The Average Complexity of Moore's State Minimization Algorithm is $\mathcal{O}(n \log \log n)$ *

Julien David

Institut Gaspard Monge, Université Paris Est
77454 Marne-la-Vallée Cedex 2, France.

Abstract. We prove that the average complexity, for the uniform distribution on complete deterministic automata, of Moore's state minimization algorithm is $\mathcal{O}(n \log \log n)$, where n is the number of states in the input automata.

1 Introduction

Due to their efficiency to represent regular languages and perform most of usual computations they involve, finite state automata are used in various fields such as linguistics, bioinformatics, program verification and data compression. A minimal automata is the smallest complete deterministic automata that can be associated to a regular language. Because this automaton is unique, up to isomorphism on the labels of the states, it is a canonical representation of a regular language and permits to test the equality between regular languages and equivalence between automata. Most state minimization algorithms compute the minimal automaton of a regular language taking a deterministic automaton as an input, by identifying the indistinguishable states.
Moore proposed the first minimization algorithm[8], which is based on the calculus of the Myhill-Nerode equivalence, by refinements of partitions 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 [1], it is proved that the average complexity of the algorithm is bounded by $\mathcal{O}(n \log n)$. 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 [6] is the best known algorithm with an $\mathcal{O}(n \log n)$ worst-case complexity. It also uses partition refinements to compute the minimal automaton, but its description is not deterministic, making its analysis complicated. Different proofs of its correctness were given [5, 7] and several authors [3, 4] proved the tightness of the upper bound of the complexity for different families of automata.
In this paper, we prove that for the uniform distribution on complete deterministic automata, the average complexity of the algorithm due to Moore is

* This work was completed with the support of the ANR project GAMMA number **07 − 2_195422**.

$\mathcal{O}(n \log\log n)$. The article is organized as follows: after recalling the basics of automata minimization (Section 2), we introduce the tools we use for the average analysis (Subsections 2.3 to 2.5). Section 3 is dedicated to the average time complexity analysis of Moore's algorithm. Due to a lack of space, the proof of Lemma 6 is not fully detailed, but an idea of the proof is given. The paper closes with a discussion on Hopcroft's algorithm executions, which are faster than Moore's algorithm ones, for any input automaton and a conjecture on the average complexity of both algorithms, for various distributions on automata.

2 Preliminaries

2.1 Definitions and notations

A *finite deterministic 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 words of length i and $A^{\leq i}$ 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 $q_0 \cdot u = p$.

A *transition structure* is an automaton where the set of final states is not specified. Given such a transition 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 transition structure with n states there are exactly 2^n distinct deterministic automata that can be built from this transition structure. Each of them corresponds to a choice of set of final states.

In the following we only consider complete deterministic automata and complete deterministic transition structures, the accessibility is not guaranteed. Consequently these objects will most of the time just be called respectively *automata* or *transition structures*. The set Q of an n-state transition structure will be denoted by $\{1, \ldots, n\}$. The set of automata and the set of transition 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* [9]. 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 $a \in A$, $p \cdot a \sim_i q \cdot a$.
(3) If for some $i \in \mathbb{N}$ $(i+1)$-equivalence is equal to i-equivalence then for every $j \geq i$, j-equivalence is equal to 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 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 [8] to compute the minimal automaton of a regular language represented by a deterministic automaton. It builds the partition of the set of states corresponding to Myhill-Nerode equivalence and mainly relies on properties (2) and (3) of Proposition 1: The partition π 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 ℓ 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 the **dependency tree** and a modification of the **dependency graph** introduced in [1]. Those tools will allow us to give an upper bound on the number of automata minimized in more than ℓ partition refinements, which is useful for the average complexity analysis.

