Coloring vertices of a graph or finding a Meyniel obstruction

Kathie Cameron, Jack Edmonds, Benjamin Lévêque, Frédéric Maffray

Coloring vertices of a graph or finding a Meyniel obstruction*

Kathie Cameron[†], Jack Edmonds[‡], Benjamin Lévêque[§], Frédéric Maffray[¶]

November 13, 2007

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

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

1 Introduction

A graph is *Meyniel* [10] 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 [10] and Markosyan and Karapetyan [9] proved that Meyniel graphs are perfect. This theorem can be stated in the following way:

*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

†Laurier University, Waterloo, Ontario, Canada N2L 3C5, kcameron@wlu.ca

‡EP Institute, Kitchener, Ontario, Canada, N2M 2M6, jackedmonds@rogers.com

§Laboratoire G-SCOP, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France, benjamin.leveque@g-scop.inpg.fr

¶C.N.R.S., Laboratoire G-SCOP, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France, frederic.maffray@g-scop.inpg.fr

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 polytime algorithm which finds, in any graph, an instance of what the Meyniel-Markosyan-Karapetyan theorem says exists. This algorithm works in time $\mathcal{O}(n^2)$ 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 [3, 4], which finds, in any graph, a clique and coloring of the same size, or a Meyniel obstrution. This is an enhancement of the $\mathcal{O}(n^2)$ algorithm of Roussel and Rusu [14], which optimally colors any Meyniel graph.

This work is motivated by the "Perfect Graph Robust Algorithm Problem" [2]: 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. According to the Strong Perfect Graph Theorem [6], proved in a different way, 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* in G is a stable set that contains a vertex of every maximal (by inclusion) clique of G. 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 $G\backslash(S_1 \cup \ldots \cup S_{\ell-1})$, and S_ℓ is the last non-empty such set, then $S_1, \ldots, S_\ell$ is a coloring of G which is the same size as some clique of G.

Ravindra [12] 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 [8] 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).

Hoang [8] give a $\mathcal{O}(n^7)$ 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 [2]. 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 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 S 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. (*Contracting* vertices $x_1, \dots, x_k$ in a graph means removing them and adding a new vertex x with an edge between x and every vertex of $G \setminus \{x_1, \dots, x_k\}$ that is adjacent to at least one of $x_1, \dots, x_k$. As usual, P_4 denotes 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. In Section 5, we give an $\mathcal{O}(n^3)$ 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 [14] 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 $\mathcal{O}(nm)$ [7] (where m is the number of edges of the input graph), Hoàng $\mathcal{O}(n^8)$ [8] and Ravindra [12].

LexColor is a greedy coloring algorithm. The integers $1, 2, \dots, n$ are viewed as colors. For each vertex x of G and each color $c \in \{1, 2, \dots, n\}$, we have a label $label_x(c)$ defined as follows. If x has no neighbor colored c, then $label_x(c)$ is equal to 0; if x has a neighbor colored c, then $label_x(c)$ 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 : $label_x <_{Lex} label_y$ if and only if there exists a color c such that $label_x(c) < label_y(c)$ and $\forall\ c' > c$, $label_x(c') = label_y(c')$. 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 neighbourhood, 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 $label_x(c) := 0$;

General step: For $i = 1, \dots, n$ do:
1. Choose an uncolored vertex x that maximizes $label_x$ for $<_{Lex}$;
2. Color x with the smallest color c not present in its neighbourhood;
3. For every uncolored neighbor y of x, if $label_y(c) := 0$ do $label_y(c) := n - i$.

This coloring algorithm is optimal on Meyniel graph and its complexity is $\mathcal{O}(n^2)$ [14].

Remark 1: This version of LexColor has a minor modification from the original version of Roussel and Rusu [14] . When x has a neighbor colored c, the integer $label_x(c)$ 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 $label_x(c)$ 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 $\overline{P}_6$ is an example of a non-Meyniel graph on which Algorithm LexColor is not optimal. The graph $\overline{P}_6$ has vertices u, v, w, x, y, z and non-edges are uv, vw, wx, xy, yz. Algorithm LexColor can color the vertices in the following order with the indicated color: $v - 1$, $y - 2$, $w - 1$, $u - 3$, $x - 2$, $z - 4$; but the graph has chromatic number 3. Since $\overline{P}_6$ is a member of many families of perfect graphs (such as brittle graphs, weakly chordal graphs, perfectly orderable graphs, etc; see [11] 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.

