Les Cahiers du GERAD ISSN: 0711–2440

Online algorithms for the maximum k-colorable subgraph problem

A. Hertz, R. Montagné,
F. Gagnon

G–2016–111

November 2016

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Online algorithms for the maximum k-colorable subgraph problem

Alain Hertz [a]
Romain Montagné [a]
Francçois Gagnon [b]

[a] *GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

[b] *Department of Electrical Engineering, École de Technologie Supérieure, Montréal (Québec) Canada, H3C 1K3*

alain.hertz@gerad.ca
exemple@gerad.ca
francois.gagnon@etsmtl.ca

November 2016

Les Cahiers du GERAD
G–2016–111

Abstract: The maximum k-colorable subgraph problem (k-MCSP) is to color as many vertices as possible with at most k colors, such that no two adjacent vertices share the same color. We consider online algorithms for this $\mathcal{NP}$-hard problem, and give bounds on their competitive ratio. We then consider a large family $\mathcal{A}$ of online sequential coloring algorithms and determine the smallest graphs for which no algorithm in $\mathcal{A}$ can produce an optimal solution to the k-MCSP. We then compare the performance of several online sequential coloring algorithms, using DIMACS benchmark instances. We finally consider the case where vertices colored at an early stage can receive a new color later on, as long as they remain colored.

Keywords: Online algorithms, maximum k-colorable subgraph, sequential vertex coloring, competitive analysis

Résumé : Le problème de la détermination du plus grand sous-graphe k-colorable (k-MCSP) consiste à colorer autant de sommets que possible avec au plus k couleurs, de telle sorte que les sommets adjacents n'aient pas la même couleur. Nous considérons des algorithmes en ligne (online) pour ce problème NP-dur, et nous donnons des bornes sur leur ratio de compétitivité. Nous considérons ensuite une famille $\mathcal{A}$ d'algorithmes de coloration séquentielle en ligne, et nous déterminons les plus petits graphes pour lesquels aucun algorithme de la famille $\mathcal{A}$ n'est capable de générer une solution optimale au problème k-MCSP. Nous comparons ensuite les performances de plusieurs algorithmes en ligne de coloration séquentielle, en utilisant les graphes de la banque de données DIMACS. Finalement, nous considérons le cas où les sommets colorés d'une certaine étape peuvent recevoir une nouvelle couleur plus tard, pour autant qu'ils demeurent colorés.

Mots clés : Algorithmes en ligne, sous-graphe k-colorable, algorithmes de coloration séquentielle, analyse de compétitivité

1 Introduction

Given an undirected graph G, the vertex coloring problem is to color the vertices of G with as few colors as possible, so that no two adjacent vertices of G share the same color. The chromatic number $\chi(G)$ of G is the minimum number of colors used in a vertex coloring of G. A stable set is a set of pairwise non adjacent vertices. Hence, a vertex coloring of G is a partition of its vertex set into stable sets. In this paper, we study the maximum k-colorable subgraph problem (k-MCSP) which is to color as many vertices as possible with at most k colors. The number of vertices in such a maximum k-colorable subgraph is denoted $\alpha_k(G)$. For $k = 1$, the k-MCSP is to find a stable set with as many vertices as possible. Also, the chromatic number of a graph G is the smallest integer k such that the maximum k-colorable subgraph of G is G itself.

Lewis and Yannakakis [1] showed that the maximum subgraph problem for hereditary properties (fulfilling certain requirements) is $\mathcal{NP}$-hard. If we define the hereditary property as ?the graph is k-colorable?, we obtain the k-MCSP. Hence, the k-MCSP is $\mathcal{NP}$-hard for any fixed k, $1 \leq k < n$. Moreover, Lund and Yannakakis [2] showed that such maximum subgraph problems for hereditary properties cannot be approximated within n^ϵ for some $\epsilon > 0$ (depending on the property) unless $\mathcal{P} = \mathcal{NP}$. Hence, the k-MCSP is hard to approximate for any fixed k, $1 \leq k < n$.

The k-MCSP was the subject of the PhD thesis of Narasimhan [3]. He showed, among other results, that for $k = 2$, the problem can be solved in polynomial time on various classes of graphs like interval graphs, circular-arc graphs and tolerance graphs.

More recently, a weighted version (where weights are on the vertices) of the k-MCSP was studied in [4], where symmetry breaking techniques are proposed for integer programming formulations of the problem. In [5], it is shown that the weighted version of the k-MCSP coupled with improper colorings (where vertices can share the same color as some neighbors, to some extent) can model the problem of determining the maximum number of simultaneously communicating mobiles in a wireless network.

In this paper, we address the online version of the k-MCSP, where vertices are revealed by clusters, or one by one. When new vertices are revealed, only the edges from this vertex to already known vertices are revealed. An online coloring algorithm has to irrevocably assign a color to the vertices as soon as they are revealed.

Online network optimization has been studied extensively. A summary of different applications can be found in [6]. Properties of online colorings are investigated, for example, in [7, 8]. The specific case with $k = 1$ (that is, the online version of the maximum stable set problem) has been analyzed in [9], where competitive analysis is detailed thoroughly for various online scenarios with vertices and edges arriving and disappearing iteratively by clusters or one by one. In particular, it is proven that no online algorithm can guarantee a competitive ratio better than $\frac{1}{n-1}$ (where n is the number of vertices in G) if vertices appear sequentially, and $\frac{1}{\sqrt{n}-1}$ if they appear by clusters. We will generalize these results in Section 2.

A similar problem was studied in [10] where the edges (instead of the vertices) of G have to be colored so that no two incident edges share the same color. The problem is then to determine a partial subgraph of G with as many edges as possible, and such that its edges can be colored using at most k colors. The authors have shown that no online algorithm can guarantee a better competitive ratio than $\frac{4}{5}$ for the 2-edge coloring of paths. Since coloring the edges of G is equivalent to coloring the vertices of its line graph $L(G)$, determining a 2-edge coloring of a path with m edges is equivalent to finding a 2-vertex coloring of a path with m vertices. It follows that the competitive ratio $\frac{4}{5}$ also holds for the 2-MCSP on paths.

