

Incremental NFA Minimization

Christian Bianchini¹, Alberto Policriti^{1*}, Brian Riccardi¹ and Riccardo Romanello¹

¹University of Udine, Italy

Abstract

Finite state automata are fundamental objects in Theoretical Computer Science and find their application in Text Processing, Compilers Design, Artificial Intelligence and many other areas. The problem of minimizing such objects can be traced back to the '50s and since then it has been the arena for developing new algorithmic ideas. There are two main paradigms to tackle the problem: top down – which builds a *descending* chain of equivalences by subsequent refinements – and bottom up – which builds an *ascending* chain of equivalences by aggregation of classes. The former approach leads to a fast $\mathcal{O}(n \log n)$ algorithm, whereas the latter is currently quadratic for any practical application. Nevertheless, the bottom up algorithm enjoys the property of being *incremental*, *i.e.* the minimization process can be stopped at any time obtaining a language-equivalent partially minimized automaton. In this work we correct a small mistake in the algorithm given by Almeida *et al.* in 2014 and we propose a simple, DFS-like and truly quadratic incremental algorithm for minimizing deterministic automata. Furthermore, we adapt the idea to the nondeterministic case obtaining an incremental algorithm which computes the maximum bisimulation relation in time $\mathcal{O}(n^2 r |\Sigma|)$, where n is the number of states, Σ is the alphabet and r is the degree of nondeterminism, improving by a factor of r the running time of the fastest known aggregation based algorithm for this problem.

Keywords

Automata, Bisimulation, Minimization, Incremental

1. Introduction

Finite state automata are fundamental objects in Theoretical Computer Science and find their application in Text Processing, Compilers Design, Artificial Intelligence and many other areas.

The minimization of an automaton is the process of constructing a new (language-equivalent) automaton which is minimal in the number of states. This problem can be traced back to the '50s by the work of Moore [1]. A fundamental result in Automata Theory is the Myhill-Nerode Theorem [2], establishing that in the deterministic case this minimal automaton is in fact *the minimum*, up to isomorphism. In the wider setting of nondeterministic automata there is no analog result and finding any state-minimal automaton is PSPACE-complete [3]. For this reason, a practical alternative is the minimization with respect to bisimulation. *Bisimilarity* is indeed

Proceedings of the 23rd Italian Conference on Theoretical Computer Science, Rome, Italy, September 7-9, 2022

*Corresponding author.

✉ bianchini.christian@spes.uniud.it (C. Bianchini); alberto.policriti@uniud.it (A. Policriti);


riccardi.brian@spes.uniud.it (B. Riccardi); riccardo.romanello@uniud.it (R. Romanello)

🌐 <http://users.dimi.uniud.it/~alberto.policriti/home> (A. Policriti); <https://riccardoromanello.github.io>

(R. Romanello)

🆔 0000-0002-2855-1221 (R. Romanello)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

a valid choice since in the deterministic case two states are bisimilar if and only if they are Myhill-Nerode equivalent.

Thus, the problem of minimizing automata reduces to the problem of computing bisimilarity between states, which in turn is equivalent to determining the coarsest partition of a set stable with respect to some binary relation. The two main paradigms to compute the aforementioned partition are *top down* and *bottom up*.

Top down algorithms start with the partition that separates states between final and non final and subsequently *refine* the partition until it is stable. By a careful choice of which block of the partition to split at each refining step, Paige and Tarjan [4] devised an algorithm that computes the maximum bisimulation equivalence in time $\mathcal{O}(m \log n)$, where n is the number of states and m is the number of transitions. The iconic Hopcroft's Algorithm [5] (which Paige and Tarjan's solution is based on) deals with the special case of deterministic automata. Furthermore, it has recently been proved that the bisimilarity computation requires $\Omega(m \log n)$ time assuming top down algorithms [6].

On the contrary, bottom up solutions start with the finest partition – the one where each state constitutes a singleton – and proceed by subsequently *merging* two blocks found to be equivalent. For this reason, the technique is also known in the literature as *partition-aggregation*. The main advantage of this paradigm is that the algorithm is *incremental*, *i.e.* it proceeds in subsequent stages where at the end of each merging step the resulting automaton is language-equivalent to the input one. In this way, the minimization process can be stopped at any time and can be resumed later. The first algorithm of this kind is due to Watson [7]. After a series of improvements Watson and Daciuk [8] reduced the running time to $\mathcal{O}(n^2 |\Sigma| \alpha(n))$ for deterministic automata with n states, alphabet Σ and where $\alpha(n)$ is related to the inverse of Ackermann's function [9] which can be treated as a constant for any practical value of n^1 . The main idea is to propagate the definition of bisimilarity: if states p and q are equivalent, then also their transitions by the same character must lead to equivalent states. This is done by a recursive function EQUIV which resembles an equivalence algorithm by Hopcroft and Karp [10]. A subsequent work by Almeida *et al.* [11] aimed at simplifying the algorithm by Watson and Daciuk maintaining its running time. Unfortunately, there is a small mistake in their version of EQUIV which leads to a $\Omega(n^3)$ algorithm in the worst case.

Above algorithms are focused on the minimization of deterministic automata. The nondeterministic case was tackled by Björklund and Cleophas [12] adapting ideas from Watson and Daciuk. They devised an incremental algorithm for computing bisimilarity in time $\mathcal{O}(n^2 r^2 |\Sigma|)$, where r is the degree of nondeterminism.

In this work we correct the algorithm by Almeida *et al.* providing a simplified version of the one by Watson and Daciuk and maintaining the quadratic running time. The solution is based on the concept of *associated graph* whose purpose is twofold: to distill the behaviour of the aforementioned algorithms by interpreting them as graph colorings and to design our own incremental procedure. Having established the clear connection between minimization and graph coloring it is natural to generalize the algorithm for solving the bisimilarity problem on nondeterministic automata. Furthermore, the proposed solution improves by a factor of r the running time of Björklund and Cleophas [13].

