Theoretical Computer Science 417 (2012) 50–65

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# Average complexity of Moore's and Hopcroft's algorithms

Julien David [*]

*LIPN UMR 7030, Université Paris 13 - CNRS, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France*

ARTICLE INFO

*Keywords:*
Finite automata
Minimization algorithm
Moore
Hopcroft
Average complexity
Generic complexity

ABSTRACT

In this paper we prove that for the uniform distribution on complete deterministic automata, the average time complexity of Moore's state minimization algorithm is $\mathcal{O}(n \log \log n)$, where $n$ is the number of states in the input automata and the number of letters in the alphabet is fixed. Then, an unusual family of implementations of Hopcroft's algorithm is characterized, for which the algorithm will be proved to be always faster than Moore's algorithm. Finally, we present experimental results on the usual implementations of Hopcroft's algorithm.

© 2011 Elsevier B.V. All rights reserved.

## 1. Introduction

The *average complexity* of an algorithm is its average cost for all the instances of a given input set, on which a probability law as been defined. If a probability distribution matches a particular context, then the average time complexity gives the expected duration of an algorithm execution in this context. Without a particular context, the distribution is supposed to be uniform. In this case, the average complexity should be considered as an additional information to the worst-case complexity: it gives a better understanding of the efficiency of the studied algorithm. The reader is referred to [14,15,26] for a general presentation of this topic.

The notion of *generic complexity* has been defined in [19]. For a given probability distribution on the input set of data, the idea is to identify the rare inputs, and to conduct a worst case analysis on the other inputs. This type of analysis is very useful for the study of NP-complete problems or undecidable problems, if the "bad" instances can be proved to happen rarely. As we will see, one can sometimes obtain the average and the generic complexity of an algorithm using the same arguments.

A *minimal automaton* is the unique smallest complete deterministic automata that can be associated to a regular language. Most state minimization algorithms compute the minimal automaton of a regular language taking a deterministic automaton as an input, by identifying the indistinguishable states. Their efficiency has an important impact on the tools in which they are involved, in various fields such as computational linguistics or bioinformatics. The worst-case complexity of those algorithms is well known, but it is only since the last decade that authors started describing minimization algorithms along with their average complexity [3,24] .

*Moore's state minimization algorithm* [22] is one of the most simple to implement. It is based on the computation of the Myhill–Nerode equivalence, by successive refinements of a partition of the set of states. There are at most $n - 2$ such refinements, each of them requiring a linear running time: in the worst case, the complexity is quadratic. Though, in [3], it is proved that the average complexity of the algorithm is bounded by $\mathcal{O}(n \log n)$. Note that since this result does not rely on the underlying graph of the automaton, it holds for any probabilistic distribution on this graph. Also, the bound is tight for unary automata.

*Hopcroft's state minimization algorithm* [17] is the best known algorithm in the general case with a $\mathcal{O}(n \log n)$ worst-case complexity. It also uses partition refinements to compute the minimal automaton, but the order of the operations

* Tel.: +33 6 61 39 16 12.
*E-mail address:* Julien.David@lipn.univ-paris13.fr.

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.10.011

is not fixed, making its analysis complicated. Different proofs of its correctness were given [16,20] and several authors [8,7,10,11] proved the tightness of the upper bound of the complexity for different families of automata. Thanks to those papers, it is known that there exist some automata for which all possible executions of Hopcroft algorithms reach the worst case complexity.

Other minimization algorithms exist, but we won't study their average complexity in this paper because it is already known. For instance, the algorithm of *Hopcroft and Ullman* [18] tests, for every pair of states of the input automaton, whether the two states are equivalent or not. It is known as quadratic in both worst and average complexity. Other algorithms are dedicated to particular classes of deterministic automata or non-deterministic automata. For instance, [25] is restricted to acyclic automata, [24] to unary automata, [5] to local automata. The algorithms [6,27] are both based on Hopcroft's algorithm and are specialized in incomplete deterministic automata. Finally, *Brzozowski's algorithm* [9,13,12] is different from the other ones. Its inputs may be non-deterministic automata. It is not based on partition refinements, but on two successive determinization steps.

In this paper, we prove that for the uniform distribution on complete deterministic automata, the average and generic time complexity of algorithms due to Moore and Hopcroft are $\mathcal{O}(n \log \log n)$. The main result is the average complexity of Moore's algorithm. Simply stated, we prove that the set of automata minimized in more than $\mathcal{O}(\log \log n)$ partition refinements by Moore's algorithm is negligible. The result on the average complexity of Hopcroft's algorithm is a consequence of the main result.

The article is organized as follows: after recalling the basics of automata minimization (Sections 2.1 and 2.2), we introduce the tools we use for the analysis (Section 2.3 to 2.5): two kinds of dependency graph and a dependency tree that models different constraints on sets of automata. Section 3 is dedicated to the study of the average time complexity analysis of Moore's algorithm. Section 4 introduces a set of Hopcroft's algorithm executions which are faster than Moore's one, for any input automata. The paper closes with conjectures on the average complexity of both algorithms, for various distributions of probability on automata.

## 2. Preliminaries

### 2.1. Definitions and notations

A *deterministic finite automaton* $\mathcal{A} = (A, Q, \cdot, q_0, F)$ is a quintuple where $Q$ is a finite set of *states*, $A$ is a finite set of *letters* called *alphabet*, the *transition function* $\cdot$ is a mapping from $Q \times A$ to $Q$, $q_0 \in Q$ is the *initial state* and $F \subset Q$ is the set of final states. An automaton is *complete* when its transition function is total. The transition function can be extended by morphism to all words of $A^*$: $p \cdot \varepsilon = p$ for any $p \in Q$ and for any $u, v \in A^*$, $p \cdot (uv) = (p \cdot u) \cdot v$. A word $u \in A^*$ is recognized by an automaton when $p \cdot u \in F$. The set of all words recognized by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. We note $A^i$ the set of all the words of length $i$ and $A^{\leq i}$ the set of all the word of length less or equal to $i$. An automaton is *accessible* when for any state $p \in Q$, there exists a word $u \in A^*$ such that $q_0 \cdot u = p$.

In the remainder of this paper, we shall consider that the alphabet is always of size $k \geq 2$.

A *transitions structure* is an automaton where the set of final states is not specified. Given such a transitions structure $\mathcal{T} = (A, Q, \cdot, q_0)$ and a subset $F$ of $Q$, we denote by $(\mathcal{T}, F)$ the automaton $(A, Q, \cdot, q_0, F)$. For a given deterministic transitions structure with $n$ states there are exactly $2^n$ distinct deterministic automata that can be built from this transitions structure. Each of them corresponds to the choice of a set of final states.

In the following we only consider complete deterministic automata and complete deterministic transitions structures, the accessibility is not guaranteed. Consequently, most of the time, these objects will just be called respectively *automata* or *transition structures*. The set $Q$ of an $n$-state transition structure will be denoted by $\{1, \dots, n\}$. The set of automata and the set of transitions structures with $n$ states will respectively be denoted $\mathcal{A}_n$ and $\mathcal{T}_n$. Also, since there are $kn$ transitions and since for each transition, there are $n$ possible arrival states, we have $|\mathcal{T}_n| = n^{kn}$ and $|\mathcal{A}_n| = 2^n n^{kn}$ (when $|E|$ is the cardinal of the set $E$). The term $2^n$ comes from the choice of the set of final states.

The *military order* on words, noted $<_{mil}$, is defined as follows: $\forall u, v \in A^*, u <_{mil} v$ if $|u| < |v|$ or $|u| = |v|$ and $u$ is smaller than $v$ in the lexicographical order.

Let *Cond* be a Boolean condition, the Iverson bracket $[\![Cond]\!]$ is equal to 1 if the condition *Cond* is satisfied and 0 otherwise.

For any non-negative integer $i$, two states $p, q \in Q$ are *i-equivalent*, denoted by $p \sim_i q$, when for all words $u \in A^{\leq i}$, $[\![p \cdot u \in F]\!] = [\![q \cdot u \in F]\!]$. Two states $p$ and $q$ are *equivalent* (noted $p \sim q$) when for all $u \in A^*$, $[\![p \cdot u \in F]\!] = [\![q \cdot u \in F]\!]$. This equivalence relation on $Q$ is called *Myhill–Nerode equivalence* [23]. This relation is said to be *right invariant*, meaning that

$$\text{for all } u \in A^* \text{ and all } p, q \in Q, \quad p \sim q \Rightarrow p \cdot u \sim q \cdot u$$

**Proposition 1.** *Let $\mathcal{A} = (A, Q, \cdot, q_0, F)$ be a deterministic automaton with n states. The following properties hold:*

(1) *For all $i \in \mathbb{N}$, $\sim_{i+1}$ is a partition refinement of $\sim_i$, that is, for all $p, q \in Q$, if $p \sim_{i+1} q$ then $p \sim_i q$.*
(2) *For all $i \in \mathbb{N}$ and for all $p, q \in Q$, $p \sim_{i+1} q$ if and only if $p \sim_i q$ and for all $a \in A$, $p \cdot a \sim_i q \cdot a$.*

**Algorithm 1:** Moore's algorithm

```
1  if $F = \emptyset$ then
2      return $(A, \{1\}, *, 1, \emptyset)$
3  if $F = \{1, \dots, n\}$ then
4      return $(A, \{1\}, *, 1, \{1\})$
5  forall the $p \in \{1, \dots, n\}$ do
6      $\pi'[p] = [\![p \in F]\!]$
7  $\pi$ = undefined
8  while $\pi \neq \pi'$ do
9      $\pi = \pi'$
10     compute $\pi'$ from $\pi$
11 return the quotient of $\mathcal{A}$ by $\pi$
```

In this description of Moore's algorithm, $*$ denotes the function such that $1 * a = 1$ for all $a \in A$. Lines 1-4 correspond to the special cases where $F = \emptyset$ or $F = Q$. In the process, $\pi'$ is the new partition and $\pi$ the former one. Lines 5-6 is the initialization of $\pi'$ to the partition of $\sim_0$, $\pi$ is initially undefined. Lines 8-10 are the main loop of the algorithm where the new partition is computed, using the second algorithm below. The *number of iterations* of Moore's algorithm is the number of times those lines are executed.

The computation of the new partition is done using the following property on associated equivalence relations:

$$p \sim_{i+1} q \Leftrightarrow \begin{cases} p \sim_i q \\ p \cdot a \sim_i q \cdot a \quad \forall a \in A \end{cases}$$

To each state $p$ is associated a signature $s[p]$ such that $p \sim_{i+1} q$ if and only if $s[p] = s[q]$. The states are then sorted according to their signature, in order to compute the new partition. The use of a lexicographic sort provides a complexity of $\Theta(kn)$ for this part of the algorithm.