The rest of the paper is organized as follows. In Section 2, we analyze the competitive ratio of online coloring algorithms. Then, in Section 3, we consider a large family of such algorithms and determine the smallest graphs for which no online algorithm in this family can produce an optimal solution to the k-MCSP. We then consider, in Section 4, several online coloring strategies, while Section 5 is devoted to a variant of the problem where it is permitted to modify the color of already colored vertices. Computational experiments are reported in Section 6, where all proposed algorithms are compared and analyzed using DIMACS benchmark instances [11].

2 Competitive analysis

Online algorithms must satisfy an unpredictable sequence of requests, completing each request without being able to see the future. Competitive analysis is a method for analyzing such algorithms. Roughly speaking, the performance of an online algorithm is compared to the performance of an optimal offline algorithm that can view the sequence of requests in advance.

Definition 1 *Consider an instance I of a maximization problem among the set $\mathcal{I}$ of all possible instances. Let $OPT(I) \geq 0$ be the optimal solution value for I. An algorithm A, that produces a solution of value $A(I)$, is said to guarantee $f(I)$ if $f(I)$ is a function with values lying in $[0,1]$ such that*

$$\frac{A(I)}{OPT(I)} \geq f(I) \quad \forall I \in \mathcal{I}.$$

The competitive ratio r_A of A is defined as $r_A = \min_{I \in \mathcal{I}} \left\{ \frac{A(I)}{OPT(I)} \right\}$.

In online competitive analysis, $OPT(I)$ is the optimal solution of the problem in its offline version. The above definition indicates that an algorithm is competitive if its competitive ratio is bounded. Unlike traditional worst-case analysis, where the performance of an algorithm is measured only for hard inputs, competitive analysis requires that an algorithm performs well both on hard and easy instances. We now determine bounds on the competitive ratio of online algorithms for the k-MCSP. From now on, n denotes the number of vertices in G.

Proposition 1 *If $n > 1$ and the vertices of G are revealed one by one, then no online algorithm for the k-MCSP has a competitive ratio strictly larger than $\frac{k}{n-1}$.*

Proof. Consider the two-person game, where one of the player is the online algorithm, while the other player, called *spoiler*, reveals the vertices and the edges of the graph. The game is played in rounds. During each round, the spoiler reveals a new vertex v and the edges linking it to previously revealed vertices. The online algorithm either decides to color v with one of the k available colors (if any) that is not yet assigned to the revealed neighbors of v, or it can decide not to color v. It is now sufficient to prove that the spoiler has a strategy that guarantees the above upper bound on the competitive ratio. Such a strategy simply consists in connecting new revealed vertices to all already colored vertices. Indeed, if the online algorithm does not color any vertex, then the bound is clearly valid since $\frac{A(G)}{\alpha_k(G)} = 0 < \frac{k}{n-1}$. So assume the online algorithm colors a set $\{v_1, \ldots, v_r\}$ of vertices ($r \geq 1$), v_i being revealed before v_j if $i < j$. Every time a vertex v_i is colored, the next revealed ones cannot use the same color as v_i since the spoiler connects them to v_i. Hence, $A(G) = r \leq k$. Now, let c_i ($1 \leq i \leq r$) be the color assigned to v_i by the online algorithm. By assigning color c_i to v_i ($i = 1, \ldots, r-1$), and color c_r to all other vertices of G, except v_r, one gets a coloring of $n-1$ vertices of G with only r colors, which means that $\alpha_k(G) \geq n-1$. In summary, $\frac{A(G)}{\alpha_k(G)} \leq \frac{k}{n-1}$. □

The above Proposition shows that the best competitive ratio for an online algorithm is in $\mathcal{O}(\frac{k}{n})$. It is not difficult to design online algorithms that achieve such an asymptotic competitive ratio. One can for example color the k first revealed vertices with a different color. It is then obvious that such an algorithm has a competitive ratio larger or equal to $\frac{k}{n}$.

Assume now $k \leq \lfloor \sqrt{n} \rfloor - 1$, and suppose that the vertices (and their incident edges) are revealed by clusters that possibly contain more than one vertex. The upper bound on the competitive ratio is then larger, as shown in the following Proposition.

Proposition 2 *If $k \leq \sqrt{n} - 1$, $n > 1$, and the vertices of G are revealed by clusters, then no online algorithm for the k-MCSP has a competitive ratio strictly larger than $\frac{k}{\sqrt{n}-1}$.*

Proof. We again consider the two-person game, where one of the players is the online algorithm, while the other player is the spoiler that reveals the vertices and the edges of the graph. Let r denote the number of already revealed vertices. During each round, the spoiler reveals a new set of $\min\{n-r, \lfloor\sqrt{n}\rfloor\}$ new vertices, and the online algorithm colors any subset of them (possibly an empty one) with the k available colors, as long as no vertex gets the same color as one of its neighbors. The spoiler's strategy simply consists of connecting the new revealed vertices to all already colored vertices (while they are pairwise non-adjacent). If the online algorithm does not color any vertex of G, then $\frac{A(G)}{\alpha_k(G)} = 0 < \frac{k}{\sqrt{n}-1}$. So assume the online algorithm colors at least one vertex. When a vertex v is colored, the next revealed ones and those revealed in previous clusters cannot have the same color as v since they are all linked to v. Hence, no color can be used more than $\lfloor\sqrt{n}\rfloor$ times, which means that $A(G) \leq k\lfloor\sqrt{n}\rfloor$. Now, let c be the coloring produced by the online algorithm, let C_i $(1 \leq i \leq k)$ be the color classes, and let W denote the set of non-colored vertices. Also, let v the last vertex colored by the online algorithm, and let j be its color (i.e., $v \in C_j$). Consider the coloring c' obtained from c by removing color j from all vertices in C_j, and by assigning color j to all vertices in W. The resulting coloring is clearly feasible and colors at least $n - \lfloor\sqrt{n}\rfloor$ vertices. Hence, $\alpha_k(G) \geq n - \lfloor\sqrt{n}\rfloor$, and we therefore have

$$\frac{A(G)}{\alpha_k(G)} \leq \frac{k\lfloor\sqrt{n}\rfloor}{n - \lfloor\sqrt{n}\rfloor} = \frac{k}{\frac{n}{\lfloor\sqrt{n}\rfloor} - 1} \leq \frac{k}{\sqrt{n}-1}.$$

□

Suppose now that revealed vertices do not have to be colored immediately, and that a penalty has to be paid for late colorings. More precisely, let $p \geq 1$ be a real number and let p^{i-j} be the profit of coloring a vertex at iteration j if it was revealed at iteration $i \leq j$. The problem to be solved is then to determine a coloring with maximum total profit. For example, if $p = 2$, the profit of coloring a vertex v one iteration after it was revealed is $\frac{1}{2}$, while it is equal to 1 if v is colored immediately.

