Coloring Meyniel graphs in linear time

Benjamin Lévêque[*], **Frédéric Maffray**[†]

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

February 8, 2020

Abstract

A Meyniel graph is a graph in which every odd cycle of length at least five has two chords. We present a linear-time algorithm that colors optimally the vertices of a Meyniel graph and finds a clique of maximum size.

1 Introduction

This paper deals with the graphs in which every odd cycle of length at least five has at least two chords. Meyniel [9] proved that such graphs are perfect [11], and it has become customary to call them *Meyniel graphs*. Chordal graphs (graphs in which every cycle of length at least four has a chord), bipartite graphs, and more generally i-triangulated graphs and parity graphs (see [3]) are examples of Meyniel graphs. Burlet and Fonlupt [3] gave a polynomial-time recognition algorithm for Meyniel graphs, and later Roussel and Rusu [15] gave another, faster, such algorithm, whose complexity is $\mathcal{O}(m(m+n))$ for a graph with n vertices and m edges.

A *coloring* of the vertices of a graph is a mapping that assigns one color to each vertex in such a way that any two adjacent vertices receive distinct colors. A coloring is *optimal* if it uses as few colors as possible. The *chromatic number* $\chi(G)$ of a graph G is the number of colors used by an optimal coloring. We are interested in polynomial-time algorithms that color the vertices of a Meyniel graphs optimally. Hoàng [8] gave such an algorithm, with complexity $\mathcal{O}(n^8)$, which uses the so-called amalgam decomposition from [3]. The concept of even pair [5] (which we recall formally below) enabled Hertz [7] to devise a coloring algorithm for Meyniel graphs that works in time $\mathcal{O}(nm)$. At each step Hertz's

[*]E.N.S. Lyon. E-mail: benjamin.leveque@imag.fr
[†]C.N.R.S. E-mail: frederic.maffray@imag.fr

algorithm finds an even pair of vertices in the graph and contracts them. Later, Roussel and Rusu [16] defined a coloring algorithm called LexColor (for Lexicographic Color); this algorithm colors the vertices of a Meyniel graph in time $\mathcal{O}(n^2)$ without contracting even pairs, but it "simulates" such a contraction, and its optimality follows from that of Hertz's algorithm. LexColor is based on LexBFS (for Lexicographic Breadth First Search), which was originally invented by Rose, Tarjan and Lueker [13] to find a simplicial elimination ordering in a chordal graph.

We propose here an algorithm that we call MCColor (for Maximum Constraint Color), which will color any graph in time $\mathcal{O}(n + m)$. Just like LexColor, in the case of a Meyniel graph MCColor "simulates" the contraction of even pairs. MCColor is based on the algorithm MCS (for Maximum Cardinality Search) due to Tarjan and Yannakakis [18], which is a simplification of LexBFS and can also be used to find a simplicial elimination ordering in a chordal graph.

