A linear algorithm for coloring vertices of a graph or finding a Meyniel obstruction

Kathie Cameron, Jack Edmonds, Benjamin Lévêque, Frédéric Maffray

A linear algorithm for coloring vertices of a graph or finding a Meyniel obstruction [1]

Kathie Cameron

Wilfrid Laurier University, Waterloo, Ontario, Canada N2L 3C5

Jack Edmonds

EP Institute, Kitchener, Ontario, Canada, N2M 2M6

Benjamin Lévêque

Laboratoire Leibniz-IMAG, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France

Frédéric Maffray [2]

Laboratoire Leibniz-IMAG, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France

Abstract

A Meyniel obstruction is an odd cycle with at least five vertices and at most one chord. A graph is Meyniel if and only if it has no Meyniel obstruction. Here we give a linear-time algorithm that, for any graph, finds either a clique and coloring of the same size or a Meyniel obstruction. We also give a polytime algorithm that, for any graph, finds either an easily recognizable strong stable set or a Meyniel obstruction.

Key words: Perfect graphs, Meyniel graphs, Coloring, Robust algorithm, Strong stable set, Existentially polytime theorem

Email addresses: kcameron@wlu.ca (Kathie Cameron), jackedmonds@rogers.com (Jack Edmonds), benjamin.leveque@imag.fr (Benjamin Lévêque), frederic.maffray@imag.fr (Frédéric Maffray).

[1] This work was partially supported by the Algorithmic Discrete Optimization Network (ADONET), the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Research Grants Program of Wilfrid Laurier University

[2] C.N.R.S.

1 Introduction

This work is motivated by the Perfect Graph Robust Algorithm Problem proposed by the first and second authors. In [2], they advocated seeking a polytime algorithm which, for any graph G, finds either a clique and a coloring of the same size or an easily recognizable combinatorial obstruction to G being perfect. A simple obstruction to perfectness is the existence of an odd hole or odd antihole.

Here we present a linear-time algorithm that gives a solution to a similar problem on the class of Meyniel graphs. A graph is *Meyniel* [13] if every odd cycle of length at least five has at least two chords. We call an odd cycle of length at least five with at most one chord a *Meyniel obstruction*. Thus a graph is Meyniel if and only if it does not contain a Meyniel obstruction. Meyniel [13] and Markosyan and Karapetyan [12] proved that Meyniel graphs are perfect. This theorem can be stated in the following way:

> *For any graph G, either G contains a Meyniel obstruction, or G contains a clique and coloring of the same size (or both).*

We give a linear-time algorithm which finds, in any graph, an instance of what the Meyniel-Markosyan-Karapetyan theorem says exists. By linear time, we mean linear in the number of vertices plus the number of edges of the input graph. This is an improvement in the complexity of the algorithm of the first and second authors [3,4], which finds, in any graph, a clique and coloring of the same size, or a Meyniel obstrution. It is also an enhancement of the linear-time algorithm of the third and fourth authors [10], which finds, for any graph, either a clique and coloring of the same size, or declares the graph not to be Meyniel, but in the latter case, does not find a Meyniel obstruction.

A *stable set* in a graph G is a set of vertices, no two of which are joined by an edge of G. A *strong stable set* in G is a stable set that contains a vertex of every maximal clique of G. By maximal, we mean maximal with respect to inclusion. Note that if one can find a strong stable set in every induced subgraph of a graph G, one can easily find an optimal coloring of G : if S_1 is a strong stable set of G, S_2 is a strong stable set of $G \backslash S_1$, ..., S_l is a strong stable set of $G \backslash (S_1 \cup \ldots \cup S_{l-1})$, and S_l is the last non-empty such set, then $S_1, \ldots, S_l$ is an optimal coloring of G.

Ravindra [14] presented the theorem that

> *For any graph G, either G contains a Meyniel obstruction, or G contains a strong stable set (or both).*

Ravindra's proof is an informal description of an algorithm which finds, in any

graph, an instance of what the theorem says exists.