¹It holds $\alpha(n) \leq 5$ for $n \leq 2^{2^{16}}$.

The paper is organized as follows: in the next section we give the basics about partitions, relations, and automata. In Section 3 we briefly describe the algorithm by Almeida *et al.* and we point out the mistake. In Section 4 we introduce our procedure on the deterministic case. Finally, in Section 5 we lift the algorithm to the nondeterministic case. Some conclusions are drawn in Section 6.

2. Preliminaries

2.1. Relations and Partitions

A *binary relation* from set A to set B is a subset $\rho \subseteq A \times B$. Its size will be denoted by $|\rho|$. We say that $a \in A$ is *in relation with* $b \in B$ whenever $(a, b) \in \rho$, and we denote this by writing $a\rho b$. If we consider binary relations over A (i.e. subsets of $A \times A$), it remains defined the *identity relation* $\iota_A = \{(a, a) : a \in A\}$.

An *equivalence relation* (or *equivalence*) is a relation which is reflexive, symmetric and transitive. Given $a \in A$, the *equivalence class* of a is the set $[a] = \{b : a\rho b\}$. The *quotient set* $A/\rho = \{[a] : a \in A\}$ forms a *partition* of A .

2.2. Languages and Automata

An *alphabet* is a non-empty set Σ of *symbols*. A *string* is a finite sequence $w = w_1 \dots w_n$ of symbols. Σ^* is the set of all finite length strings of symbols in Σ and we call a subset $L \subseteq \Sigma^*$ a *language*.

A *nondeterministic finite state automaton* (NFA) is $\mathcal{N} = \langle Q, \Sigma, q_0, \delta, F \rangle$ where Q is a non-empty finite set of *states*, Σ is the alphabet, q_0 is the *initial state*, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the *transition function* and $F \subseteq Q$ is the set of *final states*. The *degree of nondeterminism* is $r = \max_{x \in \Sigma, q \in Q} \{|\delta(q, x)|\}$. We say that \mathcal{N} is *complete* if $|\delta(q, x)| \geq 1$ for every state and symbol. In what follows we will assume complete automata: this is not a loss of generalities since it is always possible to complete an automaton by adding *one* state and suitable transitions. As usual, the transition function can be recursively extended to strings, i.e. $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$, still denoted by δ .

We say that state $q \in Q$ *accepts* a string $w \in \Sigma^*$ if $\delta(q, w) \cap F \neq \emptyset$. The set of strings accepted by q is denoted by $L(q)$. The language accepted by the automaton is $L(\mathcal{N}) = L(q_0)$. A *minimal* automaton accepting L has the minimum number of states amongst all automata accepting L .

A *deterministic finite state automaton* (DFA) is a NFA \mathcal{D} with the added condition that for each symbol x and each state q , $|\delta(q, x)| = 1$.

For an automaton \mathcal{N} we define the equivalence relation $\sim \subseteq Q \times Q$ as:

$$p \sim q \iff L(p) = L(q)$$

If $p \not\sim q$ we say they are *distinguishable* and if exactly one of the two is final we say they are *trivially distinguishable*.

Given a NFA \mathcal{N} and an equivalence ρ over its states, its *quotient* is defined as $\mathcal{N}/\rho = \langle Q/\rho, \Sigma, [q_0], \delta_\rho, F/\rho \rangle$ where $\delta_\rho([q], x) = \{[q'] \mid q' \in \delta(q, x)\}$. The *Myhill-Nerode Theorem*

[2] establishes that the quotient automaton \mathcal{D}/\sim is well defined and is the unique (up to isomorphism) minimal automaton recognizing $L(\mathcal{D})$.

Definition 1. *Given a NFA, a bisimulation is a binary relation over its states such that, for every pair $(p, q) \in B$:*

$$\mathbf{B1.} \quad p \in F \iff q \in F,$$

$$\mathbf{B2.} \quad \forall x \in \Sigma \forall p' \in \delta(p, x) \exists q' \in \delta(q, x) \wedge (p', q') \in B,$$

$$\mathbf{B3.} \quad \forall x \in \Sigma \forall q' \in \delta(q, x) \exists p' \in \delta(p, x) \wedge (p', q') \in B.$$

Two states p and q are said bisimilar if there exists a bisimulation which contains (p, q) . The set of all bisimulations over the states of \mathcal{N} is denoted by $\mathfrak{B}_{\mathcal{N}}$.

The union of two bisimulation is also a bisimulation. In fact, the following generalization of this observation is a well-known result.

Lemma 1. *$\mathfrak{B}_{\mathcal{N}}$ is closed under union and has a unique largest bisimulation \mathcal{B} , which will be called bisimilarity, it is an equivalence relation, and it relates all and only bisimilar states. $\mathcal{B} \subseteq \sim$ and if \mathcal{N} is deterministic, \mathcal{B} coincides with \sim .*

The above properties — combined with efficient algorithms to compute it — justify the use of \mathcal{B} as an approximation of \sim for nondeterministic automata.

2.3. Partition Aggregation

Given an automaton \mathcal{N} , our goal is to compute the bisimilarity relation \mathcal{B} over its set of states, so that the resulting quotient \mathcal{N}/\mathcal{B} can be returned as the *minimized* version of \mathcal{N} . The *partition-aggregation* strategy will compute an ascending chain $\iota \subseteq B_1 \subseteq \dots \subseteq B_n = \mathcal{B}$ of bisimulation-equivalences.

