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

Benjamin Lévêque, Frédéric Maffray

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

Benjamin Lévêque* Frédéric Maffray*†

September 8, 2005

Abstract

A Meyniel obstruction is an odd cycle with at most one chord and a graph is Meyniel if and only if it has no Meyniel obstruction. The authors have found in a previous work a linear-time algorithm for optimaly color the vertices of a Meyniel graph. Here we extend this algorithm to obtain a linear-time algorithm that, for any graph, colors optimally the vertices of the graph or finds a Meyniel obstruction.

1 Introduction

This work is motivated by the Perfect Graph Robust Algorithm Problem proposed by Kathie Cameron and Jack Edmonds. In [2], they advocated seeking a robust 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 of perfectness is the existence of an odd hole or odd antihole.

Here we give an answer to this problem for the class of Meyniel graphs, which is a subclass of perfect graphs. A graph is Meyniel [8] if every odd cycle of length at least five has at least two chords, so a Meyniel obstruction is an odd cycle with at most one chord. Ravindra already gave in [9] an informal description of an algorithm which, for any graph, finds either a good stable set or a Meyniel obstruction. A good stable set is a stable set

*Laboratoire Leibniz-IMAG, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France {benjamin.leveque, frederic.maffray}@imag.fr

†C.N.R.S.

that meets all maximal cliques of the graph, so Ravindra's result provides an algorithm which, for any graph, finds an optimal coloring or a Meyniel obstruction.

Recently, we defined a linear time algorithm for optimally coloring the vertices of a Meyniel graph [7] improving the complexity of previous coloring algorithms of Roussel and Rusu $\mathcal{O}(n^2)$ [11], Hertz $\mathcal{O}(nm)$ [5], Hoàng $\mathcal{O}(n^8)$ [6] and Ravindra [9]. Here, we give an extention of this algorithm to obtain a linear time algorithm which, for any graph G, finds either a clique and a coloring of the same size or a Meyniel obstruction.

2 The coloring algorithm

We recall the linear algorithm MCColor of [7] which 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.

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 neighbour y of x, add $color(x)$ to $label(y)$.

This algorithm has been extended in [7] by another greedy linear algorithm, which, in the case of a Meyniel graph and a coloring obtained by the algorithm MCColor, will produce a clique of maximum size. Let G be any graph given with a coloring of its vertices using l colors. Then we can apply the following algorithm to build a set Q:

Algorithm Q

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 $Q := \emptyset$, $c := l$, and for every vertex x set $q(x) := 0$;

General step: While $c \neq 0$ do:
Pick a vertex x of color c that maximizes $q(x)$, do $Q := Q \cup \{x\}$, for every neighbour y of x do $q(y) := q(y) + 1$, and do $c := c - 1$.

3 Finding an obstruction

Let G be a general graph (possibly not Meyniel) on which Algorithm MC-Color 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. 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 $V(G) = \{x_1^1, x_1^2, \ldots, x_1^{k_1}, x_2^1, \ldots, x_2^{k_2}, \ldots, x_l^1, \ldots, x_l^{k_l}\}$.

Then we apply Algorithm Q. At each step of the algorithm Q, we check if the picked vertex x of color c has $q(x) = l - c$. If the equality is satisfied at each step then Q is a clique of cardinality l and we have a clique and a coloring of the same size. If not, then the algorithm Q stops the first time the equality is not satisfied, records 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 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 wich is adjacent to $N(x_c^1, \ldots, x_c^i)$. The first part of the algorithm will consist in using the procedure *Loop* to search the x_c^i's to find the step so there exists an obstruction that appears in G_i^* which only uses vertices of G^*. Then we call $FindObstruction$ on a path P and a vertex z which sees both extremities of P, when we know that $z \cup P$ contains an obstruction. The path P may contains one chord and the variants $FindObstruction1, 2, 3, 4$ correspond to the possible positions of that chord.

For each vertex v of color $> c$ of G^*, let $n(v)$ be the smallest integer i such that x_c^i is adjacent to v, the integer $n(v)$ exists because v has received a color strictly greater than c (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 does not see all of Q. Let

v_1 be a vertex of Q not adjacent to x_c^i, the vertex v_1 exists because x_c^i is not adjacent to all Q, and v_1 is adjacent to w_c^{i-1} in G_{i-1}^*. Let v_2 be a vertex such that $n(v_2) = i$, the vertex v_2 is adjacent to x_c^i but not to w_c^{i-1} in G_{i-1}^*. Let $P = v_1$-v_2-x_c^i, the path w_c^{i-1}-P is an odd chordless path in G_{i-1}^*, 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 with at most one chord, and this chord is $v_{t-1}v_{t+1}$ with $1 < t < p-1$ in G_{i-1}^*, e is a list which is empty if P has no chord, e contains the chord of P otherwise)

If v_1v_3 is the chord of P, then let z be a vertex, colored before x_c^i, with a color $> c$, that is adjacent to x_c^i and not adjacent to v_1 or to v_3. Such a vertex z exists, otherwise v_1, v_3 would have been colored before x_c^i and v_3 would be the i-th vertex colored c. Else, let z be a vertex, colored before x_c^i, with a color $> c$, that is adjacent to x_c^i and not adjacent to v_1 or to v_2. Such a vertex z exists, otherwise v_1, v_2 would have been colored before x_c^i and v_2 would be the i-th vertex colored c. (It takes time $deg(x_c^i)$ to find such a vertex z.) The vertex z sees w_c^{i-1} in G_{i-1}^* because z is colored before x_c^i and received a color strictly greater than c.