Hoàng [9] strengthened this to the following:

> *For any graph G and vertex v of G, either G contains a Meyniel obstruction, or G contains a strong stable set containing v (or both).*

A disadvantage of the Ravindra-Hoàng theorem is that it is not an existentially polytime theorem in the sense of the first two authors [2]. A theorem is called *existentially polytime (EP)* if it is a disjunction of NP predicates which is always true. The predicate "G contains a strong stable set" may not be an NP-predicate because the definition of strong stable set is not a polytime certificate.

The Ravindra-Hoàng theorem is strengthened by the first two authors in [3,4] to:

> *For any graph G and vertex v of G, either G contains a Meyniel obstruction, or G contains a nice stable set containing v (or both),*

where nice stable sets are a particular type of strong stable set which have the following polytime-certifiable meaning. A *nice stable set* in a graph G is a maximal stable set linearly ordered so that there is no induced P_4 between any vertex x of S and the vertex which arises from the contraction in G of all the vertices of S that precede x. (As usual, P_4 is a path on four vertices.) The proof of the theorem in [3,4] is a polytime algorithm which for any graph and any vertex in that graph, finds an instance of what the theorem says exists. Combining ideas of [3,4,10,11], we give in Section 4 an $\mathcal{O}(n^3)$ algorithm that finds, for any graph G and vertex v of G, either a Meyniel obstruction in G or a nice stable set of G containing v, where n is the number of vertices of the input graph.

2 The coloring algorithm

We recall the algorithm MCColor of [10] which is a linear-time algorithm for optimally coloring the vertices of a Meyniel graph, thereby improving the complexity of previous coloring algorithms due to Roussel and Rusu $\mathcal{O}(n^2)$ [16], Hertz $\mathcal{O}(nm)$ [8] and Hoàng $\mathcal{O}(n^8)$ [9].

MCColor is a rather simple version of the greedy coloring algorithm. Colors are viewed as integers $1, 2, \ldots$ At each step, the algorithm considers, for every uncolored vertex x, the number of colors that appear in the neighbourhood of x, selects an uncolored vertex for which this number is maximum, assigns to

this vertex the smallest color not present in its neighbourhood, and iterates this procedure until every vertex is colored. More formally:

ALGORITHM MCCOLOR
Input: A graph G with n vertices.
Output: A coloring of the vertices of G.
Initialization: For every vertex x of G do $label(x) := \emptyset$;
General step: For $i = 1, \ldots, n$ do:
1. Choose an uncolored vertex x that maximizes $|label(x)|$;
2. Color x with the smallest color in $\{1, 2, \ldots, n\} \setminus label(x)$;
3. For every uncolored neighbor y of x, add $color(x)$ to $label(y)$.

Given a coloring of a graph, there is a greedy algorithm that chooses one vertex of each color to try to find a clique of the same size. Algorithm CLIQUE below is a linear-time implementation. In [10] it was shown that, starting with a Meyniel graph G and a coloring of its vertices obtained by Algorithm MCCOLOR, Algorithm CLIQUE produces a clique of maximum size.

ALGORITHM CLIQUE
Input: A graph G and a coloring of its vertices using l colors.
Output: A set Q that consists of l vertices of G.
Initialization: Set $Q := \emptyset$, $c := l$;
General step: While $c \neq 0$ do:
Select a vertex x of color c that maximizes $N(x) \cap Q$, do $Q := Q \cup \{x\}$, and do $c := c - 1$.

3 Finding an obstruction

Let G be a general (not necessarily Meyniel) graph on which Algorithm MCCOLOR is applied. Let l be the total number of colors used by the algorithm. For each color $c \in \{1, \ldots, l\}$, let k_c be the number of vertices colored c. Then every vertex of G can be renamed x_c^i, where $c \in \{1, \ldots, l\}$ is the color assigned to the vertex by the algorithm and $i \in \{1, \ldots, k_c\}$ is the integer such that x_c^i is the i-th vertex colored c. Thus $V(G) = \{x_1^1, x_1^2, \ldots, x_1^{k_1}, x_2^1, \ldots, x_2^{k_2}, \ldots, x_l^1, \ldots, x_l^{k_l}\}$.