2 Algorithm MCColor

Our algorithm 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 (this vertex is the most "constrained"), 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 $G$.
Step 0: 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 neighbour $y$ of $x$, add $color(x)$ to $label(y)$.
```

We will prove that this algorithm is optimal on Meyniel graphs, and actually on a larger class than the class of Meyniel graphs. A graph G is a *quasi-Meyniel graph* [7] if G contains no odd chordless cycle on at least five vertices and G has a vertex, called a *pivot*, that is an endvertex of every edge that is the unique chord of an odd cycle of G. De Figueiredo and Vušković [4] found a polynomial-time algorithm to decide if a graph G is quasi-Meyniel and, if it is, to give a pivot of G.

Let us make a few obvious observations:
(a) every Meyniel graph is a quasi-Meyniel graph;

(b) in a Meyniel graph every vertex is a pivot;
(c) if G is a quasi-Meyniel graph and z is a pivot, then $G \setminus z$ is a Meyniel graph.

We can run algorithm MCColor on any quasi-Meyniel graph with a given pivot, only imposing that the first vertex to be colored is the given pivot. By observations (a) and (b), any application of the algorithm on a Meyniel graph is simply an application on the graph viewed as a quasi-Meyniel and with an arbitrary vertex as pivot.

3 Optimality

In this section we prove that Algorithm MCColor can color every quasi-Meyniel graph G with $\omega(G)$ colors, where $\omega(G)$ is the maximum size of a clique in G. This will prove the optimality of the algorithm (and also the perfectness of G). Our proof is directly inspired by Roussel and Rusu's proof [16] that their algorithm LexColor is optimal on quasi-Meyniel graphs.

Theorem 1 *Let Algorithm* MCColor *be applied on a quasi-Meyniel graph G with a given pivot, coloring the pivot first. Then the algorithm uses exactly $\omega(G)$ colors.*

Before proving this theorem, we need to recall some notation and definitions. An *even pair* in a graph G is a pair of non-adjacent vertices such that every chordless path between them has even length (number of edges). A survey on even pairs is given in [5]. In a graph G, the neighbourhood of a vertex v is denoted by $N(v)$. Given two vertices x, y in G, the operation of *contracting* them means removing x and y and adding one vertex with an edge to each vertex of $N(x) \cup N(y)$. The next lemma states an essential result about even pairs.

Lemma 1 ([6, 10]) *The graph G' obtained from a graph G by contracting an even pair of G satisfies $\omega(G') = \omega(G)$ and $\chi(G') = \chi(G)$.*

A graph is *even contractile* [2] if there is a sequence of graphs $G_0, \ldots, G_k$ ($k \geq 0$) such that $G_0 = G$, G_k is a clique, and if $k > 0$ then for $i = 1, \ldots, k$ the graph G_i is obtained from G_{i-1} by contracting an even pair of G_{i-1}. A graph is *perfectly contractile* if every induced subgraph of G is even contractile. Hertz [7] proved that every Meyniel graph is perfectly contractile; indeed his proof is the polynomial-time algorithm mentioned in the introduction, which colors every Meyniel graph optimally through a sequence of even-pair contractions. Just like for Roussel and Rusu's algorithm [16], the optimality of our algorithm will follow from the fact that each color class produced by the algorithm corresponds to the contraction of some even pairs.

As in [16], say that a path $P = v_0\text{-}v_1\text{-}v_2\text{-}\cdots\text{-}v_p$ is *quasi-chordless* if it has at most one chord and, if it has one, then this chord is $v_{j-1}v_{j+1}$ with $1 < j < p-1$