A partition-aggregation algorithm proceeds by a sequence of merging steps where at each step i the bisimulation B_i is computed. Since each B_i is a bisimulation, the minimization process can be stopped at any step obtaining a language-equivalent automaton with no more states than the input one; the minimization process can be resumed later from this intermediate automaton. In this sense, the algorithm is *incremental*. This property is not shared with top down algorithms that proceed by *partition-refinement* — such as Hopcroft’s Algorithm and its many successors — where only the final result is a bisimulation.

3. The Algorithm Proposed by Almeida *et al.*

This section is devoted to a brief description of the algorithm proposed by Almeida *et al.* It uses the *union-find* [14, 9] data structure to manage the partition of states, so that finding and merging classes with the FIND and UNION primitives can be done in $\mathcal{O}(\alpha(n))$.

Pairs of states are recursively considered until their equivalence is established. Intermediate results are cached, so that queries on pairs of states already found to be (non-)equivalent can be answered in constant time.

Algorithm 1 Aggregation-based minimization by Almeida *et al.*

```

1: function MINIMIZEALMEIDA( $Q, \delta, F$ )
2:   for all  $q \in Q$  do
3:     MAKE( $q$ )
4:    $\bar{E} \leftarrow (F \times F^c) \cup (F^c \times F)$ 
5:
6:   for all  $(p, q) \in Q \times Q$  do
7:      $fp \leftarrow \text{FIND}(p)$ 
8:      $fq \leftarrow \text{FIND}(q)$ 
9:     if  $fp \neq fq \wedge (p, q) \notin \bar{E}$  then
10:       $E \leftarrow \emptyset$ 
11:       $H \leftarrow \emptyset$ 
12:      if EQUIV( $p, q$ ) then
13:        for all  $(p', q') \in E$  do
14:          UNION( $p', q'$ )
15:      else
16:         $\bar{E} \leftarrow \bar{E} \cup H$ 
17:
18:       $\mathcal{P} \leftarrow \{\text{CLASSOF}(p) : p \in Q\}$ 
19:      return  $\mathcal{P}$ 
20:   end function
21: function EQUIV( $p, q$ )
22:   if  $(p, q) \in \bar{E}$  then
23:     return  $\perp$ 
24:   if  $(p, q) \in H$  then
25:     return  $\top$ 
26:
27:    $H \leftarrow H \cup \{(p, q), (q, p)\}$ 
28:   for all  $x \in \Sigma$  do
29:      $(p', q') \leftarrow (\text{FIND}(\delta(p, x)), \text{FIND}(\delta(q, x)))$ 
30:     if  $p' \neq q' \wedge (p', q') \notin E$  then
31:        $E \leftarrow E \cup \{(p', q'), (q', p')\}$ 
32:       if  $\neg \text{EQUIV}(p', q')$  then
33:         return  $\perp$ 
34:
35:    $H \leftarrow H \setminus \{(p, q), (q, p)\}$ 
36:    $E \leftarrow E \cup \{(p, q), (q, p)\}$ 
37:   return  $\top$ 
38: end function

```

At lines 2–3, the identity relation is constructed and pairs of trivially non-equivalent states are added to the *memoization* table \bar{E} . In the main loop at lines 6–16, we iterate over all pairs of states to check for equivalence. If a pair is either on the same class – *i.e.* is a pair of states already found to be equivalent – or is in the memoization table – *i.e.* is a pair of distinguishable states – the minimization continues to the next iteration. Otherwise, two empty collections E and H are prepared, respectively the set of *wondering* pairs of states and the *history* pairs. E and H are considered global variables and can be accessed from EQUIV. The recursive function EQUIV is responsible for checking if states p, q are equivalent and, if so, pairs in E are merged. Otherwise, all visited pairs are set to be distinguishable and this information is used to update \bar{E} . At the end the partition in equivalence classes is returned.

The underlying idea of EQUIV(p, q) is to recursively check the transitions from p and q on all symbols. If two states are found to be cached as distinguishable, the recursion stops returning \perp . If they are found to be in visit, it is useless to continue the visit and nothing can be said (*i.e.* EQUIV returns \top postponing the decision to the upper-level of the recursion). These preliminary checks are at lines 22–25. Next, each x -transition is checked recursively in the loop at 28–33, stopping when the states are found to be distinguishable. At the end, if p and q are not found to be distinguishable, the pair is removed from the history H and added to the “wondering” pairs E and \top is returned.

Detailed proof of the algorithm’s correctness can be found in [11].

On the complexity analysis, the authors claim that the algorithm terminates in time $\mathcal{O}(n^2|\Sigma|\alpha(n))$. This comes from the assumption that each pair visited during the recursion of EQUIV will be skipped on the subsequent iterations of MINIMIZEALMEIDA (*cf.* [11, Lemma 4.9]). The assumption is wrong and a family of counterexamples can be constructed such that MINIMIZEALMEIDA terminates in time $\Theta(n^3|\Sigma|\alpha(n))$ (see Fig 1).

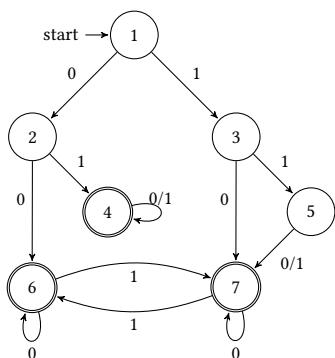


Figure 1: A small counterexample of [11, Lemma 4.9]: pair (6, 7) is visited twice. Consider $\text{EQUIV}(2, 3)$. Pair (6, 7) is visited a first time by reading 0. Symbol 1 leads to (4, 5) which stops the recursion without inserting (6, 7) in \overline{E} . At some subsequent iteration, (6, 7) is visited again via $\text{EQUIV}(6, 7)$. By generalization of this automaton (A_1) automata A_n of $8n - 1$ states can be provided such that n pairs are visited (roughly) n^2 times each.

