Coloring vertices of a graph or finding a Meyniel obstruction

Kathie Cameron\textsuperscript{a}, Benjamin Lévêque\textsuperscript{b}, Frédéric Maffray\textsuperscript{c,*}

\textsuperscript{a} Wilfrid Laurier University, Waterloo, Ontario, Canada N2L 3C5
\textsuperscript{b} Laboratoire G-SCOP, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France
\textsuperscript{c} C.N.R.S., Laboratoire G-SCOP, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France

\textbf{A B S T R A C T}

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 as an induced subgraph. Here we give a $O(n^2)$ algorithm that, for any graph, finds either a clique and a coloring of the same size or a Meyniel obstruction. We also give a $O(n^3)$ algorithm that, for any graph, finds either a strong stable set recognizable in polynomial time or a Meyniel obstruction.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

A graph $G$ is perfect if every induced subgraph $H$ of $G$ satisfies $\chi(H) = \omega(H)$, where $\chi(H)$ is the chromatic number of $H$ and $\omega(H)$ is the maximum clique size on $H$. A graph is Meyniel [11] if every odd cycle of length at least five has at least two chords. A Meyniel obstruction is an odd cycle of length at least five with at most one chord. Thus a graph is Meyniel if and only if it does not contain a Meyniel obstruction as an induced subgraph. Meyniel [11] and Markosyan and Karapetyan [10] proved independently that Meyniel graphs are perfect. Since an induced subgraph of a Meyniel graph is a Meyniel graph, this theorem can be stated in the following way:

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

This statement means that, for any graph $G$, exactly one of the following is satisfied (illustrated in Fig. 1):

- $G$ contains a Meyniel obstruction and it does not have a clique and a coloring of the same size.
- $G$ contains a Meyniel obstruction and it has a clique and a coloring of the same size.
- $G$ contains no Meyniel obstruction and (consequently) it has a clique and coloring of the same size.

We give a polytime algorithm which finds, in any graph, an instance of what the Meyniel–Markosyan–Karapetyan theorem says exists. That is, for any input graph, the algorithm finds either (a) a Meyniel obstruction or (b) a clique and a coloring of the same size. When the graph happens to contain both (a) and (b), then the algorithm returns exactly one of (a) or (b), not both, and we cannot predict which one.

\textsuperscript{*} 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 the Wilfrid Laurier University.

\textsuperscript{*} Corresponding author. Tel.: +33 476 57 47 01.
E-mail addresses: kcameron@wlu.ca (K. Cameron), benjamin.leveque@g-scop.inpg.fr (B. Lévêque), frederic.maffray@g-scop.inpg.fr (F. Maffray).
This algorithm works in time $O(n^2)$ where $n$ is the number of vertices of the input graph. This is an improvement in the complexity of the algorithm of the first and second authors [4,5], which finds, in any graph, a clique and coloring of the same size, or a Meyniel obstruction. This is an enhancement of the $O(n^2)$ algorithm of Roussel and Rusu [15] which requires that the input graph be Meyniel and returns only an optimal coloring.

This work is motivated by the “Perfect Graph Existential Polytime (EP) Problem” [3]: seek 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 hole is a chordless cycle of length at least four. An antihole is the complementary graph of a hole. The Strong Perfect Graph Theorem [7] states that a graph is perfect if and only if it contains no odd hole and no odd antihole. So a simple obstruction to perfectness is the existence of an odd hole or odd antihole.

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 [1] in $G$ is a stable set that contains a vertex of every maximal (by inclusion) clique of $G$. It is easy to see that a strong stable set is a maximal (by inclusion) stable set (indeed, if some vertex $v$ of $G \setminus S$ had no neighbor in $S$, then every maximal clique that contains $v$ would be disjoint from $S$). Note that if one can find a strong stable set in every induced subgraph of a graph $G$, then 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 $G \setminus S_1$, . . . , $S_k$ is a strong stable set of $G \setminus (S_1 \cup \cdots \cup S_{k-1})$, and $S_k$ is the last non-empty such set, then $S_1, \ldots, S_k$ is a coloring of $G$ which is the same size as some clique of $G$. This shows that if every induced subgraph of $G$ has a strong stable set, then $G$ is perfect.

Ravindra [13] presented the theorem that

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

The three cases that can occur are illustrated in Fig. 2.