Proposition 3 *If $n > 1$ and the vertices of G are revealed one by one, then no online algorithm with possible late colorings has a competitive ratio strictly larger than $\frac{k+\frac{n-k-1}{p}}{n-1}$.*

Proof. We consider again the two-person game of Proposition 1, but with the two following differences: the online algorithm can wait before deciding to color a vertex, and the spoiler links new revealed vertices only to those that were colored at the iteration at which they were revealed.

If the online algorithm never colors a vertex immediately when it is revealed, then the total profit is at most np^{-1} while its maximum value is n (since they are no edges in G). Hence, the competitive ratio is bounded by

$$\frac{\frac{n}{p}}{n} = \frac{1}{p} \leq \frac{1 + \frac{k(p-1)}{n-1}}{p} = \frac{kp + n - k - 1}{p(n-1)} = \frac{k + \frac{n-k-1}{p}}{n-1}.$$

So, assume the online algorithm colors at least one vertex immediately when it is revealed. Let c be the obtained coloring with color classes C_i $(1 \leq i \leq k)$, and let W be the set of non-colored vertices.

- If $W = \emptyset$, then the total profit is at most $k + (n-k)p^{-1}$ while its maximum value is n (since all vertices are colored). We therefore have the following guarantee ratio:

$$\frac{k + \frac{n-k}{p}}{n} = \frac{(n-1)(k + \frac{n-k}{p})}{n(n-1)} = \frac{k + \frac{n-k}{p}}{n-1} - \frac{(k - \frac{k}{p}) + \frac{n}{p}}{n(n-1)} \leq \frac{k + \frac{n-k}{p}}{n-1} - \frac{\frac{1}{p}}{n-1} = \frac{k + \frac{n-k-1}{p}}{n-1}.$$

- If $W \neq \emptyset$, then the total profit is at most $k + (n-k-1)p^{-1}$ while its maximum value is at least $n-1$. Indeed, let v be the last vertex colored immediately when it was revealed, and let j be its color (i.e., $v \in C_j$). Consider the coloring c' obtained from c by removing color j from v, and by assigning color j not only to the vertices in $C_j \setminus \{v\}$, but also to all vertices in W. The resulting coloring is clearly feasible and colors at least $n-1$ vertices. Hence, the maximum profit it at least $n-1$, which proves the upper bound on the competitive ratio.

□

Note that if p tends to ∞, then the ratio $\frac{k}{n-1}$ of Proposition 1 is reached, while if $p = 1$ (if the waiting time is not penalized at all), the algorithm can be considered as offline and the ratio equals 1. We finally prove a lower bound on the competitive ratio for the *greedy* online algorithm. Such an algorithm colors every revealed vertex v with an integer in $\{1, \ldots, k\}$ (if any) that does not appear on an already revealed vertex adjacent to v. If no such color exists, then v is not colored.

Proposition 4 *Given any graph G, the greedy online algorithm has a competitive ratio at least equal to $\frac{k}{\Delta(G)+k}$, where $\Delta(G)$ is the maximum degree in G.*

Proof. Let $A(G)$ denote the number of vertices colored by the greedy online algorithm. Since every uncolored vertex has at least k colored neighbors, we know that the number $n - A(G)$ of uncolored vertices is at most equal to $\frac{\Delta(G)A(G)}{k}$. Hence, $A(G)$ is at least equal to $\frac{n}{1+\frac{\Delta(G)}{k}} = \frac{kn}{\Delta(G)+k}$. Since $\alpha_k(G)$ is at most equal to n, we get

$$\frac{A(G)}{\alpha_k(G)} \geq \frac{\frac{nk}{\Delta(G)+k}}{n} = \frac{k}{\Delta(G)+k}.$$

□

3 Small hard instances for the k-MCSP

Competitive ratios fail to describe the behavior of algorithms in the average case. Factual evidence of this is that many well known coloring heuristics such as Largest-First [12], Dsatur [13], and Recursive-Largest-First [14] have a competitive ratio in $\mathcal{O}(n)$, while numerical experiments have shown that RLF is statistically more efficient than the two other algorithms (see for example [15]).

Hansen and Kuplinsky [16] have partially overcome this insufficiency by studying *hard-to-color* graphs. In what follows, we consider *partially specified algorithms*, as was done in [16]. One such example is the Largest-First heuristic. To apply this algorithm to a graph, one has to order the vertices of the graph according to non-increasing degrees. An implementation of the algorithm must specify how ties in the ordering should be broken. By an *instance* of this algorithm we understand any coloring resulting from the application of this rule to a specific graph, with the only restriction that the vertices should be ordered as stated above. A graph is then defined as *hard-to-color* for a given algorithm if any instance of this algorithm is a non-optimal coloring. The analysis of such graphs is interesting, since this might lead to the design of better heuristics, and to the recognition of new classes of graphs colorable optimally in polynomial time.