4. Deterministic Case

In this Section we present our idea to correct the previously presented algorithm, discussing its correctness and complexity. The main point is to run the recursive check on a richer data structure, the *associated graph* introduced below, whereby the running time of the overall algorithm is going to be $\mathcal{O}(n^2|\Sigma|\alpha(n))$.

4.1. The Associated Graph

The reason why Algorithm 1 is not quadratic on some automata is the fact that whenever a pair of distinguishable states is found the recursion stops losing reusable information gathered on elements of E . A graph associated to the automaton clarifies how pairs of states evolve when they are found to either be equivalent or distinguishable.

Definition 2. Given a DFA \mathcal{D} its associated graph $\mathcal{G} = (V, A)$ is defined as:

$$V = Q \times Q,$$

$$A = \{\langle p, q \rangle \rightarrow \langle \delta(p, x), \delta(q, x) \rangle \mid p, q \in Q, x \in \Sigma\}.$$

$\langle p, q \rangle$ is distinguishable if p and q are distinguishable and equivalent otherwise.

Coloring \mathcal{G} with distinguishable vertices black and equivalent vertices white, the problem of computing \sim can be seen as the problem of correctly coloring the associated graph.

The algorithm by Almeida *et al.* can be described as follows: starts by coloring trivially distinguishable and equivalent vertices in black and white, respectively, and in grey the remaining vertices. At each iteration of the main loop, it considers a grey vertex $\langle p, q \rangle$ and starts a visit of \mathcal{G} from it. If the visit reaches a black vertex $\langle p', q' \rangle$, the recursion stops and all vertices in the path from $\langle p, q \rangle$ to $\langle p', q' \rangle$ (saved in H) are colored in black. Otherwise, if all paths lead either to white or grey vertices all visited vertices are colored white. The main issue with Algorithm 1 is that when a black vertex is encountered all information of vertices in $E \setminus H$ gets lost.

4.2. The Algorithm for Deterministic Automata

We now present the minimization algorithm based on the observations above.

Algorithm 2 Proposed algorithm for deterministic automata.

```

1: function MINIMIZEDFA( $Q, \Sigma, \delta, F$ )
2:   for all  $\langle p, q \rangle \in Q \times Q$  do
3:     if  $p, q$  are triv. distinguishable then
4:       COLOR( $\langle p, q \rangle$ )  $\leftarrow$  BLACK
5:     else if  $p = q$  then
6:       COLOR( $\langle p, q \rangle$ )  $\leftarrow$  WHITE
7:     else
8:       COLOR( $\langle p, q \rangle$ )  $\leftarrow$  GREY
9:     for all  $\langle p, q \rangle \in Q \times Q$  do
10:      if COLOR( $\langle p, q \rangle$ ) = GREY then
11:         $\mathcal{H} \leftarrow$  EMPTYGRAPH
12:         $eq \leftarrow$  EQUIV( $\langle p, q \rangle$ )
13:        if  $\neg eq$  then
14:           $\mathcal{H} \leftarrow$  REVERSE( $\mathcal{H}$ )
15:          VISIT( $\mathcal{H}, h$ )
16:        for all  $\langle p', q' \rangle \in$  WHITEV( $\mathcal{H}$ ) do
17:          UNION( $p', q'$ )
18:      end function
19: function EQUIV( $\langle p, q \rangle$ )  $\triangleright \mathcal{H}$  and  $h$  global
20:   if  $\langle p, q \rangle \in \mathcal{H}$  then
21:     return  $\top$ 
22:   else if COLOR( $\langle p, q \rangle$ ) = BLACK then
23:      $h \leftarrow \langle p, q \rangle$ 
24:     return  $\perp$ 
25:   else if COLOR( $\langle p, q \rangle$ ) = WHITE then
26:     return  $\top$ 
27:   else  $\triangleright$  here  $\langle p, q \rangle$  is GREY and fresh
28:     COLOR( $\langle p, q \rangle$ )  $\leftarrow$  WHITE
29:   for all  $x \in \Sigma$  do  $\triangleright$  in lex. order
30:      $\langle p_x, q_x \rangle \leftarrow \langle \delta(p, x), \delta(q, x) \rangle$ 
31:     ADDARC( $\mathcal{H}, \langle p, q \rangle, \langle p_x, q_x \rangle$ )
32:      $eq \leftarrow$  EQUIV( $\langle p_x, q_x \rangle$ )
33:     if  $\neg eq$  then return  $\perp$ 
34:   return  $\top$ 
35: end function

```

The general structure of MINIMIZEDFA is the same as MINIMIZEALMEIDA rewritten in terms of colorings. The only difference is that instead of maintaining two sets E and H we maintain the global variable \mathcal{H} which represents the visited portion of \mathcal{G} . The idea is that when EQUIV($\langle p, q \rangle$) returns to the main loop, after line 15, vertices in \mathcal{H} will be correctly colored, either in WHITE or BLACK. In Algorithm 1 this information was lost while in Algorithm 2 \mathcal{H} is used to determine extra black vertices. Helper procedures REVERSE and VISIT perform, respectively, arc-reverse of a graph and the BLACK-coloring of \mathcal{H} starting from the source vertex h .

