Improved Edge-Coloring with Three Colors[☆]

Łukasz Kowalik[a,b]

[a] *Institute of Informatics, University of Warsaw, Banacha 2, 02-097, Warsaw, Poland*
[b] *Max-Planck-Institute für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany*

Abstract

We show an $O(1.344^n) = O(2^{0.427n})$ algorithm for edge-coloring an n-vertex graph using three colors. Our algorithm uses polynomial space. This improves over the previous, $O(2^{n/2})$ algorithm of Beigel and Eppstein [1]. We apply a very natural approach of generating inclusion-maximal matchings of the graph. The time complexity of our algorithm is estimated using the "measure and conquer" technique.

Key words: edge-coloring, exponential-time, algorithm, measure and conquer

1. Introduction

1.1. Problem statement and motivation

In the problem of edge-coloring the input is an undirected graph and the task is to assign colors to the edges so that edges with a common endpoint have different colors. This is one of the most natural graph coloring problems and arises in a variety of scheduling and routing applications (see e.g. [3, 11, 14, 15]).

We consider the problem of verifying whether a given graph G is edge-colorable with k colors and finding such a coloring. Let $\Delta(G)$ denote the maximum degree in graph G. Trivially at least $\Delta(G)$ colors are needed, so if $k < \Delta(G)$ the answer is "no". On the other hand, Vizing [16] proved that when $k \geq \Delta + 1$ the answer is "yes". Unfortunately, when $k = \Delta(G)$ the problem is NP-hard even for $k \geq 3$, as it was shown by Holyer [12]. In this paper we focus on the simplest NP-hard case: $k = 3$.

Note that studying Δ-edge-coloring algorithms makes sense particularly for small values of Δ. Coloring with $\Delta + 1$ colors can be done in polynomial time (e.g. by the algorithm arising from the Vizing's proof). When Δ is large it does not make a big difference whether one uses Δ or $\Delta + 1$ colors. However, for $\Delta = 3$, using four colors when three colors suffice means that the solution is 33% worse than the optimum. (In other words, $(\Delta + 1)$-coloring is a $\frac{\Delta}{\Delta+1}$-approximation.)

Moreover, we suppose that our algorithm may be useful in doing research connected with snarks. A snark is a bridgeless cubic graph which cannot be edge-colored with three colors. Snarks turn out to be the crucial case in several important graph-theoretic

[☆] A preliminary version of this work was presented at WG 2006. The preparation of the current version was supported by a grant from the Polish Ministry of Science and Higher Education, N206 005 32/0807

conjectures, like the 5-flow-conjecture. For some information on snarks see the survey [4]. Brinkmann and Steffen [2] and Cavicchioli et al. [4] used computer programs to generate all snarks of size up to 30; they also verified some claims on the generated graphs. Testing whether a cubic graph is 3-edge-colorable is an important part of these programs.

1.2. Previous results

One way to solve our problem is to apply a vertex-coloring algorithm to the line graph L of G. The currently fastest algorithm for 3-vertex-coloring a given graph L is due to Beigel and Eppstein [1] and it works in $O(1.3289^{|V(L)|})$ time. For $k = 3$, since $\Delta(G) = 3$, the line graph has at most $\frac{3}{2}n$ vertices (n denotes the number of vertices in the input graph G), hence it yields an $O(1.532^n)$ algorithm. However, for 3-edge-coloring Beigel and Eppstein get an $O(2^{n/2}) = O(1.415^n)$-time algorithm by applying nontrivial preprocessing and their algorithm for the $(3, 2)$-CSP problem.

As it was pointed out by Fomin [7], the paper of Fomin and Høie [10] implies an $O(6^{n/6}) = O(1.34801^n)$-time algorithm based on dynamic programming and path decomposition. However, such an algorithm uses exponential space.

1.3. Our result

In this paper we present a 3-edge-coloring algorithm with time complexity $O(1.344^n)$ $= O(2^{0.427n})$. The space complexity of our algorithm is polynomial (even linear). We apply the "measure and conquer" technique. Its basic idea was introduced by Beigel and Eppstein [1], and further developed by Fomin, Grandoni and Kratsch, who recognized its power and wide applicability – they used it in analysis of very simple algorithms for the minimum dominating set [8] and maximum independent set problems [9], obtaining the best known upper bounds on their complexity. In many papers, like e.g. [1], the algorithm consists on identifying one of a large number of possible local configurations in the instance, reducing the configuration in several ways obtaining several smaller instances, and solving the problem in each of them recursively (this is called *branching*). Then the time complexity analysis is rather short and trivial. This situation is reversed in the papers of Fomin, Grandoni and Kratsch [8, 9] and also in an earlier paper of Eppstein [5] on TSP in cubic graphs. In these works, the algorithm performs just a few types of different reductions, while the tedious case analysis is moved to the time complexity proof. Our algorithm follows the same approach.

2. The Algorithm and its Correctness

For the sake of simplicity, we will describe an algorithm for *deciding* whether a given graph is 3-edge-colorable. It is straightforward to extend our description to an algorithm which *finds* a coloring if one exists and which has the same time and space complexity as the decision version. Throughout the paper we will consider only *subcubic* graphs, i.e. graphs with vertices of degree at most three, since any other graph clearly is not 3-edge-colorable.

2.1. Outline

The outline of our algorithm is as follows. Let G be the input graph. Let us call a subcubic graph *semi-cubic* when it has no 1-vertices and each pair of 2-vertices is at distance at least 3. Our algorithm finds a set $\mathcal{A}$ of semi-cubic graphs such that G is 3-edge-colorable if and only if at least one of the graphs in $\mathcal{A}$ is 3-edge-colorable. Additionally, the graphs in $\mathcal{A}$ have no cycles of length smaller than 5. Generating these graphs is done using branching and $\mathcal{A}$ may have exponential size. Then we make use of the following simple fact. A matching M in graph H is called *fitting* when each connected component of $H - M$ is a path or even length cycle.

Proposition 1. *A graph is 3-edge-colorable iff it contains a fitting matching.*

For each of the generated semi-cubic graphs $H \in \mathcal{A}$ the algorithm verifies whether it contains a fitting matching. Again using branching, the algorithm checks a possibly exponential number of (not necessarily fitting) matchings. Then for each of them it can be verified in polynomial time whether it can be completed to a fitting matching.