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).

Hoàng [9] gives a $O(n^7)$ algorithm that finds, for any vertex of a Meyniel graph, a strong stable set containing this vertex. A disadvantage of the Ravindra–Hoàng theorem is that it is not an existentially polytime theorem. A theorem is called existentially polytime (EP) if it is a disjunction of NP predicates which is always true [3]. 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 (since a graph may have an exponential number of maximal cliques).

The Ravindra–Hoàng theorem is strengthened in [4,5] 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 $S = \{x_1, \ldots, x_k\}$, such that, for every $1 \leq i < k$ there is no induced $P_4$ between $x_{i+1}$ and the vertex which arises from the contraction in $G$ of $x_1, \ldots, x_i$. (Contracting vertices $x_1, \ldots, x_i$ in a graph means removing them and adding a new vertex $x$ with an edge between $x$ and every vertex of $G \setminus \{x_1, \ldots, x_i\}$ that is adjacent to at least one of $x_1, \ldots, x_i$. As usual, $P_4$ denotes a path on four vertices.)

Note that every maximal stable set of cardinality 1 is a nice stable set. Note that a maximal stable set of cardinality 2 is a nice stable set if and only if its vertices are not the endpoints of a $P_4$. In particular, the highlighted stable sets, both in the
second and in the third graphs of Fig. 2, are nice stable sets, whereas in the second graph of Fig. 2, every maximal stable set containing the vertex of degree two of the triangle is not strong and is not nice.

In Section 5, we show that a nice stable set is a strong stable set but that the converse is not true. The proof of the above displayed theorem in [4,5] is a polytime algorithm which for any graph and any vertex in that graph, finds an instance of what the theorem says exists. In Section 5, we give an $\mathcal{O}(n^4)$ algorithm for this, where $n$ is the number of vertices of the input graph.

2. The coloring algorithm

We recall the algorithm LexColor of Roussel and Rusu [15] which is a $\mathcal{O}(n^2)$ algorithm that colors optimally the vertices of a Meyniel graph, thereby improving the complexity of previous coloring algorithms due to Hertz $\mathcal{O}(nm)$ [8] (where $m$ is the number of edges of the input graph), Hoâng $\mathcal{O}(n^8)$ [9] and Ravindra [13].

LexColor is a greedy coloring algorithm. The integers $1, 2, \ldots, n$ are viewed as colors. For each vertex $x$ of $G$ and each color $c \in \{1, 2, \ldots, n\}$, we have a label $\text{label}_c(x)$ defined as follows. If $x$ has no neighbor colored $c$, then $\text{label}_c(x)$ is equal to 0; if $x$ has a neighbor colored $c$, then $\text{label}_c(x)$ is equal to the integer $i$ such that the first neighbor of $x$ colored $c$ is the $(n-i)$-th colored vertex of the graph. We consider the following (reverse) lexicographic order on the labels: $\text{label}_x < \text{Lex} \text{label}_y$ if and only if there exists a color $c$ such that $\text{label}_x(c) < \text{label}_y(c)$ and $\forall c' > c, \text{label}_x(c') = \text{label}_y(c')$. At each step, the algorithm selects an uncolored vertex which is maximum for the lexicographic order of the labels, assigns to this vertex the smallest color not present in its neighborhood, and iterates this procedure until every vertex is colored. More formally:

**Algorithm LexColor**

*Input:* A graph $G$ with $n$ vertices.
*Output:* A coloring of the vertices of $G$.
*Initialization:* For every vertex $x$ of $G$ and every color $c$ do $\text{label}_c(x) := 0$;
*General step:* For $i = 1, \ldots, n$ do:
1. Choose an uncolored vertex $x$ that maximizes $\text{label}_x$ for $\text{Lex}$;
2. Color $x$ with the smallest color $c$ not present in its neighborhood;
3. For every uncolored neighbor $y$ of $x$, if $\text{label}_c(y) := 0$ do $\text{label}_c(y) := n - i$.

This coloring algorithm is optimal on Meyniel graphs and its complexity is $\mathcal{O}(n^2)$ [15].