**Algorithm 2:** Computing$\pi'$from$\pi$

```
1  forall the $p \in \{1, \dots, n\}$ do
2      $s[p] = (\pi[p], \pi[p \cdot a_1], \dots, \pi[p \cdot a_k])$
3  compute the permutation $\sigma$ that sorts the states according to
   $s[]$
4  $i = 0$
5  $\pi'[\sigma(1)] = i$
6  forall the $p \in \{2, \dots, n\}$ do
7      if $s[p] \neq s[p-1]$ then $i = i + 1$
8      $\pi'[\sigma(p)] = i$
9  return $\pi'$
```

**Fig. 1.** Description of Moore's algorithm

(3) *If for some $i \in \mathbb{N}$, the $(i+1)$-equivalence is equal to the $i$-equivalence then for every $j \geq i$, the $j$-equivalence is equal to the Myhill–Nerode equivalence.*

For any integer $n \geq 1$ and any $m \in \mathbb{N}$, we denote by $\mathcal{A}_n^m$ the set of automata with $n$ states for which $m$ is the smallest integer such that the $m$-equivalence $\sim_m$ is equal to the Myhill–Nerode equivalence. It is well known that $m \leq n - 2$.

## 2.2. Moore's state minimization algorithm

In this section we describe Moore's algorithm [22] to compute the minimal automaton of a regular language represented by a deterministic automaton (see Fig. 1). It builds the partition of the set of states corresponding to the Myhill–Nerode equivalence and mainly relies on properties (2) and (3) of Proposition 1: the partition $\pi$ is initialized according to the 0-equivalence $\sim_0$, then at each iteration the partition corresponding to the $(i+1)$-equivalence $\sim_{i+1}$ is computed from the one corresponding to the $i$-equivalence $\sim_i$ using property (2). The algorithm halts when no new partition refinement is obtained, and the result is Myhill–Nerode equivalence according to property (3). The minimal automaton can then be computed from the resulting partition since it is the quotient automaton by Myhill–Nerode equivalence.

According to Proposition 1, if an automaton is minimized in more than $\ell$ partition refinements, then there exists at least a pair of states $p, q$ and a word $u$ of length $\ell + 1$, such that $p \sim_\ell q$ and $p \cdot u \nsim_0 q \cdot u$, that is to say at least two states are separated during the $\ell + 1$-th partition refinement. In the remainder of this section we introduce a **dependency tree** and a **dependency graph** that model constraints on sets of transitions structures and a modification of the **dependency graph** introduced in [3] that models constraints on sets of final states. Those tools will allow us to give an upper bound on the number of automata minimized in more than $\ell$ partition refinements, which is useful for the average complexity analysis.

## 2.3. The dependency tree

In the following, we introduce a dependency tree to model a set of transitions structures. To begin with, we explain how a dependency tree $\mathcal{R}(p)$ can be obtained from a fixed transitions structure $\tau$ and a fixed state $p$ and then how this object will help us to estimate the cardinal of a set of transitions structures. For a fixed transitions structure with $n$ states over a $k$-letter alphabet and a fixed state $p$, we define the function isnode mapping $A^*$ to $\{0, 1\}$ as follows:

$$isnode(w) = \begin{cases} 0 & \text{if } \exists v \in A^* \text{ such that } p \cdot w = p \cdot v \text{ and } v <_{mil} w, \\ 1 & \text{otherwise} \end{cases}$$

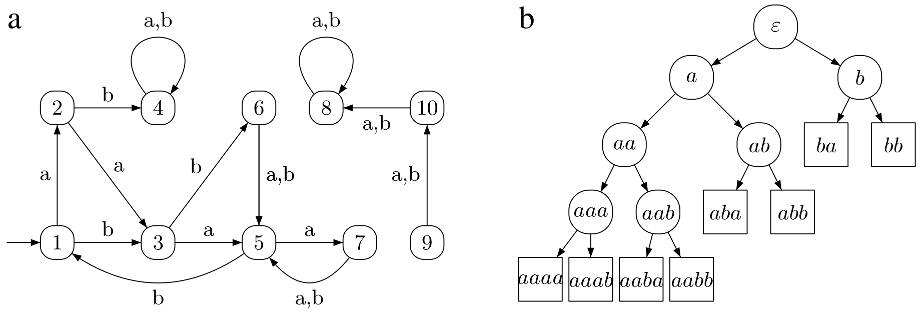

**Fig. 2.** Let (a) be the fixed transitions structure and 2 be the fixed state, (b) is the associated **dependency tree** $\mathcal{R}(2)$. We have $S_2(2) = \{\varepsilon, a, b, aa, ab\}$, $L_2(2) = \{aa, ab\}$ and $s_2(2) = \{2, 3, 4, 5, 6\}$.

Then $\mathcal{R}(p)$ is a tree in which nodes and leaves of depth $h$ are labeled by words of length $h$, built recursively, using a breadth-first traversal starting from the node $p$. For each node of depth $h$ labeled by $w$, and each letter $a$ in the alphabet, we add a **node** labeled by $wa$ at depth $h + 1$ if `isnode(wa)` is equal to 1, and a leaf otherwise. Fig. 2 gives an example of a dependency tree. Note that this construction resembles the method used in [4] to randomly generate accessible automata, but the authors use a depth-first traversal. A same dependency tree can be obtained from several fixed transitions structures and states. In the remainder of the paper, we characterize sets of transitions structures corresponding to particular dependency trees.

We introduce some notations associated to a dependency tree $\mathcal{R}(p)$: $S_h(p)$ is the set of all nodes of depth less or equal to $h$ and $L_h(p)$ denotes the set of all the nodes at given depth $h$. Since every node in the tree is labeled by a word, we note $w \in S_h(p)$ or $w \in L_h(p)$ if $w$ is a word labeling a node in these sets. We also define the set $s_h(p)$ of all the states that are reached, starting from a state $p$ and following a path labeled by a word of length less or equal to $h$. For all the transitions structures associated to a dependency tree $\mathcal{R}(p)$, we have $|s_h(p)| = |S_h(p)|$.

**Lemma 1.** *For any fixed state $p$, if a dependency tree $\mathcal{R}(p)$ contains $f$ leaves of depth less or equal to $h$, then the number of associated transitions structures is bounded above by* $|\mathcal{T}_n| \left(\frac{|S_h(p)|}{n}\right)^f$.

**Proof.** We recall that $|\mathcal{T}_n|$ is equal to the product of the cardinals of the sets of possible arrival states, for each transition. Let $wa$ be the label of a leaf at depth less than $h$. For every transitions structure associated to the tree $\mathcal{R}(p)$, the transition labeled by $a$ outgoing from the state $p \cdot w$ ends in a state $p \cdot v$, with $v \in S_h(p)$. Therefore, the number of possible arrival states for this transition is bounded above by $|S_h(p)|$ instead of $n$. This is a rough upper bound but sufficient for the needs of the proof. □

## 2.4. The $\mathcal{T}$-dependency graph

We introduce another model for sets of transitions structures. It involves two fixed states instead of only one in the case of dependency tree.

For two fixed states $p$ and $q$, two fixed $x$-tuples ($x$ is an integer) of non-empty words $\overrightarrow{u} = (u_1, \ldots, u_x)$ and $\overrightarrow{v} = (v_1, \ldots, v_x)$, two fixed sets $\varphi_p$ and $\varphi_q$ of pairs of words $(w,w')$ such that $w' <_{mil} w$, we define the set $\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ as follows:

$$\begin{aligned}\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v}) = \{\tau \in \mathcal{T}_n \mid & \forall (w_p, w'_p) \in \varphi_p, p \cdot w_p = p \cdot w'_p, \\ & \forall (w_q, w'_q) \in \varphi_q, q \cdot w_q = q \cdot w'_q, \\ & \forall i \le x,\ p \cdot u_i = q \cdot v_i\}\end{aligned}$$

We define the $x$-tuples of words $\overrightarrow{u'} = (u'_1, \ldots, u'_x)$ and $\overrightarrow{v'} = (v'_1, \ldots, v'_x)$ and the $x$-tuples of letters $\overrightarrow{\alpha'} = (\alpha'_1, \ldots, \alpha'_x)$ and $\overrightarrow{\beta'} = (\beta'_1, \ldots, \beta'_x)$, such that for all $i \le x$ we have $u_i = u'_i\alpha_i$ and $v_i = v'_i\beta_i$. In order to model some properties of $\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$, one can define the undirected graph $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$, called the *$\mathcal{T}$-dependency graph*, as follows:

- its vertices are pairs $(r, a)$, with $r \in Q$ and $a \in A$, that model transitions.
- There is an edge $((r, a), (t, b))$ in $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ if and only if for all $\tau \in \mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$, $r.a = t.b$.

The $\mathcal{T}$-Dependency graph satisfies the two following properties:

- For all $i \le x$, there is an edge $((p \cdot u'_i, \alpha_i), (q \cdot v'_i, \beta_i))$ in $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$.

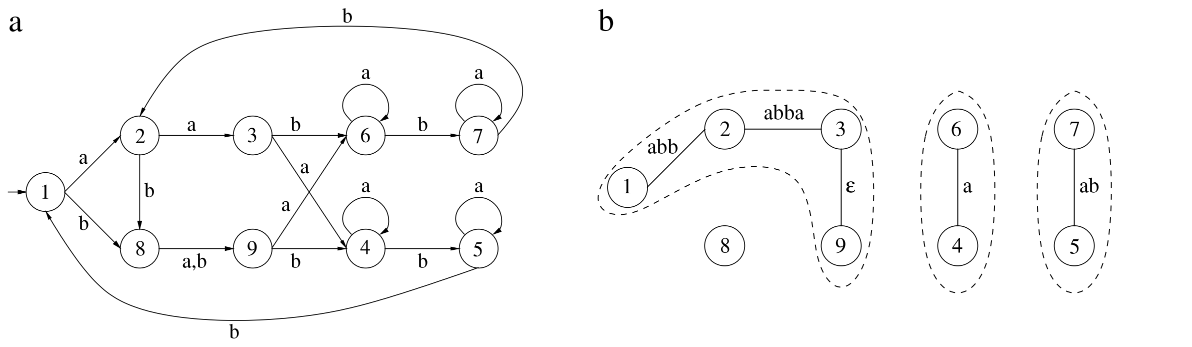

**Fig. 3.** (a) is a fixed transitions structure and (b) the $\mathcal{F}$-**dependency graph** for $p = 3$, $q = 9$ and $u = abbaa$. Thanks to (b), we know that all states in a same connected component will be either all final or all non-final. Hence, there are at most $2^4$ possible sets of final states, instead of $2^9$.