The intuition behind the above algorithm is as follows. This is an improvement of the natural algorithm which generates all maximal matchings of the input graph, and for each of them verifies (in polynomial time) whether it is fitting. (This is a counterpart of Lawler's 3-vertex-coloring algorithm [13]). Unfortunately, the number of maximal matchings may be large. However, for cubic graphs one can observe that every fitting matching is a perfect matching, and the number of perfect matchings is much lower than the number of maximal matchings. A matching in a subcubic graph will be called *semi-perfect* when every 3-vertex is matched. Clearly, every fitting matching in a subcubic graph (and in particular in a semi-cubic graph) is semi-perfect. Again, in semi-cubic graphs there are much fewer semi-perfect matchings than maximal matchings. The only problem is that the input graph is subcubic but not necessarily semi-cubic. However, it turns out that if a graph G contains a pair of 2-vertices at distance 2 (hence it is not semi-cubic), then there are two graphs such that G is 3-edge-colorable if and only if at least one of the two graphs is edge-colorable, and, what is very important, these two graphs are much smaller than G (i.e. work factor is small – see Section 3.1). Similarly, graphs with no 4-cycles have fewer semi-perfect matchings and one can get rid of these cycles using another small work factor reduction. That is why the set $\mathcal{A}$ is generated. To reduce the time complexity even further we notice that generating all semi-perfect matchings of semi-cubic graphs in $\mathcal{A}$ is not needed. Instead, we generate all matchings with some nice structure so that verifying whether they extend to a fitting semi-perfect matching takes only polynomial time.

2.2. Generating Almost Cubic Graphs

In this section we describe the part of our algorithm which generates a set $\mathcal{A}$ of semi-cubic graphs with neither 3- nor 4-cycles and such that the input graph is 3-edge-colorable iff at least one of the graphs in $\mathcal{A}$ is 3-edge-colorable. This part is implemented as a recursive procedure EdgeColor – see Pseudocode 2.1. For each graph $H \in \mathcal{A}$, the algorithm calls function FittingMatch, described in the next subsection, which verifies whether H contains a fitting matching. Note that the input graph in procedure EdgeColor is allowed to have double and triple edges. This simplifies the correctness proof (one does not need to care about keeping the graph simple during reductions) but also makes our result more general.

Pseudocode 2.1 procedure EdgeColor(G)

Input: subcubic multigraph G with no self-loops.
Output: True if G is 3-edge-colorable, False otherwise.

1: **if** exists $v \in V(G)$ such that $\deg(v) \in \{0, 1\}$ **then**
2:     **return** EdgeColor($G - v$)
3: **else if** exists $uv \in E(G)$ such that $\deg(u) = \deg(v) = 2$ **then**
4:     **return** EdgeColor($G - uv$)
5: **else if** G contains a triple edge uv **then**
6:     **return** EdgeColor($G - \{u, v\}$)
7: **else if** G contains a double edge uv **then**
8:     **if** $\deg(u) = 2$ or $\deg(v) = 2$ **then**
9:         **return** EdgeColor($G - \{u, v\}$)
10:     **else**
11:         Let u_1 (resp. v_1) be the neighbor of u (resp. v) distinct from v (resp. u)
12:         **if** $u_1 = v_1$ **then**
13:             **return** False
14:         **else**
15:             **return** EdgeColor($G - \{u, v\} + u_1v_1$)
16: **else if** exists a 3-cycle C **then**
17:     Let G' be the graph obtained from G by contracting $V(C)$ into one vertex.
18:     **return** EdgeColor(G')
19: **else if** exists a path $xuzvy$ (possibly $x = y$) such that $\deg(u) = \deg(v) = 2$ **then**
20:     $z' \leftarrow$ the neighbor of z distinct from u and v ▷ (Note that $\deg(z) = 3$)
21:     **return** EdgeColor($G - \{z, v\} + uy$) **or** EdgeColor($G - \{u, z, v\} + xz'$)
22: **else if** exists a 4-cycle $C = xyzu$ with $\deg(x) = \deg(y) = \deg(z) = 3$, $\deg(u) = 2$ **then**
23:     Let x' (resp. y', z') be the neighbor of x (resp. y, z) outside the cycle
24:     **return** EdgeColor($G - V(C) + x'y'$) **or** EdgeColor($G - V(C) + z'y'$)
25: **else if** exists a 4-cycle $C = xyzu$ with $\deg(x) = \deg(y) = \deg(z) = \deg(u) = 3$ **then**
26:     Let x' (resp. y', z', u') be the neighbor of x (resp. y, z, u) outside the cycle
27:     **return** EdgeColor($G - V(C) + \{x'y', u'z'\}$) **or** EdgeColor($G - V(C) + \{x'u', y'z'\}$)
28: **else**   ▷ G is semi-cubic and has no 3-, 4-cycles
29:     **return** FittingMatch(G,G,$\emptyset$)

Lemma 1. *Consider an execution of algorithm* EdgeColor *on an input graph G. Let $\mathcal{A}$ be the set of all graphs H such that* FittingMatch(H,H,$\emptyset$) *was executed. Then G is 3-edge-colorable iff at least one graph in $\mathcal{A}$ is 3-edge-colorable.*

Proof. It suffices to prove that whenever procedure EdgeColor(G) performs "**return** EdgeColor(G_1)" / "**return** EdgeColor(G_1) **or** EdgeColor(G_2)" then G_1 / both G_1 and G_2 are subcubic multigraphs with no self-loops, and G is 3-edge-colorable if and only if G_1 is edge colorable / at least one of graphs G_1, G_2 is edge colorable.

For an example consider the case in line 25. Let $G_1 = G - V(C) + \{x'y', u'z'\}$ and $G_2 = G - V(C) + \{x'u', y'z'\}$ (see Figure 1). Graphs G_1 and G_2 may have double edges $x'y'$, $u'z'$, $x'u'$ or $y'z'$. However, in this proof when we refer to these edges we mean the edges added to G after removing $V(C)$. Clearly both G_1 and G_2 are subcubic multigraphs. Also, both G_1 and G_2 have no self-loops since the condition in line 16 was false.