**Remark 1.** This version of LexColor has a minor modification from the original version of Roussel and Rusu [15]. When $x$ has a neighbor colored $c$, the integer $\text{label}_c(x)$ was originally defined to be the integer $i$ such that the first neighbor of $x$ colored $c$ is the $(n-i)$-th vertex colored $c$ of the graph (instead of the $(n-i)$-th colored vertex of the graph). For a color $c$, the order between $\text{label}_c(x)$ of each vertex $x$ is the same in the two versions of the algorithm, so the lexicographic order is the same and there is no difference in the two executions of the algorithm. This modification only simplifies the description of the algorithm.

**Remark 2.** The graph $P_6$ is an example of a non-Meyniel graph on which Algorithm LexColor is not optimal. Fig. 3 shows a possible execution of Algorithm LexColor on the graph $P_6$ which gives a coloring with 4 colors although there is a coloring with only 3 colors. Since $P_6$ is a member of many families of perfect graphs (such as brittle graphs, weakly chordal graphs, perfectly orderable graphs, etc; see [12] for the definitions), this algorithm will not perform optimally on these classes.

3. Finding a maximum clique

Given a coloring of a graph, there is a greedy algorithm that chooses one vertex of each color in an attempt to find a clique of the same size.
Proof of Lemma 1.

Lemma 2.
If \( G \) has a near-obstruction, then \( G \) has a Meyniel obstruction contained in the subgraph induced by \( P \cup \{ z \} \).

**Lemma 1.** If \( G_{n-1}^a \) contains a bad path, then \( G \) contains a near-obstruction.

**Lemma 2.** If \( G \) has a near-obstruction \( (P, z) \), then \( G \) has a Meyniel obstruction contained in the subgraph induced by \( P \cup \{ z \} \).

**Proof of Lemma 1.** Let \( P = w_{i-1}v_{i+1} \) be a bad path in \( G_{n-1}^a \), with the same notation as in the definition of bad path. We prove the lemma by induction on \( i \). We first claim that:

\((*)\) There exists a vertex \( z \), colored before \( x_i \) with a color \( > c \), that is adjacent to \( x_i \) and to \( w_{i-1} \) in \( G_{n-1}^a \) and satisfies the following property. If \( v_{i+1}v_3 \) is the chord of \( P \), then \( z \) is not adjacent to at least one of \( v_1 \) and \( v_3 \). If \( v_{i+1}v_3 \) is not a chord of \( P \), then \( z \) is not adjacent to at least one of \( v_1 \) and \( v_2 \).
For let us consider the situation when the algorithm selects $x_i$ to be colored. Let $U$ be the set of vertices of $G_{r-1}^* \setminus \{w_{i-1}\}$ that are already colored at that moment. We know that every vertex of $G_{r-1}^* \setminus \{w_{i-1}\}$ will have a color from $[c, c+1, \ldots, \ell]$ when the algorithm terminates. So, if $c \geq 2$, every vertex of $U$ satisfies $\forall c' < c, label_v(c') \neq 0$. For any $X \subseteq U$, let color($X$) be the set of colors of the vertices of $X$. Put $T = N(x_i) \cap U$. Every vertex of $T$ has a color $c \geq c + 1$, and so is adjacent to at least one vertex colored $c$ in $G$ and thus is adjacent to $w_{r-1}$ in $G_r^*$. Specify one vertex $v_i$ of $P$ as follows: put $r = 3$ if $v_1v_2$ is a chord of $P$; else put $r = 2$. Note that $v_i$ is not adjacent to $w_{i-1}$ and $x_i$ is $x_i$ by the definition of bad path. Suppose the claim is false: so every vertex of $T$ is adjacent to $v_i$ and $v_j$.

Since every vertex of $T$ is adjacent to $v_i$, we have $label_v(c') \geq label_v(c')$ for every color $c' > c$. Since $v_1$ is adjacent to $w_{i-1}$, we have $label_v(c') > 0$. Since $x_i$ is colored $c$, we have $label_v(c') = 0$. So $label_v \geq label_v$, which means that $v_i$ is already colored. Moreover, $color(v_i) \notin \{1, \ldots, c\} \cup \{\}$. (Take time $deg(x_i)$ to find such a vertex $z$.)

Let $j$ be the smallest integer such that both $v_1$ and $z$ have a neighbor in $\{x_1, \ldots, x_j\}$. Then $j < i$ because $z$ and $v_1$ are adjacent to $w_{i-1}$.