Let us analyze the version of EQUIV($\langle p, q \rangle$) in Algorithm 2. At lines 20–27 some base cases are checked. In particular, if $\langle p, q \rangle$ is BLACK, then it is stored in the global variable h and \perp is returned. Otherwise, if $\langle p, q \rangle$ is WHITE we return \top to continue the downstream inspection. Finally, in case $\langle p, q \rangle$ is GREY, it is colored WHITE and at lines 30–34 the for loop tries to continue the recursive visit by reading each symbol *in lexicographic* order. Before each recursive call \mathcal{H} is updated by adding arc $\langle p, q \rangle \rightarrow \langle p_x, q_x \rangle$ – we assume vertex $\langle p_x, q_x \rangle$ is added, if not already present.

Even though we do not give detailed proofs for space reasons, below we outline the main arguments for complexity and correctness.

As far as complexity is concerned it is clear that, summing over all the iterations of the main loop, line 9, the associated graph is visited at most thrice: during the “forward” recursion pass and, optionally, during REVERSE and VISIT. In fact, every vertex starts GREY and gets WHITE during the forward pass of an EQUIV-call. Possibly, if the call returns \perp , some WHITE vertices become BLACK. Altogether, considering the cost of maintaining the union-find data-structure, we have the following: (notice that $|\mathcal{G}| = n^2 + n^2|\Sigma|$)

Theorem 1. *MINIMIZEDFA algorithm terminates in time $\mathcal{O}(n^2|\Sigma|\alpha(n))$.*

To prove correctness we will show the following invariant at line 16: all vertices in \mathcal{H} are correctly colored either BLACK or WHITE – Lemma 2 below.

We start by showing some properties of the coloring performed by `EQUIV` and `VISIT`. First of all, notice that, since we are in the deterministic case, for every $u = \langle p, q \rangle \in V$ and $w \in \Sigma^*$ there is a unique path in \mathcal{G} starting from u and spelling w : denote by $\delta(u, w) = \langle \delta(p, w), \delta(q, w) \rangle$ the last vertex of this path.

Definition 3. Given $u \in V$ and $w \in \Sigma^*$, we say that w for u is:

1. simple if $u \rightsquigarrow \delta(u, w)$ in \mathcal{G} is simple,
2. avalanche² if it is simple and vertex $\delta(u, w)$ is `BLACK`.

If there exists w avalanche for u , denote by $\text{av}(u)$ the lexicographically smallest such w and name $u \rightsquigarrow \delta(u, \text{av}(u))$ the avalanche path of u .

If `EQUIV`(u_0) is called in the main loop, line 12, the visit checks all *simple* words for u_0 in lexicographic order, either until all words are explored or – if it exists – until $\text{av}(u_0)$ is found. Any vertex that can reach the avalanche path should be colored in `BLACK`.

Proposition 1. Consider \mathcal{H} upon return of `EQUIV`(u_0) at line 12. If there exists $u \in \mathcal{H}$ distinguishable, then $\text{av}(u_0)$ exists. Furthermore, if all `WHITE` vertices in $\mathcal{G} \setminus \mathcal{H}$ are equivalent, then for every $u \in \mathcal{H}$, if u is distinguishable we have that $u \rightsquigarrow \delta(u_0, \text{av}(u_0))$ exists in \mathcal{H} .

Proof. Suppose $u \in \mathcal{H}$ is distinguishable. First of all, we prove that there exists $w \in \Sigma^*$ such that $\delta(u_0, w)$ is `BLACK` – recall $\text{av}(u_0)$ is the lexicographically smallest such w . Since $u \in \mathcal{H}$, there exists w_0 such that $u = \delta(u_0, w_0)$. Since u is distinguishable, there exists w_1 such that $\delta(u, w_1)$ is *trivially* distinguishable (i.e. `BLACK` from the start). Thus, $\delta(u_0, w_0 w_1)$ is `BLACK`.

Let $h = \delta(u_0, \text{av}(u_0))$, $u \in \mathcal{H}$ distinguishable and π be the \mathcal{G} -path leading u to some *trivially* distinguishable v . We claim π must cross the avalanche path of u_0 (call it α). Suppose not. Since h is the only `BLACK` vertex in \mathcal{H} , $v \notin \mathcal{H}$. Hence, π must traverse some arc $u' \rightarrow u''$ with $u' \in \mathcal{H}$ and $u'' \notin \mathcal{H}$. By assumption on π and construction of `EQUIV` it follows that $u' \notin \alpha$ and u'' is `WHITE`. Since π leads u'' to v it follows that u'' is distinguishable contradicting the hypothesis on `WHITE` vertices in $\mathcal{G} \setminus \mathcal{H}$. \square

Lemma 2. The following hold at the end of each iteration of loop 9–17:

D1. $\{\langle p, q \rangle \mid \text{COLOR}(\langle p, q \rangle) = \text{BLACK}\} \cap \sim = \emptyset$,

D2. $\{\langle p, q \rangle \mid \text{COLOR}(\langle p, q \rangle) = \text{WHITE}\} \subseteq \sim$.

Proof. Before entering the the loop both properties hold by initialization.

D1. Assume **(D1)** and **(D2)** hold before `EQUIV`(u_0). It is sufficient to prove that at the end of the iteration for every $\langle p, q \rangle \in \mathcal{H}$ we have that $\langle p, q \rangle$ is `BLACK` if and only if $\langle p, q \rangle$ is distinguishable.

(\rightarrow) If $u = \langle p, q \rangle \in \mathcal{H}$ is `BLACK`, then it must have been colored by `VISIT`. Therefore, $eq = \perp$ and before `REVERSE` there was $u \rightsquigarrow h$ in \mathcal{H} . Since h was `BLACK` before the `EQUIV`-call, by **(D1)** it follows h distinguishable. Thus, $\langle p, q \rangle$ is distinguishable.

(\leftarrow) If $u = \langle p, q \rangle \in \mathcal{H}$ is distinguishable, then by Prop. 1 it follows that after `REVERSE` and `VISIT` pair $\langle p, q \rangle$ has been colored in `BLACK`.