Then we apply Algorithm CLIQUE. At each step of Algorithm CLIQUE, we check if the selected vertex x of color c has $q(x) = l - c$. If this equality is satisfied at each step then Q is a clique of cardinality l, and so we have a clique and a coloring of the same size, which proves the optimality of both. If not, then stop Algorithm CLIQUE the first time the equality is not satisfied, and record the current color c and the current clique Q. We know that no vertex colored c is adjacent to all of Q. Our task is now to find a Meyniel obstruction

in G.

Let $G^* = G \backslash \{x_1^1, \ldots, x_1^{k_1}, \ldots, x_{c-1}^1, \ldots, x_{c-1}^{k_{c-1}}\}$, and for $1 \leq i \leq k_c$, let G_i^* be the graph obtained from G^* by removing $x_c^1, \ldots, x_c^i$ and adding a new vertex w_c^i with edges to $N(x_c^1) \cup \cdots \cup N(x_c^i)$ (in other words, vertices $x_c^1, \ldots, x_c^i$ are contracted to become w_c^i). A path is called *odd (even)* if it has an odd (even) number of edges. The algorithm works roughly as follows. Starting from the fact that no vertex colored c is adjacent to all of Q, it finds an odd path R in G_i^* with certain properties (for a value of i which we specify later), and a vertex z adjacent to both ends of R. If it happens that R has only vertices of G^*, and it has at most one chord, and z is not adjacent to any interior vertex of R, then $R \cup \{z\}$ induces an obstruction. However, these conditions may fail to hold, so we have two procedures to handle the difficulties. The first part of the algorithm consists in using the procedure *Loop*, called iteratively, which eventually finds a path R and a vertex z that is adjacent to both ends of R, where z and the vertices of R are vertices of G^* (and thus of G). Vertex z may be adjacent to interior vertices of R. Path R has at most one chord, and if the chord exists, it is a short chord. In the second part of the algorithm, *Loop* calls the procedure *FindObstruction* on R and z to find a Meyniel obstruction. Variants $1, 2, 3, 4$ of *FindObstruction* correspond to the possible positions of the chord of R. Now we describe these procedures formally.

Initialization: For each vertex v of color $> c$ of G^*, let $n(v)$ be the smallest integer h such that x_c^h is adjacent to v. The integer $n(v)$ exists because v has received a color strictly greater than c. (Clearly the function n can be computed in linear time.) Let $i = \max\{n(v) \mid v \in Q\}$. We have $i > 1$ because x_c^1 is not adjacent to all of Q. Let v_1 be a vertex of Q not adjacent to x_c^i. Vertex v_1 exists because x_c^i is not adjacent to all of Q. Note that v_1 is adjacent to w_c^{i-1} in G_{i-1}^*. Let v_2 be a vertex of Q such that $n(v_2) = i$. Vertex v_2 is adjacent to x_c^i but not to w_c^{i-1} in G_{i-1}^*. Then $R = w_c^{i-1}$-v_1-v_2-x_c^i is an odd chordless path in G_{i-1}^*. Let $P = v_1$-v_2-x_c^i. Call $Loop(P, i, \emptyset)$.

Loop(P, i, e)

[where $i > 1$, $P = v_1$-$\cdots$-v_p with $v_p = x_c^i$ and w_c^{i-1}-P is an odd path in G_{i-1}^* with at most one chord, and such a chord (if any) is $v_{t-1}v_{t+1}$ with $1 < t < p-1$, e is the set that consists of the chord of P, if any]

Claim. There exists a vertex z, colored before x_c^i with a color $> c$, that is adjacent to x_c^i and satifies the following property. If v_1v_3 is the chord of P, z is not adjacent to at least one of v_1 and v_3. If v_1v_3 is not a chord of P, z is not adjacent to at least one of v_1 and v_2.