Suppose that $j$ is adjacent to both $v_1$ and $z$. Then $(v_1, v_2, \ldots, v_j, z)$ is a near-obstruction in $G$. Indeed, by Claim (*) it is a near-obstruction of Type 2 if $v_1v_2$ is the chord of $P$, of Type 3 if $v_1v_2$ is not a chord of $P$ and $z$ is not adjacent to $v_1$, or of Type 4 if $v_1v_2$ is not a chord of $P$ and $z$ is adjacent to $v_1$ (and thus is not adjacent to $v_j$).

Now suppose that $j$ is not adjacent to both $v_1$ and $z$. Then the definition of $j$ implies that $j > 1$ and either (a) $j$ is adjacent to $x_j$ and not to $w_{j-1}$, and $v_1$ is adjacent to $w_{i-1}$ and not to $x_j$ or (b) $v_1$ is adjacent to $x_j$ and not to $w_{j-1}$, and $v_1$ is adjacent to $w_{i-1}$ and not to $x_j$. In either case, let $k$ be the smallest integer with $k \geq 1$ such that $z$ is adjacent to $v_k$. Such a $k$ exists because $z$ is adjacent to $v_1$. Suppose that $k$ is odd. If (a) holds, then let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-2}v_{j-1}v_1v_2v_3$; if (b) holds, let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-1}v_1v_2v_3$. Then $P'$ is an odd chordless path, so $P'$ is a bad path in $G_{r-1}^*$, and the result follows by induction.

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

**Case 1:** $P$ has a chord $v_{j-1}v_{j-2}v_{j-3}$ with $t < k$. If (a) holds, then let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; if (b) holds, let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$. Then $P'$ is an odd chordless path, so $P'$ is a bad path in $G_{r-1}^*$, and the result follows by induction.

**Case 2:** $P$ has a chord $v_{j-1}v_{j-2}v_{j-3}$. When $z$ is adjacent to both $v_1$ and $v_k+2$, if (a) holds, then let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; if (b) holds, let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; in either case, $P'$ is an odd path and has only one chord, which is $v_{j-1}v_{j-2}$, so $P'$ is a bad path in $G_{r-1}^*$, and the result follows by induction. When $z$ is not adjacent to $v_1$, then $v_{j-1}v_{j-2}v_{j-3}$ is an odd chordless path, so $(v_{j-1}v_{j-2}v_{j-3}, z)$ is a near-obstruction of Type 3. When $z$ is adjacent to $v_{k+1}$ and is not adjacent to $v_{k+2}$, then $(v_{k+1}v_{k+2}v_{k+3})$ is an odd chordless path, so $(v_{k+1}v_{k+2}v_{k+3}, z)$ is a near-obstruction of Type 4.

**Case 3:** $P$ has a chord $v_{j-1}v_{j-2}v_{j-3}$. When $z$ is adjacent to both $v_1$ and $v_k$, if (a) holds, then let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; if (b) holds, let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; in either case, $P'$ is an odd path and has only one chord, which is $v_{j-1}v_{j-2}$, so $P'$ is a bad path in $G_{r-1}^*$, and the result follows by induction. When $z$ is not adjacent to $v_1$ and $v_k+2$, then $(v_{k+1}v_{k+2}v_{k+3})$ is a near-obstruction of Type 1.

**Case 4:** $P$ has no chord $v_{j-1}v_{j-2}v_{j-3}$ with $t \leq k+1$. When $z$ is adjacent to $v_{j-1}$, if (a) holds, then let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; if (b) holds, let $P' = w_{j-1}v_{j-1}v_{j-2}v_{j-3}v_{j-4}v_{j-5}v_{j-6}v_1v_2v_3$; in either case, $P'$ is an odd path and has only one chord, which is $v_{j-1}v_{j-2}$, so $P'$ is a bad path in $G_{r-1}^*$, and the result follows by induction. When $z$ is not adjacent to $v_{k+1}$ and $v_{k+2}$, then $(v_{k+1}v_{k+2}v_{k+3})$ has at most one chord, which is the chord of $P$ if it exists, so $(v_{k+1}v_{k+2}v_{k+3}, z)$ is a near-obstruction of Type 3. This completes the proof of the last case. □

Let us discuss the complexity of the algorithmic variant of this proof. When we find a new bad path, the value of $i$ decreases by at least 1, and so this happens at most $n_r$ times. Dealing with one bad path takes time $\Theta(deg(x_i) + deg(z))$ (for the corresponding $i$), and $x_i$ is different at each call since $i$ decreases. Vertex $z$ is also different at each call, because $z$ becomes a vertex of the new bad path. When the algorithm produces a new bad path to be examined, it also tells if the path has no chord or one chord, and what the chord is (if it exists); so we do not have to spend any time to find this chord. So the total complexity of this algorithm is $\Theta(m + n)$.