In the same vein, we consider here partially specified algorithms for the k-MCSP and an instance of such an algorithm is any partial coloring resulting from the application of this algorithm. Hard-k-MCSP graphs are then defined as follows

Definition 2 *Given an algorithm A for the k-MCSP, a graph is hard-k-MCSP for A if any instance of A is a non-optimal solution.*

The *sequential algorithm* colors the vertices sequentially, and assigns to every vertex v the smallest color in $\{1, \ldots, k\}$ that does not appear on a neighbor of v. If no such color exists for a vertex v, then v is not colored. Let c be an optimal solution to the k-MCSP for a graph G, and let C_i $(i = 1, \ldots, k)$ be the color classes. If the sequential algorithm considers all vertices in C_i before those in C_{i+1} $(i = 1, \ldots, k-1)$, it clearly produces an optimal solution, which proves that there are no hard-k-MCSP graphs for the sequential algorithm.

We consider here the partially specified algorithm $\mathcal{A}$ that works as the basic sequential one, but with the following additional constraint : when a color is used for the first time, the vertex that receives this color should be of maximum degree among the uncolored vertices to which the algorithm would have assigned the same color.

The above constraint is flexible enough to include many famous algorithms. In particular, the k-LF, k-Dsatur, and k-RLF algorithms described here below are all particular cases of $\mathcal{A}$. They are adaptations of the famous Largest First (LF), Dsatur, and Recursive-Largest-First (RLF) algorithms mentioned above for the classical vertex coloring problem. The difference between the proposed algorithms and the original ones is that the vertices that should receive color $k+1$ are left uncolored, the aim being to color as many vertices as possible with the k available colors, instead of minimizing the total number of colors used.

The three algorithms differ in the order in which the vertices are considered for the sequential coloring.

- k-LF orders the vertices by non-increasing degrees.
- The order used by k-Dsatur is a dynamic one based on the *saturation degree* of the vertices which is defined as the number of different colors appearing on their neighbors. The next vertex to color is then one with highest saturation degree, and if several vertices maximize this value, the algorithm chooses one with maximum degree.
- The k-RLF algorithm builds a sequence $V_1, \cdots, V_k$ of color classes. Consider the construction of one color class V_ℓ, let U denote the set of uncolored vertices, and let W be the set (initially empty) of uncolored vertices with at least one neighbor in V_ℓ. Every time a vertex in U is chosen to be moved to V_ℓ, all its neighbors in U are moved from U to W. The first vertex $v \in U$ to be included in V_ℓ is the uncolored one with largest degree. The rest of V_ℓ is built as follows : while U is not empty, the next vertex to be moved from U to V_ℓ is one having the largest number of neighbors in W. Ties are, if possible, broken by choosing a vertex with the smallest number of neighbors in U.

Let $H_{k,r}$ be the graph obtained by linking all vertices of a clique on k vertices to all vertices of a stable set on r vertices. We now show that these graphs (for $r \geq 2$) are hard-k-MCSP for $\mathcal{A}$ (hence, in particular, for k-LF, k-Dsatur, and k-RLF).

Proposition 5 *Given any integer $r \geq 2$, $H_{k,r}$ is a hard-k-MCSP graph for $\mathcal{A}$.*

Proof. Since the vertices of the clique have degree $k+r-1$, while those of the stable set have degree k, $\mathcal{A}$ first colors a vertex of the clique. Since such a vertex is adjacent to all other vertices of $H_{k,r}$, no other vertex will receive the same color. The next vertex that will be colored is therefore another one from the clique, and this process is repeated until all vertices of the clique are colored. Since no vertex of the stable set can be colored (all of them being adjacent to the k vertices of the clique), $\mathcal{A}$ colors only k vertices, which is not optimal since $\alpha_k(H_{k,r}) = k+r-1$. Indeed, it is possible to color the vertices of the stable set with one of the k colors, and $k-1$ vertices of the clique with the $k-1$ other colors. □

The above Proposition shows that in the two person game of the last section, if the spoiler reveals the vertices one by one, in the same order as $\mathcal{A}$ would color them, and if the online algorithm assigns the smallest integer in $\{1, \ldots, k\}$ that does not appear on the neighbors of the revealed vertex, then the solution produced by the online algorithm is not optimal, and the competitive ratio $\frac{k}{n-1}$ of Proposition 1 is reached. We now show that $H_{k,2}$ is the smallest hard-k-MCSP graph for $\mathcal{A}$.

Proposition 6 *Every hard-k-MCSP graph for $\mathcal{A}$ contains at least $k+2$ vertices, and $H_{k,2}$ is the only one with $k+2$ vertices.*

Proof. Clearly, a hard-k-MSCP graph for $\mathcal{A}$ contains at least $k+1$ vertices, else $\mathcal{A}$ would color all vertices, which is an optimal solution. So assume G is a hard-k-MSCP graph for $\mathcal{A}$ with $k+1$ vertices. This means that $\mathcal{A}$ produces a coloring of k vertices of G, while the optimal solution is to color all $k+1$ vertices of G. Hence, one vertex of G, call it v, could not be colored by $\mathcal{A}$, which means that v is colored last, and is adjacent to the k other vertices of G, each one having a different color. It then follows from the coloring rules that G is a clique, which contradicts the fact that $k+1$ vertices are colored in an optimal solution.

It remains to prove that $H_{k,2}$ is the only hard-k-MCSP graph for $\mathcal{A}$ with $k+2$ vertices. For $k=1$, this is clearly true since $H_{1,2}$ (which is a chain on three vertices) is the only graph with three vertices for which $\mathcal{A}$ is

not optimal: $\mathcal{A}$ colors the middle vertex of the chain, while the optimal solution is to color the two endpoints. So assume $k \geq 2$ and that the result is valid for any number $k' < k$ of colors. Let G be a hard-k-MCSP graph for $\mathcal{A}$ with $k+2$ vertices, and let $\mathcal{A}(G)$ denote the number of vertices colored by $\mathcal{A}$.

Case 1 : G contains at least one vertex v of degree $k+1$. Such a vertex is the first one colored by $\mathcal{A}$, it receives color 1, and no other vertex can receive color 1. So, let G' be the graph obtained by removing v from G. We then have $\mathcal{A}(G') = \mathcal{A}(G) - 1 < \alpha_k(G) - 1 \leq \alpha_k(G')$, which means that G' is a hard-$(k-1)$-MCSP graph for $\mathcal{A}$ with $k+1$ vertices, and it follows from the induction hypothesis that G' is isomorphic to $H_{k-1,2}$. Hence, G (which is obtained from G' by linking v to all vertices of G') is isomorphic to $H_{k,2}$.