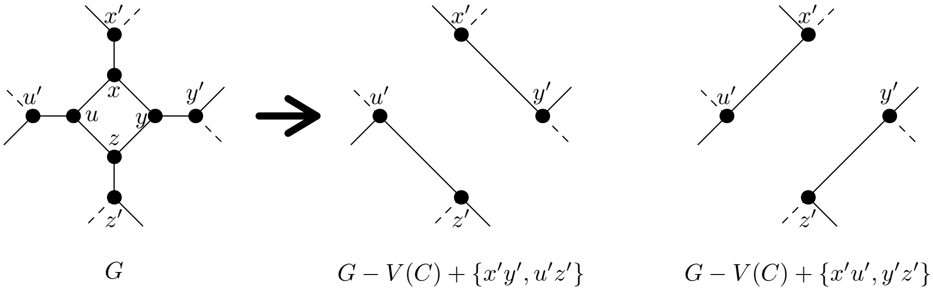


Figure 1: Branching in line 25 of Pseudocode 2.1

Assume there is a 3-edge-coloring of G. We will show that one of G_1, G_2 is 3-edge-colorable. Then cycle C is either colored with two or three colors. In the first case all the four edges xx', yy', zz' and uu' have the same color, say a. Hence one gets a 3-edge-coloring of G_1 by copying the colors of edges in $E(G) \cap E(G_1)$ from G and coloring both edges $x'y'$ and $u'z'$ with color a. In the second case one color appears in $E(C)$ twice, and each of the two other once. By symmetry, we can assume w.l.o.g that edges xy, yz, zu, ux have colors b, a, b, c. Then both yy' and zz' have color c and both uu' and xx' have color a. Hence one gets a 3-edge-coloring of G_2 by copying the colors of edges in $E(G) \cap E(G_2)$ from G, coloring edge $y'z'$ with color c and $u'x'$ with color a.

Now assume G_1 is 3-edge-colorable (by symmetry there is no need for checking G_2 separately). Now we show how to color G. The common edges of G_1 and G inherit their colors from G_1. If edges $x'y'$ and $u'z'$ have the same color, say a, then in G edges xx', yy', zz' and uu' are colored with a, xy and zu with b, and yz and ux with c. Finally assume that $x'y'$ has color a and $u'z'$ has color b. Then xx', yy' and zu are colored with a, while zz', uu' and xy are colored with b, and both xu and yz are colored with c.

The easy proofs of the other cases are left to the reader. $\square$

2.3. Finding a Fitting Matching

In this section we describe a recursive procedure FittingMatch (G_0, G, M). The parameters G and G_0 are simple semi-cubic graphs, and $G \subseteq G_0$. The parameter M is a matching in G_0 such that $V(M) \cap V(G) = \emptyset$ and every 3-vertex in $V(G_0) - V(G)$ is matched. The procedure verifies whether G_0 contains a fitting matching $M' = M \cup N$ such that $N \subseteq E(G)$.

We will use the following auxiliary definitions. Any vertex in G which has degree 3 in G_0 will be called *forced.* A *switch* is a 4-path P in G such that P forms a connected component in G and the two inner vertices of P are forced, while the endvertices are not forced. An edge e in G will be called *allowed* if both of its endvertices are forced and e does not belong to a switch. The *weight* of an edge is the sum of degrees of its endvertices.

Below we give procedure FittingMatch (G_0, G, M). When every connected component of G is a switch it calls procedure SetSwitches, described in the following section.

Pseudocode 2.2 procedure FittingMatch(G_0,G,M)

if every connected component of G is a switch **then**
    return SetSwitches(G_0,G,M)
else if exists a forced vertex $v \in V(G)$ such that $\deg_G(v) = 0$ **then**
    return False
else if exists a non-forced vertex $v \in V(G)$ such that $\deg_G(v) = 0$ **then**
    return FittingMatch(G_0, $G - \{v\}$, M)
else if exists a forced vertex $v \in V(G)$ such that $\deg_G(v) = 1$ **then**
    $u \leftarrow$ the neighbor of v in G
    return FittingMatch(G_0, $G - \{u, v\}$, $M \cup \{uv\}$)
else
    $uv \leftarrow$ any allowed edge in G with the highest weight. ▷ (it exists, see proof of Th. 1)
    return FittingMatch(G_0, $G - \{u, v\}$, $M \cup \{uv\}$) **or** FittingMatch(G_0, $G - uv$, M)

2.4. Setting Switches

In this section we describe a procedure SetSwitches (G_0, G, M) and prove the following lemma.

Lemma 2. *Let G_0 be a simple semi-cubic graph, and let G be a subgraph of G_0 in which each connected component is a switch. Let M be a matching in G_0 such that $V(M) \cap V(G) = \emptyset$ and every 3-vertex in $V(G_0) - V(G)$ is matched. Then procedure* SetSwitches *(G_0, G, M) verifies whether G_0 contains a fitting matching M' such that $M' = M \cup N$ for some $N \subseteq E(G)$.*

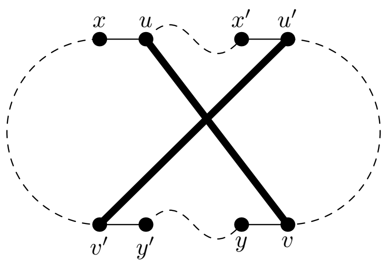


Figure 2: A pair $\{s, s'\}$ of crossing switches, $s = xuvy$, $s' = x'u'v'y'$. Edges of $(E(s) \cup E(s')) \cap M'$ are bold. Cycle C consists of thin edges: the solid edges are from $(E(s) \cup E(s')) - M'$ and the dashed edges are from $E(G_0) - (M' \cup E(s) \cup E(s'))$.

Let us assume that G_0, G and M satisfy the assumptions of the above lemma. Let $s = xuvy$ be a switch and let M' be a semi-perfect matching in G_0 such that $M' = M \cup N$ for some $N \subseteq E(G)$. Observe that either $uv \in M'$ (then we will say that the switch is *closed* in M') or $xu, vy \in M'$ (the switch is *open* in M'). Let $M'' = M' \oplus E(s)$, where $\oplus$ denotes the xor operation. Clearly, then M'' is a semi-perfect matching (recall that $V(M) \cap V(G) = \emptyset$). Also, if s was open in M', it is closed in M'' and vice versa. Recall that $G_0 - M'$ is a collection of paths and cycles. Let C be a cycle in $G_0 - M'$ and let $s = xuvy$ and $s' = x'u'v'y'$ be a pair of switches closed in M' such that $V(s) \subseteq V(C)$ and $V(s') \subseteq V(C)$ and let P be any of the two paths in C between x and y. When P

contains exactly one of the vertices x', y', we will say that the switches s, s' are *crossing* (see Fig. 2). We will say that a set of switches S *improves* matching M' when for matching $M'' = M' \oplus \bigcup_{s \in S} E(s)$ graph $G_0 - M''$ has fewer odd cycles than $G_0 - M'$ or the total length of odd cycles in $G_0 - M''$ is larger than in $G_0 - M'$.