(so the endvertices of P are not incident to the chord). We will use the following lemma which is essentially from [16] (the first item of the lemma was also proved for Meyniel graphs in [9, 12]).

Lemma 2 ([16]) *Let G be a quasi-Meyniel graph. Let $P = v_0\text{-}v_1\text{-}\cdots\text{-}v_{2k}\text{-}v_{2k+1}$ (with $k \geq 0$) be a quasi-chordless odd path in G. Suppose that v_0 is a pivot of G, and that there is a vertex w adjacent to both v_0, v_{2k+1}. Then:*

- *If P is chordless, then w is adjacent to every vertex of P.*
- *If P has a chord $v_{j-1}v_{j+1}$ (with $1 < j < 2k$), then w is adjacent to every vertex of $P \setminus v_j$.*

Proof of Theorem 1. Let G be a quasi-Meyniel graph on which Algorithm MCColor is applied, so that a given pivot is the first vertex to be colored. 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. Therefore 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 $V(G) = \{x_1^1, x_1^2, \ldots, x_1^{k_1}, x_2^1, \ldots, x_2^{k_2}, \ldots, x_l^1, \ldots, x_l^{k_l}\}$.

Define a sequence of graphs and vertices as follows. Put $G_1^1 = G$ and $w_1^1 = x_1^1$ (which is a pivot of G). For $i = 2, \ldots, k_1$, call G_1^i the graph obtained from G_1^{i-1} by contracting w_1^{i-1} and x_1^i into a new vertex w_1^i. In the graph $G_1^{k_1}$, we remark that $w_1^{k_1}$ is adjacent to all other vertices of $G_1^{k_1}$; for otherwise, there is a vertex y that is not adjacent to $w_1^{k_1}$, which means that y has no neighbour of color 1, so y should have received color 1, a contradiction. Let us call simply w_1 the vertex $w_1^{k_1}$.

The sequence continues as follows. For each $c \in \{2, \ldots, l\}$, put $G_c^1 = G_{c-1}^{k_{c-1}}$ and $w_c^1 = x_c^1$. For $i = 2, \ldots, k_c$, call G_c^i the graph obtained from G_c^{i-1} by contracting vertices w_c^{i-1} and x_c^i into a new vertex w_c^i. In $G_c^{k_c}$, we can again remark that $w_c^{k_c}$ is adjacent to all other vertices of $G_c^{k_c}$, for the same reason as above, and we call simply w_c the vertex $w_c^{k_c}$. So the last graph in the sequence, $G_l^{k_l}$, is a clique of size l with vertices $w_1, \ldots, w_l$, where each w_c is obtained by the contraction of the vertices of color c.

Lemma 3 *For every $c \in \{1, \ldots, l\}$ and $i \in \{1, \ldots, k_c - 1\}$, there is no quasi-chordless odd path from w_c^i to x_c^{i+1} in G_c^i.*

Proof. Suppose on the contrary that there exists a quasi-chordless odd path $P = v_0\text{-}v_1\text{-}\cdots\text{-}v_{2k}\text{-}v_{2k+1}$ from $v_0 = w_c^i$ to $v_{2k+1} = x_c^{i+1}$ in G_c^i. We have $k > 0$ since w_c^i, x_c^{i+1} are not adjacent. Note that every vertex of P has a non-neighbour in G_c^i. Put $W_1 = \emptyset$ and $W_c = \{w_1, \ldots, w_{c-1}\}$ if $c \geq 2$, and recall that any $w \in W_c$ is a vertex of G_c^i that is adjacent to all vertices of $G_c^i \setminus w$. So P contains no vertex of W_c. We know that every vertex of $G_c^i \setminus W_c$ will have a color from $\{c, c+1, \ldots, l\}$ when the algorithm terminates. So, if $c \geq 2$, every vertex v of $G_c^i \setminus W_c$ (in particular every vertex of P) satisfies $label(v) \supseteq \{1, 2, \ldots, c-1\}$.

Let us consider the situation when the algorithm selects x_c^{i+1}. Let U be the set of vertices 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 $T = \{v \in N(x_c^{i+1}) \cap U,\ color(v) \geq c+1\}$. We have $|label(x_c^{i+1})| = (c-1) + |color(T)|$. Every vertex of T is adjacent to at least one vertex colored c in G and thus is adjacent to w_c^i in G_c^i. 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 v_0 and $v_r \neq x_c^{i+1}$ since P is quasi-chordless. Since every vertex of T is adjacent to both v_0, v_{2k+1}, by Lemma 2 every vertex of T is adjacent to v_1 and v_r.