Case 2 : the maximum degree in G is at most k. Since $\mathcal{A}(G) < \alpha_k(G)$, at least one vertex, call it v, is left uncolored by $\mathcal{A}$. Hence, v is adjacent to at least one vertex of each color $1, \ldots, k$. It follows that v has degree k, which means that the first vertex colored by $\mathcal{A}$, call it w, also has degree k. Therefore, one vertex in G, call it u, is not adjacent to w and necessarily receives color 1. Hence $\mathcal{A}(G) \geq k+1$, and the vertices of G, except u, v, w receive color $2, \ldots, k$ and are all adjacent to v. In summary, w, v and these $k-1$ vertices form a clique with $k+1$ vertices, which implies $\alpha_k(G) \leq k+1 \leq \mathcal{A}(G)$, a contradiction. □

4 Coloring strategies

When the vertices are revealed by clusters with more than one vertex, one has to decide in which order the new revealed vertices are colored. We will use the same rules as those in k-LF or k-Dsatur. Hence, new revealed vertices are either ordered by non-increasing degree, or by non-increasing saturation degree.

When coloring a vertex v, the most natural strategy is to assign the smallest available color in $\{1, \ldots, k\}$, if any, that does not appear on a neighbor of v. This strategy, called *First-Fit* [7, 8] is the one used by the partially specified algorithm $\mathcal{A}$ of the previous section. It may create unbalanced color classes since the first colors are preferred to the last ones. Another possible strategy, called *Next-Fit*, it to choose a color according to the last used: if the last colored vertex received color c, the next one will receive the first available color in the ordered sequence $(c+1, c+2, \ldots, k, 1, 2, \ldots, c)$. Such a coloring strategy is shown to have a competitive ratio of $2\sqrt{3} - 3$ for k-edge coloring [10]. Given any graph G, it is interesting to observe that while the First-Fit strategy guarantees that the number of colored vertices increases with k (if the order of the vertices is not changed), this is not necessarily the case with the Next-Fit strategy. This is illustrated in Figure 1 where the vertices are considered in the order $v_1, \ldots, v_6$ and colored using the Next-Fit strategy: while six vertices are colored with $k = 2$, only five of them are colored with $k = 3$.

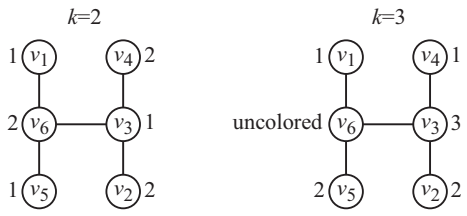

Figure 1: 2 and 3-colorings with the Next-Fit strategy

In addition to First-Fit and Next-Fit, one can also consider the following *Best-Fit* strategy proposed in [17]. For a revealed vertex v, let C_v denote the cluster of vertices revealed at the same time as v, and let N_v be the set of not yet colored vertices in C_v that are adjacent to v. The color chosen for v by the Best-Fit strategy is a color that does not appear on its colored neighbors, and which appears in the neighborhood of as many vertices in N_v as possible. Ties are broken by choosing the smallest color in $\{1, \ldots, k\}$. For the example of Figure 2, assuming $k = 5$, Best-Fit would choose color 2. Indeed, colors 1 cannot be assigned to v, colors 3 and 5 appear in the neighborhood of only one vertex in N_v, while colors 2 and 4 appear in the neighborhood of two vertices in N_v.

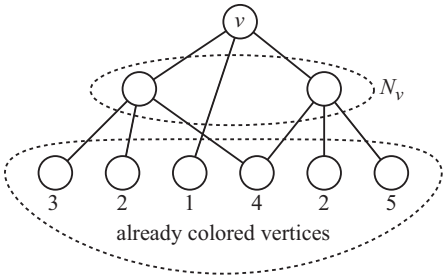

Figure 2: Illustration of the Best-Fit strategy

Best-Fit has similarities with k-RLF since both try to assign the same color to vertices having a large number of common neighbors. Note that if the vertices are revealed one by one, then Best-Fit is identical to First-Fit (since $N_v = \emptyset$).

In what follows, we will test the following online algorithms: when the vertices are revealed one by one, we either use First-Fit or Next-Fit as coloring strategy; when the vertices are revealed by clusters having more than one vertex, we use k-LF or k-Dsatur for ordering new revealed vertices, and First-Fit, Next-Fit or Best-Fit for choosing the color to assign to them.

5 Recoloring strategies

In this section, we consider the possibility of modifying the color of vertices revealed and colored in previous clusters. We try to minimize color changes, but accept them if they allow coloring new revealed vertices. In other words, while the decision to color or not a revealed vertex is irrevocable, the choice of color can be modified. In practice, color changes correspond to new channels assigned to communicating users of a wireless network. These recolorings are made with two different strategies described in the next subsections.

5.1 Sequential recoloring

When a new cluster of vertices is revealed, we first try to color these vertices as explained in the previous section, using k-LF or k-Dsatur coupled with First-Fit, Next-Fit, or Best-Fit. Let W be the set of vertices revealed in previous clusters, and let C be its subset of colored vertices. Also, let R be the new set of revealed vertices. We thus obtain a coloring c_1 where the vertices of C and a subset R' of R are colored. If at least one new revealed vertex is not colored (i.e., $R' \neq R$), we try to generate a coloring with more colored vertices, but with the constraint that all vertices in C must remain colored. This is done by using again k-LF or k-Dsatur coupled with First-Fit, Next-Fit, or Best-Fit, but starting from scratch (i.e. with no colored vertex in $W \cup R$), and with two small differences :

- If ties must be broken for choosing the next vertex to be colored, we give preference to a vertex in C, the reason being that we want to maximize the chance that all vertices in C are colored.
- If color $c_1(v)$ is available when coloring a vertex of C, we give that color to v instead of the one that would be chosen by First-Fit, Next-Fit, or Best-Fit, the reason being that we try to minimize the number of color changes.

