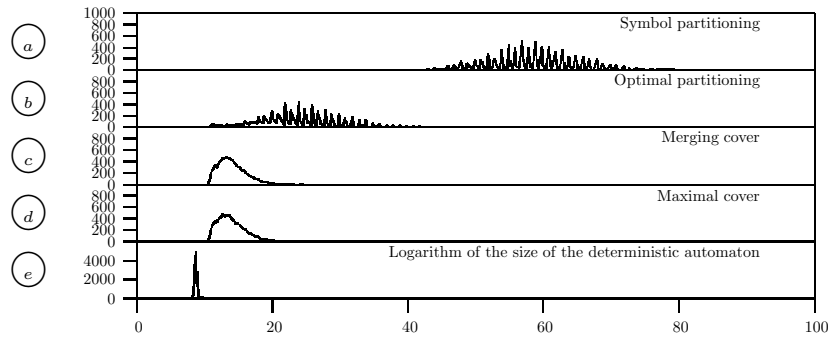


Fig. 3. Case of `RandText` expressions of size approximately 500.

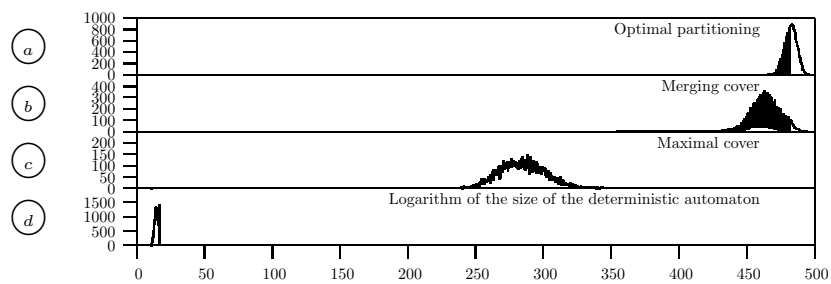


Fig. 4. Case of random 500-NFAs on an alphabet of size 2.

Conclusion

Deterministic covers enable a consequent reduction of brute force determinization complexity, so that we can reasonably handle 500-long regular expressions on text (Figure 3). We expect them also to reduce the complexity of determinization by reachability. The first optimization described in paper [4] has been implemented in the software CCP [6]. We shall soon design a new version implementing improvements of this paper.

References

1. A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. *Nucleic Acids Research*, 28(1):45–48, 2000.
2. C. Campeanu, A. Paun, and S. Yu. An efficient algorithm for constructing minimal cover automata for finite languages. *Int. Journal of Foundations of Computer Science*, 13(1):83–97, 2002.
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, F. Coulon, and T. Paranthoën. Compact and fast algorithms for safe regular expression search. *Intern. J. of Computer. Math.*, 81(4):383–401, 2004.
5. J.-M. Champarnaud, G. Hansel, T. Paranthoën, and D. Ziadi. Nfas random generation models. In *Proceedings of DCFs 2002*, 2002.
6. F. Coulon. CCP software. <http://www.univ-rouen.fr/LIFAR/aia/ccp.html>.
7. J.E.F. Friedl. *Mastering Regular Expressions, Second edition*. O'Reilly, 2002.
8. J. Glenn and W.I. Gasarch. Implementing WS1S via finite automata: Performance issues. In *Lecture Notes in Computer Science, Workshop on Implementing Automata*, volume 1260, pages 75–86. Springer, 1997.
9. V.-M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.
10. J.H. Johnson and D. Wood. Instruction computation in subset construction. In D.R. Raymond, D. Wood, and S. Yu, editors, *Lecture Notes in Computer Science*, volume 1260, pages 64–71. Springer, 1997.
11. 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*, number 2141, pages 1–12. Springer-Verlag, Berlin, 2001.
12. 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.
13. S. Wu and U. Manber. Fast text searching algorithm allowing errors. *CACM*, 35(10):83–91, October 1992.