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 $Q := \emptyset$;

General step: For $c = \ell, \dots, 1$ do:
Select a vertex x of color c that maximizes $N(x) \cap Q$, do $Q := Q \cup \{x\}$.

Algorithm Clique can be implemented in time $\mathcal{O}(m+n)$.

We claim that when the input consists of a Meyniel graph G with the coloring produced by LexColor, then the output Q of Algorithm Clique is a clique of size ℓ. This result is a consequence of the next section, we show that when the output of the algorithm is not a clique, we can find a Meyniel obstruction.

4 Finding a Meyniel obstruction

Let G be a general (not necessarily Meyniel) graph on which Algorithm LexColor is applied. Let ℓ be the total number of colors used by the algorithm. Then we apply Algorithm Clique. At each step, we check whether the selected vertex x of color c is adjacent to all of Q (this can be done without increasing the complexity of Algorithm Clique by maintaining a counter which for each vertex counts the number of its neighbors in Q). If this holds at every step, then the final 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 Algorithm Clique stops the first time $Q \cup \{x\}$ is not a clique and records the current color c and the current clique Q. So we know that no vertex colored c is adjacent to all of Q. Let us show now how to find a Meyniel obstruction in G. As usual, a path is called odd or even if its length (number of edges) is respectively odd or even.

Let n_c be the number of vertices colored c, and for $i = 1, \dots, n_c$ let x_i be the i-th vertex colored c. Let G^* be the subgraph of G obtained by removing the vertices of colors $< c$. Let G^*_i be the graph obtained from G^* by removing $x_1, \dots, x_i$ and adding a new vertex w_i with an edge to every vertex that is adjacent to one of $x_1, \dots, x_i$ (in other words, vertices $x_1, \dots, x_i$ are contracted into w_i).

Let $h \leq n_c$ be the smallest integer such that every vertex of color $> c$ has a neighbor in $\{x_1, \dots, x_h\}$. Integer h exists because n_c has that property.

There is a vertex a of Q that is not adjacent to x_h, because x_h is not adjacent to all of Q. Thus $h \geq 2$. Note that a is adjacent to w_{h-1} in G^*_{h-1}. There is a vertex b of Q that is adjacent to x_h and not to w_{h-1}, by the definition of h. Then w_{h-1}-a-b-x_h is a chordless odd path in G^*_{h-1}.

For any $i > 1$, a *bad path* is any odd path $P = w_{i-1}$-v_1-$\cdots$-v_p in G^*_{i-1} such that $v_p = x_i$, path P has at most one chord, and such a chord (if any) is $v_{t-1}v_{t+1}$ with $1 < t < p-1$. Note that the path w_{h-1}-a-b-x_h obtained at the end of the preceding paragraph is a bad path.

A *near-obstruction* in G is any pair (P, z), where P is a path v_0-$\cdots$-v_p, with odd $p \geq 3$, P has at most one chord, such a chord (if any) is $v_{t-1}v_{t+1}$ with $0 < t < p-1$, vertex z is a vertex of $G \setminus P$ that is adjacent to both v_0, v_p, and the pair (P, z) satisfies one of the following conditions:
Type 1: v_0v_2 is the only chord of P, and z is not adjacent to either of v_1, v_2.
Type 2: v_1v_3 is the only chord of P, and z is not adjacent to one of v_1, v_3.
Type 3: v_0v_2 is not a chord of P, and z is not adjacent to v_1.
Type 4: v_0v_2 and v_1v_3 are not chords of P, and z is adjacent to v_1 and not to v_2.

The following lemmas show that the existence of a bad path is a certificate that the graph is not Meyniel. The proof of the first lemma can easily be read as a linear-time algorithm which, given a bad path, finds explicitly a near-obstruction. Likewise, the proof of the second lemma can easily be read as a linear-time algorithm which, given a near-obstruction, finds explicitly an obstruction. Since G^*_{h-1} contains the bad path w_{h-1}-a-b-x_h, these two lemmas imply that G contains a Meyniel obstruction.