²The word “catastrophically” leads to the `BLACK` vertex.

D2. It follows from **(D1)** and the fact that all vertices in \mathcal{H} are either BLACK or WHITE. □

Theorem 2. *Algorithm 2 is correct and incremental.*

Proof. Both correctness and incrementality follow from Lemma 2 and the fact that — upon termination — all vertices of \mathcal{G} are either BLACK or WHITE. In particular, **(D2)** of Lemma 2 proves that UNION is always correct. □

5. Nondeterministic Case

Algorithm 2 is not directly applicable to the nondeterministic case, the reason being that reaching a pair of non-bisimilar states — i.e. sufficient condition to color in BLACK a node by Algorithm 2 — is not a sufficient condition now to declare a pair of states distinguishable.

To tackle this issue we first turn the associated graph into a bipartite graph. In the definition below, for each state p we introduce the *shadow* state \bar{p} as a distinct copy of the *real* p .

Definition 4. *Let \mathcal{N} be a complete NFA. The associated graph $\mathcal{G}(\mathcal{N})$ is a bipartite directed graph with vertices $V_0 \cup V_1$ and arcs $A_0 \cup A_1$, defined as:*

$$\begin{aligned} V_0 &= Q \times Q, \\ V_1 &= \{ \langle p, \bar{q}, x \rangle, \langle \bar{p}, q, x \rangle \mid p, q \in Q, x \in \Sigma \}, \\ A_0 &= \{ \langle p, q \rangle \rightarrow \langle p', \bar{q}, x \rangle, \langle p, q \rangle \rightarrow \langle \bar{p}, q', x \rangle \mid p' \in \delta(p, x), q' \in \delta(q, x) \}, \\ A_1 &= \{ \langle p', \bar{q}, x \rangle \rightarrow \langle p', q' \rangle, \langle \bar{p}, q', x \rangle \rightarrow \langle p', q' \rangle \mid p' \in \delta(p, x), q' \in \delta(q, x) \}. \end{aligned}$$

$\langle p, q \rangle$ in “left” V_0 will be called *equivalent* or *distinguishable* as in Def. 2.

The bisimilarity between p and q (Def. 1) can be checked in two steps: 0) choose $x \in \Sigma$ and $p' \in \delta(p, x)$, and 1) respond with suitable $q' \in \delta(q, x)$. The idea is to mimic step 0) traversing arcs of A_0 and step 1) traversing arcs of A_1 . The triplet $\langle p', \bar{q}, x \rangle \in V_1$ indicates that we have chosen symbol x , state $p' \in \delta(p, x)$, and we are expecting to respond with some $q' \in \delta(q, x)$ (\bar{q} provides the information on the state that must respond). A_1 arcs do something similar.

Then we need to tune up nondeterminism and BLACK coloring of vertices in \mathcal{G} . Observe that $u \in V_0$ needs only one BLACK child to be colored in BLACK, while it needs all children WHITE to be colored in WHITE. Dually, $u \in V_1$ behaves the same but with reversed colors. A check will be performed using the variable $\text{DOUBTS}(u)$ which, roughly speaking, counts how many BLACK neighbours are needed to mark u as BLACK.

5.1. The Algorithm for Nondeterministic Automata

We present Algorithm 3 for the nondeterministic case, whose essential ingredients are those of Algorithm 2. Details are left to the reader.

First of all, notice that we are actually dealing with *four* colors: \perp (never been explored), GREY (in visit), BLACK (distinguishable) and WHITE (equivalent).

Algorithm 3 Proposed algorithm for nondeterministic automata, adapted from DFA case.

```

1:  $\{\perp, \text{BLACK}, \text{WHITE}, \text{GREY}\} \leftarrow \{-1, 0, 1, 2\}$ 
2:
3: function MINIMIZE_NFA( $Q, \Sigma, \delta, F$ )
4:   for all  $\langle p, q \rangle \in Q \times Q$  do
5:     if  $p, q$  are triv. distinguishable then
6:        $\text{COLOR}(\langle p, q \rangle) \leftarrow \text{BLACK}$ 
7:     else if  $p = q$  then
8:        $\text{COLOR}(\langle p, q \rangle) \leftarrow \text{WHITE}$ 
9:     else
10:       $\text{COLOR}(\langle p, q \rangle) \leftarrow \perp$ 
11:   for all  $\langle p, q \rangle \in Q \times Q$  do
12:      $\mathcal{H} \leftarrow \text{EMPTY\_GRAPH}$ 
13:      $\text{EQUIV}(\langle p, q \rangle, 0)$ 
14:     for all  $u \in V_0 \cap \mathcal{H}$  do
15:        $\langle p', q' \rangle \leftarrow u$ 
16:       if  $\text{COLOR}(u) \neq \text{BLACK}$  then
17:          $\triangleright u$  is either GREY or WHITE
18:          $\text{COLOR}(u) \leftarrow \text{WHITE}$ 
19:          $\text{UNION}(p', q')$ 
20:   end function
21:
22: procedure RELAX( $v$ )  $\triangleright \mathcal{H}$  is global
23:   for  $u \in \text{ADJ}(\mathcal{H}, v)$  do
24:      $\text{DOUBTS}(u) \leftarrow \text{DOUBTS}(u) - 1$ 
25:     if  $\text{DOUBTS}(u) = 0$  then
26:        $\text{COLOR}(u) \leftarrow \text{BLACK}$ 
27:        $\text{RELAX}(u)$ 
28:   end procedure
29: function EQUIV( $u, s$ )  $\triangleright \mathcal{H}$  is global
30:   if  $\text{COLOR}(u) \neq \perp$  then
31:     return  $\text{COLOR}(u)$ 
32:
33:    $\text{COLOR}(u) \leftarrow \text{GREY}$ 
34:    $\text{DOUBTS}(u) \leftarrow 0$ 
35:
36:   for  $v \in \text{ADJ}(\mathcal{G}, u) \wedge \text{COLOR}(u) \neq s$  do
37:      $col \leftarrow \text{EQUIV}(v, 1 - s)$ 
38:     if  $col = s$  then
39:        $\text{COLOR}(u) \leftarrow col$ 
40:        $\text{DOUBTS}(u) \leftarrow 0$ 
41:     else if  $col = \text{GREY}$  then
42:        $\text{ADD\_ARC}(\mathcal{H}, v, u)$ 
43:        $\text{DOUBTS}(u) \leftarrow \text{DOUBTS}(u) + 1$ 
44:
45:   if  $\text{COLOR}(u) = \text{GREY}$  then
46:     if  $\text{DOUBTS}(u) = 0$  then
47:        $\text{COLOR}(u) \leftarrow 1 - s$ 
48:     else if  $s = 0$  then
49:        $\text{DOUBTS}(u) \leftarrow 1$ 
50:
51:   if  $\text{COLOR}(u) = \text{BLACK}$  then
52:      $\text{RELAX}(u)$ 
53:   return  $\text{COLOR}(u)$ 
54: end function