Proof. Let $R = w_c^{i-1}$-P. We know that every vertex of G_{i-1}^* will have a color

from $\{c, c+1, \ldots, l\}$ when the algorithm terminates. So, if $c \geq 2$, every vertex v of G^*_{i-1} (in particular every vertex of R) satisfies $label(v) \supseteq \{1, 2, \ldots, c-1\}$.

Let us consider the situation when the algorithm selects x^i_c to be colored. Let U be the set of vertices of G that are already colored at that moment. For any $X \subseteq V$, let $color(X)$ be the set of colors of the vertices of $X \cap U$. So for every vertex $v \in V \setminus U$ we have $label(v) = color(N(v))$. Put $T = \{v \in N(x^i_c) \cap U \mid color(v) \geq c+1\}$. We have $|label(x^i_c)| = (c-1) + |color(T)|$. Every vertex of T is adjacent to at least one vertex colored c in G and thus is adjacent to w^{i-1}_c in G^*_{i-1}. Specify one vertex v_r of R as follows: put $r = 3$ if v_1v_3 is a chord of R; else put $r = 2$. Note that v_r is not adjacent to v_0 and $v_r \neq x^i_c$ according to the possible position of the chord of R.

Suppose the claim is false: every vertex of T is adjacent to v_1 and v_r.

Suppose v_1 is not colored yet (i.e., $v_1 \notin U$). Since $label(v_1) \supseteq \{1, 2, \ldots, c-1\}$ if $c \geq 2$, and $N(v_1) \supseteq T \cup \{v_0\}$ and v_0 has color c, we have $|label(v_1)| = |color(N(v_1))| \geq c + |color(T)| > |label(x^i_c)|$, which contradicts the fact that the algorithm is about to color x^i_c. So v_1 is already colored; moreover $color(v_1) \notin \{1, \ldots, c\} \cup color(T)$.

Suppose v_r is not colored yet. Since $label(v_r) \supseteq \{1, 2, \ldots, c-1\}$ if $c \geq 2$ and v_r is adjacent to all of $T \cup \{v_1\}$, we have $|label(v_r)| = |color(N(v_r))| \geq (c-1) + |color(T \cup \{v_1\})| = c + |color(T)| > |label(x^i_c)|$, again a contradiction. So v_r is already colored. However, v_r is not adjacent to w^{i-1}_c, so c was the smallest color available for v_r when it was colored; but this contradicts the definition of w^{i-1}_c and x^i_c. This completes the proof of the claim. $\square$

Let z be a vertex that satisfies the claim above. Vertex z is adjacent to w^{i-1}_c in G^*_{i-1} because z was colored before x^i_c and received a color strictly greater than c. (It takes time $deg(x^i_c)$ to find such a vertex z.)

Let $j = max(n(z), n(v_1))$. We have $j < i$ because z and v_1 are adjacent to w^{i-1}_c.

Suppose $n(z) = n(v_1)$. Then x^j_c is adjacent to v_1 and z. If v_1v_3 is the chord of P, call $FindObstruction2(z, x^j_c\text{-}P)$. If v_1v_3 is not a chord of P and z is not adjacent to v_1, call $FindObstruction3(z, x^j_c\text{-}P, e)$. If v_1v_3 is not a chord of P and z is adjacent to v_1 (and thus is not adjacent to v_2), call $FindObstruction4(z, x^j_c\text{-}P, e)$.

Now suppose $n(z) \neq n(v_1)$. Then $j > 1$, one of w^{j-1}_c, x^j_c is adjacent to v_1 and is not adjacent to z, and the other is not adjacent to v_1 and is adjacent to z. Let k be the smallest integer such that z is adjacent to v_k. Such a k exists because z is adjacent to v_p. (It takes time $deg(z)$ to compute k.)

Suppose that k is odd. If $n(z) = j$, then let $P' = v_1\text{-}\cdots\text{-}v_k\text{-}z\text{-}x_c^j$; else let $P' = z\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$. If e is a chord of P', then call $Loop(P', j, e)$; else call $Loop(P', j, \emptyset)$.