If no color in $\{1, \ldots, k\}$ can be assigned to a vertex of C, we stop the recoloring process, keep c_1, and wait for a new cluster of revealed vertices. Otherwise, we obtain a coloring c of all vertices in C as well as some other vertices in $W \cup R$. If more vertices are colored with c than with c_1, we permute the color classes of c to minimize the number of color changes of the vertices in C. This is done as follows. Consider the weighted complete bipartite graph $B = (V_1, V_2, E)$ with $V_1 = V_2 = \{1, \ldots, k\}$, and where the weight of the edge linking a vertex $i \in V_1$ to a vertex $j \in V_2$ is equal to the number of vertices in C having color i in c and color j in c_1. We determine a matching of maximum weight in B, and then construct c_2 from c by permuting colors i and j if the edge of B that links $i \in V_1$ to $j \in V_2$ belongs to the optimal matching.

Consider for example the chain on three vertices v_1, v_2, v_3, where v_2 is linked to v_1 and v_3, and assume $k = 1$. If v_1 is the first revealed vertex, we color it, and wait for the next revealed vertices. If both v_2 and v_3 are revealed in the next cluster, we first try to color them. Since v_2 cannot be colored (while v_3 gets color 1), we start the recoloring process, where $C = \{v_1\}$. Both k-LF and k-Dsatur choose v_2 as first vertex to color, which means that $v_1 \in C$ cannot be colored. The process is therefore stopped, and we keep the coloring c_1 where v_1 and v_3 are the colored vertices.

The situation where c_2 is preferred to c_1 is illustrated in Figure 3. Suppose that the first cluster contains vertices $v_1, \ldots, v_9$ which are colored using k-LF coupled with First-Fit. If the vertices are colored in the order of their label (which corresponds to an order by non-increasing degree), we get a coloring c_1 where v_5 is the only non-colored vertex (shown in grey). Assume now $R = \{w_1, \ldots, w_6\}$ is the new set of revealed vertices. By again using k-LF coupled with First-Fit, we can consider these six new vertices in the order of their label and get a coloring where w_4 and w_6 are left uncolored. Since not all vertices of R have been colored, we use again k-LF coupled with First-Fit for these 15 vertices, but starting from scratch. A possible order is $w_1, w_2, v_4, v_1, v_2, v_8, w_3, w_4, w_5, v_3, v_5, v_7, w_6, v_6, v_9$. Notice that, for example, v_8 appears before w_3 while both vertices have degree 4, the reason being that v_8 belongs to the set C of colored vertices of the first cluster. The resulting coloring is shown at the bottom left of the Figure. Only w_5 and w_6 are left uncolored (while v_5 from the first cluster is colored). Since 13 vertices are colored with c while only 12 were colored with c_1, we perform a permutation of the color classes. The weighted complete bipartite graph is shown in the middle of the Figure. For example, the edge linking vertex 3 in V_1 to vertex 2 in V_2 has weight 2 since two vertices (v_2 and v_6) of C have color 3 in c and color 2 in c_1. The optimal matching is shown with bold lines and indicates that color classes 2 and 3 must be permuted. The resulting coloring c_2 is shown at the bottom right. We keep this coloring that has one more colored vertex than c_1. Observe that 5 vertices $(v_1, v_2, v_3, v_6, v_9)$ of C have the same color in c_1 and c_2, while only three of them (v_1, v_8, v_9) had the same color in c_1 and c.

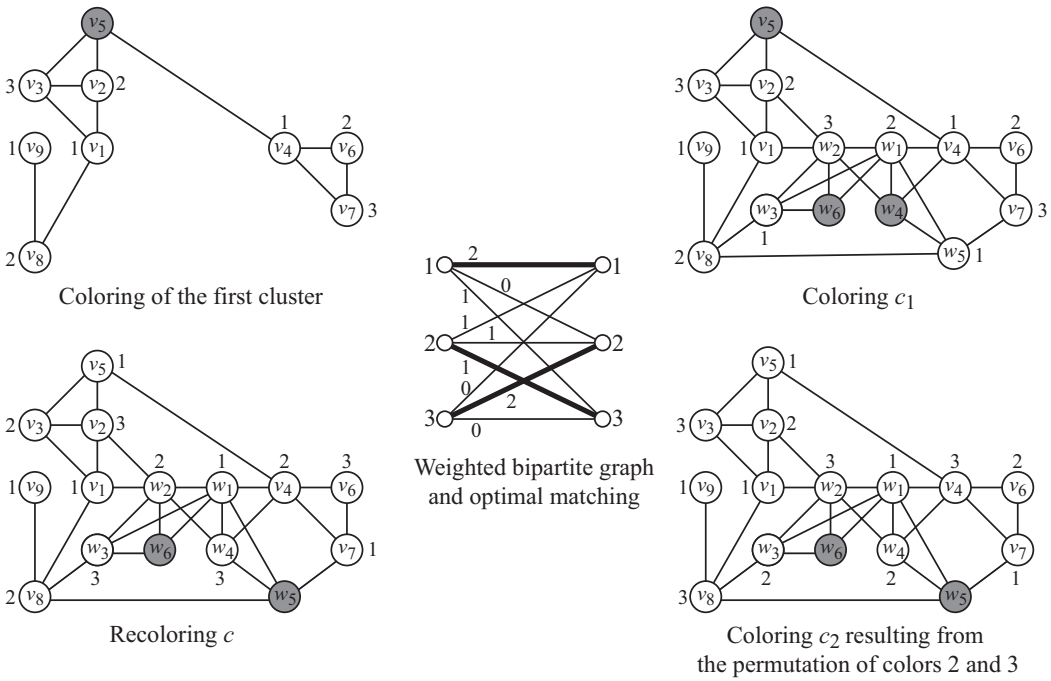

Figure 3: Illustration of the first recoloring strategy

The optimal matching can be computed in $\mathcal{O}\left(|E|^2\sqrt{|V_1|+|V_2|}\log n\right) = \mathcal{O}\left(k^{9/2}\log n\right)$ time [18], taking into account that no weight in B is larger than n.