2.3 The Dependency Tree

In the following, we introduce the dependency tree to model a set of transition structures. To begin with, we explain how a dependency tree $\mathcal{R}(p)$ can be obtained from a fixed transition structure τ and a fixed state p and then how this object will help to estimate the cardinal of a set of transition structures. For a

Algorithm 1: Moore's algorithm

```
1  if F = ∅ then
2      return (A, {1}, ∗, 1, ∅)
3  if F = {1, ..., n} then
4      return (A, {1}, ∗, 1, {1})
5  forall p ∈ {1, ..., n} do
6      π'[p] = [[p ∈ F]]
7  π = undefined
8  while π ≠ π' do
9      π = π'
10     compute π' from π
11 return the quotient of A by π
```

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, π' is the new partition and π the former one. Lines 5-6 is the initialization of π' to the partition of $\sim_0$, π 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 π' from π

```
1  forall p ∈ {1, ..., n} do
2      s[p] = (π[p], π[p · a_1], ..., π[p · a_k])
3  compute the permutation σ that
   sorts the states according to s[]
4  i = 0
5  π'[σ(1)] = i
6  forall p ∈ {2, ..., n} do
7      if s[p] ≠ s[p − 1] then i = i + 1
8      π'[σ(p)] = i
9  return π'
```

Fig. 1. Description of Moore's algorithm

fixed transition 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:

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

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

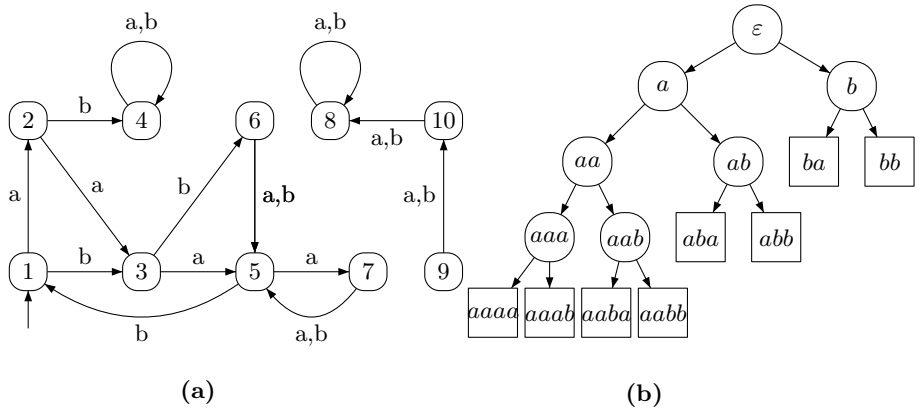

Fig. 2. Let **(a)** be the fixed transition structure and 2 be the fixed state, **(b)** is the associated **dependency tree** $\mathcal{R}(2)$. We have $S_2(2) = \{\varepsilon, a, b, aa, ab\}$, $L_2(2) = \{aa, ab\}$ and $s_2(2) = \{2, 3, 4, 5, 6\}$.

characterize sets of transition structures corresponding to particular dependency trees.

We introduce some notations associated to a dependency tree $\mathcal{R}(p)$: $S_h(p)$ denotes the set of all nodes of depth less or equal to h, $L_h(p)$ denotes the set of all the nodes at given depth h. Since every node in the tree is labelled by a word, we note $w \in S_h(p)$ or $w \in L_h(p)$ if w is a word labelling a node in those sets. We also define the set $s_h(p)$ of all the states that are reached from a state p by following a path labelled by a word of less or equal to h. For all the transition 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 at a depth less or equal to h, then the number of associated transition 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 transition structure associated to the tree $\mathcal{R}(p)$, the transition labelled 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 transition structures. For two fixed states p and q, two fixed x-tuples (x is a fixed integer) of non-empty words $\overrightarrow{u} = (u_1, \ldots, u_x)$ and $\overrightarrow{v} = (v_1, \ldots, v_x)$, two fixed sets φ_p and φ_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:

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

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 $i \leq x$ we have $u_i = u'_i\alpha_i$ and $v_i = v'_i\beta_i$. From $\mathcal{T}_n(p,q,\varphi_p,\varphi_q,\overrightarrow{u},\overrightarrow{v})$, 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 $\tau \in \mathcal{T}_n(p,q,\varphi_p,\varphi_q,\overrightarrow{u},\overrightarrow{v})$, $r\cdot a = t\cdot b$.

