A quadratic algorithm for road coloring

Marie-Pierre Béal, Dominique Perrin

A quadratic algorithm for road coloring

Marie-Pierre Béal and Dominique Perrin
Université Paris-Est
Laboratoire d'informatique Gaspard-Monge, CNRS
77454 Marne-la-Vallée Cedex 2, France
{beal,perrin}@univ-mlv.fr

Abstract

The road coloring theorem states that every aperiodic directed graph with constant out-degree has a synchronized coloring. This theorem had been conjectured during many years as the road coloring problem before being settled by A. Trahtman.

Trahtman's proof leads to an algorithm that finds a synchronized labeling with a cubic worst-case time complexity. We show a variant of his construction with a worst-case complexity which is quadratic in time and linear in space. We also extend the road coloring theorem to the periodic case.

1 Introduction

Imagine a map with roads which are colored in such a way that a fixed sequence of colors, called a homing sequence, leads the traveler to a fixed place whatever be the starting point. Such a coloring of the roads is called synchronized and finding a synchronized coloring is called the road coloring problem. In terms of graphs, it consists in finding a synchronized labeling in a directed graph.

The road coloring theorem states that every aperiodic directed graph with constant out-degree has a synchronized coloring (an aperiodic graph is strongly connected and the gcd of its cycles is equal to 1). It has been conjectured under the name of the road coloring problem by Adler, Goodwin, and Weiss [2], and solved for many particular types of automata (see for instance [2], [8], [7], [10]). Trahtman settled the conjecture in [11].

The problem is not only a puzzle but has many applications in various areas like coding or design of computational systems. That systems are often modeled by finite-state automata (*i.e.* graphs with labels). Due to some noise, the system may take a wrong transition. This noise may for instance result from the physical properties of sensors, from unreliability of computational hardware, or from insufficient speed of the computer with respect

to the arrival rate of input symbols. It turns out that the asymptotic behavior of synchronized automata is better than the behavior of unsynchronized ones (see [5]). Synchronized automata are thus less sensitive to the effect of noise.

In the domain of coding, automata with outputs (*i.e.* transducers) can be used as encoders or decoders. When they are synchronized, the behavior of the coder or of the decoder is improved in the presence of noise or errors. For instance, the well known Huffman compression scheme leads to a synchronized decoder provided the lengths of the codewords of the Huffman code are mutually prime. It is also a consequence of the road coloring theorem that coding schemes for constrained channels can have sliding block decoders and synchronized encoders (see [1] and [9]).

Trahtman's proof is constructive and leads to an algorithm that finds a synchronized labeling with a cubic worst-case time complexity [12]. The algorithm consists in a sequence of flips of edges with the same origin so that the resulting automaton is synchronized. One first searches a sequence of flips leading to an automaton with a stable pair of states (s, t) (*i.e.* with good synchronizing properties). One then quotients the automaton by the congruence generated by (s, t) and iterates the process on this smaller automaton. Trahtman's method for finding the flips leading to a stable pair has a worst-case quadratic time complexity, which makes his algorithm cubic.

In this paper, we design a worst-case linear time algorithm for finding a sequence of flips until the automaton has a stable pair. This makes the algorithm quadratic in time and linear in space. The sequence of flips is obtained by fixing a color, say red, and by considering the red cycles formed with red edges, taking into account the positions of the root of red trees attached to each cycle. The prize to pay for decreasing the time complexity is some more complication in the choice of the flips.

The article is organized as follows. In Section 2, we give some definitions to formulate the problem in terms of finite automata instead of graphs. In Section 3 we describe Trahtman's algorithm and our variant. We give both an informal description of the algorithm with pictures illustrating the constructions, and a pseudocode. The complexity is analyzed in Section 5.

2 The road coloring theorem

In order to formulate the *road coloring problem* we introduce the notion of automaton. A (finite) *automaton* $\mathcal{A}$ over some (finite) alphabet A is composed of a finite set Q of states and a finite set E of edges which are triples (p, a, q) where p, q are states and a is a symbol from A called the *label* of the edge.

An automaton is *deterministic* if, for each state p and each letter a, there is at most one edge starting in p and labeled with a. It is *complete*

deterministic if, for each state p and each letter a, there is exactly one edge starting in p and labeled with a. This implies that, for each state p and each word w, there is exactly one path starting in p and labeled with w. If this unique path ends in a state q, we denote by $p \cdot w$ the state q.

An automaton is *irreducible* if its underlying graph is strongly connected. It is *aperiodic* if it is irreducible and if the gcd of the cycles of its graph is equal to 1[1].

A *synchronizing word* of a deterministic complete automaton is a word w such that for any states p, q, one has $p \cdot w = q \cdot w$. A synchronizing word is also called a *reset sequence* or a *magic sequence*, or also a *homing word*. An automaton which has a synchronizing word is called *synchronized* (see an example on the right part of Figure 1).

Two automata which have isomorphic underlying graphs are called *equivalent*. Hence two equivalent automata differ only by the labeling of their edges. In the sequel, we shall consider only complete deterministic automata.

The *road coloring theorem* can be stated as follows.

Theorem 1 (A. Trahtman [11])**.** *Any aperiodic complete deterministic automaton is equivalent to a synchronized one.*

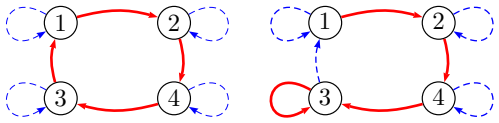

Figure 1: Two complete aperiodic deterministic automata labeled on $A = \{a, b\}$. A thick red plain edge is an edge labeled by a while a thin blue dashed edge is an edge labeled by b. The automaton on the left is not synchronized. The one on the right is synchronized; for instance, the word aaa is a synchronizing word. The two automata are equivalent.

3 An algorithm to find a synchronized coloring

Trahtman's proof of Theorem 1 is constructive and gives an algorithm to find a labeling that makes the automaton synchronized provided it is aperiodic.

In the sequel $\mathcal{A}$ denotes an n-state deterministic and complete automaton over an alphabet A. We fix a particular letter $a \in A$. Edges labeled by a are also called *red edges*. The other ones are called *b-edges*.

A *synchronizable pair* of states in an automaton is a pair of states (p, q) such that there is a word w with $p \cdot w = q \cdot w$. It is well known that an automaton is synchronized if all its pairs of states are synchronizable.

[1]Note that this notion, which is usual for graphs, is not the notion of aperiodic automata used elsewhere and which refers to the period of words labeling the cycles (see *e.g.* [6]).

A *stable pair* of states in an automaton is a pair of states (p, q) such that, for any word u, $(p \cdot u, q \cdot u)$ is a synchronizable pair. In a synchronized automaton, any pair of states is stable. Note that if (p, q) is a stable pair, then for any word u, $(p \cdot u, q \cdot u)$ also is a stable pair, hence the terminology. Note also that, if (p, q) and (q, r) are stable pairs, then (p, r) also is a stable pair. It follows that the relation defined on the set of states by $p \equiv q$ if (p, q) is a stable pair, is an equivalence relation. It is actually a congruence (*i.e.* $p \cdot u \equiv q \cdot u$ whenever $p \equiv q$) called the *stable pair congruence*. More generally, a congruence such that any pair of states in a same class is stable, is called a *stable pair congruence*. The congruence *generated by a stable pair* (s, t) is the least congruence such that s and t belong to a same class. It is a stable pair congruence.