Let $j = max(n(z), n(v_1))$, we have $j < i$ because z and v_1 see w_c^{i-1}. When $n(z) = n(v_1)$, we have x_c^j sees v_1 and z. If v_1v_3 is the chord of P, call $FindObstruction2(z, x_c^j$-$P)$, else, when z misses v_1, call $FindObstruction3($ z, x_c^j-$P, e)$, and when z sees v_1 and misses v_2, call $FindObstruction4(z, x_c^j$-$P, e)$.

When $n(z) \neq (v_1)$, we have $j > 1$, one of w_c^{j-1}, x_c^j sees v_1 and misses z, the other one misses v_1 and sees z. Let k be the smallest integer such that z sees v_k, the integer k exists because z sees v_p (it takes time $deg(z)$ to compute k). When k is odd, if $n(z) = j$, then let $P' = v_1$-$\cdots$-v_k-z-x_c^j, else let $P' = z$-v_k-$\cdots$-v_1-x_c^j. If e is a chord of P', then call $Loop(P', j, e)$, else call $Loop(P', j, \emptyset)$. When k is even, p is odd so $k < p$, we consider the different following case :

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

Case 2 : P has a chord $v_{k-1}v_{k+1}$. When z sees v_{k+1} and v_{k+2}, if $n(z) = j$, then let $P' = v_1$-$\cdots$-v_{k-1}-v_{k+1}-v_{k+2}-z-x_c^j, else let $P' = z$-v_{k+2}-v_{k+1}-v_{k-1}-$\cdots$-

v_1-x_c^j, call $Loop(P', j, zv_{k+1})$. When z misses v_{k+1}, call $FindObstruction3(z, v_k\text{-}\cdots\text{-}v_p, \emptyset)$. When z sees v_{k+1} and misses 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 sees 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 misses v_{k+1} and sees 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 misses v_{k+1} and 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 sees 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 misses 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 sees v_0, v_p and misses v_1 and v_2)

Let r be the smallest integer > 0 such that z sees v_r, the integer r exists because z sees v_p. We have $r \geq 3$ because z misses v_1 and 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 sees v_0, v_p and misses v_1 or v_3)

Case 1 : z misses v_1, misses v_2. Let r be the smallest integer > 0 such that z sees 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 misses v_1, sees v_2, misses v_3. Call $FindObstruction3(z, v_2\text{-}\cdots\text{-}v_p, \emptyset)$.

Case 3 : z misses v_1, sees v_2, sees v_3, misses v_4. Call $FindObstruction4(z, v_2\text{-}\cdots\text{-}v_p, \emptyset)$,

Case 4 : z misses v_1, sees v_2, sees v_3, sees 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 sees v_1. The vertex z misses v_3 because z misses 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 sees v_0, v_p and misses v_1, e is a list which is empty if P has no chord, e contains the chord of P otherwise)

Let r be the smallest integer > 0 such that z sees v_r ($r \geq 2$). When r is odd, return $z, v_0, \ldots, v_r$ which induce an odd hole. When r is even, we consider the different following case :

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 sees 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 misses v_{r+1}, call $FindObstruction3(z$,$v_r\text{-}\cdots\text{-}v_p$,$\emptyset)$. When z sees v_{r+1} and misses 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 sees v_{r+1}, return $z, v_0, \cdots, v_{r+1}$ which induce an odd cycle with only one chord zv_r. When z misses v_{r+1} and sees 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 misses v_{r+1} and 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 sees v_{r+1}, return $z, v_0, \cdots, v_{r+1}$ which induce an odd cycle with only one chord zv_r. When z misses 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 sees v_0, v_1, v_p and misses v_2, e is a list which is empty if P has no chord, e contains the chord of P otherwise)

Let r be the smallest integer > 1 such that z sees 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.

This algorithm can be implemented in linear time.

At each pass in $Loop(P, i)$, i decreases at least by 1, so there are at least k_c passes in this loop. Each pass takes time $\mathcal{O}(deg(x_c^i) + deg(z))$. The vertex x_c^i is different in each loop because i decreases. The vertex z is also different in each loop, 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 $n(z) = j$.

Each call to a *FindObstruction* is made with the same vertex z, so we can compute the adjacency array of z in linear time, the first time a *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 in one of *FindObstruction*, the length of the path P decreases by at least r.

4 Comments

The algorithm presented here is completly different from a recognition algorithm for Meyniel graphs, this algorithm cannot be used to say if a graph is Meyniel or not. It can happen that the input graph is not Meyniel and the output is a clique and a coloring of the same size.

The fastest known recognition algorithm for Meyniel graph, which improves the previous algorithm of Burlet and Fonlupt [1], is due to Roussel and Rusu [10] and its complexity is $\mathcal{O}(m(m+n))$. So it seems that it is 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, Cornuejols, Liu, Seymour and Vuscovic [3] which in fact recognizes Berge graphs (graphs that contains 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, Robertson, Seymour and Thomas [4].

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] M. Chudnovsky, G. Cornuejols, X. Liu, P. Seymour and K. Vuscovic Recognizing Berge graphs, Combinatorica 25 (2005) 143–186.

[4] M. Chudnovsky, N. Robertson, P. Seymour, R. Thomas, The strong perfect graph theorem. Manuscript, Princeton Univ., 2002.

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

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

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

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

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

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

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