The $\mathcal{T}$-dependancy graph $G_n(p,q,\varphi_p,\varphi_q,\overrightarrow{u},\overrightarrow{v})$ satisfies the two following properties:

- For all $i \leq x$, there exists an edge $((p\cdot u'_i,\alpha_i),(q\cdot v'_i,\beta_i))$.
- For all $(w_1,w_2) \in \varphi_p$ (resp. $(w_3,w_4) \in \varphi_q$), we have $w_1 = w'_1a_1$ and $w_2 = w'_2a_2$ with $a_1,a_2 \in A$ and such that there exists an edge $((p\cdot w'_1,a_1),(p\cdot w'_2,a_2))$ (resp. $((q\cdot w'_3,a_3),(q\cdot w'_3,a_3))$).

Lemma 2. *If $G_n(p,q,\varphi_p,\varphi_q,\overrightarrow{u},\overrightarrow{v})$ contains an acyclic subgraph induced by a subset of nodes 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 transitions in the same connected components of $G_n(p,q,\varphi_p,\varphi_q,$-$\overrightarrow{u},\overrightarrow{v})$ share 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 [1]. Let τ be a fixed transition structure with n states and ℓ be an integer such that $1 \leq \ell < n$. Let p,q be two states of τ such that $p \neq q$ and u a word of length ℓ. We define $\mathcal{F}_\tau(p,q,u)$ as the set of sets of final states F for which in the automaton (τ,F) the states p and q are separated by the word u. That is to say :

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

From the set $\mathcal{F}_\tau(p,q,u)$ one can define the undirected graph $\mathcal{G}_\tau(p,q,u)$, called the *$\mathcal{F}$-dependency graph*, as follows:

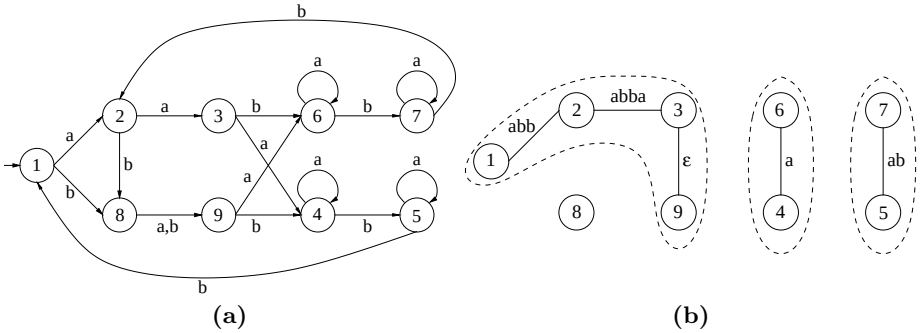

Fig. 3. (a) is a fixed transition structure and **(b)** the **$\mathcal{F}$-dependency graph** for $p = 3$, $q = 9$ and $u = abbaa$. Thanks to **(b)**, we know that all states in a same connected component will be either all final or all non-final. Hence, there are at most 2^4 possible sets of final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

- its set of vertices is $\{1, \ldots, n\}$, the set of states of τ;
- 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 ℓ 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 $|u|$ iterations because of p, q and u. Figure 3 shows an example of a F-dependency graph.

Lemma 3. *[1] 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 subsections 3.2 and 3.3 to obtain upper bounds on the cardinal of sets of automata with given properties and prove that their contribution to the average complexity is negligible.