- For all $(w_1, w_2) \in \varphi_p$ (resp. $(w_3, w_4) \in \varphi_q$), we have $w_1 = w_1'a_1$ and $w_2 = w_2'a_2$ with $a_1, a_2 \in A$ and such that there is an edge $((p \cdot w_1', a_1), (p \cdot w_2', a_2))$ (resp. $((q \cdot w_3', a_3), (q \cdot w_3', a_3))$) in $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$.

Note that this dependency graph models an equivalence relation on transitions, based on the state in which they arrive.

**Lemma 2.** *If $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ contains an acyclic subgraph with j edges, then:*

$$|\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})| \leq \frac{|\mathcal{T}_n|}{n^j}$$

**Proof.** Two vertices in the same connected components of $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ model two transitions of the transitions structure, ending in the same arrival state. Hence if $x$ is the number of connected components in the graph, then $|\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})| \leq n^x$. If $G_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ contains an acyclic subgraph with exactly $j$ edges, then there is at most $kn - j$ connected components. □

## 2.5. The $\mathcal{F}$-dependency graph

In this subsection, we slightly modify the notion of dependency graph introduced in [3]. Let $\tau$ be a fixed transitions structure with $n$ states and $\ell$ be an integer such that $1 \leq \ell < n$. Let $p, q$ be two states of $\tau$ such that $p \neq q$ and $u$ a word of length $\ell$. We define $\mathcal{F}_\tau(p, q, u)$ as the set of sets of final states $F$ for which in the automaton $(\tau, F)$ the states $p$ and $q$ are separated by the word $u$. That is to say:

$$\mathcal{F}_\tau(p, q, u) = \{F \subset \{1, \dots, n\} \mid \text{ for } (\tau, F),\ p \sim_{|u|-1} q \text{ and } [\![p \cdot u \in F]\!] \neq [\![q \cdot u \in F]\!]\}$$

In order to obtain a upper bound on the cardinal of the set $\mathcal{F}_\tau(p, q, u)$, we define the undirected graph $\mathcal{G}_\tau(p, q, u)$, called the *$\mathcal{F}$-dependency graph*, as follows:

- its set of vertices is $\{1, \dots, n\}$, the set of states of $\tau$;
- there is an edge $(s, t)$ between two vertices $s$ and $t$ if and only if there exists a word $w$ of length less than $\ell$ such that $s = p \cdot w$ and $t = q \cdot w$ and for all $F \in \mathcal{F}_\tau(p, q, u)$, $[\![s \in F]\!] = [\![t \in F]\!]$.

The $\mathcal{F}$-dependency graph contains some information that is a basic ingredient of the proof: it is a convenient representation of necessary conditions for a set of final states to be in $\mathcal{F}_\tau(p, q, u)$, that is, for Moore's algorithm to require more than $|u|$ partition refinements because of $p$, $q$ and $u$. Fig. 3 shows an example of a $\mathcal{F}$-dependency graph.

**Lemma 3** ([3]). *If $\mathcal{G}_\tau(p, q, u)$ contains an acyclic subgraph with at least i edges, then $|\mathcal{F}_\tau(p, q, u)| \leq 2^{n-i}$.*

The notions of dependency graphs and dependency tree will be used in Sections 3.2 and 3.3 to obtain upper bounds on the cardinal of sets of automata with given properties and prove that their contributions to the average complexity is negligible.

## 3. Moore's algorithm: average case analysis

In [3], it is proved that the average complexity of Moore's state minimization algorithm is $\mathcal{O}(n \log n)$. Since the result is obtained by studying only properties on the sets of final states of automata minimized in a given complexity, it holds for any distribution on the set of transitions structures. In this paper, in order to improve the upper bound on the average complexity, we also have to study some properties of transitions structures. Since the enumeration of accessible automata with given properties is difficult, we focus our study on the uniform distribution over the set of complete deterministic automata.

## 3.1. Main result and decomposition of the proof

The main result of this paper is the following.

**Theorem 1.** *For any fixed integer $k \geq 2$ and for the uniform distribution over the deterministic and complete automata with $n$ states over a $k$-letter alphabet, the average complexity of Moore's state minimization algorithm is $\mathcal{O}(n \log \log n)$.*

Recall that the number of partition refinements made by Moore's state minimization algorithm is smaller or equal to $n-2$ and that $\mathcal{A}_n^i$ is the set of automata of $\mathcal{A}_n$ for which $i$ is the smallest integer such that $\sim_i$ is equal to Myhill–Nerode equivalence. The average number of iterations is given by:

$$\mathcal{N}_n = \frac{1}{|\mathcal{A}_n|}\left(\sum_{i=0}^{n-2}(i+1) \times |\mathcal{A}_n^i|\right)$$

We define $\lambda_n = \lceil \log_k \log_2 n^3 + 2 \rceil$, which will be used in the sequel. We gather the sets $\mathcal{A}_n^i$ , in order to obtain the following upper bound:

$$\mathcal{N}_n \leq \underbrace{\frac{\lambda_n + 1}{|\mathcal{A}_n|}\sum_{i \leq \lambda_n}|\mathcal{A}_n^i|}_{\textbf{(S1)}} + \underbrace{\frac{(5\log_2 n + 1)}{|\mathcal{A}_n|}\sum_{i=\lambda_n+1}^{5\log_2 n}|\mathcal{A}_n^i|}_{\textbf{(S2)}} + \underbrace{\frac{n-1}{|\mathcal{A}_n|}\sum_{i=5\log_2 n+1}^{n-2}|\mathcal{A}_n^i|}_{\textbf{(S3)}}$$

**(S1)** is less than $\lambda_n$ and therefore equal to $\mathcal{O}(\log \log n)$. In [3], it is proved that:

$$\sum_{i=5\log_2 n+1}^{n-2}|\mathcal{A}_n^i| \leq \frac{|\mathcal{A}_n|}{n}$$

Thus we know that **(S3)** is equal to $\mathcal{O}(1)$. Hence, in the following, we prove that:

$$\textbf{(S2)}\ \frac{(5\log_2 n + 1)}{|\mathcal{A}_n|}\sum_{i=\lambda_n+1}^{5\log_2 n}|\mathcal{A}_n^i| = \mathcal{O}(\log \log n) \tag{1}$$

For any $\ell > 0$, we define the set $\mathcal{A}_n^{>\ell}(p, q)$ as the set of automata with $n$ states, where the states $p$ and $q$ are separated during the $\ell$-th partition refinement:

$$\mathcal{A}_n(p, q, \ell) = \{(\tau, F) \in \mathcal{A}_n \mid \tau \subset \mathcal{T}_n,\ F \subseteq \{1, \dots, n\},\ p \sim_{\ell-1} q,\ p \nsim_\ell q\}$$

**Remark 1.** Note that if in the automaton $(\tau, F)$, for all letter $a \in A$, $p \cdot a = q \cdot a$, then either $p \nsim_0 q$ or $p \sim q$. Therefore $(\tau, F) \notin \mathcal{A}_n(p, q, \ell)$ with $\ell > 0$. Consequently, in the remainder of the proof, in all sets of transitions structures where $p$ and $q$ are fixed, there exists a letter $a$ such that $p \cdot a \neq q \cdot a$.

The following statement comes from the definition of the sets it involves:

$$\bigcup_{i>\lambda_n}\mathcal{A}_n^i = \bigcup_{p,q\in\{1,\dots,n\}}\mathcal{A}_n(p, q, \lambda_n + 1)$$

Let $\tau$ be a transitions structure, and $p$, $q$ be two distinct states. If $A$ is a $k$-letter alphabet, and $\mu$ a positive integer, we define two properties associated to transitions structures:

(1) *noIntersection*$(\tau, p, q)$ is *true* if and only if $s_{\lambda_n-2}(p) \cap s_{\lambda_n-2}(q) = \emptyset$.
(2) *largeTree*$(\tau, p, \mu)$ is *true* if and only if $\mathcal{R}(p)$ contains at most one leaf of depth less or equal to $\mu$. Note that this implies that for all integer $i$, with $i \leq \mu$, $|s_i(p)| \geq k^i - 1$. Indeed, if $\mathcal{R}(p)$ contains at most one leaf of depth less or equal to $\mu$, then there exist $k-1$ letters $a \in A$ such that $\mathcal{R}(p \cdot a)$ does not contain any leaf of depth less than $\mu$. Therefore we have $|S_i(p)| \geq k^i - 1$. Fig. 4 illustrates this proof.

For fixed states $p$ and $q$, we partition the set of transitions structures $\mathcal{T}_n$ in three subsets:

- $\mathcal{X}_n$ **is the set of all transitions structures $\tau$ such that**:
  - there exists a state $r \in Q$ such that *largeTree*$(\tau, r, \lambda_n)$ is *false*.

  Note that this set does not rely on the values of $p$ and $q$.
- $\mathcal{Y}_n(p, q)$ **is the set of transitions structures $\tau$ such that**:
  - for all state $r \in Q$, the property *largeTree*$(\tau, r, \lambda_n)$ is *true*,
  - for all words $w \in A^{\leq 2}$, the property *noIntersection*$(\tau, p \cdot w, q \cdot w)$ is *false*.
- $\alpha_n(p, q)$ **is the set of transitions structures $\tau$ such that**:
  - for all state $r \in Q$, the property *largeTree*$(\tau, r, \lambda_n)$ is *true*,
  - there exists a word $w \in A^{\leq 2}$ such that *noIntersection*$(\tau, p \cdot w, q \cdot w)$ is *true*.

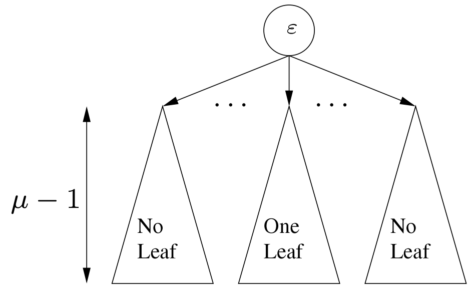

**Fig. 4.** For any dependency tree $\mathcal{R}(r)$, if there exists at most one leaf of depth less or equal to $\mu$, since there are $k$ transitions starting from $r$, then there are $k-1$ subtrees that do not contain any leaf of depth less than $\mu$, that is to say each one contains $\frac{k^{\mu}-1}{k-1}$ nodes.