**Proof of Lemma 2.** We use the same notation for $P$ as in the definition of near-obstruction. We prove the lemma by induction on $p$. If $p = 3$, then the hypothesis implies immediately that $P \cup \{z\}$ induces an obstruction. Now let $p \geq 5$.

If $(P, z)$ is a near-obstruction of Type 1, 2 or 3, then let $r$ be the smallest integer $\geq 3$ such that $z$ is adjacent to $v_r$. If $(P, z)$ is of Type 4, then let $r$ be the smallest integer $\geq 3$ such that $z$ is adjacent to $v_r$.

First assume that $(P, z)$ is a near-obstruction of Type 1. So $r \geq 3$. If $r$ is odd, then $z$, $v_0$, $v_1$, $v_2$, $v_3$ induce an odd cycle with only one chord $v_0v_2$. If $r$ is even, then $z$, $v_0$, $v_2$, $v_3$, $v_4$ induce an odd hole.
Now assume that \((P, z)\) is a near-obstruction of Type 2.

Case 2.1: \(z\) is not adjacent to either of \(v_1, v_2, v_3\). So \(r \geq 3\). If \(r\) is odd, then \(z, v_0, \ldots, v_r\) induce an odd cycle with only one chord \(v_1 v_3\). If \(r\) is even, then \(r \geq 4\), and \(z, v_0, v_1, v_3, \ldots, v_r\) induce an odd hole.

Case 2.2: \(z\) is not adjacent to \(v_1\) and is adjacent to \(v_2, v_3\). So \(r = 2\). If \(z\) is not adjacent to \(v_3\), then \(z, v_0, v_1, v_3\) induce an odd cycle with only one chord \(v_1 v_2\). Suppose \(z\) is adjacent to \(v_3\). If \(z\) is also adjacent to \(v_4\), then \(z, v_0, v_1, v_3, v_2\) induce an odd cycle with only one chord \(v_3 v_4\). If \(z\) is not adjacent to \(v_4\), consider the path \(P' = v_2 \cdots v_p\). Then \(P'\) is an odd chordless path and \(|P'| = |P| - r\), so \((P', z)\) is a near-obstruction of Type 4, and the result follows by induction.

Case 2.3: \(z\) is adjacent to \(v_1\). So \(r = 1, 2\) is not adjacent to \(v_3\) by the definition of Type 2. Consider the path \(P' = v_3 v_4 \cdots v_p\). Then \(P'\) is an odd chordless path and \(|P'| = |P| - r - 1\), so \((P', z)\) is a near-obstruction of Type 3, and the result follows by induction.

Now assume that \((P, z)\) is a near-obstruction of Type 3. If \(r\) is odd, then \(z, v_0, \ldots, v_r\) induce an odd cycle with at most one chord. If \(r\) is even, we consider the following cases:

Case 3.1: \(P\) has a chord \(v_{t-1} v_{t+1}\) with \(t < r\). Then \(z, v_0, \ldots, v_{t-1}, v_{t+1}, \ldots, v_r\) induce an odd hole.

Case 3.2: \(P\) has a chord \(v_r v_0\). If \(z\) is not adjacent to \(v_{r+1}\), then \(z, v_0, v_1, \ldots, v_r, v_0\) induce an odd cycle with only one chord \(v_r v_0\). If \(z\) is also adjacent to \(v_{r+2}\), then \(z, v_0, v_1, \ldots, v_r, v_{r+1}, v_{r+2}\) induce an odd cycle with only one chord \(v_r v_{r+1}\). If \(z\) is not adjacent to \(v_{r+2}\), then \(p \geq r + 3\). Consider the path \(P' = v_r \cdots v_p\). Then \(P'\) is an odd chordless path and \(|P'| = |P| - r\), so \((P', z)\) is a near-obstruction of Type 4, and the result follows by induction.