Pseudocode 2.3 procedure SETSWITCHES(G_0,G,M)

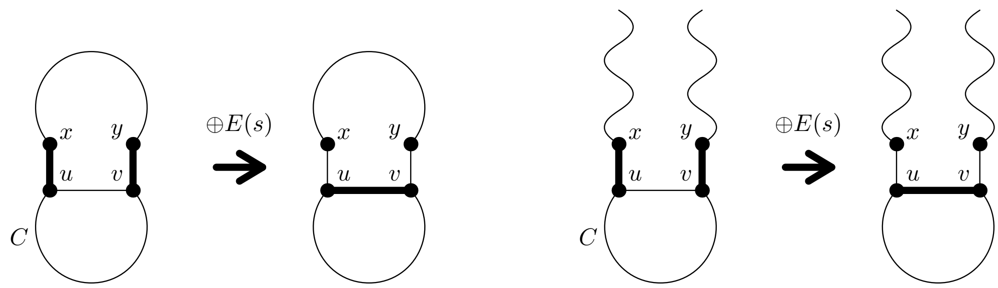
```
1: M' ← M
2: for each switch s = xuvy in G do
3:     M' ← M' ∪ {xu, vy}                          ▷ Set all the switches as open
4: while G_0 − M' contains an odd cycle C do
5:     if there is a switch s or pair of crossing switches s_1, s_2 which improve M' then
6:         M' ← M' ⊕ E(s), (resp. M' ← M' ⊕ (E(s_1) ∪ E(s_2))
7:     else
8:         return FALSE
9: return TRUE
```

Now we are ready to prove Lemma 2.

Proof of Lemma 2. Assume that the procedure returned "TRUE". Note that after the loop in lines 2–3 is performed M' is a semi-perfect matching. Hence $G_0 - M'$ is a collection of paths and cycles. Since the condition in line 4 was eventually not satisfied all the cycles in $G_0 - M'$ are even, hence M' is fitting. The condition that M' is an extension of M using edges of G is also trivially satisfied.

Now it suffices to prove that when the algorithm returns "FALSE", then there is no such a matching in G_0. Since "FALSE" was returned, there is an odd cycle C in $G_0 - M'$ and no switch/pair of crossing switches which improve the current matching M'. Let T be the set of all the switches that touch C, i.e. T contains all switches s such that $V(s) \cap V(C) \neq \emptyset$.

Claim 1: Any switch $s = xuvy$ in T is closed and $V(s) \subseteq V(C)$.



a) b)

Figure 3: Proof of Claim 1: if s is open then s improves M'.

First assume that s is open. Then $x, y \notin V(C)$ since they are of degree 2 in G_0 and they are matched by M'. Hence u or v is in $V(C)$. As $uv \notin M'$ it implies that both u and v are in $V(C)$. The connected component of $G_0 - M'$ containing x is a path, similarly for y. If it is just one path with x and y as endpoints, then either $G_0 - (M' \oplus E(s))$ has 1 odd cycle fewer than $G_0 - M'$ (when the path is of even length) or $G_0 - (M' \oplus E(s))$ has larger total length of odd cycles (when it is odd) than $G_0 - M'$. Hence s improves M', which is a contradiction. Similarly, when those two paths are distinct, $G_0 - (M' \oplus E(s))$ has one odd cycle fewer than $G_0 - M'$ (C transforms into a path), a contradiction again.

Hence assume s is closed. Then $x, u \in V(C)$ or $v, y \in V(C)$. Assume w.l.o.g. $x, u \in V(C)$. Now assume that $v \notin V(C)$. Then the connected component of $G_0 - M'$ containing u (i.e. cycle C) is distinct from the connected component K_v of $G_0 - M'$ containing v. Hence, in $G_0 - [M' \oplus E(s)]$ cycle C is replaced by a path. If K_v is a cycle, then operation $M' \oplus E(s)$ splits it into a path. Also, if K_v is a path, it splits into two paths. Hence the number of odd cycles in $G_0 - [M' \oplus E(s)]$ is smaller than it was in $G_0 - M'$, a contradiction. Hence we are left with the case $v \in V(C)$. Then also $y \in V(C)$. This establishes the claim.

Let $C = x_0 x_1 \ldots x_{|C|-1}$. Let s be a switch in T. By Claim 1, $V(s) \subset V(C)$. We will say that s is *C-shaped* when $s = x_i x_{i+1} x_j x_{j+1}$, for some $i, j \in \{0, \ldots, |C|-1\}$. (From now on indices at x_i are modulo $|C|$). Note that by this definition, if $s = x_{j+1} x_j x_{i+1} x_i$ then s is also C-shaped. (See Fig. 4 — all switches in that figure are C-shaped.) If T contains a switch s which is not C-shaped, i.e. $s = x_i x_{i+1} x_j x_{j-1}$ (or $s = x_i x_{i-1} x_j x_{j+1}$, which is symmetric) then in $M' \oplus E(s)$ cycle C transforms into the path $x_i x_{i-1} \ldots x_j x_{i+1} x_{i+2} \ldots x_{j-1}$, so s improves M', a contradiction. This establishes our next claim.

Claim 2: All switches in T are C-shaped.

Now assume T contains a pair of crossing switches s_1, s_2. By Claims 1 and 2, we can assume that $s_1 = x_i x_{i+1} x_j x_{j+1}$ and $s_2 = x_k x_{k+1} x_l x_{l+1}$ for some $k \in \{i+2, \ldots, j-1\}$, $l \in \{j+2, \ldots, i-1\}$ (the other cases are symmetric). Then, in $M' \oplus (E(s_1) \cup E(s_2))$ cycle C is replaced by two paths, namely $x_{j+1} x_{j+2} \ldots x_l x_{k+1} x_{k+2} \ldots x_j x_{i+1} x_{i+2} \ldots x_k$ and $x_{l+1} x_{l+2} \ldots x_i$. Hence $\{s_1, s_2\}$ improves M', a contradiction which implies:

Claim 3: T does not contain a pair of crossing switches.