Now suppose that k is even. Then $k < p$ since p is odd. We consider the following cases:

Case 1 : P has a chord $v_{t-1}v_{t+1}$ with $t < k$. If $n(z) = j$, then let $P' = v_1\text{-}\cdots\text{-}v_{t-1}\text{-}v_{t+1}\text{-}\cdots\text{-}v_k\text{-}z\text{-}x_c^j$; else let $P' = z\text{-}v_k\text{-}\cdots\text{-}v_{t+1}\text{-}v_{t-1}\text{-}\cdots\text{-}v_1\text{-}x_c^j$. Call $Loop(P', j, \emptyset)$.

Case 2 : P has a chord $v_{k-1}v_{k+1}$. When z is adjacent to both v_{k+1} and v_{k+2}, if $n(z) = j$, then let $P' = v_1\text{-}\cdots\text{-}v_{k-1}\text{-}v_{k+1}\text{-}v_{k+2}\text{-}z\text{-}x_c^j$; else let $P' = z\text{-}v_{k+2}\text{-}v_{k+1}\text{-}v_{k-1}\text{-}\cdots\text{-}v_1\text{-}x_c^j$; call $Loop(P', j, zv_{k+1})$. When z is not adjacent to v_{k+1}, call $FindObstruction3(z, v_k\text{-}\cdots\text{-}v_p, \emptyset)$. When z is adjacent to v_{k+1} and is not adjacent to v_{k+2}, call $FindObstruction4(z, v_k\text{-}\cdots\text{-}v_p, \emptyset)$.

Case 3 : P has a chord v_kv_{k+2}. When z is adjacent to v_{k+1}, if $n(z) = j$, then let $P' = v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+1}\text{-}z\text{-}x_c^j$; else let $P' = z\text{-}v_{k+1}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$; call $Loop(P', j, zv_k)$. When z is not adjacent to v_{k+1} and is adjacent to v_{k+2}, if $n(z) = j$, then let $P' = v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+2}\text{-}z\text{-}x_c^j$; else let $P' = z\text{-}v_{k+2}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$; call $Loop(P', j, zv_k)$. When z is not adjacent to v_{k+1} or to v_{k+2}, call $FindObstruction1(z, v_k\text{-}\cdots\text{-}v_p)$.

Case 4 : P has no chord $v_{t-1}v_{t+1}$ with $t \leq k+1$. When z is adjacent to v_{k+1}, if $n(z) = j$, then let $P' = v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+1}\text{-}z\text{-}x_c^j$; else let $P' = z\text{-}v_{k+1}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$; call $Loop(P', j, zv_k)$. When z is not adjacent to v_{k+1}, call $FindObstruction3(z, v_k\text{-}\cdots\text{-}v_p, e)$.

$FindObstruction1(z, P)$

[where $P = v_0\text{-}\cdots\text{-}v_p$ is an odd path with only one chord v_0v_2, z is adjacent to both v_0 and v_p and is not adjacent to either v_1 or v_2]

Let r be the smallest integer > 0 such that z is adjacent to v_r. This integer r exists because z is adjacent to v_p. We have $r \geq 3$ because z is not adjacent to v_1 or v_2. When r is odd, return $z, v_0, \ldots, v_r$, which induce an odd cycle with only one chord v_0v_2. When r is even, we have $r < p$ and $r \geq 4$; return $z, v_0, v_2, \ldots, v_r$, which induce an odd hole.

$FindObstruction2(z, P)$

[where $P = v_0\text{-}\cdots\text{-}v_p$ is an odd path $p > 3$ with only one chord v_1v_3, z is adjacent to both v_0 and v_p and is not adjacent to one of v_1, v_3]

Case 1 : z is not adjacent to either v_1 or v_2. Let r be the smallest integer > 0 such that z is adjacent to v_r; we have $r \geq 3$. When r is odd, return $z, v_0, \ldots, v_r$, which induce an odd cycle with only one chord v_1v_3. When r is even, we have $r \geq 4$; return $z, v_0, v_1, v_3, \ldots, v_r$, which induce an odd hole.

Case 2 : z is not adjacent to v_1, is adjacent to v_2, is not adjacent to v_3. Call $FindObstruction3(z, v_2\text{-}\cdots\text{-}v_p, \emptyset)$.