The computation of the congruence generated by (s, t) can be performed by using UNION/FIND functions computing respectively the union of two classes and the leader of the class of a state. After merging two classes whose leaders are p and q respectively, we need to merge the classes of $p \cdot \ell$ and $q \cdot \ell$ for any $\ell \in A$. A pseudocode for merging classed in given in Procedure MERGE below.

```
MERGE( stable pair (s, t))
1  x ← FIND(s)
2  y ← FIND(t)
3  if x ≠ y
4     then UNION(x, y)
5          for ℓ ∈ A
6               do MERGE(x · ℓ, y · ℓ)
```

If $\mathcal{A} = (Q, E)$ is an automaton, the *quotient* of $\mathcal{A}$ by a stable pair congruence is the $\mathcal{B}$ whose states are the classes of Q under the congruence. The edges of B are the triples $(\bar{p}, \ell, \bar{q})$ where (p, ℓ, q) is an edge of $\mathcal{A}$. The automaton $\mathcal{B}$ is complete deterministic when $\mathcal{A}$ is complete deterministic. The automaton $\mathcal{B}$ is irreducible (resp. aperiodic) when $\mathcal{A}$ is irreducible (resp. aperiodic).

Lemma 2 (Culik *et al.* [4])**.** *If the quotient $\mathcal{B}$ of an automaton $\mathcal{A}$ by a stable pair congruence is equivalent to a synchronized automaton $\mathcal{B}'$, then $\mathcal{A}$ is equivalent to a synchronized automaton $\mathcal{A}'$.*

Proof. Let $\mathcal{B}'$ be a synchronized labeling of $\mathcal{B}$. The equivalent automaton $\mathcal{A}'$ is defined as follows. If there is an edge (p, ℓ, q) in $\mathcal{A}$ but no edge $(\bar{p}, \ell, \bar{q})$ in $\mathcal{B}'$, then there is an edge $(\bar{p}, c, \bar{q})$ in $\mathcal{B}'$ with $\ell \neq c$. We flip the labels of the two edges labeled ℓ and c going out of p in $\mathcal{A}$. This definition is consistent with the definition of the edges of $\mathcal{B}$. After all flips, $\mathcal{B}'$ is still a quotient of $\mathcal{A}'$.

Let us show that $\mathcal{A}'$ is synchronized. Let w be a synchronizing word of $\mathcal{B}'$ and r the state ending any path labeled by w in $\mathcal{B}'$. Let p, q be two

states of $\mathcal{A}'$. Then $p \cdot w$ and $q \cdot w$ belong to the congruence class of r. Hence $(p \cdot w, q \cdot w)$ is a stable pair of $\mathcal{A}'$. Therefore (p, q) is a synchronizable pair of $\mathcal{A}'$. All pairs of $\mathcal{A}'$ being synchronizable, $\mathcal{A}'$ is synchronized. □

Trahtman's algorithm for finding a synchronized coloring of an aperiodic automaton $\mathcal{A}$ consists in finding an equivalent automaton $\mathcal{A}'$ of $\mathcal{A}$ that has a stable pair (s, t), then in recursively finding a synchronized coloring $\mathcal{B}'$ for the quotient automaton $\mathcal{B}$ by the congruence generated by (s, t), and finally in lifting up this coloring into the initial automaton as follows. If there is an edge (p, ℓ, q) in $\mathcal{A}$ but no edge $(\bar{p}, \ell, \bar{q})$ in $\mathcal{B}'$, then there is an edge $(\bar{p}, c, \bar{q})$ in $\mathcal{B}'$ with $\ell \neq c$. Then we flip the labels of the two edges labeled ℓ and c going out of p in $\mathcal{A}$. The algorithm is described in the following pseudocode, where the procedure FindStablePair, which finds a stable pair of states in any aperiodic automaton of size at least 2, is described in the next section.

```
FindColoring( aperiodic automaton A)
1  A_0 ← A
2  while A has at least two states
3        do A, (s,t) ← FindStablePair(A)
4           lift the coloring of A up to A_0
5           A ← the quotient of A by the congruence generated par (s,t)
6  return A_0
```

The termination of the algorithm is guaranteed by the fact that the number of states of the quotient automaton of $\mathcal{A}$ is strictly less than the number of states of $\mathcal{A}$.

4 Finding a stable pair

In this section, we consider an aperiodic complete deterministic automaton $\mathcal{A}$ over the alphabet A. We design an algorithm for finding an equivalent automaton with a stable pair, which has a linear-time complexity.

In order to describe the algorithm, we give some definitions and notations.

Let $\mathcal{R}$ be the subgraph of the graph of $\mathcal{A}$ made of the red edges. The graph $\mathcal{R}$ is a disjoint union of connected component called *clusters*. Since each state has exactly one outgoing edge in $\mathcal{R}$, each cluster contains a unique (red) cycle with trees attached to the cycle at their root. If r is the root of such a tree, its *children* are the states p such that p is not on the a red cycle and (p, a, r) is an edge. If p, q belong to a same tree, p is an *ancestor* of q (or q is a *descendant* of p) in the tree if there is a red path from q to p.

For each state p of the cluster, we define the *level* of p as the distance between p and the root of the tree containing p. If p belongs to the cycle, its level is null. For each state we denote by *root*[p] the root of the tree

containing p. A *maximal state* is a state of maximal level. A *maximal cluster* is a cluster containing a maximal state. A *maximal tree* is a tree containing at least one maximal state and rooted at a state of level 0. The *level of an automaton* is the level of a maximal state. A *maximal root* is the root of a maximal tree.

Let $\mathcal{C}$ be the red cycle of a cluster and k its length. We number its states from 0 to $k-1$ along the cycle. If p belongs to the cycle, we denote this number by $num[p]$.

The algorithm relies on the following key lemma due to Trahtman [11]. It uses the notion of minimal images in an automaton. An *image* in an irreducible automaton $\mathcal{A} = (Q, E)$ is a set of states $I = Q \cdot w$, where w is a word and $Q \cdot w = \{q \cdot w \mid q \in Q\}$. A *minimal image* in an automaton is an image which is a minimal element of the set of images for set inclusion. In an irreducible automaton two minimal images have the same cardinality which is called the *minimal rank* of $\mathcal{A}$.

Lemma 3 (Trahtman [11])**.** *Let $\mathcal{A}$ be an irreducible complete deterministic automaton with a positive level. If all maximal states in $\mathcal{A}$ belong to the same tree, then $\mathcal{A}$ has a stable pair.*

Proof. Since $\mathcal{A}$ is irreducible, there is a minimal image I containing a maximal state p. Let $\ell > 0$ the level of p (*i.e.* the distance between p and the root r of its tree). Let us assume that there is a state $q \neq p \in I$ of level ℓ. Then the cardinal of $I \cdot a^l$ is strictly less than the cardinal of I, which contradicts the minimality of I. Thus all states but p in I have a level strictly less than ℓ.

Let m be a common multiple of the lengths of all red cycles. Let $\mathcal{C}$ be the red cycle containing r. Let s_0 be the predecessor of r in $\mathcal{C}$ and s_1 the child of r containing p in its subtree. Let $J = I \cdot a^{l-1}$ and $K = J \cdot a^m$. Since the level of all states of I but p is less than $l-1$, the set J is equal to $\{s_1\} \cup R$, where R is a set of states belonging to some red cycle. Since for any state q in a red cycle, $q \cdot a^m = q$, we get $K = \{s_0\} \cup R$.

Let w be a word of minimal rank (*i.e.* $Q \cdot w$ is a minimal image). For any word v, the minimal images $J \cdot vw$ and $K \cdot vw$ have the same cardinal equal to the cardinal of I. This implies that $s_0 \cdot vw = s_1 \cdot vw$. As a consequence $(s_0 \cdot v, s_1 \cdot v)$ is synchronizable and thus (s_0, s_1) is a stable pair. □

We say that an automaton satisfies Condition (*) if *all maximal states belong to the same tree*. Note that Condition (*) implies that all maximal states belong to a same cluster.