```

The procedure MINIMIZE_NFA is structurally the same as MINIMIZE_DFA, the twist being the usage and maintenance of \mathcal{H} . Function EQUIV takes two inputs: the current vertex u and the “side” $s \in \{0, 1\}$ of the bipartition. If u has already been encountered we return its color. Otherwise, it is colored in GREY with zero DOUBTS. At lines 36–43 each successor v of u is recursively visited. In case $s = 0$ ($u \in V_0$) if v is recursively found BLACK, then u can be safely marked BLACK. Otherwise, there is not enough information to safely assign a BLACK/WHITE color to u . In particular, if v is GREY we add arc $v \rightarrow u$ to \mathcal{H} (notice that it is reversed *w.r.t.* the transition) and we increment $\text{DOUBTS}(u)$ – BLACKNESS of u depends on the (possible) future BLACKNESS of v . Case $s = 1$ is dual.

Upon leaving the loop at lines 45–49, if u is still GREY we consider two cases: if there are no doubts (*i.e.* $\text{DOUBTS}(u) = 0$) each of its successors has the same color (either BLACK or WHITE) and this can be safely assigned to u ; otherwise, in case $u \in V_0$ we set $\text{DOUBTS}(u) = 1$.

Proceeding at lines 51–53, if u is made BLACK this information is propagated (RELAXED) to its neighbours in \mathcal{H} . Notice that in this case we explicitly define the procedure RELAX: in Algorithm 2 the corresponding procedure VISIT’s purpose, was to color in BLACK all vertices reachable from some distinguishable vertex. In Algorithm 3 we also need to consider the doubts of each vertex v by coloring in BLACK only non-doubtful vertices.

The complexity is again easy to be determined. Observe that in case the automaton has n states and m transitions, $|\mathcal{G}| \leq 7nm$. EQUIV performs the equivalent of a visit of \mathcal{G} and RELAX visits every arc of \mathcal{H} at most once. Hence, their cost over all the execution of Algorithm 3 is

bounded by the size of \mathcal{G} . Considering the cost of maintaining the union-find data structures and recalling that r is the degree of nondeterminism, we have the following:

Theorem 3. *MINIMIZE_{NFA} terminates in time $\mathcal{O}(nm\alpha(n)) \subseteq \mathcal{O}(n^2|\Sigma|r\alpha(n))$.*

Let us briefly discuss correctness.

Proposition 2. *Let $p \neq q$ and $u = \langle p, q \rangle$ be WHITE. Then, for every arc $u \rightarrow v \in A_0$, vertex v is either GREY or WHITE.*

Proof. Since $p \neq q$, there are only two places at which $u = \langle p, q \rangle$ changed from GREY to WHITE:

Line 18. In this case, upon termination of $\text{EQUIV}(u, 0)$ the pair is GREY and $\text{DOUBTS}(u) = 1$ (line 49). In the loop of its EQUIV -call every neighbour was recursively found to be either GREY or WHITE – or u would have been colored in BLACK. Finally, it cannot be the case that some GREY neighbour v became BLACK afterward: $\text{RELAX}(v)$ would have colored u in BLACK by setting $\text{DOUBTS}(u) = 0$.

Line 47. In this case, inside the loop of $\text{EQUIV}(u, 0)$ none of u 's neighbours were found to be either BLACK or GREY. Thus, they must all be WHITE.

Therefore, every neighbour of u is either GREY or WHITE. □

Proposition 3. *At line 19 (end of the main loop), for every $v \in V_1$ if v is either GREY or WHITE, then there exists $v \rightarrow u' \in A_1$ such that u' is WHITE.*

Proof. If v is WHITE, then it must have been colored at line 39 after finding a WHITE neighbour. If v is GREY, upon termination of EQUIV at line 13 some of its neighbours must have been found to be GREY. Since every GREY left node is colored in WHITE before the end of the iteration, it follows again that v has some WHITE neighbour. □

Lemma 3. *The following hold at line 19 (end of the main loop):*

N1. $\mathcal{R}_W = \{\langle p, q \rangle \mid \text{COLOR}(\langle p, q \rangle) = \text{WHITE}\} \subseteq \mathcal{B}$,

N2. $\mathcal{R}_B = \{\langle p, q \rangle \mid \text{COLOR}(\langle p, q \rangle) = \text{BLACK}\} \cap \mathcal{B} = \emptyset$.

Proof.