Case 3 : z is not adjacent to v_1, is adjacent to v_2, is adjacent to v_3, is not adjacent to v_4. Call $FindObstruction4(z, v_2\text{-}\cdots\text{-}v_p, \emptyset)$,

Case 4 : z is not adjacent to v_1, is adjacent to v_2, is adjacent to v_3, is adjacent to v_4. Return z, v_0, v_1, v_3, v_4, which induce an odd cycle with only one chord zv_3.

Case 5 : z is adjacent to v_1. The vertex z is not adjacent to v_3 because z is not adjacent to one of v_1 or v_3; call $FindObstruction3(z, v_1\text{-}v_3\text{-}\cdots\text{-}v_p, \emptyset)$.

$FindObstruction3(z, P, e)$

[where $P = v_0\text{-}\cdots\text{-}v_p$ is an odd path $p \geq 3$ with at most one chord, and this chord is $v_{t-1}v_{t+1}$ with $1 < t < p-1$, z is adjacent to both v_0 and v_p and is not adjacent to v_1, and e is the set that consists of the chord of P, if any]

Let r be the smallest integer > 0 such that z is adjacent to v_r ($r \geq 2$). When r is odd, return $z, v_0, \ldots, v_r$, which induce an odd hole or an odd cycle with only one chord e. When r is even, we consider the different following cases :

Case 1 : P has a chord $v_{t-1}v_{t+1}$ with $t < r$. Return $z, v_0, \cdots, v_{t-1}, v_{t+1}, \cdots, v_r$, which induce an odd hole.

Case 2 : P has a chord $v_{r-1}v_{r+1}$. When z is adjacent to both v_{r+1} and v_{r+2}, return z,$v_0, \cdots, v_{r-1}$,v_{r+1},v_{r+2}, which induce an odd cycle with only one chord zv_{r+1}. When z is not adjacent to v_{r+1}, call $FindObstruction3(z,v_r\text{-}\cdots\text{-}v_p,\emptyset)$. When z is adjacent to v_{r+1} and is not adjacent to v_{r+2}, call $FindObstruction4($ $z,v_r\text{-}\cdots\text{-}v_p,\emptyset)$.

Case 3 : P has a chord v_rv_{r+2}. When z is adjacent to v_{r+1}, return $z, v_0, \cdots,$ v_{r+1}, which induce an odd cycle with only one chord zv_r. When z is not adjacent to v_{r+1} and is adjacent to v_{r+2}, return $z, v_0, \cdots, v_r, v_{r+2}$, which induce an odd cycle with only one chord zv_r. When z is not adjacent to either v_{r+1} or v_{r+2}, call $FindObstruction1(z, v_r\text{-}\cdots\text{-}v_p)$.

Case 4 : P has no chord $v_{t-1}v_{t+1}$ with $t \leq r+1$. When z is adjacent to v_{r+1}, return $z, v_0, \cdots, v_{r+1}$, which induce an odd cycle with only one chord zv_r.

When z is not adjacent to v_{r+1}, call $FindObstruction3(z, v_r\text{-}\cdots\text{-}v_p, e)$.

$FindObstruction4(z, P, e)$

[where $P = v_0\text{-}\cdots\text{-}v_p$ is an odd path $p \geq 3$ with at most one chord, and this chord is $v_{t-1}v_{t+1}$ with $2 < t < p-1$, z is adjacent to v_0, v_1 and v_p and is not adjacent to v_2, and e is the set that consists of the chord of P, if any]

Let r be the smallest integer > 1 such that z is adjacent to v_r; we have $r \geq 3$.

Case 1 : P has a chord $v_{t-1}v_{t+1}$ with $2 < t < r$. When r is odd, return $z, v_1, \ldots, v_{t-1}, v_{t+1}, \ldots, v_r$, which induce an odd hole. When r is even, return $z, v_1, \ldots, v_r$, which induce an odd cycle with only one chord $v_{t-1}v_{t+1}$.