A set of edges going out of a state p is called a *bunch* if these edges all end in a same state q. Note that if a state q has two incoming bunches from p, p', then (p, p') is a stable pair.

We describe below sequences of flips of edges that make the resulting equivalent automaton have a stable pair.

4.1 The case of null maximal level

We first assume that the level of the automaton is $\ell = 0$. The subgraph $\mathcal{R}$ of red edges is a disjoint union of cycles. If the set of outgoing edges of each state is a bunch, then there is only one red cycle and the automaton is not aperiodic. Let p be a state whose set of outgoing edges is not a bunch. There exists $b \neq a$ and $q \neq r$ such that (p, a, q) and (p, b, r) are edges. We flip these edges. This gives an automaton $\mathcal{A}$ which satisfies Condition (*). Let s be the state which is the predecessor of r on its red cycle. It follows from the proof of Lemma 3 that the pair (p, s) is stable.

This case is described in the pseudocode FlipEdgesLevelZero where *previous*$[q]$ is the predecessor of q on its red cycle. The function FlipEdgesLevelZero($\mathcal{A}$) returns an automaton equivalent to $\mathcal{A}$ together with a stable pair.

```
FlipEdgesLevelZero(automaton A)
    We assume that the maximal level of the states is 0
1   for each state p on a red cycle C
2       do
3           if the set of outgoing edges of p is not a bunch
4              then let (p,a,q) and (p,b,r) be edges with b ≠ a and q ≠ r
5                   flip these edges
6                   s ← previous[q]
7                   return A, (p,s)
8   return error(A is not aperiodic)
```

4.2 The case of non-null maximal level

4.2.1 Preliminary flips

Let us now assume that the level of the automaton is $\ell > 0$. We assume that the level of the states have been computed with a depth-first exploration of the subgraph $\mathcal{R}$ of red edges in a linear time.

In a preliminary step, we perform a sequence of flips of $\mathcal{A}$ which leads to an equivalent automaton satisfying the following Condition (**): *for any state of maximal level p, there is no b-edge (t, b, p) where t is an ancestor of p distinct from the root in the tree containing p.*

In order to achieve this goal, we compute for each maximal tree $\mathcal{T}$ a set S of states t such that there is a b-edge (t, b, p) with t ancestor of p in $\mathcal{T}$, p of maximal level, t distinct from the root of $\mathcal{T}$, and there is no other state t' ancestor of t satisfying the same condition.

The sets S are computed in linear time by performing a depth-first exploration of each tree. For each tree, we start at its root following the red edges in the backward direction and examining the b-edges (t, b, p) in the forward direction. We assume that a function Exploration returns all

states t in S together with a b-edge (t, b, p), p maximal, satisfying the above conditions.

We then flip (t, b, p) and the red edge going out of t for each $t \in S$ as long as there still remains at least one maximal tree in the automaton.

This sequence of flips is described in the function ScanTrees below. In the pseudocode, the value of $previous[r, p]$ is the stable pair (s_0, s_1), where s_0 is the predecessor of r on its red cycle, and s_1 is the child of r which is an ancestor of p.

Note that each such flip creates a new red cycle and hence a new cluster. The level of each state belonging to each new cluster decreases strictly. In particular, the level of p and t becomes 0. One can know whether there remains maximal states after each flip without recomputing all levels: with a depth-first search exploration of the trees, one computes for each state $t \in S$ the number of maximal states descendant of t, and we update the whole number of maximal states after each flip. This can be done in a linear time complexity for the whole sequence of flips.

The procedure either stops when we have flipped two outgoing edges of all states in S (and there still remains states of maximal level ℓ), or when we have flipped two outgoing edges of all states in S but one. In the first case, for any state of maximal level p, there is no b-edge (t, b, p) where t is an ancestor of p distinct from the root of the tree of p (*i.e.* Condition(**) is fulfilled). In the second case, the resulting automaton $\mathcal{A}$ has a single maximal tree. Hence it satisfies Condition (*). Thus the function ScanTrees($\mathcal{A}$) returns either an equivalent automaton together with a stable pair of states, or just an equivalent automaton satisfying Condition (**).

```
ScanTrees( automaton A)
    We assume that the maximal level of the states is positive
 1  M ← the number of states of maximal level
 2  S ← Exploration(A)
    (to each state t ∈ S is associated a b-edge (t, b, p))
 3  for each state t ∈ S
 4        do n(t) ← the number of maximal states descendant of t
 5  for each state t ∈ S
 6        do
 7           if n(t) < M
 8              then flip (t, b, p) and the red edge going out of t
 9                   M ← M − n(t)
10                   S ← S − {t}
11  if S = ∅
12     then return A
13     else  r ← the root of the unique maximal tree of A
14           return A, previous[r, p]
```

The preliminary treatment is done only one time at the beginning and it has a linear time complexity.

4.2.2 Main treatment

We now assume that the automaton satisfies Condition (**). We compute the level of the states with a depth-first exploration of the subgraph $\mathcal{R}$ of red edges (following the edges backward) in a linear time. For each node, one can access the root of his tree in constant time.

Let $\mathcal{C}$ be a red cycle containing a maximal tree $\mathcal{T}$ rooted at r. Let s_0 be the predecessor of r in the red cycle. We denote by $s_1, s_2, .., s_\rho$ the children of r which are ancestors of a maximal state, and by p_i a maximal state descendant of s_i for $1 \leq i \leq \rho$. We denote by t_i a predecessor of p_i by a b-edge (t_i, b_i, p_i). We describe a procedure FlipEdges$(\mathcal{A}, r)$ which is a sequence of flips of edges such that the automaton obtained after the sequence of flips has a unique maximal tree. Note that the levels will not be recomputed after each flip (which would increase the time complexity too much). Each update will be explicitly mentioned.

We consider several cases depending on the position of the states t_i in $\mathcal{A}$ satisfying Condition (**). If there is a state t_i which is not in the same cluster as r or if $level[t_i] > 0$, then an automaton equivalent to $\mathcal{A}$ satisfying Condition (*) is obtained by flipping the edge (t_i, b_i, p_i) and the red edge going out of t_i. Indeed, one may easily check that, after the flip, all states of maximal level belong to the same tree as p_i.

If there is a state t_i that belongs to the cycle $\mathcal{C}$ but which is not contained in the interval $[r_k \,..\, r[$, where $r, r_2, .. r_k$ is the ordered list of *maximal* roots, the same conclusion holds by flipping the edge (t_i, b_i, p_i) and the red edge going out of t_i, as is shown in Figure 2.

Let us now assume that all states t_i belong to the cycle $\mathcal{C}$ and $t_i \in [r_k \,..\, r[$ for $1 \leq i \leq \rho$.

We first consider the case where there are at least two distinct states t_i (see Figure 3). Without loss of generality, we can for instance assume that $t_1 < t_2$ in $[r_k \,..\, r[$. We flip the edge (t_1, b_1, p_1) and the red edge going out of t_1. We denote by $\mathcal{T}'$ the new tree rooted at r. If the height of $\mathcal{T}'$ is greater than ℓ, the automaton satisfies Condition (*) (see the right part of Figure 3). Otherwise the height of $\mathcal{T}'$ is at most ℓ (see the left part of Figure 4). In that case, we also flip the edge (t_2, b_2, p_2) and the red edge going out of t_2. The new equivalent automaton satisfies Condition (*) (see the right part of Figure 4). The computation of the size of $\mathcal{T}'$ is detailed in the complexity issue.