5.2 Recoloring with tabu search

The second recoloring strategy works as follows. As was the case in the previous section, let W be the set of vertices revealed in previous clusters, let C be its subset of colored vertices, and let R be the new set of revealed vertices. We first try to color the vertices in R by using k-LF or k-Dsatur coupled with First-Fit, Next-Fit, or Best-Fit. When a vertex $v \in R$ cannot be colored, we use a tabu search algorithm to try to color v, with the constraint that all already colored vertices must remain colored (but possibly with different colors).

Tabu search is a local search technique that visits a search space S by moving step by step from a current solution $s \in S$ to a *neighbor* solution $s' \in N(s)$, where $N(s)$ is a subset of S called the *neighborhood* of s. A *tabu list* forbids some moves which would bring the search back to a recently visited solution. The best non-tabu move is chosen at each iteration. Tabu search was introduced in [19]. A description of the method and its concepts can be found in [20]. Tabu search was successfully applied to many complex combinatorial optimization problems, including classical vertex coloring [21].

Recoloring with tabu search is done as follows. Let R' be the subset of already colored vertices in R, and let v the current vertex in R to which we are unable to assign a color. We consider the subgraph induced by the vertices in $C \cup R' \cup \{v\}$. Let c be the current coloring of the vertices in $C \cup R'$. The initial solution considered by the tabu search has color $c(w)$ on all vertices $w \in C \cup R'$, and any color in $\{1, \ldots, k\}$ on v. A *conflicting edge* is an edge with the same color on its endpoints. Hence, in the initial solution, all conflicting edges have v as endpoint. Given any coloring s we denote by $f_1(s)$ the number of conflicting edges in s and by $f_2(s)$ the number of vertices in C which have a color that differs from that they had before the vertices in R were revealed. Our objective is to minimize $f(s) = f_1(s) + pf_2(s)$, where p is a parameter that gives more or less importance to the color changes. For example, by setting $p = 0.1$, we consider that 10 color changes are acceptable if these changes allow reducing the number of conflicting edges by one unit.

A move to a neighbor solution is performed by modifying the color of an endpoint of a conflicting edge. If the color of a vertex w is changed from i to j, we put the pair (w, i) in the tabu list, which means that it is forbidden, for some iterations, to give back color i to w. As was done in [21], the pair (w, i) remains in a tabu list for $\sqrt{f_1(s)}$ iterations. Let f^* denote the best value encountered during the tabu search, and let s^* be the corresponding coloring (i.e., $f(s^*) = f^*$). If the assignment of color j to a vertex w is tabu (i.e., (w, j) belongs to the tabu list) while such a change of color would lead to a solution of value strictly smaller than f^*, then the tabu status of (w, j) is cancelled, which means that the color change is permitted.

The tabu search stops when a solution of value 0 is found, or when a given number of iterations are performed without improving f^*. If the output of the tabu search is a coloring s^* with at least one conflicting edge (i.e., $f_1(s^*) > 0$), then v is not colored and we keep the coloring we had before applying the tabu search; otherwise s^* becomes the new current coloring. We then proceed with the next vertex in R (if any).

6 Computational experiments

As mentioned in the introduction, the chromatic number of a graph G is the smallest integer k such that the maximum k-colorable subgraph of G is G itself. In order to evaluate the performance of the proposed online algorithms, we test them on the 50 DIMACS benchmark instances [11] having $11 \leq n \leq 559$ vertices and $20 \leq m \leq 18\,707$ edges. The chromatic number of these graphs is known and ranges from 4 to 73. We consider a number k of colors equal to $\chi(G), \chi(G) + 1, \ldots, \chi(G) + 10$, where $\chi(G)$ denotes the chromatic number of G. The optimal solution for all these tests is therefore to color all vertices. The vertices being revealed in a random order, we run each algorithm 10 times, using 10 different vertex orders. Average results are reported in the following Tables and Figures. We first compare k-LF and k-Dsatur coupled with First-Fit, Next-Fit and Best-Fit, without allowing recolorings.

6.1 Results with no recoloring

6.1.1 Vertices are revealed one by one

Assume first that the vertices are revealed one by one, which means that they do not have to be ordered within their cluster (that contains only one element). Since Best-Fit is equivalent to First-Fit, we only compare First-Fit with Next-Fit. The first column of Tables 1 and 2 indicates the name of the considered graphs, their number of vertices and their chromatic number. We then report the number of colored vertices (column #) of both strategies, as well as the computing time in seconds (column t) needed to produce such colorings. Optimal solutions (i.e., when all vertices are colored) appear with bold letters.

A more visual picture of these results is given in Figure 4 where we indicate the performance profiles of both strategies. More precisely, for each value of $k \in \{\chi(G), \chi(G)+1, \ldots, \chi(G)+10\}$, every algorithm is run 500 times (10 times for each of the 50 instances), and we indicate the percentage of runs that produce an optimal solution (i.e., a solution where all vertices are colored). For each graph and each value of k, we also consider the best and the worst of the 10 runs and indicate how often they are optimal. For example, for $k = \chi(G) + 3$, 71.4% of the 500 runs, 76% (38 out of 50) of the best runs, and 62% (31 out of 50) of the worst runs produced by First-Fit are optimal. With $k = \chi(G)$, First-Fit finds, in average, 39.6% of the optimal solutions, while the percentage for Next-Fit is only 2.2%. If we allow $k = \chi(G) + 10$ colors, First-Fit produces optimal solutions for 47 graphs (which corresponds to 94% of the instances), while this is the case for only 25 instances (50%) with Next-Fit.

6.1.2 All vertices are revealed in one cluster

We now consider the other extreme, where all vertices are revealed in one cluster, which means that the algorithms can be considered as offline. We give in Figure 5 the performance profiles of the six algorithms obtained by coupling k-LF and k-Dsatur with First-Fit, Next-Fit and Best-Fit. We observe that k-Dsatur with Best-Fit outperforms all other strategies, with up to 68% of the instances solved to optimality. Its curve always lies above the other ones, and with $k = \chi(G) + 9$ colors, all vertices of every instance are colored. As already mentioned, this is not surprising, since this strategy is similar to the RLF algorithm which has proven to be more effective than LF or Dsatur in many numerical tests [15]. We also observe that First-Fit and Best-Fit should be preferred to Next-Fit for both k-LF and k-Dsatur. When setting $k = \chi(G)$, k-LF with Best-Fit is nearly as competitive as k-Dsatur with First-Fit or Best-Fit, with approximately 65% of the instances solved to optimality. This suggests that the Best-Fit policy outperforms the other policies, and that Next-Fit is to be avoided.