Now assume that there exists a fitting matching F in G_0 such that $F = M \cup N$ and $N \subseteq E(G)$, contradicting our lemma. Hence there is a set of switches S such that $M' \oplus \bigcup_{s \in S} E(s) = F$. Observe that $S \cap T \neq \emptyset$, for otherwise $G_0 - F$ contains the odd cycle C. Since by Claim 3 among the switches in T there is no crossing pair, we can enumerate switches in $S \cap T$ from s_1 to $s_{|S \cap T|}$ so that after removing the forced vertices of any switch in $S \cap T$, the cycle C splits into two parts such that one part contains all the switches with smaller numbers and the other with larger numbers (informally, we enumerate the switches from left to right, see Figure 4). Note that Claim 2 implies

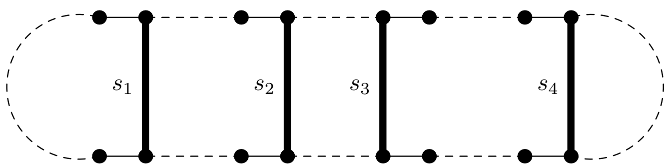


Figure 4: Enumerating switches touching odd cycle C. For $i \in \{1, 2, 4\}$ switch s_i has the cycle $C(s_i)$ succeeding and s_3 has the cycle $C(s_3)$ preceding.

that for each switch s in T, after performing the operation $M' \oplus E(s)$ cycle C splits into a path and a (shorter) odd cycle. Let us denote this resulting shorter cycle by $C(s)$.

Observe that cycle $C(s)$ either contains all the switches preceding s (then $C(s)$ will be called preceding), or all the switches succeeding s (then $C(s)$ will be called succeeding) — see Fig. 4. If every switch s in $T \cap S$ has the cycle $C(s)$ succeeding then the cycle $C(s_{|S\cap T|})$ is odd, and it exists in $G_0 - F$, hence F is not fitting, a contradiction. Let $s_i \in T \cap S$ be a switch with preceding cycle $C(s_i)$ such that for any $j < i$, cycle $C(s_j)$ is succeeding (in the situation from Fig. 4, $i = 3$). If $i = 1$ then the cycle $C(s_1)$ is odd, and it exists in $G_0 - F$, hence F is not fitting, a contradiction. Hence $i > 1$. Let us denote the forced vertices of s_i by u_i, v_i, and the forced vertices of s_{i-1} by u_{i-1}, v_{i-1} in such a way that they appear around cycle C in the order $u_{i-1}, u_i, v_i, v_{i-1}$. Let a, b, c and d denote the length of the path in C joining u_{i-1} with u_i, u_{i-1} with v_{i-1}, v_{i-1} with v_i, and v_i with u_i, respectively (there are always two such paths, but we mean the path which does not contain the other two forced vertices of switches s_{i-1}, s_i). Then $a+b+c+1 \equiv 1 \pmod 2$ and $a + c + d + 1 \equiv 1 \pmod 2$, since the cycles $C(u_i)$ and $C(u_{i-1})$ have odd length. It implies that $b + d \equiv 0 \pmod 2$. As C has odd length, $a + b + c + d \equiv 1 \pmod 2$. Hence $a + c \equiv 1 \pmod 2$ and so $a + c + 2$ is odd. However, $a + c + 2$ is the length of a cycle that appears after performing operation $M' \oplus (E(s_{i-1}) \cup E(s_i))$. This cycle exists also in $G_0 - F$, which is a contradiction. It ends the proof. □

Now we are ready to state the correctness of our coloring algorithm.

Theorem 1. *Algorithm* EdgeColor *correctly verifies whether a given subcubic graph is 3-edge-colorable.*

Proof. By Lemma 1 and Proposition 1 it suffices to show that FittingMatch(G,G,$\emptyset$) correctly verifies whether the semi-cubic graph G has a fitting matching. Using Lemma 2 for the base case it easy to show by induction on $|V(G)|$ that FittingMatch(G_0,G,M), with parameters satisfying assumptions of Lemma 2, verifies whether G_0 contains a fitting matching $M' = M \cup N$ for some $N \subseteq E(G)$. The only unclear issue is why in line 11 graph G must contain an allowed edge. To see this consider any connected component K of G. Let v be any vertex in K. If v is not forced then since the condition in line 5 was false v has at least one neighbor, which must be forced since G_0 is semi-cubic. Hence component K has a forced vertex; let us denote it by w. Since the conditions in lines 3 and 7 were false $\deg_G(w) \geq 2$. Since G_0 is semi-cubic at least one of the neighbors of w, say u, is forced. This proves that every connected component contains an edge with both endpoints forced. The condition in line 1 was false hence in line 11 graph G contains a component which is not a switch so it contains an allowed edge. □

3. Time Complexity Analysis

The aim of this section is to prove the time complexity of algorithm EdgeColor. We start with recalling a standard technique for solving recurrences arising from “branch-and-reduce” algorithms.

3.1. Solving Recurrences

Our algorithm uses the standard “branch-and-reduce” approach. Here we recall what it means and we recall some standard tools for analysis of such algorithms.

In the “branch-and-reduce” approach, the algorithm is recursively applied to the problem instance and uses two types of *rules*. *Reduction rules*, (see e.g. lines 1-18 of

EdgeColor procedure) simplify the instance. *Branching rules* (see e.g. lines 21, 24 of EdgeColor) also simplify the instance, but in several ways, generating several smaller instances in such a way that the initial instance (graph) is 3-edge-colorable iff one of the simplified ones is. Then the problem is solved recursively for each of the smaller instances. Hence execution of procedure EdgeColor is a traversing of a recursion tree, with nodes corresponding to single calls of procedures EdgeColor, FittingMatch and SetSwitches. Reducing rules correspond to nodes with only one child, hence their number is only polynomially larger than the number of nodes corresponding to branching rules. It follows that in order to bound the time complexity up to a polynomial factor it suffices to focus on branching rules. Then for each branching rule we act like it was the only rule in the algorithm. Consider a branching rule generating smaller instances I_1, I_2 (there are never more ones in our algorithm) such that the size of I_1, I_2 is smaller than the initial instance by r_1 and r_2, respectively. This leads to a recurrence of the form $T(s) = T(s - r_1) + T(s - r_2)$, whose solution is the unique positive zero of the function $f(x) = 1 - x^{-r_1} - x^{-r_2}$. This solution will be called (as in [1]) a *work factor* and denoted as $\lambda(r_1, r_2)$. After finding the zeroes of all the functions corresponding to branching rules we choose the largest one, say λ. Then the time complexity of the algorithm is $O(\lambda^n p(n))$ for some polynomial p. The intuition behind it is that in the worst case the "weakest branching rule" may apply in all nodes of the search tree. In this paper we use only numerically obtained (not sharp) upper bounds of the work factors. It follows that we can omit the polynomial factor in time complexity, since $\lambda^n p(n) = O((\lambda + \epsilon)^n)$ for any $\epsilon > 0$.