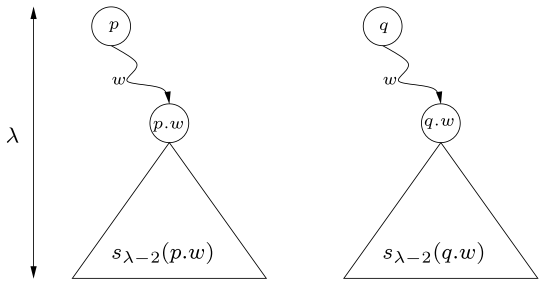

**Fig. 5.** For all transitions structures in $\alpha_n(p, q)$, there exists a word $w$ of length 2 such that the sets of states, reached following paths starting in $p \cdot w$ and $q \cdot w$ and labeled by words of length less or equal to $\lambda_n - 2$, are disjoint and both have a cardinal greater or equal to $\log n^3 - 1$.

According to those three definitions, for all fixed states $p$ and $q$, the following equalities hold:

$$\mathcal{T}_n = \mathcal{X}_n \cup \mathcal{Y}_n(p, q) \cup \alpha_n(p, q)$$
$$\mathcal{X}_n \cap \mathcal{Y}_n(p, q) \cap \alpha_n(p, q) = \emptyset$$

In the following, we study each of those sets separately in order to obtain combinatorial properties or, more precisely, in order to obtain an upper bound on the number of automata in a set $\mathcal{A}_n(p, q, \lambda_n + 1)$. We show that:

- if a transitions structure $\tau$ is in a set $\alpha_n(p, q)$, there exist only a few sets of final states $F$ such that $(\tau, F) \in \mathcal{A}_n(p, q, \lambda_n+1)$,
- there exist a few automata whose transitions structures is in $\mathcal{X}_n$ or in a set $\mathcal{Y}_n(p, q)$.

## 3.2. *Transitions structures with a huge $\mathcal{F}$-dependency graph*

Fig. 5 illustrates the properties of transitions structures in $\alpha_n(p, q)$.

**Lemma 4.** *For any fixed transitions structures $\tau \in \alpha_n(p, q)$, and a fixed word $u$ of length $\lambda_n + 1$, the following property holds: every $\mathcal{F}$-dependency graph $\mathcal{G}_\tau(p, q, u)$ contains an acyclic subgraph with at least $k^{\lambda_n-2} - 1$ edges.*

**Proof.** We recall that for all transitions structure $\tau$ in the set $\alpha_n(p, q)$, there exists a word $w \in A^{\leq 2}$ such that the property *noIntersection*$(\tau, p \cdot w, q \cdot w)$ is *true*. If there exist several words of length at most 2 satisfying this property, $w$ denotes the smallest in the lexicographical order. Let $\mathcal{G}'$ be the subgraph $\mathcal{G}_\tau(p, q, u)$ defined as follows: there exists an edge $(s, t)$ in $\mathcal{G}'$, if and only if $s = p \cdot wv$ and $t = q \cdot wv$, where $w$ is the word previously mentioned and $v \in S_{\lambda_n-2}(p \cdot w)$, i.e. $v$ labels a node in the tree $\mathcal{R}(p \cdot w)$. The subgraph $\mathcal{G}'$ contains exactly $|s_{\lambda_n-2}(p \cdot w)|$ edges, since for all $v \in S_{\lambda_n-2}(p \cdot w)$, the states $p \cdot wv$ are all pairwise distinct. Since *largeTree*$(\tau, r, \lambda_n)$ is true for all state $r \in Q$, we have $|s_{\lambda_n-2}(p \cdot w)| \geq k^{\lambda_n-2} - 1$. As every edge in $\mathcal{G}'$ has exactly one tip in $S_{\lambda_n-2}(p \cdot w)$ and as the degree of each vertex in $S_{\lambda_n-2}(p \cdot w)$ is equal to 1, the subgraph $\mathcal{G}'$ is acyclic. □

**Corollary 1.** *For $\lambda_n = \lceil \log_k \log_2 n^3 + 2 \rceil$, the number of automata belonging to $\mathcal{A}_n(p, q, \lambda_n + 1)$ and whose transitions structures are in $\alpha_n(p, q)$ is $\mathcal{O}\left(|\mathcal{T}_n| \frac{2^n \log n}{n^3}\right)$.*

**Proof.** This follows directly from Lemmas 3 and 4: for any distinct states $p$ and $q$, any word $u$ of length $\lambda_n + 1$, and any fixed transitions structure $\tau \in \alpha_n(p, q)$, we have

$$|\mathcal{F}_\tau(p, q, u)| = \mathcal{O}\left(2^{n - \log_2 n^3}\right)$$

For a fixed transitions structure $\tau \in \alpha_n(p,q)$, since the number of words of length $\lambda_n + 1$ is $\mathcal{O}(\log n)$, the number of choices of sets of final states such that the automata are in $\mathcal{A}_n(p,q,\lambda_n+1)$ is bounded above by:

$$\sum_{u \in A^{\lambda_n+1}} |\mathcal{F}_\tau(p,q,u)| = \mathcal{O}\left(\frac{2^n \log n}{n^3}\right)$$

This upper bound being valid for all transitions structures in $\alpha_n(p,q)$, we obtain the announced result. □

## 3.3. Negligible sets of transitions structures

**Lemma 5.** *The number of transitions structures in $\mathcal{X}_n$ is $\mathcal{O}\left(|\mathcal{T}_n| \frac{\log^5 n}{n}\right)$.*

**Proof.** For a fixed state $r$ and a fixed integer $\mu \in \{1, \ldots, \lambda_n\}$, we define the sets $\mathcal{X}_n(r,\mu)$ of all transitions structures $\tau$ for which $\mu$ is the smallest integer such that the property *largeTree*$(\tau, r, \mu)$ is false. We have:

$$\mathcal{X}_n = \bigcup_{r \in \{1,\ldots,n\}} \left( \bigcup_{\mu \in \{1,\ldots,\lambda_n\}} \mathcal{X}_n(r,\mu) \right) \text{ and } \bigcap_{\mu \in \{1,\ldots,\lambda_n\}} \mathcal{X}_n(r,\mu) = \emptyset$$

For all transitions structures in $\mathcal{X}_n(r,\mu)$, the dependency tree $\mathcal{R}(r)$ contains at least two leaves of depth less or equal to $\mu$ and at least one leaf of depth less than $\mu$. In order to obtain an upper bound on the cardinal of a set $\mathcal{X}_n(r,\mu)$, we partition the set of possible dependency trees $\mathcal{R}(r)$ into two subsets:

1. All leaves are at level $\mu$. Let $k$ be the size of the alphabet and $f$ the number of leaves, the number of trees of this kind is equal to:

$$\sum_{f=2}^{k^\mu} \binom{k^\mu}{f}$$

2. There exists exactly one leaf of depth $h$ (with $h < \mu$), and at least one leaf of depth $\mu$. The number of trees of this kind is at most:

$$\sum_{h=1}^{\mu-1} \left( k^h \sum_{f=1}^{k^\mu} \binom{k^\mu}{f} \right)$$

In the remainder of the proof, we suppose $n$ to be large. Using the upper bound of Lemma 1 on the number of transitions structures associated to a dependency tree, we obtain:

$$|\mathcal{X}_n(r,\mu)| \le \sum_{f=2}^{k^\mu} \left[ \binom{k^\mu}{f} |\mathcal{T}_n| \left(\frac{|S_\mu(r)|}{n}\right)^f \right] + \sum_{h=1}^{\mu-1} \left( k^h \sum_{f=1}^{k^\mu} \left[ \binom{k^\mu}{f} |\mathcal{T}_n| \left(\frac{|S_\mu(r)|}{n}\right)^{f+1} \right] \right)$$

Since $\mu \le \lambda_n$, we have $|S_\mu(r)| < k^{\lambda_n+1}$ and:

$$|\mathcal{X}_n(r,\mu)| < |\mathcal{T}_n| \left( \sum_{f=2}^{k^\lambda_n} \left[ \binom{k^\lambda_n}{f} \left(\frac{k^{\lambda_n+1}}{n}\right)^f \right] + \frac{\lambda_n k^{2\lambda_n+1}}{n} \sum_{f=1}^{k^\lambda_n} \left[ \binom{k^\lambda_n}{f} \left(\frac{k^{\lambda_n+1}}{n}\right)^f \right] \right)$$

Since we have $\binom{k^\lambda_n}{f} \left(\frac{k^{\lambda_n+1}}{n}\right)^f \le \left(\frac{k^{2\lambda_n+1}}{n}\right)^f$:

$$|\mathcal{X}_n(r,\mu)| < |\mathcal{T}_n| \left( \frac{k^{4\lambda_n+2}}{n^2} \sum_{f=0}^{\infty} \left(\frac{k^{2\lambda_n+1}}{n}\right)^f + \frac{\lambda_n k^{4\lambda_n+2}}{n^2} \sum_{f=0}^{\infty} \left(\frac{k^{2\lambda_n+1}}{n}\right)^f \right)$$

Since $\sum_{f=0}^{\infty} \left(\frac{k^{2\lambda_n+1}}{n}\right)^f = \mathcal{O}(1)$, we have:

$$|\mathcal{X}_n(r,\mu)| = \mathcal{O}\left(|\mathcal{T}_n| \frac{\lambda_n k^{4\lambda_n}}{n^2}\right) = \mathcal{O}\left(|\mathcal{T}_n| \frac{\log^4 n^3 \times \log\log n^3}{n^2}\right)$$

Since this upper bound holds for any $\mu \in \{1, \ldots, \lambda_n\}$ and any $r \in Q$, we obtain:

$$|\mathcal{X}_n| \le \left( \sum_{r \in \{1,\ldots,n\}} \sum_{\mu \in \{1,\ldots,\lambda_n\}} |\mathcal{X}_n(r,\mu)| \right) = \mathcal{O}\left(|\mathcal{T}_n| \frac{\log^4 n^3 \times \log^2 \log n^3}{n}\right) \quad \square$$

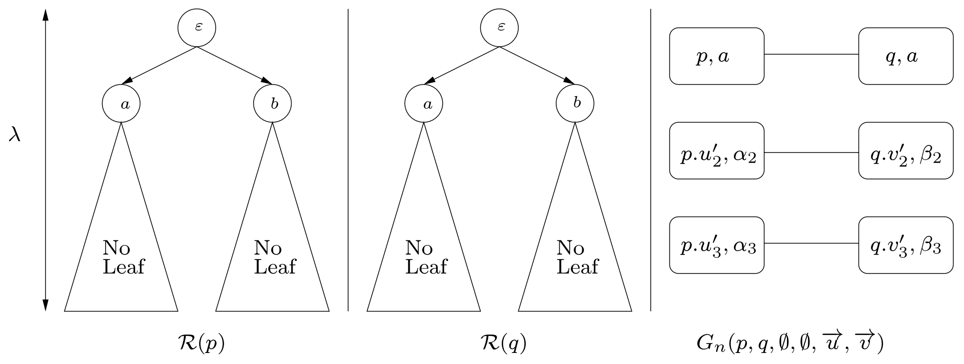

**Fig. 6.** First case of the study of $\mathcal{Y}_n(p, q)$: in this example, $u_1 = v_1 = a$, $u_2$ and $v_2$ have a common prefix $ba$, $u_3$ and $v_3$ have a common prefix $bb$.