We now consider the case where there all states t_i are equal to a same state t. This includes the case where the number ρ of children of r having a maximal descendant, is equal to 1.

We first treat the case of $\rho > 1$ and all states t_i equal to t. Let $\mathcal{T}_0$ be the tree rooted at r obtained if we flip (t, b_1, p_1) and the red edge going out of t, and if we keep only r and the subtree rooted at the child s_0. The states of the tree $\mathcal{T}_0$ rooted at r are represented in salmon in the left part of Figure 5.

This step again needs a computation of the height of $\mathcal{T}_0$ explained in the complexity issue. If the height of $\mathcal{T}_0$ is greater than the height of $\mathcal{T}$, we do the flip and the equivalent automaton satisfies Condition (*). If the height of $\mathcal{T}_0$ is less than the height of $\mathcal{T}$, we also do the flip of (t, b_1, p_1) and the red edge going out of t. We then call again the procedure FlipEdges$(\mathcal{A}, r)$ with this new red cycle. This time, the height of the new tree $\mathcal{T}_0$, denoted T_0' is equal to the height of $\mathcal{T}$. Hence this call is done at most one time for a given maximal root r. Finally, we consider the case where the height of $\mathcal{T}$ and $\mathcal{T}_0$ are equal (see the left part of Figure 5). If the set of outgoing edges of s_0 is a bunch and there is a state s_i, for $1 \leq i \leq \rho$, whose set of outgoing edges also is a bunch, we get a trivial stable pair (s_0, s_i). If the set of outgoing edges of s_0 is a bunch but none of the sets of outgoing edges of s_i for $1 \leq i \leq \rho$ is a bunch (as in the left part of Figure 5), we flip (t, b_1, p_1) and the red edge going out of t. The height of the new tree $\mathcal{T}_0$ (obtained if we flip (t, b_2, p_2) and the red edge going out of t and keep only r and the subtree rooted at the child s_1) has the same height as $\mathcal{T}$. We then call again the procedure FlipEdges$(\mathcal{A}, r)$ with this new red cycle. This time, the height of the new tree $\mathcal{T}_0$ is equal to the height of $\mathcal{T}$ and the set of outgoing edges of the predecessor of r on the cycle is not a bunch. This call is thus performed at most one time.

If the set of outgoing edges of s_0 is not a bunch, let (s_0, b_0, q_0) be a b-edge going out of s_0 with $q_0 \neq r$. If q_0 does not belong to $\mathcal{T}$, we get an equivalent automaton satisfying Condition (*) by flipping (s_0, b_0, q_0) and the red edge going out of s_0. If q_0 belongs to $\mathcal{T}$, we flip (s_0, b_0, q_0) and the red edge going out of s_0. We also flip (t, b_1, p_1) and the red edge going out of t if q_0 is not a descendant of s_1, or (t, b_2, p_2) and the red edge going out of t in the opposite case. Note that $s_0 \neq t$ since the height of $\mathcal{T}_0$ is equal to the non-null height of $\mathcal{T}$. We get an equivalent automaton satisfying Condition (*) (see the right part of Figure 6).

We now treat the case of $\rho = 1$. As before we denote by $\mathcal{T}_0$ the tree rooted at r obtained if we flip (t, b_1, p_1) and the red edge going out of t and keep only r and the subtree rooted at the child s_0. If the height of $\mathcal{T}_0$ is greater than the height of $\mathcal{T}$, we do the flip and the equivalent automaton satisfies Condition (*). If the height of $\mathcal{T}_0$ is less than the height of $\mathcal{T}$, we do not flip the edge (t, b_1, p_1) and the red edge going out of t, and return the automaton together with the edge (t, b_1, p_1). We now come to the case where the height of $\mathcal{T}_0$ is equal to the height of $\mathcal{T}$. If the sets of outgoing edges of s_0 and s_1 are bunches, there is a trivial stable pair (s_0, s_1). If the set of outgoing edges of s_0 is a bunch and the set of outgoing edges of s_1 are not a bunch (see the left part of Figure 7), we flip the edge (t, b_1, p_1) and the red edge going out of t. We then call the procedure FlipEdges$(\mathcal{A}, r)$ with this new red cycle. The root r has now a unique child (s_1) ancestor of maximal state whose set of outgoing edges is a bunch (see the right part of Figure 7). This call is thus performed at most one time. Finally, if s_0 is a not a bunch,

let (s_0, b_0, q_0) be a b-edge with $q_0 \neq r$. If q_0 does not belong to $\mathcal{T}$ we flip the edge (s_0, b_0, q_0) and the red edge going out of s_0. The equivalent automaton satisfies Condition (*). It q_0 belongs to $\mathcal{T}$ and is not a descendant of s_1, we flip the edge (t, b_1, p_1) and the red edge going out of t and we flip the edge (s_0, b_0, q_0) and the red edge going out of s_0. The equivalent automaton satisfies Condition (*). If q_0 belongs to $\mathcal{T}$ and is a descendant of s_1, we return the automaton together with the edge (s_0, b_0, q_0).