Lemma 1 *If G^*_{i-1} 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_1-$\cdots$-v_p be a bad path in G^*_{i-1}, with the same notation as above. 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^*_{i-1} and satisfies the following property. If v_1v_3 is the chord of P, then z is not adjacent to at least one of v_1 and v_3. If v_1v_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^*_{i-1} that are already colored at that moment. We know that every vertex of G^*_{i-1} will have a color from $\{c, c+1, \dots, \ell\}$ when the algorithm terminates. So, if $c \geq 2$, every vertex v 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 $\geq c+1$, and so is adjacent to at least one vertex colored c in G and thus is adjacent to w_{i-1} in G^*_{i-1}. Specify one vertex v_r of P as follows: put $r = 3$ if v_1v_3 is a chord of P; else put $r = 2$. Note that v_r is not adjacent to w_{i-1} and $v_r \neq x_i$ by the definition of bad path. Suppose the claim is false: so every vertex of T is adjacent to v_1 and v_r.

Since every vertex of T is adjacent to v_1, we have $label_{v_1}(c') \geq label_{x_i}(c')$ for every color $c' > c$. Since v_1 is adjacent to w_{i-1}, we have $label_{v_1}(c) > 0$. Since x_i is colored c, we have $label_{x_i}(c) = 0$. So $label_{v_1} >_{Lex} label_{x_i}$, which means that v_1 is already colored. Moreover, $color(v_1) \notin \{1, \dots, c\} \cup color(T)$.

Since every vertex of T is adjacent to v_r, we have $label_{v_r}(c') \geq label_{x_i}(c')$ for every color $c' > c$. Since v_r is adjacent to v_1, we have $label_{v_r}(color(v_1)) > 0$. Since $color(v_1) \notin \{1, \dots, c\} \cup color(T)$ we have $label_{x_i}(color(v_1)) = 0$. So $label_{v_r} >_{Lex} label_{x_i}$, which means that v_r is already colored. However, v_r is not adjacent to w_{i-1}, so c was the smallest color available for v_r when it was colored, which contradicts the definition of w_{i-1} and x_i. This completes the proof of Claim (*).

Now let z be a vertex given by Claim (*). (It takes 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 $\{x_1, \dots, x_j\}$. Then $j < i$ because z and v_1 are adjacent to w_{i-1}.

Suppose that x_j is adjacent to both v_1 and z. Then $(x_j\text{-}v_1\text{-}\cdots\text{-}v_p, z)$ is a near-obstruction in G. Indeed, by Claim (*), it is a near-obstruction of Type 2 if v_1v_3 is the chord of P, of Type 3 if v_1v_3 is not a chord of P and z is not adjacent to v_1, or of Type 4 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).

Now suppose that x_j is not adjacent to both v_1 and z. Then the definition of j implies that $j > 1$ and either (a) z is adjacent to x_j and not to w_{j-1}, and v_1 is adjacent to w_{j-1} and not to x_j or (b) v_1 is adjacent to x_j and not to w_{j-1}, and z is adjacent to w_{j-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_p.

Suppose that k is odd. If (a) holds, then let $P' = w_{j-1}\text{-}v_1\text{-}\cdots\text{-}v_k\text{-}z\text{-}x_j$; if (b) holds, let $P' = w_{j-1}\text{ -}z\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_j$. Then P' has at most one chord, which is the chord of P if it exists and if its two end-vertices are in P', so

P' is a bad path in G^*_{j-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 $v_{t-1}v_{t+1}$ with $t < k$. If (a) holds, then let $P' = w_{j-1}\text{-}v_1\text{-}\cdots\text{-}v_{t-1}\text{-}v_{t+1}\text{-}\cdots\text{-}v_k\text{-}z\text{-}x_j$; if (b) holds, let $P' = w_{j-1}\text{-}z\text{-}v_k\text{-}\cdots\text{-}v_{t+1}\text{-}v_{t-1}\text{-}\cdots\text{-}v_1\text{-}x_j$. Then P' is chordless, so P' is a bad path in G^*_{j-1}, and the result follows by induction.