**Lemma 6.** *For any distinct states p and q, the number of transitions structures in $\mathcal{Y}_n(p, q)$ is $\mathcal{O}\left(|\mathcal{T}_n| \frac{\log^6 n}{n^3}\right)$.*

**Proof.** Recall that for two fixed states $p$ and $q$ and all transitions structure in $\mathcal{Y}_n(p, q)$, we have:

(i) For any state $r \in Q$, the property *largeTree*$(\tau, r, \lambda_n)$ is satisfied and the tree $\mathcal{R}(r)$ contains at most one leaf of depth less than $\lambda_n$.

(ii) For any word $w$ of length less than 2, the property *noIntersection*$(\tau, p \cdot w, q \cdot w)$ is not satisfied.

We prove that for any transitions structure in $\mathcal{Y}_n(p, q)$, the associated $\mathcal{T}$-dependency graph contains an acyclic subgraph with at least 3 edges and use Lemma 2 to obtain the announced result. Recall that a $\mathcal{T}$-dependency graph is defined using to tuples of words $\overrightarrow{u}$ and $\overrightarrow{v}$. In the following, those tuples only contains words that are a label in a node of $\mathcal{R}(p)$ and $\mathcal{R}(q)$. Consequently, the set of edges in the $\mathcal{T}$-dependency graph, induced by tuples $\overrightarrow{u}$ and $\overrightarrow{v}$, forms a matching, meaning that the edges are pairwise disjoint. Indeed, for two distinct words $u_i$ and $u_j$ in the tuple $\overrightarrow{u}$, if there existsd a path in the $\mathcal{T}$-dependency graph between the two nodes associated to $u_i$ and $u_j$, then $p \cdot u_i = p \cdot u_j$. This is impossible since both words are distinct and both label a node in $\mathcal{R}(p)$. The same applies on $\overrightarrow{v}$ by symmetry.

We shall partition the proof according to the presence/absence of a leaf in $\mathcal{R}(p)$ and $\mathcal{R}(q)$. In each case, we characterize a set of transitions structures that satisfy necessary conditions to be in $\mathcal{Y}_n(p, q)$ and obtain an upper bound on the cardinal of this set. Therefore some transitions structures might be counted twice but this is sufficient for what we want to prove.

*Both trees do not contain a leaf of depth less or equal to $\lambda_n$.* For any word $w$ of length less than 2, the property *noIntersection*$(\tau, p \cdot w, q \cdot w)$ is not satisfied, more precisely:

- either $|w| = 1$ ($w$ is a letter). Let $E \subsetneq A$ be a set of $e$ letters such that for all $a \in E$, we have $p \cdot a = q \cdot a$. According to Remark 1, we have $e < k$.
- or $|w| \geq 2$. For all letters $b \in A \setminus E$ and $c \in A$, the property *noIntersection*$(\tau, p \cdot bc, q \cdot bc)$ is false. There are exactly $k(k - e)$ such words of length 2.

Let $\overrightarrow{u}$ and $\overrightarrow{v}$ be two tuples of words of size $x = e + k(k - e)$, such that:

- for all integers $j$ with $1 \leq j \leq e$, we have $u_j = v_j = a_j$ with $a_j \in E$,
- for all integers $i$ with $e < i \leq x$, the words $u_i$ and $v_i$ have a common prefix $ab$ with $a \in A \setminus E$.

Any transitions structure satisfying those properties can be associated to a set $\mathcal{T}_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$. We compute an upper bound on the number of transitions structures associated to such sets.

- First, we obtain a bound on the number of possible tuples $\overrightarrow{u}$ and $\overrightarrow{v}$. There are $2^k - 1$ possible subsets $E$ and less than $k^{2\lambda_n(x-e)}$ possible choices for $u_i, v_i \in A^{\leq \lambda_n}$, for $e < i \leq x$.
- Now we obtain an upper bound on the cardinal of a fixed set $\mathcal{T}_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$. According to previous arguments, the set of edges implied by $\overrightarrow{u}$ and $\overrightarrow{v}$ forms a matching. Thus, $G_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$ contains an acyclic graph with $x$ edges. Fig. 6 illustrates the dependency trees and the $\mathcal{T}$-dependency graph associated to this subset. Therefore, according to Lemma 2, there is at most $\frac{|\mathcal{T}_n|}{n^x}$ transitions structures in a set $\mathcal{T}_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$.

Using previous remarks, we obtain the following upper bound for the cardinal of this subset of $\mathcal{Y}_n(p, q)$:

$$(2^k - 1) \times k^{2\lambda_n(x-e)} \times \frac{|\mathcal{T}_n|}{n^x}$$

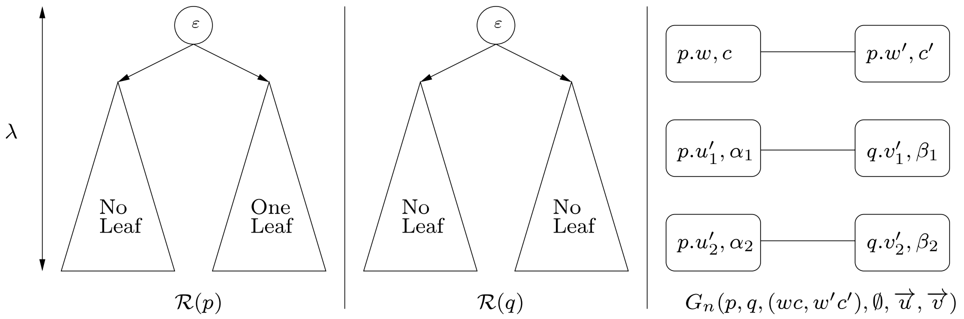

**Fig. 7.** Second case of the study of $\mathcal{Y}_n(p, q)$: in this example, $w$ labels the leaf in $\mathcal{R}(p)$, $u_1$ and $v_1$ have a common prefix $a$, $u_2$ and $v_2$ have a common prefix $b$.

As $x \geq k+1$ (for $e = k-1$). We obtain a new upper bound:

$$\begin{aligned}
&(2^k - 1) \times \frac{|\mathcal{T}_n|}{n^{k+1}} \times k^{2k\lambda_n} \\
&\leq (2^k - 1) \times \frac{|\mathcal{T}_n| k^{4\lambda_n}}{n^3} \times \left(\frac{k^{2k\lambda_n}}{n}\right)^{k-2} \\
&= \mathcal{O}\left(\frac{|\mathcal{T}_n| k^{4\lambda_n}}{n^3}\right).
\end{aligned}$$

*Only one tree contains a leaf of depth less or equal to $\lambda_n$.* Assume by symmetry that the unique leaf is in $\mathcal{R}(p)$.

- Let $wc$ be the word labeling the leaf in $\mathcal{R}(p)$ and $w'c'$ a word such that $w'c' <_{mil} wc$ and $p \cdot wc = p \cdot w'c'$ ($c, c' \in A$).
- For any letter $b \in A$, the property *noIntersection*$(\tau, p \cdot b, q \cdot b)$ is *false*.

We characterize the $k$-tuples $\overrightarrow{u}$ and $\overrightarrow{v}$. For all $i \leq k$, each $v_i$ starts with $k$ distinct letters and $u_i \in A^*$. Any transitions structure satisfying those properties can be associated to a set $\mathcal{T}_n(p, q, \{(wc, w'c')\}, \emptyset, \overrightarrow{u}, \overrightarrow{v})$. We compute an upper bound on the number of transitions structures associated to such sets.

- There are fewer than $k^{2\lambda_n(k+1)}$ possible choices for the words $wc, w'c' \in A^{\leq \lambda_n}$ and $u_i, v_i \in A^{\leq \lambda_n}$, such that $i \leq k$.
- Since all words $v_i$ label a node in $\mathcal{R}(p)$, we use the same arguments as in the previous case and obtain that the subgraph composed with the $k$ edges given by $\overrightarrow{u}$ and $\overrightarrow{v}$ is acyclic. By adding the edge between the vertices associated to $p \cdot wc$ and $p \cdot w'c'$, it is impossible to create a cycle. Hence, the subgraph $G_n(p, q, \{(wa, w'a')\}, \emptyset, \overrightarrow{u}, \overrightarrow{v})$ contains an acyclic subgraph with $k+1$ edges (see Fig. 7).

Using previous remarks, taking into account that the leaf can either be in $\mathcal{R}(p)$ or $\mathcal{R}(q)$, we obtain the following upper bound:

$$2 \times k^{2\lambda_n(k+1)} \times \frac{|\mathcal{T}_n|}{n^{k+1}}$$

which matches the upper bound stated in Lemma 6.

*Both trees contain a leaf of depth less or equal to $\lambda_n$.* Let $w_p a_p$ (resp. $w_q a_q$) be the label of the leaf in $\mathcal{R}(p)$ (resp. $\mathcal{R}(q)$) and $w'_p a'_p$ the word such that $w'_p a'_p <_{mil} w_p a_p$ and $p \cdot w_p a_p = p \cdot w'_p a'_p$. For all letters $b_i \in A$, such that $b_i$ is not a prefix of $w_p a_p$, the property *noIntersection*$(\tau, p \cdot b_i, q \cdot b_i)$ is *false*. For $\overrightarrow{u}$ and $\overrightarrow{v}$ of size $k-1$, we consider that for all $i \leq k-1$, the letter $b_i \in A$ is prefix of both words $u_i$ and $v_i$. Any transitions structure satisfying those properties can be associated to a set $\mathcal{Y}_n(p, q, (w_p, w'_p), (w_q, w'_q), \overrightarrow{u}, \overrightarrow{v})$. There are less than $k^{2\lambda_n(k+1)}$ possible choices for the pairs $(w_p, w'_p)$, $(w_q, w'_q)$ and the pairs $(u_i, v_i)$, for all $i \leq (k-1)$. Using the same argument as in the previous case, one can prove that in the subgraph composed with the $k-1$ edges $((p \cdot u'_i, \alpha_i), (q \cdot v'_i, \beta_i))$ is a perfect matching. The word $w'_p$ (resp. $w'_q$) and all words $u_i$ (resp. $v_i$) label a total of $k$ distinct nodes in $\mathcal{R}(p)$ (resp. $\mathcal{R}(q)$). If we add the edge between the vertices associated to $p \cdot w_p$ and $p \cdot w'_p$ (resp. $q \cdot w_q$ and $q \cdot w'_q$), there can not exist a path between the vertex associated to $p \cdot w'_p$ (resp. $q \cdot w'_q$) and a vertex $(p \cdot u'_i, \alpha_i)$ (resp. $(q \cdot v'_i, \beta_i)$). Hence, the subgraph $G_n(p, q, \{(w_p, w'_p)\}, \{(w_q, w'_q)\}, \overrightarrow{u}, \overrightarrow{v})$ contains an acyclic subgraph with $k+1$ edges (see Fig. 8).