The previous sequence of flips is described below in the pseudocode FlipEdges. The value of $previous[r, p]$ is the pair (s_0, s_1), where s_0 is $previous[r]$, *i.e.* the predecessor of r on its red cycle, and s_1 is the child of r which is an ancestor of p. When all states t_i are equal to a same state t, the code is split into to procedures, FlipEdgesChild and FlipEdgesChildren corresponding respectively the case of $\rho = 1$ and the case of $\rho > 1$.

```
FlipEdges( automaton A, root r of a maximal tree)
    We assume that the maximal level of the states is positive and that A satisfies Condition (**)
 1  let s_1, s_2, ..., s_ρ be the children of r which are ancestors of a maximal state
 2  let p_i be a maximal state descendant of s_i (1 ≤ i ≤ ρ)
 3  let s_0 be the predecessor of r in its red cycle
 4  let r, r_2, .. r_k be the ordered list of maximal roots
 5  let T be the tree rooted at r
 6  if there is a predecessor of p_i by a b-edge (t_i, b_i, p_i)
    such that t_i ∉ cluster[r] or level[t_i] ≠ 0 or (t_i ∈ cycle[r] and t_i ∉ [r_k .. r[)
 7      then flip (t_i, b_i, p_i) and the red edge going out of t_i
 8           (s_0, s_i) ← previous[r, p]
 9           return A, (s_0, s_i)
10      else (t_i ∈ [r_k .. r[)
11           if there are two distinct states t_i (say t_1, t_2)
12              then let us assume that t_1 < t_2
13                   flip (t_1, b_1, p_1) and the red edge going out of t
14                   let T' be the new tree rooted at r
15                   if height[T'] > height[T]
16                      then s'_0 ← previous[r]
17                           (s_0, s_1) ← (previous[r], s'_0)
18                           return A, (s_0, s_1)
19                      else (height[T'] ≤ height[T])
20                           flip (t_2, b_2, p_2) and the red edge going out of t'
21                           (s_1, s_2) ← previous[r, p_2]
22                           return A, (s_1, s_2)
23              else (all states t_i are equal to a state t)
24                   if ρ > 1
25                      then return FlipEdgesChildren(A, r, t, (s_i))
26                      else return FlipEdgesChild(A, r, t, s_1)
```

```
FlipEdgesChild( automaton A, root r of a maximal tree, t, s_1)
    We assume here that ρ = 1
 1  let T_0 be the tree rooted at r obtained if we flip (t, b_1, p_1) and the red edge
    going out of t and keep only r and the subtree rooted at the child s_0
 2  if height[T_0] > height[T]
 3     then flip (t, b_1, p_1) and the red edge going out of t
 4          s'_0 ← previous[r]
 5          (s_0, s_1) ← (previous[r], s'_0)
 6          return A, (s_0, s_1)
 7  elseif height[T_0] < height[T]
 8     then return A, (t, b_1, p_1)
 9     else (height[T_0] = height[T])
10          if the sets of outgoing edges of s_0 and s_1 are bunches
11            then return A, (s_0, s_1)
12          elseif the set of outgoing edges of s_0 is a bunch
            and the set of outgoing edges of s_1 is not a bunch
13            then flip (t, b_1, p_1) and the red edge going out of t
                   (we still have height[T_0] = height[T])
14                 return FlipEdges(A, r)
15            else (the set of outgoing edges of s_0 is not a bunch)
16                 let (s_0, b_0, q_0) a b-edge going out of s_0 with q_0 ≠ r
17                 if q_0 ∉ tree[r]
18                   then flip (s_0, b_0, q_0) and the red edge going out of s_0
19                        let r' ← root[q_0]
20                        (s_0, s_1) ← previous[r', p]
21                        return A, (s_0, s_1)
22                 elseif q_0 is not a descendant of s_1
23                   then flip the edge (t, b_1, p_1) and the red edge going out of t
24                        flip the edge (s_0, b_0, q_0) and the red edge going out of s_0
25                        let r' ← root[q_0]
26                        (s_0, s_1) ← previous[r', p]
27                        return A, (s_0, s_1)
28
29                   else (q_0 is a descendant of s_1)
30                        return A, (s_0, b_0, q_0)
```

```
FlipEdgesChildren( automaton A, root r of a maximal tree, t, (s_i))
    We assume here that ρ > 1 and all states t_i are equal to a state t
 1  let T_0 be the tree rooted at r obtained if we flip (t, b_1, p_1) and the red edge
    going out of t and keep only r and the subtree rooted at the child s_0
 2  if height[T_0] > height[T]
 3     then flip (t, b_1, p_1) and the red edge going out of t
 4          s'_0 ← previous[r]
 5          (s_0, s_1) ← (previous[r], s'_0)
 6          return A, (s_0, s_1)
 7  elseif height[T_0] < height[T]
 8     then flip (t, b_1, p_1) and the red edge going out of t
            (now height[T_0] = height[T] and ρ is decremented)
 9          return FlipEdges(A, r)
10     else (height[T_0] = height[T])
11          if the set of outgoing edges of s_0 is a bunch and there is an
            integer i ≥ 1 such that the set of outgoing edges of s_i is a bunch
12             then return A, (s_0, s_i)
13          elseif the set of outgoing edges of s_0 is a bunch
            and the sets of outgoing edges of s_i for i ≥ 1 are not bunches
14             then flip (t, b_1, p_1) and the red edge going out of t
                    (we still have height[T_0] = height[T])
15                  return FlipEdges(A, r)
16             else (the set of outgoing edges of s_0 is not a bunch)
17                  let (s_0, b, q_0) a b-edge going out of s_0 with q_0 ≠ r
18                  if q_0 ∉ tree[r]
19                     then flip (s_0, b_0, q_0) and the red edge going out of s_0
20                  elseif q_0 is not a descendant of s_1
21                     then flip the edge (t, b_1, p_1) and the red edge going out of t
22                          flip (s_0, b_0, q_0) and the red edge going out of s_0
23                     else flip the edge (t, b_2, p_2) and the red edge going out of t
24                          flip (s_0, b_0, q_0) and the red edge going out of s_0
25                  let r' ← root[q_0]
26                  (s_0, s_1) ← previous[r', p]
27                  return A, (s_0, s_1)
```

The function FlipEdges$(\mathcal{A}, r)$ returns either an equivalent automaton together with a stable pair of states, or an equivalent automaton or an equivalent automaton together with a b-edge.

After running FlipEdges$(\mathcal{A}, r)$ on all maximal roots r, we get either an automaton with a stable pair, or an automaton where each cluster fulfills the following conditions

1. the root of each maximal tree $\mathcal{T}_i$ has a unique child;
2. there is a b-edge (x_i, b_i, y_i) such that r_i is no more a maximal root after flipping this edge and the red edge going out of x_i.

If the latter case, we flip the b-edge (x_i, b_i, y_i) and the red one starting at x_i for all maximal roots r_i but one. We get an equivalent automaton which

satisfies Condition (*) as is shown in Figure 8. The pseudocode for this final treatment is given in procedure FindStablePair.

```
FindStablePair( automaton A)
 1  if the maximal level of states of A is 0
 2     then return FlipEdgesLevelZero(A)
 3  else
 4  A, R ← ScanTrees(A)
 5  if R is a (stable) pair of states (s, t)
 6     then return A, (s, t)
 7  else (R is empty)
 8  for each maximal root r
 9        do A, R ← FlipEdges(A, r)
10           if R is a (stable) pair of states (s, t)
11              then return A, (s, t)
12              else  (R is a b-edge)
13                    e[r] ← R
    At this stage FlipEdges has not returned a stable pair
14  r_0 ← a maximal root
15  for each maximal root r ≠ r_0
16        do flip the edge e[r] = (x, b, y) and the red edge going out of x
17  p ← a maximal node in the tree rooted at r_0
18  (s, t) ← previous[r_0, p]
19  return A, (s, t)
```

5 The complexity issue

We establish the time and space complexity of our algorithm in the following proposition. We denote by k the size of the alphabet A and by n the number of states of $\mathcal{A}$. Since $\mathcal{A}$ is complete deterministic, it has kn edges.

Proposition 4. *The worst-case complexity of* FindColoring *applied to an n-state aperiodic automaton is $O(kn^2)$ in time and $O(kn)$ in space.*

Proof. The edges of the automaton can be stored in tables indexed by the states and labels. For each state, we compute in linear time two lists of predecessor states by red and b-edges respectively.

For each state p of the automaton we memorize some data like its cluster, its cycle, the root of the tree containing p, the level of p. If $level[p] > 0$, the data contains also the child of the root of the tree containing p which is an ancestor of p. If $level[p] = 0$, the data contains also the height of the tree rooted at p, and the number of p on its cycle. All these data can be computed in a linear time with a depth-first search that starts at the states that have no predecessors by a red edge. We denote by ℓ the maximal level of the states.

We first prove that the complexity of FindColoring is at most n times the complexity of FindStablePair, when we do not take into account

the cost induced by Line 5 of FindColoring for computing the quotient automata. Let us show that FindStablePair can be performed in linear time.

Checking at Line 6 of FlipEdges whether $t_i \in [r_k, \ldots, r[$ is done in constant time for each t_i using the numbers of states at level 0.

In the worst case, Line 15 of FlipEdges takes a time at most equal to the number of states contained in the trees whose roots belong to $]t, \ldots, r[$. Indeed, in Line 15, we have to compute the maximal value of $height[x] + num[r] - num[x]$ for all $x \in]t, \ldots, r[$ on the red cycle, where $height[x]$ denotes the height of the tree rooted at x. Hence the overall time spent for computing FlipEdges$(\mathcal{A}, r)$ for all maximal roots r is $O(kn)$.