Case 3.3: \(P\) has a chord \(v_r v_{r+2}\). If \(z\) is adjacent to \(v_{r+1}\), then \(z, v_0, v_1, \ldots, v_r, v_{r+2}\) induce an odd cycle with only one chord \(v_r v_{r+1}\). Suppose \(z\) is not adjacent to \(v_{r+1}\), then \(z, v_0, v_1, \ldots, v_r, v_{r+1}\) induce an odd cycle with only one chord \(v_r v_{r+1}\). If \(z\) is not adjacent to \(v_{r+2}\), then \(p \geq r + 3\). Consider the path \(P' = v_r \cdots v_p\). Then \(P'\) is an odd path with at most one chord, which is the chord of \(P\) (if it exists) and \(|P'| = |P| - r\), so \((P', z)\) is a near-obstruction of Type 3, and the result follows by induction.

Case 3.4: \(P\) has no chord \(v_{t-1} v_{t+1}\) with \(t \leq r + 1\). If \(z\) is adjacent to \(v_{r+1}\), then \(z, v_0, v_1, \ldots, v_{r+1}\) induce an odd cycle with only one chord \(v_r v_{r+1}\). If \(z\) is not adjacent to \(v_{r+1}\), then consider the path \(P' = v_r \cdots v_p\). Then \(P'\) is an odd path with at most one chord, which is the chord of \(P\) (if it exists) and \(|P'| = |P| - r\), so \((P', z)\) is a near-obstruction of Type 3, and the result follows by induction.

Now assume that \((P, z)\) is a near-obstruction of Type 4.

Suppose \(P\) has a chord \(v_{t-1} v_{t+1}\) with \(2 < t < r\). If \(r\) is odd, then \(z, v_1, v_3, \ldots, v_t, v_{t+1}, \ldots, v_r\) induce an odd hole. If \(r\) is even, then \(z, v_1, v_3, \ldots, v_t, v_{t+1}, \ldots, v_r\) induce an odd cycle with only one chord \(v_{t-1} v_{t+1}\).

Now \(P\) has no chord \(v_{t-1} v_{t+1}\) with \(2 < t < r\). If \(r\) is odd, then \(z, v_0, \ldots, v_r\) induce an odd cycle with only one chord \(v_{t-1} v_{t+1}\). If \(r\) is even, then \(z, v_1, \ldots, v_r\) induce an odd hole. This completes the proof of the four cases.

In the algorithmic variant of this proof, each recursive call happens with the same vertex \(z\), so we need only run once through the adjacency array of \(z\). Note that the first near-obstruction is produced by the algorithm of Lemma 1, so we already know if \(P\) has no chord or one chord, and what its chord is, if it exists. Computing the value of \(r\) takes time \(\Theta(r)\), and the rest of each call takes constant time. Note that either a Meyniel obstruction is output, or a near-obstruction \((P', z)\) is obtained.

Now let \(|P'| \leq |P| - r\), and we know if \(P'\) has no chord or one chord, and what its unique chord is (if it exists); so we do not have to spend any time to find this chord. So the total running time is \(\Theta(|P| + \deg_P(z))\) (where \(\deg_P(z)\) is the number of vertices of \(P\) that are adjacent to \(z\)).

Algorithms LexColor and Clique run in time \(\Theta(n^2)\) and \(\Theta(n + m)\) respectively, so the total time for finding, in any graph, either a clique and a coloring of the same size, or a Meyniel obstruction is \(\Theta(n^2)\).

**Remark.** As mentioned earlier, the execution of Algorithm LexColor on the graph \(\overline{F_6}\) is not optimal. Given the coloring shown in Fig. 3, Algorithm Clique will stop on color 1 and clique \(Q = \{f, a, d\}\). We observe that no vertex of color 1 is adjacent to all \(Q\), because vertex \(c\) is not adjacent to \(d\) and \(b\) is not adjacent to \(a\). So \(c-a-d-b\) is a chordless path of length three between \(c\) and \(b\). Vertex \(c\) was colored before \(d\) because of \(e\), which is not adjacent to \(d\), and the algorithm will return the Meyniel obstruction induced by \(\{c, a, d, b, e\}\) (which is a cycle of length five with only one chord \(ae\)).