Using the same calculus as before, we obtain the announced upper bound. □

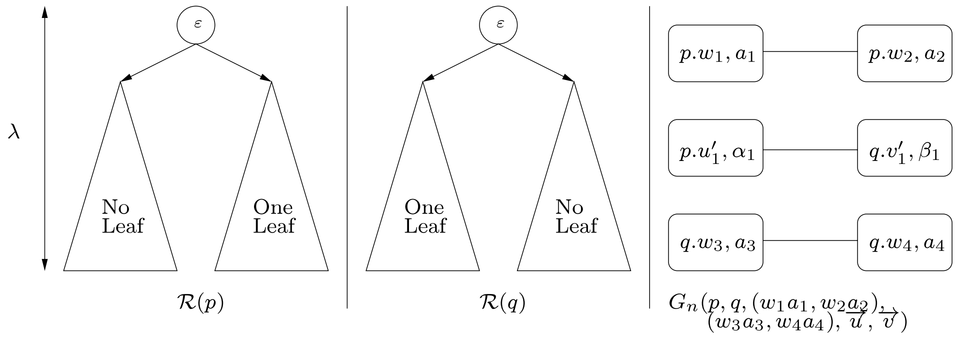

**Fig. 8.** Third case of the study of $\mathcal{Y}_n(p, q)$: $w_1$ labels the leaf in $\mathcal{R}(p)$, $w_3$ labels the leaf in $\mathcal{R}(q)$.

## 3.4. Concluding the proof

Recall that we want to prove that:

$$\frac{(5\log_2 n + 1)}{|\mathcal{A}_n|}\sum_{i=\lambda_n+1}^{5\log_2 n}|\mathcal{A}_n^i| = \mathcal{O}(\log\log n)$$

We define $\widetilde{\mathcal{X}_n}$, $\widetilde{\alpha_n}(p, q)$ and $\widetilde{\mathcal{Y}_n}(p, q)$ as the sets of automata whose transitions structures are respectively in $\mathcal{X}_n$, $\alpha_n(p, q)$ and $\mathcal{Y}_n(p, q)$. We have:

$$\bigcup_{i>\lambda_n}\mathcal{A}_n^i = \bigcup_{p,q\in\{1,\ldots,n\}}\mathcal{A}_n(p, q, \lambda_n + 1) \subseteq \widetilde{\mathcal{X}_n} \cup \left(\bigcup_{p,q\in\{1,\ldots,n\}}\widetilde{\alpha_n}(p, q) \cup \widetilde{\mathcal{Y}_n}(p, q)\right)$$

Using Lemmas 4–6 we obtain:

$$\begin{aligned}\sum_{i>\lambda_n}|\mathcal{A}_n^i| &\le 2^n|\mathcal{T}_n|\frac{\log^5 n}{n^3} + \sum_{p,q\in\{1,\ldots,n\}}2^n|\mathcal{T}_n|\left(\frac{\log n}{n^3} + \frac{\log^6 n}{n^3}\right)\\ &= \mathcal{O}\left(|\mathcal{A}_n| \times \frac{\log^6 n}{n}\right),\end{aligned}$$

and therefore

$$\frac{(5\log_2 n + 1)}{|\mathcal{A}_n|}\sum_{i=\lambda_n+1}^{5\log_2 n}|\mathcal{A}_n^i| = \mathcal{O}\left(\frac{\log^7 n}{n}\right) = \mathcal{O}(\log\log n)$$

Hence $\mathcal{N}_n = \mathcal{O}(\log\log n)$, this concludes the proof of the Theorem 1.

## 3.5. The Bernouilli distribution

**Theorem 2.** *Let $k$ be a fixed integer such that $k \ge 2$. For the distribution where the transitions structure is chosen uniformly amongst the deterministic and complete transitions structures with $n$ states over a $k$-letter alphabet, and where each state has a probability $x \in ]0, 1[$ to be final, the average complexity of Moore's state minimization algorithm is $\mathcal{O}(n\log\log n)$.*

**Proof.** Let $x$ be a fixed real number with $0 < x < 1$. Consider the distribution on the sets of final states defined such that each state as a probability $x$ of being final. For any fixed transitions structure, the probability of a given subset $F$ of $\{1, \ldots, n\}$ is $\mathbb{P}(F) = x^{|F|}(1 - x)^{n-|F|}$. For fixed parameters $\tau, p, q, u$, let $P_\tau(p, q, u)$ be the probability for a set of final states to be in $\mathcal{F}_\tau(p, q, u)$. Since $G_\tau(p, q, u)$ only models a subset of constraints on the set $\mathcal{F}_\tau(p, q, u)$, $P_\tau(p, q, u)$ is less than the probability for a set of final states to verify the constraints given by the $\mathcal{F}$-dependency graph.

Let $r$ be the real number defined by $r = \max\{x, 1 - x\} \in [1/2, 1[$. In [3], the authors prove that for this distribution, the contribution to the average number of partition refinements given by automata minimized in more than $5\log_{1/r} n$ is $O(1)$. We define the constant $\lambda_{n,r} = \lceil\log_k\log_{1/r} n + 2\rceil$. The contribution of automata minimized in less than $\lambda_{n,r}$ partition refinements is $\mathcal{O}(\log\log n)$. We slightly modify the definition of the sets $\widetilde{\mathcal{Y}_n}(p, q)$ and $\widetilde{\mathcal{X}_n}$: we replace $\lambda_n$ by $\lambda_{n,r}$. Using the same method, one can prove that their contribution does not change from the previous case.

Hence, what we need to prove is:

$$\frac{(5\log n + 1)}{|\mathcal{T}_n|} \times \sum_{p,q\in\{1,\ldots,n\}}\left(\sum_{\tau\in\alpha_n(p,q)}\left[\sum_{u\in A^{\lambda_{n,r}}}P_\tau(p, q, u)\right]\right) = \mathcal{O}(\log\log n)$$

Let $m$ be the number of connected components in $\mathcal{G}'$ containing at least two nodes, and $c_1, \ldots, c_m$ the sizes of these components. Since $\mathcal{G}'$ is acyclic and contains exactly $\log_{1/r} n^3$ edges, the following equality holds:

$$\sum_{i=1}^{m} c_i = m + \log_{1/r} n^3$$

The probability that a random set satisfy the conditions implied by the $i$-th connected component is equal to $x^{c_i} + (1-x)^{c_i}$ (either all states of the subset are final, either none). For all $n \geq 1, x^n + (1-x)^n \leq r^{n-1}(x+1-x) = r^{n-1}$. Hence

$$P_\tau(p, q, u) \leq \prod_{i=1}^{m} r^{c_i - 1} = r^{\sum_{i=1}^{m}(c_i-1)} = \left(\frac{1}{r}\right)^{-\log_{1/r} n^3}$$

Therefore we obtain:

$$\frac{(5 \log n + 1)}{|\mathcal{T}_n|} \times \sum_{p,q \in \{1,\ldots,n\}} \left( \sum_{\tau \in \alpha_n(p,q)} \left[ \sum_{u \in A^{\lambda n, r}} P_\tau(p, q, u) \right] \right) = \mathcal{O}\left(\frac{\log n}{n}\right),$$

concluding the proof. □

## 3.6. Generic complexity

In this section, we study the generic complexity of Moore's algorithm. We recall that the notion of generic complexity has been defined in [19]. Let $E$ be the set of inputs of an algorithm. A size parameter is to be determined on $E$: let $E_n$ be the set of inputs of size $n$ and $B_n = \bigcup_{i \leq n} E_i$ be the set of inputs of size at most $n$. Let $X$ be a set of inputs. For a fixed probability distribution, $X$ is *negligible* if and only if, for all instances $e$ of $B_n$, the probability that $e$ belongs $X$ tends to 0 when $n$ tends to infinity which, for the uniform distribution, is equivalent to:

$$\lim_{n \to +\infty} \frac{|X \cap B_n|}{|B_n|} = 0$$

and $X$ is *generic* if and only if for all instances $e$ of $B_n$, the probability that $e$ belongs $X$ tends to 1 which, for the uniform distribution, is equivalent to:

$$\lim_{n \to +\infty} \frac{|X \cap B_n|}{|B_n|} = 1.$$

The generic complexity of an algorithm is the worst-case complexity of a generic set of inputs of the algorithm.

**Remark 2.** In the particular case where

$$\lim_{n \to +\infty} \frac{|B_{n-1} \cap B_n|}{|B_n|} = 0$$

in order to show that $X$ is generic for the uniform distribution on $B_n$, it is sufficient to prove that

$$\lim_{n \to +\infty} \frac{|X \cap E_n|}{|E_n|} = 1$$

and in order to show that $X$ is negligible for the uniform distribution on $B_n$, it is sufficient to prove that

$$\lim_{n \to +\infty} \frac{|X \cap E_n|}{|E_n|} = 0$$

In order to study the generic complexity of Moore's algorithm, we start by identifying the negligible sets. We recall that $\lambda_n = \lceil \log_k \log_2 n^3 + 2 \rceil$.