Similarly, in the worst case, FlipEdgesChild$(\mathcal{A}, r, t, s_1)$ takes a time at most equal to the number of states contained in trees whose roots belong to $]r_k, \ldots, r]$, where r_k is the maximal root before r in the red cycle of r. In order to perform the test of Line 2 of FlipEdgesChild, we have to compute the maximal value of $height[x] + num[r] - num[x]$ for all $x \in]t, \ldots, r[$ on the red cycle. In the case where this value is equal to ℓ and the set of outgoing edges of s_0 is a bunch, we flip two outgoing edges of t. We then only have to update the data of the states contained in the trees whose roots are $x \in]t, \ldots, r]$ before the second (and last) call to FlipEdges$(\mathcal{A}, r)$. It is not needed to update $num[p]$ for all states p in the new cycle. It is enough to mark the states $x \in [t, \ldots, r[$ on the new red cycle (*i.e.* the states t and all states on the path from p to r in the tree of r before the flip). The complexity for all calls to FlipEdgesChild is thus at most the number of edges, *i.e.* at most kn. The same result holds for FlipEdgesChildren.

Moreover, Lines 14 to 16 of FindStablePair can be performed in time at most the number of maximal roots, *i.e.* at most n. Hence FindStablePair has a $O(kn)$-time complexity.

Suppose that the class of a state for the current stable pair congruence is stored in an array giving the leader of the class. The Union of two classes can be performed in time at most $O(n)$ while the class of a state is found in constant time with the Find operation.

Since lifting a coloring from an automaton up to another one uses only Find operations, it can be done in time $O(kn)$.

Finally, we show that the computation of all quotient automata in Line 5 of FindColoring during the while loop has an overall quadratic-time complexity. Since the number of states decreases when two classes are merged, the number of merges realized is at most n. Thus the total time for the computation of all quotient automata is at most kn^2. Note that this complexity may be improved with a more clever implementation of Union-Find operations [3].

Hence FindColoring has a $O(kn^2)$-time complexity. The space complexity is $O(kn)$. Indeed, only linear additional space is needed to perform the computation of the current congruence. □

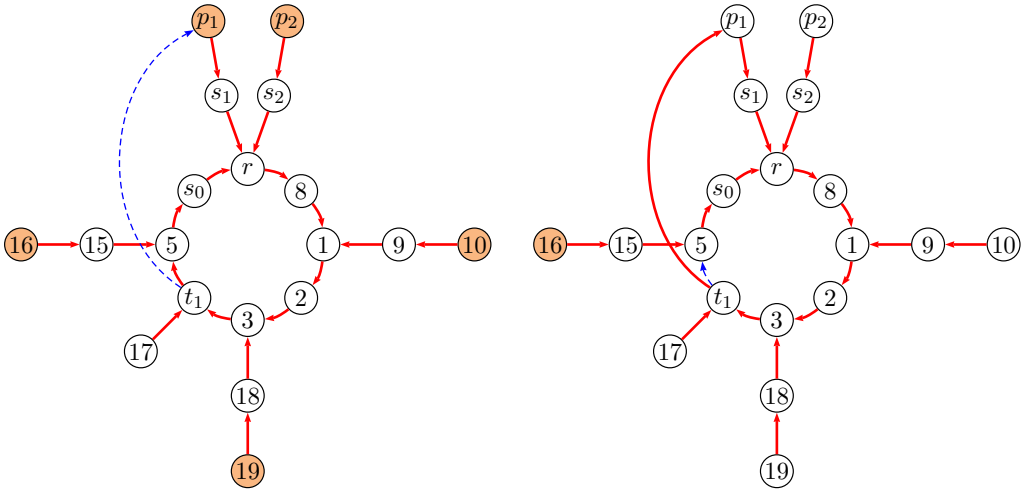

Figure 2: FlipEdges Lines 6-9. The picture on the left illustrates the case corresponding to line 6 of FlipEdges. After the execution of Lines 6-9 of FlipEdges, *i.e.* after flipping the edge (t_1, b_1, p_1) and the red edge going out of t_1, we get the automaton on the right. It satisfies the Condition (*), *i.e.* it has a unique maximal tree (here rooted at r). Maximal states are colored and the (dashed) b-edges of the automaton are not all represented.

6 The case of periodic graphs

Let the period of an automaton be the gcd of the lengths of the cycles in its graph. If the automaton $\mathcal{A}$ is an n-state complete deterministic irreducible automaton which is not aperiodic, it is not equivalent to a synchronized automaton. Nevertheless, the previous algorithm can be modified as follows to find an equivalent automaton with the minimal possible rank in a quadratic time.

PeriodicFindColoring(aperiodic automaton $\mathcal{A}$)

1 $\mathcal{A}_0 \leftarrow \mathcal{A}$
2 **repeat**
3 $\mathcal{A}, (s,t) \leftarrow$ FindStablePair($\mathcal{A}$)
4 **if** there is a stable pair (s,t)
5 **then** lift the coloring of $\mathcal{A}$ up to $\mathcal{A}_0$
6 **else return** $\mathcal{A}_0$

At Line 3, it may happen that FindStablePair does not return a stable pair of states. In this case, the condition at Line 4 is false. At the end of PeriodicFindColoring, we get an automaton $\mathcal{A}$ that has no stable pair

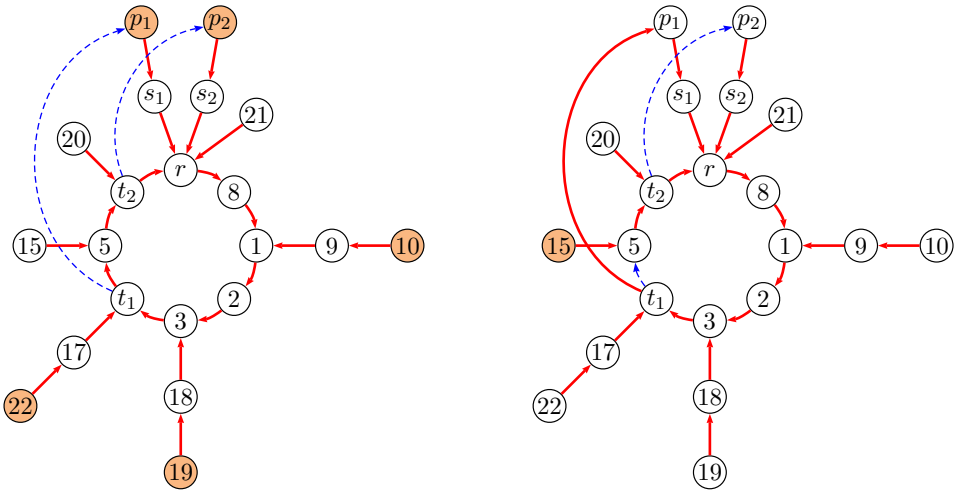

Figure 3: FlipEdges Lines 12-18. The picture on the left illustrates the case corresponding to lines 15 of FlipEdges. The height of the tree $\mathcal{T}'$ obtained after flipping the edge (t_1, b_1, p_1) and the red edge going out of t_1, is greater than the maximal level. We get a unique maximal tree rooted at r in the same cluster. The picture on the right illustrates the result.