Suppose v_1 is not colored yet. 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 $|label(v_1)| = |color(N(v_1))| \geq c + |color(T)| > |label(x_c^{i+1})|$, which contradicts the fact that the algorithm is about to color x_c^{i+1}. 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 $|label(v_r)| = |color(N(v_r))| \geq (c-1) + |color(T \cup \{v_1\})| = c + |color(T)| > |label(x_c^{i+1})|$, again a contradiction. So v_r is already colored. However, v_r is not adjacent to w_c^i, so c is the smallest color available for v_r; but this contradicts the definition of w_c^i and x_c^{i+1}. This completes the proof of Lemma 3.

Lemma 4 *For every color $c \in \{1, \ldots, l\}$ and integer $i \in \{0, 1, \ldots, k_c - 1\}$, the following two properties hold:*

(A_i) If $i \geq 1$, then w_c^i and x_c^{i+1} form an even pair of G_c^i.

(B_i) G_c^{i+1} is a quasi-Meyniel graph and w_c^{i+1} is a pivot of this graph.

Proof. Let $c \in \{1, \ldots, l\}$. We show by induction on i that (A_i) and (B_i) hold.

Property (A_0) holds by vacuity. Property (B_0) is just the general assumption when $c = 1$, so suppose $c \geq 2$. In the graph G_c^1, every vertex w_h with $h \in \{1, \ldots, c-1\}$ is adjacent to all other vertices of the graph; moreover $G_c^1 \setminus \{w_1, \ldots, w_{c-1}\}$ is a Meyniel graph by observation (c) above, since it is a subgraph of $G \setminus x_1^1$ and x_1^1 is a pivot of G. It follows that G_c^1 is actually a Meyniel graph and so w_c^1 is a pivot of this graph.

Now suppose that $i \geq 1$ and that (A_{i-1}) and (B_{i-1}) hold. Lemma 3 implies immediately that (A_i) holds. To prove (B_i), suppose on the contrary that G_c^{i+1} is not a quasi-Meyniel graph with pivot w_c^{i+1}. This means that G_c^{i+1} contains an odd cycle C of length at least 5, with vertices $v_1, \ldots, v_{2k+1}$ ($k \geq 2$) and edges v_iv_{i+1} modulo $2k+1$, such that either C is chordless or C has exactly one chord and w_c^{i+1} is not an endvertex of that chord. By taking such a C as short as possible, we may assume that if it has a chord then this chord is $v_{j-1}v_{j+1}$ for some $j \in \{1, \ldots, 2k+1\}$. Then w_c^{i+1} must be in C, for otherwise C would be a cycle in $G_c^i \setminus w_c^i$, contradicting (B_{i-1}). So we may assume $w_c^{i+1} = v_1$. A vertex $x \in \{w_c^i, x_c^{i+1}\}$ cannot be adjacent to both v_2, v_{2k+1} in G_c^i, for otherwise replacing w_c^{i+1} by x in C gives an odd cycle in G_c^i that contradicts (B_{i-1}). So we may assume that w_c^i is adjacent to v_2 and not to v_{2k+1} and that x_c^{i+1} is adjacent to v_{2k+1} and not to v_2. If C has a chord v_2v_{2k+1}, then w_c^i-v_2-v_{2k+1}-x_c^{i+1} is a chordless odd path from w_c^i to x_c^{i+1} in G_c^i, which contradicts Lemma 3. So either C is chordless or its unique chord is $v_{j-1}v_{j+1}$ with $j \in \{3, \dots, 2k\}$. But then w_c^i-v_2-$\cdots$-v_{2k+1}-x_c^{i+1} is a quasi-chordless odd path from w_c^i to x_c^{i+1} in G_c^i, which contradicts Lemma 3. So (B_i) holds. This completes the proof of Lemma 4.

Lemma 4 implies that in the sequence $G = G_1^1, \dots, G_l^{k_l}$, each graph other than the first one is obtained from its predecessor by contracting an even pair of the predecessor. Then Lemma 1 applied successively along the sequence implies that $\omega(G) = \omega(G_l^{k_l})$ and $\chi(G) = \chi(G_l^{k_l})$; but $\chi(G_l^{k_l}) = \omega(G_l^{k_l}) = l$ since $G_l^{k_l}$ is a clique of size l; so the algorithm does color the input graph optimally with $\omega(G)$ colors.