**Lemma 7.** *For the uniform distribution on the set of complete deterministic with n states over a finite k-letter alphabet, the set of automata minimized in more than $\lambda_n$ partition refinements is negligible.*

**Proof.** According to the proof of Theorem 1, we have:

$$\sum_{i > \lambda_n} |\mathcal{A}_n^i| = \mathcal{O}\left( |\mathcal{A}_n| \frac{\log^6 n}{n} \right)$$

Hence we have:

$$\lim_{n \to +\infty} \frac{\sum_{i > \lambda_n} |\mathcal{A}_n^i|}{|A_n|} = 0$$

Moreover, for a fixed $k$-letter alphabet,

$$\lim_{n\to+\infty}\frac{\sum_{i=\lambda_{n-1}}|\mathcal{A}_i|}{\sum_{j=\lambda_n}|A_j|}\leq\lim_{n\to+\infty}\frac{1}{n^{k-1}}=0$$

We conclude the proof using Remark 2. □

**Theorem 3.** *For the uniform distribution on the set of complete deterministic with $n$ states over a finite $k$-letter alphabet, the generic complexity of Moore's state minimization algorithm is $\mathcal{O}(n\log\log n)$.*

**Proof.** According to Lemma 7, the set of minimized in less than $\lambda_n$ partition refinements is generic, which concludes the proof. □

In the following section, we use the results on Moore's algorithm to obtain similar results on Hopcroft's algorithm average and generic complexities.

## 4. Hopcroft's state minimization algorithm

The reader is invited to consult [21] for a detailed description of the implementation of Hopcroft's algorithm. Before explaining the algorithm, note that a pair $(C, a)$ represents the set of all the transitions labeled by $a$ ending in a state of the equivalence class $C$ and is called a *block*. Also, a class $B$ in the partition is `splitted` by a block $(C, a)$, extracted from a `waiting set`, into two classes $B'$ and $B''$, if $B' = \{q \mid q \in B, q\cdot a \in C\}$ and $B'' = \{q \mid q \in B, q\cdot a \notin C\}$ with $B'$ and $B''$ both non-empty. In the usual description of the algorithm, the order of insertions and extractions of blocks $(C, a)$ in the waiting set is not defined, since the algorithm is correct for any order. We introduce a description of Hopcroft's algorithm, in which a partial order is determined, for which the algorithm easier to compare with Moore's algorithm. All possible implementations of this new description were already feasible with the classical algorithm, since we will only guarantee a partial order, for the extraction and the insertion, between certain blocks. Therefore we will provide neither proof of its correctness nor proof of its worst case complexity.

### 4.1. Redescription of Hopcroft's algorithm

Though Hopcroft's algorithm computes the Myhill–Nerode equivalence, it is not easy to know, at a given step of the algorithm, if two states inside the same class are $i$-equivalent, for any integer $i \in \{1, \ldots, n-2\}$. In the following description (see Fig. 9), we solve this problem by dividing the waiting set into a `Current` waiting set, and a `Next` waiting set. The algorithm is initialized with the partition $\{F, Q\setminus F\}$, just like Moore's algorithm. The initial waiting set contains all blocks $(min\{F, Q\setminus F\}, a)$ (Algorithm 3, Lines 1–2). This algorithm contains two loops:

- Lines 6–11: A block $(P, a)$ is taken from `Current` (Line 7). If a class $B$ is splitted by $(P, a)$, we replace it by $B'$ and $B''$ in the partition and for all letters $a$ in the alphabet, we `Update` the waiting sets as follows (Algorithm 4): if the block $(B, a)$ was already `Current`, we replace it by $(B', a)$ and $(B'', a)$. If the block was in `Next`, we replace it by $(B', a)$ and $(B'', a)$. Otherwise, we simply add $(min\{B', B''\}, a)$ in the set `Next`. Note that these are the only ways to add a block in a waiting set. In the remainder of this section, we call this block of instructions an **iteration** of Hopcroft's algorithm, by analogy with Moore's algorithm.
- Lines 3–11: as long as the waiting set `Next` is not empty, meaning that at least one class has been splitted during the previous iteration, the waiting sets are swapped, and a new iteration starts.

The algorithm ends when both waiting sets are empty. Like in the classical description, this implies that the partition corresponds to the Myhill–Nerode equivalence.

Since a block $(C, a)$ is the set of all transitions ending in a state of class $C$, a waiting set can be seen as a set of transitions: the union of all the blocks it contains.

**Lemma 8.** *For any deterministic automaton, at the beginning of any iteration of Hopcroft's algorithm, the following properties hold:*

(i) *The waiting set `Current` contains at most $kn$ transitions.*
(ii) *For each block $(B, a)$ in `Current`, the class $B$ is the result of a split made at the previous iteration.*
(iii) *The set of transitions contained in `Current` is exactly the set of transitions that will be used to split the partition, during the current iteration.*

**Proof.** The property (i) is obvious since the automaton is deterministic: there is at most $kn$ transitions in an automaton and each of them belongs to at most one block otherwise it would imply that the transition ends in two states of two different classes.

The property (ii) follows directly from the description of the algorithm: a transition is in a block $(C, a)$ of `Current` at the beginning of an iteration if and only if it was added in `Next` during the previous iteration, which happens only in the function `Update`, when a class has been splitted.

```
Algorithm 3: Hopcroft
Data: $Partition = \{F, \overline{F}\}, C = min\{F, \overline{F}\}, Current = \emptyset, Next = \emptyset$
foreach $a \in A$ do
    Add $(C, a)$ in $Next$
while $Next \neq \emptyset$ do
    $Current = Next$
    $Next = \emptyset$
    while $Current \neq \emptyset$ do
        $(P, a) = First(Current)$
        foreach $B \in Partition$ such that $B$ is refined by $(P, a)$ do
            $B', B'' \leftarrow Refine(B, P, a)$
            $Breakblock(B, B', B'', Partition)$
            $Update(Current, Next, B, B', B'')$
```

```
Algorithm 4: $Update(Current, Next, B, B', B'')$
Data: $C = min\{B', \overline{B''}\}$
foreach $b \in A$ do
    if $(B, b) \in Current$ then
        Replace $(B, b)$ by $(B', b)$ and $(B'', b)$ in Current
    else
        if $(B, b) \in Next$ then
            Replace $(B, b)$ by $(B', b)$ and $(B'', b)$ in Next
        else
            Add $(C, b)$ in $Next$
```

**Fig. 9.** Re-description of Hopcroft's algorithm

The property (iii) also follows from the algorithm: a block is extracted from `Current` only to be treated. In the function `Update`, `Current` is only modified to replace a given set of transition $(C, a)$ by sets $(C', a)$ and $(C'', a)$, such that $(C, a) = (C', a) \cup (C'', a)$. Hence, this operation does not change the global set of transitions contained in `Current`. □

**Remark 3.** From Lemma 8, we know that the time complexity of an iteration of Hopcroft's algorithm is bounded by $\mathcal{O}(kn)$: indeed the functions `Update` and `BreakBlock` can be performed in $\mathcal{O}(1)$, by choosing properly the data structures. Since, the loop **Foreach** (Line 8) depends on the number of transitions contained in a block, the complexity of an iteration is bounded by the number of transitions in `Current`.

### *4.2. Analogy between Moore's and Hopcroft's complexities*

**Lemma 9.** *For any deterministic automaton, any integer $i \in \{0, \ldots, n-2\}$, any states $p$ and $q$, if $p \nsim_i q$, then $p$ and $q$ are not in the same class of the partition at the end of iteration $i$ of Hopcroft's algorithm.*

**Proof.** By induction: we call iteration 0 the initialization of the algorithm, which is exactly the same for both algorithms. Suppose now that the property is true for a given iteration $i$. Let $p$ and $q$ be two states such that $p \sim_i q$ and $p \nsim_{i+1} q$. According to the definition of the Myhill–Nerode equivalence, there exists a letter $a$ such that $p \cdot a \nsim_i q \cdot a$. Then, $p \cdot a$ and $q \cdot a$ are not in the same class at the end of iteration $i$ of Hopcroft's algorithm, meaning that the transitions labeled by $a$ and starting from $p$ and $q$ have been added in two different blocks, either in `Current` or in `Next` during an iteration $j$, with $j \leq i$. This implies that $p$ and $q$ are separated at the iteration $j$ or $j+1$. In any case, they are not in the same equivalence class at the end of iteration $i+1$ of Hopcroft's algorithm. □

**Corollary 2.** *For any deterministic automaton, Hopcroft's algorithm stops either at the same iteration as Moore's algorithm, or at a previous one.*

Note that this lemma does not prove that a Moore's algorithm implementation is never faster than a Hopcroft's algorithm implementation. Indeed, Hopcroft's algorithm uses several data structures to guarantee its worst-case complexity. For a single operation on a transition, Hopcroft's algorithm requires more computation than Moore's algorithm. Therefore it is still possible to obtain an execution of Moore's algorithm which is slightly faster than all executions of Hopcroft's algorithm.

We use the Theorems 1–3 associated to Moore's algorithm to obtain similar results on Hopcroft's algorithm.

**Theorem 4.** *For the uniform distribution on the set of complete deterministic automata with $n$ states over a $k$-letter alphabet, there exists a family of implementations of Hopcroft's algorithm whose average complexity is bounded by $\mathcal{O}(n \log \log n)$.*