N1. By Lemma 1 it is sufficient to prove that \mathcal{R}_W is a bisimulation.

First, notice that pairs violating **(B1)** are colored in BLACK from the start. Consider $\langle p, q \rangle \in \mathcal{R}_W$. If $p = q$, **(B2)** and **(B3)** trivially hold. Otherwise, let $x \in \Sigma$ and $p' \in \delta(p, x)$. From Prop. 2 it follows that $v = \langle p', \bar{q}, x \rangle$ is either GREY or WHITE. From Prop. 3 it follows that some neighbour u' of v is in \mathcal{R}_W . By Def. 4 we have $u' = \langle p', q' \rangle$ for some $q' \in \delta(q, x)$. Thus, **(B2)** holds for $\langle p, q \rangle$. The very same argument can be used to prove that **(B3)** holds for $\langle p, q \rangle$. Hence, \mathcal{R}_W is a bisimulation.

N2. The result follows from **(N1)** and the fact that vertices in $V_0 \cap \mathcal{H}$ are either BLACK or WHITE. □

Theorem 4. *MINIMIZE_{NFA} is correct and incremental.*

Proof. The inclusion $\mathcal{R}_W \subseteq \mathcal{B}$ is **(N1)**. For the converse inclusion notice that it can be easily checked that, upon termination, every pair $\langle p, q \rangle \in V_0$ is either BLACK or WHITE. Therefore,

$$\mathcal{B} = V_0 \cap \mathcal{B} = (\mathcal{R}_B \cup \mathcal{R}_W) \cap \mathcal{B} = \emptyset \cup (\mathcal{R}_W \cap \mathcal{B}) \subseteq \mathcal{R}_W.$$

Incrementality follows again from **(N1)**. □

6. Conclusions

Bisimilarity is a fundamental (equivalence) relation among the states of finite automata, finding applications and variants in a number of different areas. Algorithms for computing bisimilarity are a *classic* and can be subdivided in two categories: top-down and a bottom-up. The former (partition refinement) approach starts with a coarse partition and refines it until the result is produced, while the latter (partition aggregation) starts from a singleton-classes equivalence relation and merges classes until possible.

Although algorithms belonging to the bottom-up category are, to the best of our knowledge, still currently asymptotically slower than their alternative ones, aggregation based techniques enjoy the property of being *incremental*: automata resulting at intermediate stages of the computation are *partially minimized yet language-equivalent to the input one*.

Moreover, partition aggregation algorithms, even though less celebrated than partition refinement ones – introduced by Hopcroft and generalized by Paige and Tarjan –, are interesting (at least) for two reasons. The first is theoretical: if two methods computes the same relation (just one from “above” and the other from “below”), why is there a complexity gap? Is there some (hidden) *cost* involved in maintaining incrementality? The second is practical: some applicative contexts can greatly benefit from having a partially minimized *equivalent* automaton, especially when, as alternative, long sequences of refinement steps are involved.

In this work, while fixing a minor mistake in the algorithm by Almeida *et al.*, we reduced bisimilarity computation to a coloring problem on an associated graph. We then extended the algorithm to nondeterministic case, obtaining a complexity improvement on the best known bound for this case. The time complexity of both algorithms carry the $\alpha(n)$ factor which we aim to shave off as future work. As a further line of research, it will be interesting to investigate the effect of applying the technique introduced here to the *color refinement algorithm* (a.k.a. Weisfeiler-Leman-1 algorithm, see [15]), currently implemented using an algorithm by Cardon and Crochemore (see [16]) belonging to the top-down/partition-refinement category.

References

- [1] E. F. Moore, *Gedanken-Experiments on Sequential Machines*, Princeton University Press, 1956, pp. 129–154. doi:10.1515/9781400882618-006.
- [2] J. E. Hopcroft, R. Motwani, J. D. Ullman, Pearson Education, 2014.
- [3] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: SWAT, 1972.

- [4] R. Paige, R. E. Tarjan, Three partition refinement algorithms, *SIAM J. Comput.* 16 (1987) 973–989.
- [5] J. E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, 1971.
- [6] J. F. Groote, J. Martens, E. P. de Vink, Lowerbounds for bisimulation by partition refinement (2022). URL: <https://arxiv.org/abs/2203.07158>.
- [7] B. W. Watson, Taxonomies and toolkits of regular language algorithms, 1995.
- [8] B. W. Watson, J. Daciuk, An efficient incremental dfa minimization algorithm, *Natural Language Engineering* 9 (2003) 49 – 64.
- [9] R. E. Tarjan, Efficiency of a good but not linear set union algorithm, *J. ACM* 22 (1975) 215–225. doi:10.1145/321879.321884.
- [10] J. E. Hopcroft, A linear algorithm for testing equivalence of finite automata, volume 114, Defense Technical Information Center, 1971.
- [11] M. Almeida, N. Moreira, R. Reis, Incremental dfa minimisation, *RAIRO - Theoretical Informatics and Applications* 48 (2014) 173–186. doi:10.1051/ita/2013045.
- [12] J. Björklund, L. G. Cleophas, Minimization of finite state automata through partition aggregation, in: *LATA*, 2017.
- [13] J. Björklund, L. Cleophas, Aggregation-based minimization of finite state automata, 2020. doi:10.1007/s00236-019-00363-5.
- [14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to algorithms*, 3rd edition, 2009, pp. 170–173.
- [15] M. Grohe, K. Kersting, M. Mladenov, P. Schweitzer, *Color refinement and its applications*, 2021.
- [16] A. Cardon, M. Crochemore, Partitioning a graph in $o(|a| \log^2 |v|)$, *Theor. Comput. Sci.* 19 (1982) 85–98. doi:10.1016/0304-3975(82)90016-0.