4 Complexity

We now analyze the complexity of Algorithm MCColor. Let the input be a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$. We assume that, for every vertex x, we are given the set $N(x)$ as a list, of size $degree(x)$. So the total size of the input is $n + m$.

We will not compute explicitly the set $label(x)$ for $x \in V$. Instead, we consider the counter $|label(x)|$ for every $x \in V$ and maintain an $n \times n$ array A such that, for every vertex $x \in V$ and every color i, the entry $A(x, i)$ is set to 1 if x has a neighbour of color i, else it is set to 0.

Ordering the vertices according to the value of $|label(x)|$ can be done with the usual techniques, such as bucket sort [1]: For each $j = 0, 1, \dots, n-1$, we maintain the set L_j of the uncolored vertices x such that $|label(x)| = j$. This set is implemented as a doubly linked list, where each element also points to the head of the list, which is the integer j. The heads of the non-empty L_j's are themselves put in decreasing order into a doubly linked list M.

During the initialization step we must initialize each L_j, M and the array A. All vertices are put into L_0, and L_0 is the only element of M. Thus the initialization of the L_j's and of M takes time $\mathcal{O}(n)$. Initializing every entry of A would take time $\mathcal{O}(n^2)$; but we can skip this by using an argument from [1, ex. 2.12] (see also [17, p. 9]), which ensures that only those entries that are actually accessed during the algorithm are initialized, on the first occasion they are accessed. It will be obvious below that for every vertex x at most $degree(x) + 1$ entries $A(x, .)$ are accessed. So the total time for initializing A will be $\mathcal{O}(m + n)$.

Consider line 1 of the general step. Using the data structure we just described, this line can be done in constant time for each vertex: we get the largest integer j in M and the first vertex x in L_j, remove x from L_j, and, if L_j becomes empty, remove j from M. So the total time over all vertices is $\mathcal{O}(n)$.

Now consider line 2. Given x, we scan the entries $A(x,1)$, $A(x,2)$, …, until we find the first entry $A(x,c)$ that is not equal to 1, and we assign color c to x. Since each entry equal to 1 corresponds to a color assigned to a different neighbour of x, we will scan at most $degree(x)+1$ entries for each x. So the total time over all vertices is $\mathcal{O}(m+n)$.

Finally consider line 3. For every neighbour y of x we check whether $A(y,c)$ is equal to 1 or not. If it is 1 we do nothing, else we update vertex y: we set $A(y,c) := 1$, move y from the list L_j that contains it to the list L_{j+1}, and update M accordingly (i.e., if $j+1$ was not in M we insert it between j and the predecessor of j, and if L_j becomes empty we remove j from M). Using the data structure this takes constant time for each y. So line 3 takes time $\mathcal{O}(degree(x))$. Note that it happens only once for each x. So the total complexity over all vertices is $\mathcal{O}(m)$.

In conclusion, the total running time of the algorithm is $\mathcal{O}(n+m)$.

5 Finding a maximum clique

We can extend our algorithm so that, in the case of a quasi-Meyniel graph, it produces in linear time a clique of maximum size. This idea is implicit in [16] but the algorithmic aspects were not worked out there. Assume that G is any graph and that we are given a coloring of its vertices using l colors. Then we can apply the following algorithm to build a set Q:

```
Input: A graph $G$ and a coloring of its vertices using $l$ colors.
Set $Q := \emptyset$ and for every vertex $x$ set $q(x) := 0$;
For $c = l, l-1, \ldots, 1$ in decreasing order, pick a vertex $x_c$ of color $c$
that maximizes $q(x_c)$, do $Q := Q \cup \{x_c\}$, and for every neighbour $y$
of $x_c$ do $q(y) := q(y) + 1$;
Output the set $Q$.
```

We claim that when the input consists of a quasi-Meyniel graph G with the coloring produced by MCColor, then the output Q is a clique of size l. Actually this will be true in a more general framework. Let G be any graph. Suppose that we are given a coloring of its vertices using l colors and that the vertices of G are named $x_1^1, x_1^2, \ldots, x_1^{k_1}, x_2^1, \ldots, x_2^{k_2}, \ldots, x_l^1, \ldots, x_l^{k_l}$, so that vertices of subscript c have color c. Define the corresponding sequence of graphs G_c^i and vertices w_c^i ($1 \le c \le l$, $1 \le i \le k_c$) as in the preceding section. Suppose that for each $c = 1, \ldots, l$ and $i = 1, \ldots, k_c - 1$, vertices w_c^i and x_c^{i+1} satisfy Lemma 3 in G_c^i. Note that this is exactly the situation after Algorithm MCColor is applied to a quasi-Meyniel graph. Actually every perfectly contractile graph admits such a sequence. Then we have:

Lemma 5 *For any $c \in \{1, \ldots, l\}$, let Q be a clique of size $l - c$ such that if $c < l$ then Q consists of one vertex of each color $j = c + 1, \ldots, l$. Then there exists a vertex of color c that is adjacent to all of Q.*

Proof. We prove this lemma by descending induction on c. The lemma holds for $c = l$. Suppose $c \in \{1, \ldots, l - 1\}$ and assume by induction that the lemma holds for $c + 1$. For $i = 1, \ldots, k_c$, consider the following Property P_i: "In the graph G_c^i, vertex w_c^i is adjacent to all of Q." We may assume that Property P_1 does not hold, for otherwise the lemma holds with vertex $x_c^1 = w_c^1$. Recall that Property P_{k_c} holds. So there exists an integer $i \in \{2, \ldots, k_c\}$ such that P_i holds and P_{i-1} does not. Then, in the graph G_c^{i-1} vertex x_c^i is adjacent to all of Q, for otherwise Q contains vertices a, b such that a is adjacent to w_c^{i-1} and not to x_c^i and b is adjacent to x_c^i and not to w_c^{i-1} and then w_c^{i-1}-a-b-x_c^i is a path of length 3, contradicting Lemma 3. So the lemma holds for c with vertex x_c^i, which completes the proof of the lemma.

Lemma 5 implies that at every step $c = l, l - 1, \ldots, 1$ the algorithm will find a vertex of color c that is adjacent to all of the current set Q. So, at termination the algorithm will return a clique that consists of one vertex of each color. This procedure can be implemented in time $\mathcal{O}(m)$, using again bucket sort with respect to the counter q.

6 Comments

Our algorithm is not "robust" in the sense that if the input graph is not quasi-Meyniel the algorithm will not detect this fault. The algorithm will just produce a coloring, which may be non-optimal. An example is the graph $\overline{P}_6$ whose vertices are u, v, w, x, y, z and whose non-edges are uv, vw, wx, xy, yz. A possible application of the algorithm produces the ordering v, y, x, z, u, w with the corresponding colors $1, 2, 2, 3, 1, 4$ although the graph has chromatic number 3. Since $\overline{P}_6$ is in many classical families of perfect graphs (such as brittle graphs, weakly chordal graphs, perfectly orderable graphs, etc—see [11] for the definition of these classes), our algorithm will not perform optimally on these classes. We remark that $\overline{P}_6$ is also a graph which the algorithms LexColor [16] and LexBFS + Color (which consists simply of the greedy algorithm applied on an ordering produced by LexBFS) [14] may fail to color optimally.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman. *The Design and analysis of computer algorithms.* Addison-Wesley, Menlo Park, California, 1974.

[2] M.E. Bertschi, Perfectly contractile graphs. *J. Comb. Th. B* 50 (1990), 222–230.

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

[4] C.M.H. De Figueiredo, K. Vušković. Recognition of quasi-Meyniel graphs. *Disc. Appl. Math.* 113 (2001), 255–260.

[5] H. Everett, C.M.H. de Figueiredo, C. Linhares Sales, F. Maffray, O. Porto, B.A. Reed. Even pairs. In [11], 67–92.

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

[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] H. Meyniel. The graphs whose odd cycles have at least two chords. *Ann. Disc. Math.* 21 (1984), 115–119.

[10] H. Meyniel. A new property of critical imperfect graphs and some consequences. *Eur. J. Comb.* 8 (1987), 313–316.

[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] D.J. Rose, R.E. Tarjan, G.S. Lueker. Algorithmic aspects of vertex elimination of graphs. *SIAM J. Comput.* 5 (1976), 266–283.

[14] F. Roussel, I. Rusu. A linear algorithm to color i-triangulated graphs. *Inf. Proc. Lett.* 70 (1999), 57–62.

[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.

[17] J.P. Spinrad. *Efficient Graph Representations.* Fields Institute Monographs, Am. Math. Soc., Providence, R.I., 2003.

[18] R.E. Tarjan, M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.* 13 (1984), 566–579.