**Proof.** The proof follows directly from the Theorem 1, Corollary 2 and Properties 1 and 3 given in Lemma 8. □

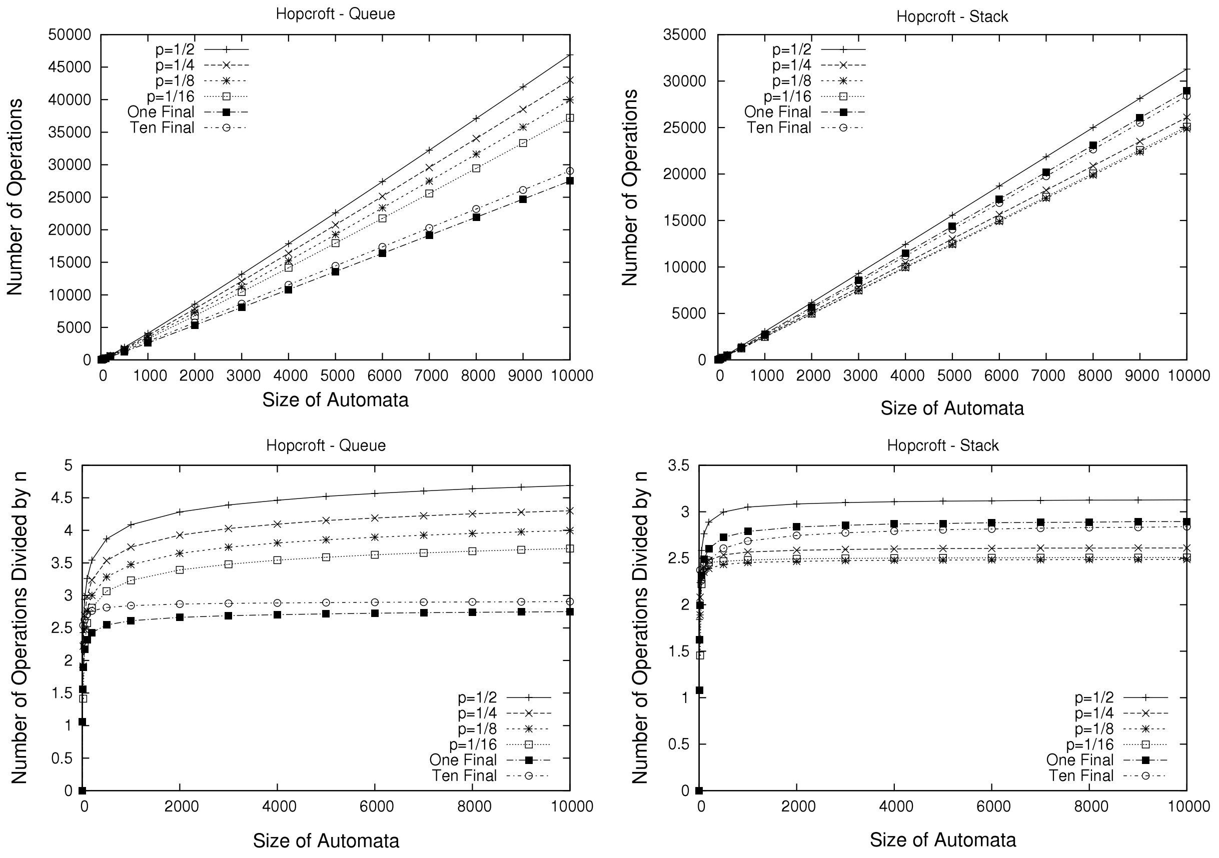

**Fig. 10.** The pictures at the top illustrates the evolution of the average number of operations made by Hopcroft's algorithm. On the left is the queue implementation: it seems that the less final states there is in the input automaton, the faster the algorithm is. On the right is the stack implementation, for which there seems to be less operations than in the other case, especially when there is a Bernoulli distribution on the set of final states. The pictures at the bottom are the same values divided by *n*, which is the size of the automaton.

**Theorem 5.** *For any fixed integer $k \geq 2$, the uniform distribution over the deterministic and complete transitions structures with n states over a k-letter alphabet, and the distribution over the set of final states where each state has a probability $x \in ]0, 1[$ to be final, there exists a family of implementations of Hopcroft's algorithm whose average complexity is bounded by $\mathcal{O}(n \log \log n)$.*

**Proof.** We use Theorem 2, Corollary 2 and Properties 1 and 3 given in Lemma 8. □

**Theorem 6.** *For the uniform distribution on the set of complete deterministic automata with n states over a k-letter alphabet, there exists a family of implementations of Hopcroft's algorithm whose generic complexity is bounded by $\mathcal{O}(n \log \log n)$.*

**Proof.** We use Theorem 3, Corollary 2 and Properties 1 and 3 given in Lemma 8. □

## *4.3. On the usual implementations of Hopcroft's algorithm*

In [1], the authors compare two types of implementations of Hopcroft's algorithm when the *waiting set* is implemented with a stack and when it is implemented with a queue. The benchmarks they made seemed to indicate that the stack implementation is slightly faster than the queue, but the tests were made on restricted families of automata. Using the library REGAL [2], that allows to generate large random complete deterministic accessible automata, and we obtained the same conclusions. In fact, experimentally, all the implementations of Hopcroft's algorithm we tested only differed from a small constant in average time complexity, including the ones using two *waiting sets*.

A natural extension of Theorem 4 would be to find out if the average complexity of Hopcroft's algorithm is still $\mathcal{O}(n \log \log n)$ with those implementations. For distributions where the transitions structures are chosen uniformly, and where either each state has a fixed probability $p \in \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}$ to be final, or a fixed number of final states is randomly chosen amongst $Q$, we made some tests on what we consider to be the best indicator of the average number of operations made by Hopcroft's algorithm: the sum of the numbers of transitions associated to all the sets extracted from the *waiting set*.

Given the results illustrated in Fig. 10, we conjecture that these usual implementations of Hopcroft's algorithm also have an $\mathcal{O}(n \log \log n)$ average complexity.

## 5. Conclusion and open discussions

In this paper, we established that the average complexity of both Moore's algorithm and Hopcroft's algorithm is $\mathcal{O}(n \log \log n)$, for the uniform distribution on complete deterministic automata. It is a first step to prove the conjecture made in the conclusion of [3]: for the uniform distribution on complete deterministic accessible automata, the average complexity of Moore algorithm is $\Theta(n \log \log n)$. To show this conjecture is not an easy task, since it requires a better knowledge of the average size of the accessible part in a complete deterministic automaton, but also of the average number of minimal automata amongst the complete deterministic and accessible. Also, the actual results on the average complexity of both algorithms do not include probability distributions where the number of final states is fixed. We conjecture that for those distributions, the average complexity of Moore's algorithm is $\mathcal{O}(n \log n)$ and the average complexity of Hopcroft's algorithm is still $\mathcal{O}(n \log \log n)$.

## References

[1] Manuel Baclet, Claire Pagetti, Around Hopcroft's algorithm, in: Implementation and Application of Automata, 11th International Conference, CIAA 2006, Taipei, Taiwan, August 21–23, 2006. Proceedings, in: Lecture Notes in Computer Science, vol. 4094, 2006, pp. 114–125.

[2] Frederique Bassino, Julien David, Cyril Nicaud, REGAL: a library to randomly and exhaustively generate automata, in: Jan Holub, Jan Žďárek (Eds.), Implementation and Application of Automata, 12th International Conference, CIAA'07, in: Lecture Notes in Computer Science, vol. 4783, Springer, 2007, pp. 303–305.

[3] Frederique Bassino, Julien David, Cyril Nicaud, On the average complexity of Moore's state minimization algorithm, in: Susanne Albers, Jean-Yves Marion (Eds.), 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, Freiburg, Germany, in: LIPIcs, vol. 3, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009, pp. 123–134.

[4] Frederique Bassino, Cyril Nicaud, Enumeration and random generation of accessible automata, Theoret. Comput. Sci. 381 (2007) 86–104.

[5] Marie-Pierre Beal, Maxime Crochemore, Minimizing local automata, in: IEEE International Symposium on Information Theory, ISIT'07, 2007, pp. 1376–1380.

[6] Marie-Pierre Beal, Maxime Crochemore, Minimizing incomplete automata, in: Finite-State Methods and Natural Language Processing, FSMNLP'08, 2008, pp. 9–16.

[7] Jean Berstel, Luc Boasson, Olivier Carton, Continuant polynomials and worst-case behavior of Hopcroft's minimization algorithm, Theoeret. Comput. Sci. 410 (30–32) (2009) 2811–2822.

[8] Jean Berstel, Olivier Carton, On the complexity of Hopcroft's state minimization algorithm, in: M. Domaratzki, A. Okhotin, K. Salomaa, S. Yu (Eds.), CIAA'04, in: Lecture Notes in Computer Science, vol. 3317, Springer, 2004, pp. 35–44.

[9] Janusz A. Brzozowski, Canonical regular expressions and minimal state graphs for definite events, in: Symposium on the Mathematical Theory of Automata, vol. 12, Polytechnic Press, Polytechnic Institute of Brooklyn, New York, 1962, pp. 529–561.

[10] Giusi Castiglione, Antonio Restivo, Marinella Sciortino, Hopcroft's algorithm and cyclic automata, in: Language and Automata Theory and Applications: Second International Conference, LATA 2008, Tarragona, Spain, March 13–19, 2008. Revised Papers, Springer, Berlin, Heidelberg, 2008, pp. 172–183.

[11] Giusi Castiglione, Antonio Restivo, Marinella Sciortino, On extremal cases of Hopcroft's algorithm, in: S. Maneth (Ed.), Implementation and Application of Automata, 14th International Conference, CIAA'09, in: Lecture Notes in Computer Science, vol. 5642, Springer, 2009, pp. 14–23.

[12] Jean-Marc Champarnaud, Gérard Duchamp, Brzozowski's derivatives extended to multiplicities, in: Bruce W. Watson, Derick Wood (Eds.), CIAA'01, in: Lecture Notes in Computer Science, vol. 2494, Springer, 2001, pp. 52–64.

[13] Jean-Marc Champarnaud, Ahmed Khorsi, Thomas Paranthoën, Split and join for minimizing: Brzozowski's algorithm, in: PSC'02 Proceedings, 2002, pp. 96–104.

[14] Philippe Flajolet, Robert Sedgewick, An Introduction to the Analysis of Algorithms, Addison Wesley, 1996.

[15] Philippe Flajolet, Robert Sedgewick, Analytic Combinatorics, Cambridge University Press, 2008.

[16] David Gries, Describing an algorithm by Hopcroft, Acta Inform. 2 (1973) 97–109.

[17] John E. Hopcroft, An n log n algorithm for minimizing states in a finite automaton, Technical Report, Stanford University, Stanford, CA, USA, 1971.

[18] John E. Hopcroft, Jeffrey D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[19] Ilya Kapovich, Alexei Myasnikov, Paul Schupp, Vladimir Shpilrain, Generic-case complexity, decision problems in group theory and random walks, J. Algebra 264 (2003) 665–694.

[20] Timo Knuutila, Re-describing an algorithm by Hopcroft, Theor. Comput. Sci. 250 (1–2) (2001) 333–363.

[21] Lothaire, Applied Combinatorics on Words, in: Encyclopedia of Mathematics and its Applications, vol. 105, Cambridge University Press, 2005.

[22] Edward F. Moore, Gedanken experiments on sequential machines, in: Automata Studies, Princeton U, 1956, pp. 129–153.

[23] Anil Nerode, Linear automaton transformation, in: Proc. American Mathematical Society, 1958, pp. 541–544.

[24] Cyril Nicaud, Average state complexity of operations on unary automata, in: M. Kutylowski, L. Pacholski, T. Wierzbicki (Eds.), MFCS, in: Lecture Notes in Computer Science, vol. 1672, Springer, 1999, pp. 231–240.

[25] Dominique Revuz, Minimisation of acyclic deterministic automata in linear time, Theor. Comput. Sci. 92 (1) (1992) 181–189.

[26] Wojciech Szpankowski, Average Case Analysis of Algorithms on Sequences, John Wiley & Sons, Inc., New York, NY, USA, 2001.

[27] Antti Valmari, Petri Lehtinen, Efficient minimization of DFAs with partial transition, in: Susanne Albers, Pascal Weil (Eds.), 25th International Symposium on Theoretical Aspects of Computer Science, STACS 2008, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 1, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2008, pp. 645–656.