3 Moore's Algorithm: average case analysis

In [1], 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 transition structures. In this paper, in order to improve the upper bound on the average complexity, we also have to study some properties of transition structures. Since the enumeration of accessible automata with given properties is an open problem, 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 partition refinements 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 $\lambda = \lceil \log_k \log_2 n^3 + 2 \rceil$, which will be used in the sequel. We gather the sets $\mathcal{A}_n^i$, in order to obtain the following upper bound:

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

S1 is less than λ and equal to $\mathcal{O}(\log\log n)$.
In [1], it is proved that $\sum_{i=5\log_2 n+1}^{n-2}|\mathcal{A}_n^i| \leq \frac{|\mathcal{A}_n|}{n}$. Therefore we know that **S3** is equal to $\mathcal{O}(1)$. Hence, in the following, we prove that:

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

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

$$\mathcal{A}_n(p,q,\ell) = \{(\tau,F) \in \mathcal{A}_n \mid \tau \subset \mathcal{T}_n,\ F \subseteq \{1,\ldots,n\},\ p \sim_{\ell-1} q,\ p \not\sim_\ell q\}$$

Remark 1. Note that if in the automaton (τ,F), for all letter $a \in A$, $p\cdot a = q\cdot a$, then either $p \not\sim_0 q$ 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 transition 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:

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

Let τ be a transition structure, and p,q be two distinct states. Recalling that $s_h(p)$ is defined in Section 2.3, if A is a k-letter alphabet, and μ a positive integer, we define two properties associated to transition structures:

(1) $largeTree(\tau, p, \mu)$ is *true* if and only if $|s_\mu(p)| \geq k^\mu - 1$.
Note that this implies that $|s_{\mu-2}(p)| \geq k^{\mu-2} - 1$.
(2) $noIntersection(\tau, p, q)$ is *true* if and only if $s_{\lambda-2}(p) \cap s_{\lambda-2}(q) = \emptyset$.

For fixed states p and q, an automaton is in $\mathcal{A}_n(p, q, \lambda + 1)$ if its associated transition structure is in one of the three distinct sets we are about to define:

- $\mathcal{X}_n$ **is the set of all transition structures** τ **such that**:
  - there exists a state $r \in Q$ such that $largeTree(\tau, r, \lambda)$ 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 transition structures** τ **such that**:
  - for all state $r \in Q$, the property $largeTree(\tau, r, \lambda)$ 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 transition structures** τ **such that**:
  - for all state $r \in Q$, the property $largeTree(\tau, r, \lambda)$ is *true*,
  - there exists $w \in A^{\leq 2}$ such that $noIntersection(\tau, p \cdot w, q \cdot w)$ is *true*.

3.2 Transition Structures with a Huge $\mathcal{F}$-Dependency Graph

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

Proof. Let $\mathcal{G}'$ be the subgraph $\mathcal{G}_\tau(p, q, u)$ defined as follows: there exists an edge $(p \cdot wv, q \cdot wv)$ in $\mathcal{G}'$, if and only if v labels a node in $S_{\lambda-2}(p \cdot w)$. $\mathcal{G}'$ contains exactly $|s_{\lambda-2}(p \cdot w)|$ edges, since for all $v \in S_{\lambda-2}(p \cdot w)$, the states $p \cdot wv$ are all pairwise distinct. Since $largeTree(\tau, r, \lambda)$ is true for all state $r \in Q$, we have $|s_{\lambda-2}(p \cdot w)| \geq k^{\lambda-2} - 1$. $\mathcal{G}'$ is acyclic: indeed, the property $noIntersection(\tau, p \cdot w, q \cdot w)$ indicates that the set of nodes connected to at least one edge forms a bipartite graph (the nodes of $s_{\lambda-2}(p \cdot w)$ on one side and the nodes of $s_{\lambda-2}(q \cdot w)$ on the other), where all the nodes of $s_{\lambda-2}(p \cdot w)$ are only connected to one edge.