5. Strong stable sets

In this section, we show that, in the case of a Meyniel graph, the set of vertices colored 1 by Algorithm LexColor is a strong stable set. But there are non-Meyniel graphs for which Algorithm LexColor 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 find a Meyniel obstruction. We describe below a \(\Theta(n^2)\) algorithm that, for any graph \(G\) and vertex \(v\) of \(G\), finds a Meyniel obstruction or a nice stable set containing \(v\).

**Lemma 3.** 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 \(G'\) be the graph obtained from \(G\) by contracting \(x_1, \ldots, x_k\) into \(w_i\). For \(i = 1, \ldots, k\), consider the following property \(P_i^0:\) “In the graph \(G'\), vertex \(w_i\) is adjacent to all of \(Q\).” Note that Property \(P_i^0\) holds by the maximality of \(S\) and by the definition of \(w_i\).

Now assume that \(P_i^0\) does not hold by the maximality of \(Q\). So there is an integer \(i \in \{2, \ldots, k\}\) such that \(P_i^0\) 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 LexColor on a graph $G$, choosing $v$ to be the first vertex to be colored. Let $S = \{s_1, \ldots, s_{n_1}\}$ be the set of vertices colored 1. So $S$ is a maximal stable set. We can check in time $O(n^3)$ whether $S$ is a nice stable set. If $S$ is a nice stable set, then $S$ is a strong stable set by Lemma 3. If $S$ is not a nice stable set, then the checking procedure returns some $i \in \{2, \ldots, n_1\}$ such that there is an induced path $t_{i-1} - a - b - s_i$, where $t_{i-1}$ is the vertex obtained by contracting $s_1, \ldots, s_{i-1}$. Applying the procedure described in Lemmas 1 and 2 of Section 4 to this bad path $t_{i-1} - a - b - s_i$ gives a Meyniel obstruction in $G$.

Remark. The graph in Fig. 4 is an example of a non-Meyniel graph for which Algorithm LexColor followed by Algorithm Cliquer 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. Given the coloring shown in Fig. 4, Algorithm Cliquer can output the clique $\{a, i, a\}$. None of the color classes is a strong stable set (vertices colored 1 miss the maximal clique $\{b, c\}$, vertices colored 2 miss the maximal clique $\{e, f\}$, and vertices colored 3 miss the maximal clique $\{a, b\}$). Vertices $a, f, h$ are colored 1 in this order. Let $(af)$ be the contraction of $a$ and $f$. The bad path found by the algorithm is $(af) - b - c - h$; this is the $P_4$ between $(af)$ and $h$ that shows that $a, f, h$ is not a nice stable set. Vertex $g$ is a neighbor of $(af)$ and $h$ that causes $f$ to be colored before $c$. When decontracting $(af)$, vertex $f$ is still adjacent to $g$ and $b$, so the algorithm will output the Meyniel obstruction induced by $\{f, b, c, h, g\}$ (which is a cycle of length five with only one chord $hg$). This graph contains a strong stable set $b, e, h$ that is not nice. On the graph in Fig. 4, Algorithm LexColor can color the vertices in the order $b, f, g, h, e, a, i, c, d$, and the set of vertices colored 1 is $\{b, e, h\}$, which is a strong stable set but not a nice stable set (because all three pairs $\{b, e\}, \{b, h\}$ and $\{e, h\}$ are respectively endpoints of $P_4$'s).

6. 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 [14] and its complexity is $O(m(m + n))$, where $n$ is the number of vertices and $m$ is the number of edges, which beats the complexity of the algorithm of Berge and Fonlupt [2]. So it appears to be easier to solve the Meyniel Graph EP Problem than to recognize Meyniel graphs. It could be the same for perfect graphs: it might be simpler to solve the Perfect Graph EP Problem than to recognize perfect graphs. Currently, the recognition of perfect graphs is done by an $O(n^3)$ algorithm due to Chudnovsky et al. [6] which actually recognizes Berge graphs (graphs that contain no odd hole and no odd antihole). The class of Berge graphs is exactly the class of perfect graphs by the Strong Perfect Graph Theorem of Chudnovsky et al. [7].
References