3.2. Measure and Conquer

We apply the "measure and conquer" approach (see [1, 8, 9]) for estimating the number of nodes of search tree. This approach consists in using a carefully selected measure of the size of an instance of our problem. For example, a natural measure of the instance size in our case is the number of vertices of the graph. However, it is clear that in procedure FittingMatch, a forced vertex of degree 1 should not be counted into the instance size, because it disappears from the graph in polynomial time. We can go further: intuitively, forced 2-vertex x should contribute less to the instance size than a 3-vertex, because there are only two choices: either one or the other edge incident with x belongs to a fitting matching (or, the 2-vertex is closer to becoming a 1-vertex, which does not contribute to the size). This suggests using weights of vertices, and defining the size of the instance as the sum of vertices weights. Clearly, the time complexity (solution of a relevant recurrence) depends heavily on the size measure used. The weights are chosen in a way which minimizes the time complexity, i.e., the largest work factor. This was done by quasi-convex programming algorithm due to Eppstein [6].

3.3. Analysis

Theorem 2. *Algorithm* EdgeColor *works in* $O(1.344^n)$ *time for any* n*-vertex input graph.*

Proof. Let G be the graph passed to procedure EdgeColor or FittingMatch. A vertex in G may be in one of the following states. Either it is *unmarked*, or it is *marked as forced* or it is *marked as unforced*. We assume that in the input graph all the vertices are unmarked, while in the moment of calling procedure FittingMatch, all the vertices of degree 3 are marked as forced and all the other are marked as unforced. Now we define a non-standard measure $s(G)$ of the size of G as the sum of weights of vertices, which are assigned as follows. Forced 2-vertices have weight α and unforced 1-vertices have weight β. Values of these parameters will be adjusted later; at this moment let us merely put a bound $0 \leq \alpha, \beta \leq 1$. Isolated vertices and forced 1-vertices have weight 0. All the remaining vertices — i.e. unmarked vertices, forced 3-vertices and unforced 2-vertices — have weight 1. Note that the size of the instance passed to procedure EdgeColor is simply the number of non-isolated vertices. We observe that when there is no branching, i.e. a procedure calls another procedure just once, the size of the instance does not increase.

Now for each possible branching rule in our algorithm we are going to determine its work factor. Some of the work factors will depend on the values of parameters α and β. After identifying all work factors we will set these parameters in such a way that the largest work factor is minimized.

First we focus on branchings in procedure EdgeColor. There are three of them, in lines 21, 24 and 27. They have the following work factors: $\lambda(2,3) \leq 1.325$ for the first one and $\lambda(4,4) \leq 1.190$ for the two latter ones.

Now we consider the branching rule in procedure FittingMatch (in line 12). This will require more involved analysis. Let uv be the edge picked by the algorithm (recall it is the heaviest allowed edge in G). Assume w.l.o.g. that $\deg(u) \leq \deg(v)$.

Case 1. $\deg(u) = \deg(v) = 3$. Consider the graph with vertices u and v removed. As graph G_0 is semi-cubic at most one neighbor of u is unforced (analogously for v). An unforced neighbor decreases its weight either from 1 to β (when it has degree 2) or from β to 0 (when it is a 1-vertex). A forced 3-neighbor decreases its weight from 1 to α. Any forced 2-neighbor becomes a forced 1-vertex and hence it will be matched with its unique neighbor before the next branching happens. This unique neighbor has weight α, β, or 1. Then both of them are removed from the graph so the size decreases by at least $\alpha + \min\{\alpha, \beta\}$. The last statement is not true when two neighbors of u and v are forced 2-neighbors with a common neighbor, but then the relevant call of FittingMatch returns FALSE in polynomial time (without performing any branching). The other possibilities of counting some weight reduction twice are excluded because of the lack of 3- and 4-cycles. It follows that the size of the instance is reduced by at least $2 + 2\min\{1-\alpha, \alpha + \min\{\alpha,\beta\}\} + 2\min\{\beta, 1-\beta, 1-\alpha, \alpha + \min\{\alpha,\beta\}\}$. On the other hand, when only the edge uv is removed, the size reduces by $2(1-\alpha)$. Hence we get the following upper bound on the work factor: $\lambda(2 + 2\min\{1-\alpha, \alpha + \min\{\alpha,\beta\}\} + 2\min\{\beta, 1-\beta, 1-\alpha, \alpha + \min\{\alpha,\beta\}\}, 2(1-\alpha))$.

Case 2. $\deg(u) < 3$. Since uv was an allowed edge, u is forced. As reduction rules were not applied, $\deg(u) = 2$. Consider the maximal path P of forced 2-vertices containing u. Observe that in each of the two recursive calls all the vertices of P are matched and removed from graph G without branching (in $O(|V(P)|)$ time).

Case 2.1 P is a cycle. If $|V(P)|$ is odd then within $O(|V(P)|)$ steps the function returns FALSE, because there appears a forced 0-vertex. If $|V(P)|$ is even, we can assume that $|V(P)| \geq 6$, since the graph contains no 4-cycles. Then in both of the recursive calls the size is reduced by at least 6α, which corresponds to the work factor $\lambda(6\alpha, 6\alpha)$.

Case 2.2 P is a simple path, $P = v_1 \cdots v_k$. Let x and y be the neighbors of v_1 and v_k

outside P, respectively. Assume w.l.o.g. that $\deg(x) \geq \deg(y)$.

Case 2.2.1. $|V(P)| \geq 3$.

Case 2.2.1.1. $\deg(x) = 1$ and $\deg(y) = 1$. In one of the two recursive calls x is matched with v_1 and v_2 with v_3. These vertices disappear, reducing the size by $\beta + 3\alpha$. The neighbor of v_3 distinct from v_2 is either y (and then it is removed as an unforced 0-vertex) or v_4 (and then it is matched with its other neighbor). This gives us further reduction in size by at least $\min\{\beta, \alpha + \min\{\alpha, \beta\}\}$. In the other recursive call v_1 is matched with v_2, and v_3 is matched with its another neighbor (either v_4 or y). A reasoning similar as before shows that the reduction in size is also $\beta + 3\alpha + \min\{\beta, \alpha + \min\{\alpha, \beta\}\}$. To sum up, this case has work factor $\lambda(\beta + 3\alpha + \min\{\beta, \alpha + \min\{\alpha, \beta\}\}, \beta + 3\alpha + \min\{\beta, \alpha + \min\{\alpha, \beta\}\})$.