Corollary 1. *For $\lambda = \lceil \log_k \log_2 n^3 + 2 \rceil$, the number of automata in $\mathcal{A}_n(p, q, \lambda + 1)$ whose transition 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 $\lambda + 1$, and any fixed transition 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 transition structure $\tau \in \alpha_n(p, q)$, since the number of words in $A^{\lambda+1}$ is $\mathcal{O}(\log n)$, the number of choices of sets of final states such that the automata are in $\mathcal{A}_n(p, q, \lambda + 1)$ is bounded above by:

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

3.3 Negligible Sets of Transition Structures

Lemma 5. *The number of transition structure 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 $\mu \in \{1, \ldots, \lambda\}$, we define the sets $\mathcal{X}_n(r, \mu)$ of all transition structures τ for which μ is the smallest integer such that the property $largeTree(\tau, r, \mu)$ is false. We have :

$$\mathcal{X}_n = \bigcup_{r\in\{1,\ldots,n\}} \left(\bigcup_{\mu\in\{1,\ldots,\lambda\}} \mathcal{X}_n(r,\mu) \right)$$

For all transition structures in $\mathcal{X}_n(r, \mu)$, the dependency tree $\mathcal{R}(r)$ contains at least two leaves of depth less or equal to μ. Indeed, if $\mathcal{R}(r)$ contains at most one leaf of depth less than μ, then there exist $k-1$ letters $a \in A$ such that $\mathcal{R}(r{\cdot}a)$ does not contain any leaf of depth less than μ. Therefore we have $|S_\mu(p)| \geq k^\mu - 1$. We decompose the possible dependency trees $\mathcal{R}(r)$ into two different kinds:

1. All leaves are at level μ. 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 of depth μ. The number of trees of this kind is at most $\sum_{h=1}^{\mu-1} \left(k^h \sum_{f=1}^{k^\mu} \binom{k^\mu}{f} \right)$.

Using the upper bound of Lemma 1 on the number of transition structures counted by each tree, we obtain:

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

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

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

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

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

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

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

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

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

Proof. For any transition structure in $\mathcal{Y}_n(p,q)$, for all words $w \in A^{\leq 2}$, there exist two words $u, v \in A^{\leq \lambda-2}$ such that $p \cdot wu = q \cdot wv$. We partition the set $\mathcal{Y}_n(p,q)$ according to the leaves the dependency trees contain.

Both trees do not contain a leaf of depth less or equal to λ : let E be the set of letters, such that for all $a \in E$, $p \cdot a = q \cdot a$. We define $e = |E|$. According to Remark 1, we have $e < k$. For all $b \in A \setminus E$ and all $c \in A$, $noIntersection(\tau, p \cdot bc, q \cdot bc)$ is $false$. For $\overrightarrow{u}$ and $\overrightarrow{v}$ of size $x = e + k(k-e)$, such that for all $1 \leq j \leq e$, we have $u_j = v_j = a_j$ with $a_j \in E$ and such that for all $e < j' \leq x$, w'_j is a prefix of $u_{j'}$ and $v_{j'}$, where w'_j is a word of the form bc. This subset is included in:

$$\bigcup_{\substack{E \subsetneq A \\ \forall a \in E,\ p \cdot a = q \cdot a}} \bigcup_{\overrightarrow{u}, \overrightarrow{v}} \mathcal{T}_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$$

There are $2^k - 1$ possible subsets E. There are less than $k^{2\lambda(x-e)}$ possible choices for $u_i, v_i \in A^{\leq \lambda}$, for $e < i \leq x$. For all $j, l \leq x$, $j \neq l$, since u_j and u_l (resp. v_j and v_l) label nodes in $\mathcal{R}(p)$ (resp. $\mathcal{R}(q)$), setting $u_j = u'_j\alpha_j$ and $u_l = u'_l\alpha_l$, we have $(p \cdot u'_j, \alpha_j) \neq (p \cdot u'_l, \alpha_l)$ and there is no path between $(p \cdot u'_j, \alpha_j)$ and $(p \cdot u'_l, \alpha_l)$ since if would imply that $p \cdot u_j = p \cdot u_l$ and that either u_j or u_l labels a leaf. Therefore, $G_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$ contains an acyclic graph with x edges $((p \cdot u'_j, \alpha_j), (q \cdot v'_j, \beta_j))$. Since $x \geq k+1$ (for $e = k-1$), using Lemma 2, we obtain the upper bound stated above.