Case 2 : P has no chord $v_{t-1}v_{t+1}$ with $2 < t < r$. When r is odd, return $z, v_0, \ldots, v_r$, which induce an odd cycle with only one chord zv_1. When r is even, return $z, v_1, \ldots, v_r$ which induce an odd hole.

We show now that this algorithm can be implemented in linear time. At each pass of $Loop(P, i)$, i decreases by at least 1, so there are at most k_c passes of this loop. Each pass takes time $\mathcal{O}(deg(x_c^i) + deg(z))$, and x_c^i is different at each pass since i decreases. Vertex z is also different at each pass, because if $n(z) = j$, then z is not adjacent to w_{j-1}^c and so z will not be adjacent to any future vertex x_c^i, while if $n(z) \neq j$ then z becomes the second vertex on the path P until in a future loop we have $n(z) = j$.

Each call to a variant of *FindObstruction* happens with the same vertex z, so we need only run once through the adjacency array of z, the first time *FindObstruction* is called. Then computing the value of r takes time $\mathcal{O}(r)$ and the rest of each *FindObstruction* takes constant time. At each pass of *FindObstruction*, the length of path P decreases by at least r.

4 Strong stable sets

It was shown in [10] that in the case of a Meyniel graph, the set of vertices colored 1 by Algorithm MCColor is a strong stable set. But there are non-Meyniel graphs for which Algorithm MCColor and Algorithm Clique give a coloring and a clique of the same size but none of the color classes of the coloring is a strong stable set (see the example at the end of this section). In that case we would like to be able to find a Meyniel obstruction. This can be

done in time $\mathcal{O}(n^3)$ as described below.

Lemma 1 *Every nice stable set is a strong stable set.*

Proof. Let $S = \{x^1, \ldots, x^k\}$ be a nice stable set of a graph G. Suppose there exists a maximal clique Q with $Q \cap S = \emptyset$. Let G^i be the graph obtained from G by contracting $x^1, \ldots, x^i$ into w^i. For $i = 1, \ldots, k$, consider the following Property P^i: "In the graph G^i, vertex w^i is adjacent to all of Q." Note that Property P^k holds by the maximality of S and by the definition of w^k and that Property P^1 does not hold by the maximality of Q. So there is an integer $i \in \{2, \ldots, k\}$ such that P^i holds and P^{i-1} does not. Vertex x^i is not adjacent to all of Q by the maximality of Q. So, in the graph G^{i-1}, the clique Q contains vertices a and b such that a is adjacent to w^{i-1} and not to x^i and b is adjacent to x^i and not to w^{i-1}, and then the path w^{i-1}-a-b-x^i is a P_4, which contradicts the property that S is nice. □

Now, for any graph G and any vertex v of G, we can find a Meyniel obstruction or a strong stable set containing v by the following algorithm :

Apply the algorithm MCColor on a graph G, choosing v to be the first vertex to be colored. With the same notation as in Section 3, let $S = \{x_1^1, \ldots, x_1^{k_1}\}$ be the set of vertices colored 1. So S is a maximal stable set. We can check in time $\mathcal{O}(n^3)$ if S is a nice stable set. If S is a nice stable set, then S is a strong stable set by Lemma 1. If S is not a strong stable set, then the checking procedure returns $i \in \{2, \ldots, k_1\}$ such that there is an induced $P_4 : w_1^{i-1}$-v_1-v_2-x_1^i. Let $P = v_1$-v_2-x_1^i. Applying the procedure $Loop(P, i, \emptyset)$ of Section 3 gives a Meyniel obstruction in G.

Here is an example of a non-Meyniel graphs for which Algorithm MCColor followed by Algorithm Clique can give a coloring and a clique of the same size but none of the color classes of the coloring is a strong stable set. Consider the graph G form by the 3 triangles $\{a, d, e\}$, $\{b, f, g\}$, $\{c, h, i\}$ plus the edges af, ah, bd, bi, ce, cg. Algorithm MCColor can color the vertices in that order and with the given color: d-1, b-2, f-1, g-3, e-2, a-3, c-1, h-2, i-3, and Algorithm Clique returns the clique $\{a, d, e\}$. The algorithms give a coloring and a clique of the same size but none of the color classes $\{c, d, f\}$, $\{b, e, h\}$ or $\{a, g, i\}$ is a strong stable set.