Case 2.2.1.2. $\deg(x) \geq 2$. Either x is of degree 2 and then it must be unforced by the definition of P or x is of degree 3 and then it must be forced. In both cases x has weight 1. First consider the recursive call where x is matched with v_1 and v_2 with v_3. Because cycles have length at least 5 neighbors of x are distinct from v_2, v_3. Consider such a neighbor $\tilde{x}$. Then $\tilde{x}$ decreases its weight either by β (if $\tilde{x}$ is an unforced 1-vertex) or by $1 - \beta$ (if $\tilde{x}$ is an unforced 2-vertex) or by $1 - \alpha$ (if $\tilde{x}$ is a forced 3-vertex) or by α (if $\tilde{x}$ is a forced 2-vertex). Hence the size of the instance decreases by at least 1 (for x) plus 3α (for v_1, v_2, v_3) plus $\min\{\beta, 1 - \beta, \alpha, 1 - \alpha\}$ (for $\tilde{x}$).

Now consider the other recursive call, with v_1 matched with v_2 and v_3 matched with its neighbor distinct from v_2, say $\tilde{v}_4$. The weight of x decreases either by $1 - \beta$ (when it is of degree 2) or by $1 - \alpha$ (when it is of degree 3). As before, vertices v_1, v_2, v_3 are matched which causes reduction in size of 3α. If $\tilde{v}_4$ is a forced 2-vertex (i.e. $|V(P)| \geq 4$), we consider its neighbor $\tilde{v}_5$, $\tilde{v}_5 \neq v_3$. If $\tilde{v}_5 \neq x$, the weight of $\tilde{v}_5$ decreases by $\min\{\beta, 1 - \beta, \alpha, 1 - \alpha\}$. If $\tilde{v}_5 = x$, x will be removed without branching and hence x decreases its weight further by α or β. Hence, if $\tilde{v}_4$ is a forced 2-vertex, $\tilde{v}_4$ and its neighbor give further reduction in instance size of at least $\alpha + \min\{\beta, 1 - \beta, \alpha, 1 - \alpha\}$. If $\tilde{v}_4$ is not a forced 2-vertex, it has weight at least $\min\{1, \beta\} = \beta$ (and it is removed from the graph). To sum up, in this recursive call within $O(|V(P)|)$ steps the instance reduces its size by at least $\min\{1 - \beta, 1 - \alpha\} + 3\alpha + \min\{\alpha + \min\{\beta, 1 - \beta, \alpha, 1 - \alpha\}, \beta\}$ without branching.

Let us write down the work factor for this case: $\lambda(1 + 3\alpha + \min\{\beta, \alpha, 1 - \beta, 1 - \alpha\}, \min\{1 - \beta, 1 - \alpha\} + 3\alpha + \min\{\alpha + \min\{\beta, 1 - \beta, \alpha, 1 - \alpha\}, \beta\})$.

Case 2.2.2. $|V(P)| = 2$. Since P is not a part of a switch and there are no forced 1-vertices, at least one vertex of x, y is of degree ≥ 2.

Case 2.2.2.1. $\deg(x) = 2$ and $\deg(y) = 1$. Observe that x is unforced by the definition of P and y is unforced because forced 1-vertices are excluded. It follows that weights of x and y are 1 and β, respectively. It also implies that $\{u, v\} = \{v_1, v_2\}$. Hence allowed edges have both ends of degree 2, for otherwise the algorithm would choose an edge distinct from uv. Let z be the neighbor of x outside P. Vertex z is forced, since G_0 is semi-cubic. Hence $\deg(z) \geq 2$. The neighbors of z distinct from x are also forced, again because G_0 is semi-cubic. It follows that edges joining z and its neighbors distinct from x are allowed. Consequently z has degree 2 and its only neighbor distinct from x, say $\tilde{z}$ is also of degree 2.

In one of the recursive calls before any branching is performed, v_2 is matched with y, v_1 with x and z with $\tilde{z}$, which gives reduction in instance size of $\alpha + \beta + \alpha + 1 + 2\alpha$. In the other recursive call v_1 is matched with v_2, x reduces its weight from 1 to β and y is removed as an unforced 0-vertex, which gives reduction of $2\alpha + (1-\beta) + \beta$. Hence we get the following work factor: $\lambda(4\alpha + \beta + 1, 2\alpha + 1)$.

Case 2.2.2.2. $\deg(x) = 3$ and $\deg(y) = 1$. Then x has weight 1 and y has weight β. Note that $\{u, v\} = \{x, v_1\}$. Hence neither of the neighbors of x is a 3-vertex, for otherwise the algorithm could choose an allowed edge with larger weight. In one of the recursive calls, v_2 is matched with y and v_1 with x. Moreover, both neighbors of x distinct from v_1 reduce their weight. The reduction is from α to 0 if such a neighbor was a forced 2-vertex, from 1 to β if it was an unforced 2-vertex and from β to 0 if it was an unforced 1-vertex. Hence the instance size reduction is $\alpha + \beta + \alpha + 1 + 2\min\{\alpha, 1-\beta, \beta\}$. In the other recursive call, v_1 is matched with v_2, y is removed and x decreases its weight from 1 to α, which gives a size reduction of $2\alpha + \beta + (1-\alpha)$. We get the following work factor: $\lambda(2\alpha + \beta + 1 + 2\min\{\alpha, 1-\beta, \beta\}, \alpha + \beta + 1)$.

Case 2.2.2.3. $\deg(x) \geq 2$ and $\deg(y) \geq 2$. Then neither x nor y is a forced 2-vertex, for otherwise P would be longer. Hence both x and y have weight 1.

First we consider the recursive call where v_1 is matched with x and v_2 with y. Note that either one of x, y is of degree 3 and then it has two neighbors outside P which reduce their weights or both x and y have degree 2, each of them has a neighbor outside P which reduces its weight and these neighbors are distinct, because G_0 is semi-cubic. Hence in both cases there are 2 vertices which reduce their weights. It follows that the total reduction of size in this recursive call is $2+2\alpha+2\min\{\beta, 1-\beta, \alpha, 1-\alpha\}$. In the other recursive call v_1 is matched with v_2 and the weight of both x and y is reduced from 1 to either α or β. It implies that the instance reduces its size by $2\alpha+2\min\{1-\beta, 1-\alpha\}$. We get the following work factor: $\lambda(2+2\alpha+2\min\{\beta, 1-\beta, \alpha, 1-\alpha\}, 2\alpha+2\min\{1-\beta, 1-\alpha\})$.