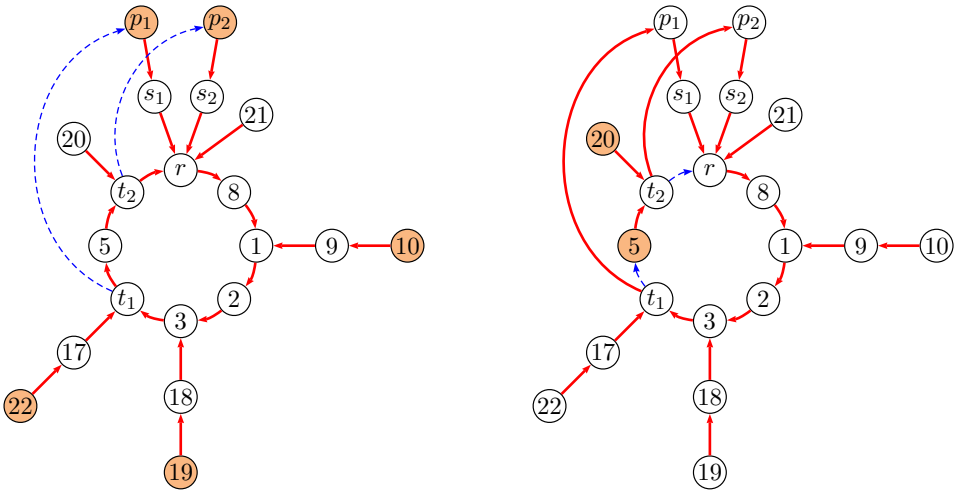

Figure 4: FlipEdges Lines 19-22. The picture on the left illustrates the case corresponding to the hypothesis of Line 19 of FlipEdges. The height of the tree $\mathcal{T}'$ obtained after flipping the edge (t_1, b_1, p_1) and the red edge going out of t_1, is not greater than the maximal level. In this case, we also flip the edge (t_2, b_2, p_2) and the red edge going out of t_2. We get a unique maximal tree rooted at r in the same cluster. The picture on the right gives the resulting cluster.

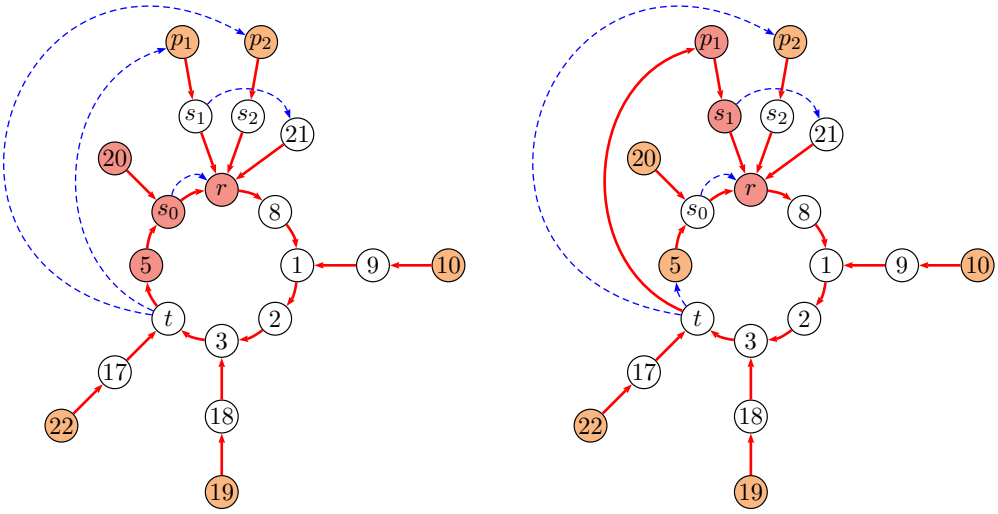

Figure 5: FlipEdgesChildren Lines 13-14. The picture on the left illustrates the case corresponding to the hypothesis of Line 13 of FlipEdgesChildren. Let $\mathcal{T}_0$ be the tree rooted at r obtained if we flip (t, b_1, p_1) and the red edge going out of t and keep only r and the subtree rooted at the child s_0. The states of the tree $\mathcal{T}_0$ rooted at r are represented in salmon in the left part of the figure. The state s_0 is a bunch. After flipping the edge (t, b_1, p_1) and the red edge going out of t, we get the automaton pictured in the right part of the figure. The The tree $\mathcal{T}_0'$ (new tree $\mathcal{T}_0$) is the tree rooted at r obtained if we flip (t, b_2, p_2) and the red edge going out of t and keep only r and the subtree rooted at the child s_1. Its states are colored in salmon. Its height is the same as before.

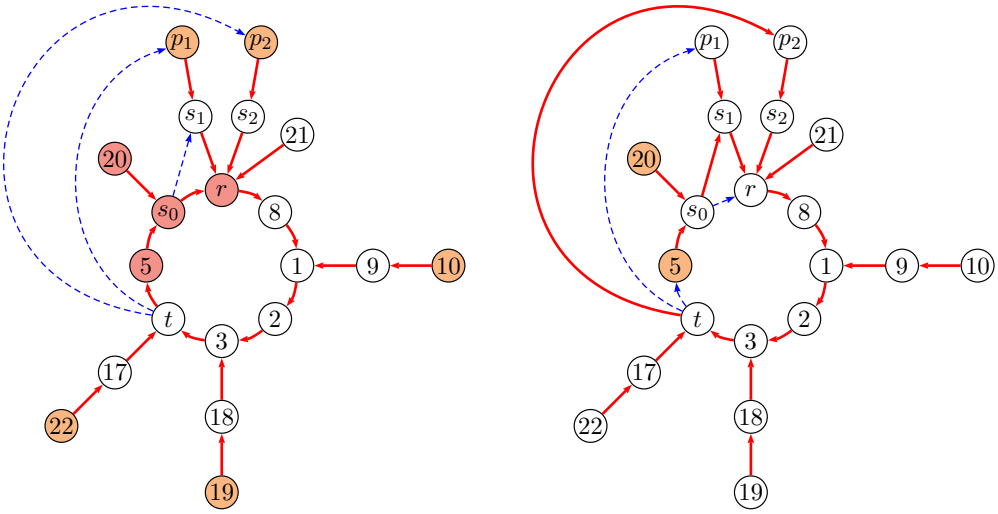

Figure 6: FlipEdgesChildren Lines 23-27. The picture on the left illustrates the case corresponding to the hypothesis of Line 23 of FlipEdgesChildren. Let $\mathcal{T}_0$ be the tree rooted at r obtained if we flip (t, b_1, p_1) and the red edge going out of t and keep only r and the subtree rooted at the child s_0. The states of the tree $\mathcal{T}_0$ rooted at r are represented in salmon in the left part of the figure. The state s_0 is not a bunch: it has a b-edge (s_0, b_0, q_0) with $q_0 = s_1$. After flipping the edge (t, b_2, p_2) and the red edge going out of t, and flipping (s_0, b_0, q_0) and the red edge going out of s_0, we get a unique maximal tree rooted at r in the same cluster (see the right part of the figure).

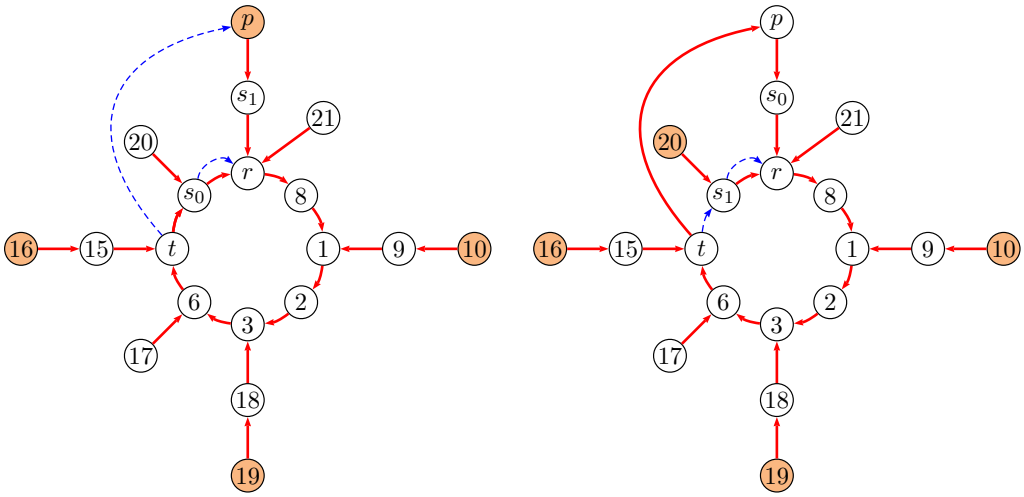

Figure 7: FLIPEDGESCHILD Lines 12-14. The picture on the left illustrates the case corresponding to Line 12 of FLIPEDGESCHILD. After the execution of lines 12-14 of FLIPEDGESCHILD, *i.e.* after flipping the edge (t, b_1, p_1) and the red edge going out of t, we get the automaton on the right. The root r has a new single child s_1 ancestor of a maximal state, whose set of outgoing edges is a bunch. The new tree rooted at r has here the same level $\ell = 2$ as before and FLIPEDGES$(\mathcal{A}, r)$ is called a second and last time.