5 Comments

The algorithms presented here are not recognition algorithms for Meyniel graphs. It can happen that the input graph is not Meyniel and yet the output is a clique and a coloring of the same size.

The fastest known recognition algorithm for Meyniel graph is due to Roussel and Rusu [15] and its complexity is $\mathcal{O}(m(m+n))$, (where n is the number of vertices and m is the number of edges), which beats the complexity of the algorithm of Burlet and Fonlupt [1]. So it appears to be easier to solve the Meyniel Graph Robust Algorithm Problem than to recognize Meyniel graphs. It could be the same for perfect graphs: it might be simpler to solve the Perfect Graph Robust Algorithm Problem than to recognize perfect graphs. Currently, the recognition of perfect graphs is done by an $\mathcal{O}(n^9)$ algorithm due to Chudnovsky, Cornuéjols, Liu, Seymour and Vušković [5] which actually recognizes Berge graphs (graphs that do not contain an odd hole or an odd antihole). The class of Berge graphs is exactly the class of perfect graphs by the Strong Perfect Graph Theorem of Chudnovsky, Robertson, Seymour and Thomas [6].

References

[1] M. Burlet, J. Fonlupt, Polynomial algorithm to recognize a Meyniel graph, Ann. Disc. Math. 21 (1984) 225–252.

[2] K. Cameron, J. Edmonds, Existentially polytime theorems, DIMACS Series Discrete Mathematics and Theoretical Computer Science 1 (1990) 83–99.

[3] K. Cameron, J. Edmonds, If it's easy to recognize, and you know it's there, can it be hard to find?, SIAM Conference on Discrete Mathematics, San Diego, August 2002.

[4] K. Cameron, J. Edmonds, Finding a strong stable set or a Meyniel obstruction in any graph, manuscript, presented at EuroComb 2005, Berlin, Germany, August 2005.

[5] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, K. Vušković, Recognizing Berge graphs, Combinatorica 25 (2005) 143–186.

[6] M. Chudnovsky, N. Robertson, P. Seymour, R. Thomas, The strong perfect graph theorem, Manuscript, Princeton Univ., 2002, to appear in Annals of Mathematics.

[7] J. Fonlupt, J.P. Uhry, Transformations which preserve perfectness and h-perfectness of graphs, Ann. Disc. Math. 16 (1982) 83–85.

[8] A. Hertz, A fast algorithm for coloring Meyniel graphs, J. Comb. Th. B 50 (1990) 231–240.

[9] C.T. Hoàng, On a conjecture of Meyniel, J. Comb. Th. B 42 (1987) 302–312.

[10] B. Lévêque, F. Maffray, Coloring Meyniel graphs in linear time, Cahiers Leibniz 105 (2004), to appear in Information Processing Letters, available on `http://hal.ccsd.cnrs.fr/ccsd-00001574`.

[11] B. Lévêque, F. Maffray, A linear algorithm for coloring vertices of a graph or finding a Meyniel obstruction, manuscript, 2005, available on `http://hal.ccsd.cnrs.fr/ccsd-00008557_v1`.

[12] S.E. Markosyan, I.A. Karapetyan, Perfect graphs, Akad. Nauk Armjan. SSR. Dokl. 63 (1976) 292–296.

[13] H. Meyniel, On the perfect graph conjecture, Disc. Math. 16 (1976) 334–342.

[14] G. Ravindra, Meyniel's graphs are strongly perfect, Ann. Disc. Math. 21 (1984) 145–148.

[15] F. Roussel, I. Rusu, Holes and dominoes in Meyniel graphs, Int. J. Found. Comput. Sci. 10 (1999) 127–146.

[16] F. Roussel, I. Rusu, An $O(n^2)$ algorithm to color Meyniel graphs, Disc. Math. 235 (2001) 107–123.