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 (a) holds, then let $P' = w_{j-1}\text{-}v_1\text{-}\cdots\text{-}v_{k-1}\text{-}v_{k+1}\text{-}v_{k+2}\text{-}z\text{-}x_j$; if (b) holds, let $P' = w_{j-1}\text{-}z\text{-}v_{k+2}\text{-}v_{k+1}\text{-}v_{k-1}\text{-}\cdots\text{-}v_1\text{-}x_j$; in either case, P' has only one chord, which is zv_{k+1}, so P' is a bad path in G^*_{j-1}, and the result follows by induction. When z is not adjacent to v_{k+1}, then $v_k\text{-}\cdots\text{-}v_p$ is a chordless path, so $(v_k\text{-}\cdots\text{-}v_p, z)$ is a near-obstruction of Type 3. When z is adjacent to v_{k+1} and is not adjacent to v_{k+2}, then $v_k\text{-}\cdots\text{-}v_p$ is a chordless path, so $(v_k\text{-}\cdots\text{-}v_p, z)$ is a near-obstruction of Type 4.

Case 3: P has a chord v_kv_{k+2}. When z is adjacent to v_{k+1}, if (a) holds, then let $P' = w_{j-1}\text{-}v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+1}\text{-}z\text{-}x_j$; if (b) holds, let $P' = w_{j-1}\text{-}z\text{-}v_{k+1}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_j$; in either case, P' has only one chord, which is zv_k, so P' is a bad path in G^*_{j-1}, and the result follows by induction. When z is not adjacent to v_{k+1} and is adjacent to v_{k+2}, if (a) holds, then let $P' = w_{j-1}\text{-}v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+2}\text{-}z\text{-}x_j$; if (b) holds, let $P' = w_{j-1}\text{-}z\text{-}v_{k+2}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_j$; in either case, P' has only one chord, which is zv_k, so P' is a bad path in G^*_{j-1}, and the result follows by induction. When z is not adjacent to v_{k+1} or to v_{k+2}, then $(v_k\text{-}\cdots\text{-}v_p, z)$ is a near-obstruction of Type 1.

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 (a) holds, then let $P' = w_{j-1}\text{-}v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+1}\text{-}z\text{-}x_j$; if (b) holds, let $P' = w_{j-1}\text{-}z\text{-}v_{k+1}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_j$; in either case, P' has only one chord, which is zv_k, so P' is a bad path in G^*_{j-1}, and the result follows by induction. When z is not adjacent to v_{k+1}, then $v_k\text{-}\cdots\text{-}v_p$ has at most one chord, which is the chord of P if it exists and if its two end-vertices are in P', so $(v_k\text{-}\cdots\text{-}v_p, z)$ 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_c times. Dealing with one bad path takes time $\mathcal{O}(deg(x_i)+deg(z))$ (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 $\mathcal{O}(m+n)$.