At least one tree contains a leaf of depth less or equal to λ : due to a lack of space, we will not describe this set. The idea is that, just like in the previous case, we are able to guarantee that a $\mathcal{T}$-dependency graph always contains an acyclic subgraph with $k+1$ edges and that there is $\mathcal{O}(\log^{2(k+1)} n)$ possible graphs.

3.4 Concluding the proof

Recall that we want to prove Equation 1. We define $\widetilde{\mathcal{X}_n}$, $\widetilde{\alpha_n}(p,q)$ and $\widetilde{\mathcal{Y}_n}(p,q)$ as the sets of automata whose transition structure are respectively in $\mathcal{X}_n$, $\alpha_n(p,q)$ and $\mathcal{Y}_n(p,q)$. We have:

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

Using Lemmas 4,5 and 6 we obtain:

$$\sum_{i>\lambda} |\mathcal{A}_n^i| \leq |\mathcal{T}_n| \frac{2^n \log^5 n}{n} + n^2 \left(|\mathcal{T}_n| \times \frac{2^n \log n}{n^3} + |\mathcal{T}_n| \frac{2^n \log^6 n}{n^3} \right) \leq |\mathcal{A}_n| \frac{\log^6 n}{n}$$

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

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

4 Conclusion

In this paper, we obtained a new upper bound on the average complexity of Moore's state minimization algorithm, for the uniform distribution on complete deterministic automata. Also, it is possible to describe a set of Hopcroft's algorithm executions which, for any deterministic automata, compute the equivalence in less steps than Moore's algorithm (due to a lack of space, we are not giving the description of those executions in this paper). Hence, for the uniform distribution on complete deterministic automata with n states, there exists an execution of Hopcroft's algorithm whose average complexity is $\mathcal{O}(n \log\log n)$. This paper is a first step to prove the conjecture made in the conclusion of [1]: for the uniform distribution on complete deterministic accessible automata, the average complexity of Moore algorithm is $\Theta(n \log\log n)$. To prove 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 the average number of minimal automata amongst the complete deterministic and accessible.

I would like to thank Phillipe Duchon for the fruitful discussion on upper bound of the cardinal of the set $\mathcal{X}_n$, but also Cyril Nicaud and Frederique Bassino for their advices and comments.

References

1. Frederique Bassino, Julien David, and Cyril Nicaud. On the average complexity of Moore's state minimization algorithm. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009), Freiburg, Germany.*, volume 3 of *LIPIcs*, page 123–134. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.
2. Frederique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theor. Comput. Sci.*, 381:86–104, 2007.
3. Jean Berstel and Olivier Carton. On the complexity of Hopcroft's state minimization algorithm. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *CIAA'04*, volume 3317 of *Lecture Notes in Computer Science*, pages 35–44. Springer, 2004.
4. Giusi Castiglione, Antonio Restivo, and Marinella Sciortino. On extremal cases of Hopcroft's algorithm. In S. Maneth, editor, *CIAA'09*, volume 5642 of *Lecture Notes in Computer Science*, pages 14–23. Springer, 2009.
5. David Gries. Describing an algorithm by Hopcroft. *Acta Inf.*, 2:97–109, 1973.
6. John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.
7. Timo Knuutila. Re-describing an algorithm by Hopcroft. *Theor. Comput. Sci.*, 250(1-2):333–363, 2001.
8. Edward F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.
9. Anil Nerode. Linear automaton transformation. In *Proc. American Mathematical Society*, pages 541–544, 1958.