Case 2.2.3. $|V(P)| = 1$. Since G_0 is semi-cubic and P is maximal, at least one of vertices x and y has degree 3. Hence $\deg(x) = 3$. Let z_1, z_2 be the neighbors of x outside P. Note that w.l.o.g. the edge uv picked by the algorithm is equal to xv_1. Since it is the heaviest edge, $\deg(z_1), \deg(z_2) \leq 2$. As G_0 is semi-cubic, at least one of z_1, z_2 (say, z_2) is forced. Hence $\deg(z_2) = 2$. Let w be the neighbor of z_2 distinct from x. (See Fig. 5).

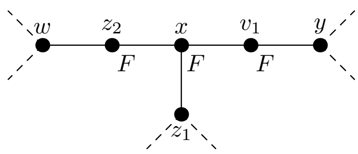


Figure 5: Situation in Case 2.2.3. Vertices z_2, x, v_1 are forced (marked by F), while w, z_1, y may be forced or not. $\deg_G(z_2) = \deg_G(v_1) = 2$, $\deg_G(x) = 3$, while w, z_1, y may have any degree from 1 to 3.

In one of the recursive calls, v_1 is matched with x, z_2 with w and both y and z_1 reduce their weight. Let $\mathrm{red}_1(y)$ denote the reduction of weight of y in this recursive call. Vertex z_1 reduces its weight either from 1 to β (when it is an unforced 2-vertex) or from α to 0 (when it is a forced 2-vertex) or from β to 0 (when it is an unforced 1-vertex). Hence it reduces its weight by at least $\min\{1-\beta, \beta, \alpha\}$. Let $\mathrm{weight}(w)$ denote the weight of w. Then the size of the instance reduces by at least $\alpha + 1 + \alpha + \mathrm{weight}(w) + \mathrm{red}_1(y) + \min\{1-\beta, \beta, \alpha\}$.

In the other recursive call, v_1 is matched with y and x reduces its weight from 1 to α. Let weight(y) denote the weight of y (it is either 1 or β). Then the size of the instance reduces by $\alpha + \text{weight}(y) + (1-\alpha) = \text{weight}(y) + 1$.

Case 2.2.3.1. $\deg(y) = 1$ and $\deg(w) = 1$. Then weight$(y) = \beta$, red$_1(y) = \beta$ and weight$(w) = \beta$. This gives the following work factor: $\lambda(2\alpha+2\beta+1+\min\{1-\beta,\beta,\alpha\}, \beta+1)$.

Case 2.2.3.2. $\deg(y) = 1$ and $\deg(w) \geq 2$. Then weight$(y) = \beta$, red$_1(y) = \beta$ and weight$(w) \in \{1, \alpha\}$. If weight$(w) = \alpha$, i.e. w is a forced 2-vertex, then in the first recursive call considered by us, the neighbor of w distinct from z_2 reduces its weight (by at least $\min\{1-\beta, 1-\alpha, \beta, \alpha\}$). Note that since there are no cycles shorter than 5 and y is of degree 1, this neighbor of w is neither of vertices z_1, v_1, y. Hence in the first recursive call either w reduces its weight from 1 to 0 or w reduces its weight from α to 0 and its neighbor reduces its weight by $\min\{1-\beta, 1-\alpha, \beta, \alpha\}$. In any case we get reduction of at least $\min\{1, \alpha + \min\{1-\beta, 1-\alpha, \beta, \alpha\}\} = \alpha + \min\{1-\beta, 1-\alpha, \beta, \alpha\}$. This gives the following work factor: $\lambda(3\alpha+\beta+1+\min\{1-\beta,\beta,\alpha\}+\min\{1-\beta,1-\alpha,\beta,\alpha\}, \beta+1)$.

Case 2.2.3.3. $\deg(y) \geq 2$. Note that y is not a forced 2-vertex because of the maximality of P. Then weight$(y) = 1$, red$_1(y) \in \{1-\alpha, 1-\beta\}$ and weight$(w) \in \{\alpha, \beta, 1\}$. This gives a work factor: $\lambda(2\alpha+1+\min\{\alpha,\beta,1\}+\min\{1-\alpha,1-\beta\}+\min\{1-\beta,\beta,\alpha\}, 2)$.

We numerically obtained the following values of parameters: $\alpha = 0.39082$ and $\beta = 0.58623$. For these values one can easily check (by finding zeroes of the 11 polynomials corresponding to cases 1 – 2.2.3.3) that the highest work factors correspond to Case 1, Case 2.1 and Case 2.2.1.1. and are bounded by 1.344. This implies that the algorithm works in time $O(1.344^{s(G)})$, where G is the input graph. This settles the theorem, since $s(G) = n$. $\square$

Acknowledgments

The author wishes to thank anonymous referees for careful reading and helpful suggestions.

References

[1] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *J. Algorithms*, 54(2):168–204, February 2005.

[2] G. Brinkmann and E. Steffen. Snarks and reducibility. *Ars Comb.*, 50, 1998.

[3] J. D. Carpinelli and A. Y. Oruc. Applications of matching and edge-coloring algorithms to routing in clos networks. *Networks*, 24(6):319–326, 1994.

[4] A. Cavicchioli, M. Meschiari, B. Ruini, and F. Spaggiari. A survey on snarks and new results: Products, reducibility and a computer search. *Journal of Graph Theory*, 28(2):57–86, 1998.

[5] D. Eppstein. The traveling salesman problem for cubic graphs. In *Proc. 8th Int. Workshop on Algorithms and Data Str. (WADS'03)*, volume 2748 of *LNCS*, pages 307–318, 2003.

[6] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 781–790, 2004.

[7] F. Fomin. Personal communication, 2006.

[8] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: Domination – a case study. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, pages 191–203, 2005.

[9] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 18–25, 2006.

[10] F. Fomin and K. Høie. Pathwidth of cubic graphs and exact algorithms. *Information Processing Letters*, 97(5):191–196, 2006.
[11] C. Gotlieb. The construction of class-teachers time-tables. In *Proc. IFIP Congress '62*, pages 73–77, Amsterdam, 1963. North Holland.
[12] I. Holyer. The np-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.
[13] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, (5):66–67, 1976.
[14] G. F. Lev, N. Pippenger, and L. G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Trans. Computers*, 30(2):93–100, 1981.
[15] S.-I. Nakano, X. Zhou, and T. Nishizeki. Edge-coloring algorithms. In J. van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 1995.
[16] V. G. Vizing. On the estimate of the chromatic class of a p-graph. *Diskret. Analiz*, 3:25–30, 1964.