Proof of Lemma 2. We use the same notation for P as above. 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 ≥ 1 such that z is adjacent to v_r. If (P, z) is of Type 4, then let r be the smallest integer ≥ 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 $z, v_0, \ldots, v_r$ induce an odd cycle with only one chord v_0v_2. If r is even, then $z, v_0, v_2, \ldots, v_r$ 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. So $r \geq 3$. If r is odd, then $z, v_0, \ldots, v_r$ induce an odd cycle with only one chord v_1v_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. So $r = 2$. If z is not adjacent to v_3, then z, v_0, v_1, v_3, v_2 induce an odd cycle with only one chord v_1v_2. So suppose z is adjacent to v_3. If z is also adjacent to v_4, then z, v_0, v_1, v_3, v_4 induce an odd cycle with only one chord zv_3. If z is not adjacent to v_4, then $p \geq 5$. Consider the path $P' = v_2\text{-}\cdots\text{-}v_p$. Then P' is chordless 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 $r = 1$, z is not adjacent to v_3 by the definition of Type 2, and so $p \geq 5$. Consider the path $P' = v_1\text{-}v_3\text{-}\cdots\text{-}v_p$. Then P' is chordless 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 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 $v_{r-1}v_{r+1}$. If z is not adjacent to v_{r+1}, then $z, v_0, \ldots, v_{r-1}, v_{r+1}, v_r$ in this order induce an odd cycle with only one chord $v_{r-1}v_r$. So suppose z is adjacent to v_{r+1}. If z is also adjacent to v_{r+2}, then z, $v_0, \ldots, v_{r-1}$, v_{r+1}, v_{r+2} induce an odd cycle with only one chord zv_{r+1}. If z is not adjacent to v_{r+2}, then $p \geq r+3$. Consider the path $P' = v_r\text{-}\cdots\text{-}v_p$. Then P' is chordless 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_rv_{r+2}. If z is adjacent to v_{r+1}, then $z, v_0, \ldots,$ v_{r+1} induce an odd cycle with only one chord zv_r. So suppose z is not adjacent to v_{r+1}. If z is adjacent to v_{r+2}, then $z, v_0, \ldots, v_r, v_{r+2}$ induce an odd cycle with only one chord zv_r. If z is not adjacent to v_{r+2}, then $p \geq r+3$. Consider the path $P' = v_r\text{-}\cdots\text{-}v_p$. Then $v_r v_{r+2}$ is the unique chord of P' and $|P'| = |P| - r$, so (P', z) is a near-obstruction of Type 1, 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 $z, v_0, \ldots, v_{r+1}$ induce an odd cycle with only one chord zv_r. If z is not adjacent to v_{r+1}, then consider the path $P' = v_r\text{-}\cdots\text{-}v_p$. Then P' has 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 that P has a chord $v_{t-1}v_{t+1}$ with $2 < t < r$. If r is odd, then $z, v_1, \ldots, v_{t-1}, v_{t+1}, \ldots, v_r$ induce an odd hole. If r is even, then $z, v_1, \ldots, v_r$ 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 zv_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 $\mathcal{O}(r)$, and the rest of each call takes constant time. At each call, either a Meyniel obstruction is output, or a near-obstruction (P', z) is obtained. Note that $|P'| \leq |P| - r$, and we know if P' has no chord or one chord, and what its unique chord is (if it exits); so we do not have to spend any time to find this chord. So the total running time is $\mathcal{O}(|P| + deg_P(z))$.

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

Remark: As mentioned earlier, on the graph $\overline{P}_6$ with vertices u, v, w, x, y, z and non-edges uv, vw, wx, xy, yz, a possible execution of LexBFS colors the vertices in the following order and with the given color: $v-1$, $y-2$, $w-1$, $u-3$, $x-2$, $z-4$. On this coloring, Algorithm Clique will stop when $c = 1$ and $Q = \{x, u, z\}$. No vertex of color 1 is adjacent to all Q : w is not adjacent to x and v is not adjacent to u. So w-u-x-v is a chordless path of length 3 between w and v. Vertex w was colored before x because of y, which is not adjacent to x, and $\{w, u, x, v, y\}$ induces a Meyniel obstruction.

5 Strong stable sets

It can be proved 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 be able to find a Meyniel obstruction. This can be done in time $\mathcal{O}(n^3)$ as described below.

Lemma 3 *Every nice stable set is a strong stable set.*

Proof. Let $S = \{x_1, \dots, x_k\}$ 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 $x_1, \dots, x_i$ into w_i. For $i = 1, \dots, k$, 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 $i \in \{2, \dots, k\}$ 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. $\square$

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 $S = \{s_1, \dots, s_{n_1}\}$ be the set of vertices colored 1. So S is a maximal stable set. We can check in time $\mathcal{O}(n^3)$ 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 $i \in \{2, \dots, n_1\}$ such that there is an induced path t_{i-1}-a-b-s_i, where t_{i-1} is the vertex obtained by contracting $s_1, \dots, s_{i-1}$. 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: Here is an example of a non-Meyniel graph for which Algorithm LexColor 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 LexColor can color the vertices in the following order and with the given color: $d - 1$, $b - 2$, $f - 1$, $g - 3$, $c - 1$, $i - 3$, $h - 2$, $a - 3$, $e - 2$, 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.

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 [13] 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, Annals of Mathematics 164 (2006), 51–229.

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

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

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

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

[11] J.L. Ramírez-Alfonsín, B.A. Reed, Perfect Graphs, Wiley Interscience, 2001.

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

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

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