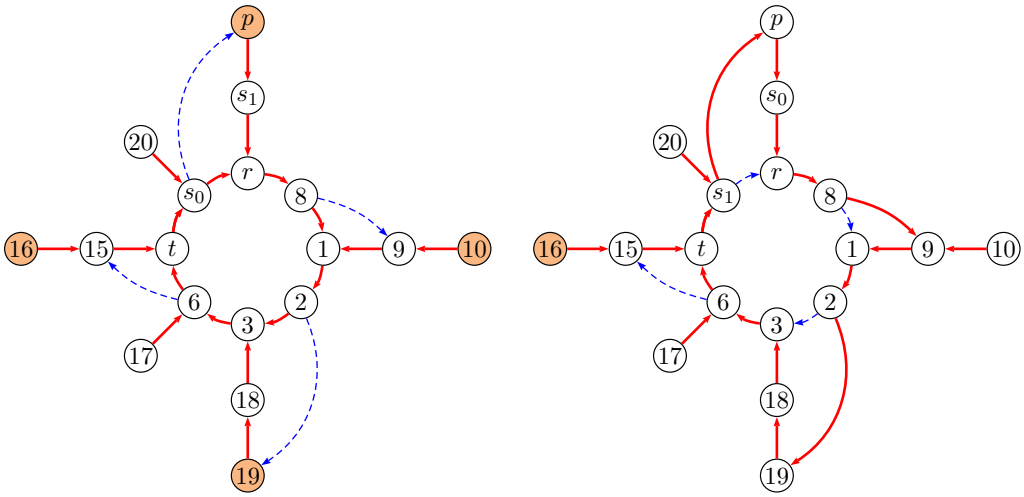

Figure 8: FINDSTABLEPAIR Lines 14-18. The picture on the left illustrates the case corresponding to the conditions at Line 14 of FINDSTABLEPAIR. After the execution of FINDSTABLEPAIR, two edges of the predecessors in the red cycle of all but one maximal roots are flipped. The new cluster is pictured on the right part of the figure. It has a unique maximal tree. By Lemma 3 the pair $(6, 15)$ is stable.

(it is made of a cycle where the set of outgoing edges of any state is a bunch). Lifting up this coloring of to the initial automaton $\mathcal{A}_0$ leads to a coloring of the initial automaton whose minimal rank is equal to its period.

This result can be stated as the following theorem, which extends the road coloring theorem to the case of periodic graphs.

Theorem 5. *Any irreducible automaton $\mathcal{A}$ is equivalent to a an automaton whose minimal rank is the period of $\mathcal{A}$.*

Proof. We prove the result by recurrence on the number of states of $\mathcal{A}$.

Let us assume that $\mathcal{A}$ is equivalent to an automaton $\mathcal{A}'$ which has a stable pair (s, t). Let $\mathcal{B}'$ be the quotient of $\mathcal{A}'$ by the congruence generated by (s, t). Let d be the period of $\mathcal{A}'$ (equal to the period of $\mathcal{A}$) and d' the period of $\mathcal{B}'$. It is clear that d' divides d. Let w be the label of a cycle around a state s of $\mathcal{B}'$. Let $\{p_1, \ldots, p_k\}$ be the congruence class of the states of $\mathcal{A}'$ that corresponds to s. Hence, after renumbering the states of s, there is an integer i and a cycle in $\mathcal{A}'$ of length $|w| \times i$:

$$p_1 \xrightarrow{w} p_2 \xrightarrow{w} \ldots \xrightarrow{w} p_{i+1} = p_1.$$

Let us assume that $i > 1$. Since (p_1, p_2) is a stable pair, there is a word u such that $p_1 \cdot u = p_2 \cdot u$. Since $\mathcal{A}'$ is irreducible, there is a word v labeling a path from $p_1 \cdot u$ to p_1. Thus there are cycles around p_1 with labels wuv and uv. This implies that d' divides $|w|$ and thus that $d = d'$. This still holds if $i = 1$.

Suppose that $\mathcal{B}'$ has a coloring $\mathcal{B}''$ which has rank d. We lift this coloring up to an automaton $\mathcal{A}''$ equivalent to $\mathcal{A}''$. Let us show that $\mathcal{A}''$ has rank d. Let I be a minimal image of $\mathcal{A}''$ and J be the set of classes of the states of I in $\mathcal{B}''$. The set J is a minimal image of $\mathcal{B}''$. Two states of I cannot belong to the same class since I would not be minimal otherwise. As a consequence I has the same cardinal as J.

Let us now assume that $\mathcal{A}$ has no equivalent automaton with a stable pair. In this case, we know that $\mathcal{A}$ is made of one red cycle where the set of edges going out of any state is a bunch. Since $\mathcal{A}$ has rank d, the length of the cycles is d. Hence $\mathcal{A}$ has minimal rank d. □

Since the modification of FindColoring into PeriodicFindColoring does not change its complexity, we obtain the following corollary.

Corollary 6. *Procedure* PeriodicFindColoring *finds a coloring of minimal rank for an n-state irreducible automaton in $O(kn^2)$.*

Acknowledgments The authors would like to thank Florian Sikora and Avraham Trahtman for pointing us a missing configuration in the algorithm.

References

[1] R. L. Adler, D. Coppersmith, and M. Hassner. Algorithms for sliding block codes. *IEEE Trans. Inform. Theory*, IT-29:5–22, 1983.

[2] R. L. Adler, L. W. Goodwyn, and B. Weiss. Equivalence of topological Markov shifts. *Israel J. Math.*, 27(1):48–63, 1977.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[4] K. Culik, II, J. Karhumäki, and J. Kari. A note on synchronized automata and road coloring problem. In *Developments in language theory (Vienna, 2001)*, volume 2295 of *Lecture Notes in Comput. Sci.*, pages 175–185. Springer, Berlin, 2002.

[5] B. Delyon and O. Maler. On the effects of noise and speed on computations. *Theoret. Comput. Sci.*, 129(2):279–291, 1994.

[6] S. Eilenberg. *Automata, Languages, and Machines. Vol. B.* Academic Press, New York, 1976.

[7] J. Friedman. On the road coloring problem. *Proc. Amer. Math. Soc.*, 110(4):1133–1135, 1990.

[8] J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoret. Comput. Sci.*, 295(1-3):223–232, 2003.

[9] D. A. Lind and B. H. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge, 1995.

[10] D. Perrin and M.-P. Schützenberger. Synchronizing prefix codes and automata and the road coloring problem. In *Symbolic dynamics and its applications*, volume 135 of *Contemp. Math.*, pages 295–318. Amer. Math. Soc., 1992.

[11] A. N. Trahtman. The road coloring problem. To appear in *Israel J. Math.*

[12] A. N. Trahtman. A subquadratic algorithm for road coloring. arXiv:0801.2838v1 [cs.DM].