6.1.3 Intermediate cluster sizes

In order to compare the results obtained with intermediate cluster sizes, we report the Average Relative Percentage Deviation (ARPD) of every algorithm, which is defined as follows. Let S be the considered set of DIMACS instances. Also, for an instance $s \in S$, let n_s be the value of the optimal solution to the k-MCSP, and let $a_s(A)$ be the number of vertices colored when applying algorithm A on s:

$$\text{ARPD}(A) := \frac{100}{|S|} \sum_{s \in S} \frac{n_s - a_s(A)}{n_s}$$

We have tested clusters of size 1, $\lfloor \frac{n}{10} \rfloor$, $\lfloor \frac{n}{2} \rfloor$, and n. Figure 6 compares the coloring strategies First-Fit, Next-Fit and Best-Fit for both algorithms k-LF and k-Dsatur. For each algorithm A, we report the value ARPD(A) obtained by setting $a_s(A)$ equal to the average result (over the 10 runs) obtained by applying A on s with $k = \chi(G)$. We also indicate the ARPD measured by setting $a_s(A)$ equal to the best or to the worst of the 10 runs of A on s. Note that when all vertices are revealed in one cluster, there is no random ordering of the vertices, and the 10 runs of the algorithms are therefore all equivalent.

As expected, we observe that more vertices are colored when the clusters have larger sizes. For example, the ARPD of k-LF with Best-Fit decreases from 9.3% with clusters of 1 vertex, to 7.3%, with one cluster of n vertices. The only exception is for k-Dsatur with Best-Fit, where results with n clusters of 1 vertex are better than those with 10 clusters of size $\frac{n}{10}$ vertices. Best-Fit seems to be the winning coloring strategy when the clusters have big sizes, while First-Fit appears as a slightly better strategy when clusters have medium or small sizes. This is particularly true when using k-Dsatur rather than k-LF.

Figure 7 compares k-LF with k-Dsatur for every coloring strategy. It appears clearly that k-LF should be preferred to k-Dsatur when using Best-Fit with clusters of medium or small sizes, while k-Dsatur is always the winner when using First-Fit or Next-Fit.

6.2 Results with recolorings

We now report results obtained by using the recoloring strategies. We compare the following algorithms:

- LF : the standard k-LF algorithm with Best-Fit, but with no recoloring;
- LF+SR : the standard k-LF algorithm with Best-Fit, and with a sequential recoloring (SR) based on k-LF and Best-Fit, but without the final permutation of the color classes;
- LF+SRP : the same algorithm as the previous one, but with the final permutation of the color classes;
- LF+10TS : the standard k-LF algorithm with Best-Fit, with a Tabu Search (TS) for the recoloring that stops when the best solution s^* is not improved during 10 iterations;
- LF+50TS : the same algorithm as the previous one, but with a Tabu Search that stops when 50 iterations have been performed without improvement of s^*.

We report in Table 3 the results obtained with clusters of size $\lfloor \frac{n}{10} \rfloor$, and with $k = \chi(G)$. For each algorithm, we indicate the number of colored vertices (column #), the total number of color changes (column ch), and the computing time in seconds (column t). Figure 8 summarizes these results in a two dimensional space, where the x-axis corresponds to the number of color changes, while the y-axis indicates the ARPD of the considered algorithm. An ideal algorithm with respect to both objectives (i.e., that maximizes the number of colored vertices and minimizes the number of color changes) would have a point at the lower left corner of this space.

We observe that if no recoloring is performed, then 9.2% of the vertices are not colored. On the other end of the scale, the LF+SR algorithm has 8.7% of non-colored vertices with 2033 color changes. The permutation of the color classes (algorithm LF+SRP) allows to save about 182 color changes. Between these two extreme values, LF+10TS reduces the number of non-colored vertices from 9.2% to 8.3%, with only 64 color changes, and LF+50TS produces solutions with 7.9% of non-colored vertices and 159 color changes.

As indicated in Table 3, LF+50TS can be very time consuming for some instances (up to 25 minutes for Leighton's graphs), whereas LF+SRP is very fast (all instances are solved in less than half a second). But LF+50TS typically colors more vertices than LF+SRP, with fewer color changes, and in a reasonable amount of time. For example, LF+SRP leaves 2 uncolored vertices for `miles250`, at the expense of 55 color changes, while LF+50TS needs only 6 seconds to color one more vertex with only 2 color changes.

We find it important to mention that the tabu search recoloring strategy is more flexible in various aspects since the stopping criterion can be chosen so that the time spent in recoloring never exceeds a given time limit, and the importance of the number of color changes ($f_2(s)$) versus the number of colored vertices ($f_1(s)$) can be controlled by modifying the value of parameter p in the objective function $f(s) = f_1(s) + pf_2(s)$ used by the tabu search.

6.3 Permanent regime

The last experiments aim to analyze the behavior of the proposed algorithms in a situation, where revealed vertices disappear after a while. This is typically the case in channel assignment problems, where users appear

and disappear in the network. We consider a fixed *traffic rate* r, which means that, at each period, r new vertices are revealed, while r already known vertices disappear. Experiments are performed on random graphs having 50 vertices, edge density 0.25, and a number $k = 9$ of available colors. This value for k is justified by the fact that it corresponds to a situation where at least 98% of the 50 vertices can be colored, which is the typical accepted coverage ratio in wireless networks (that corresponds to a call-blocking probability of 2%.).

To perform our analysis, we first color the random graphs using k-LF with Best-Fit, to obtain initial colorings. We then consider 15 time periods with traffic rate r. At each period, we not only try to color the r new revealed vertices, but also the non-colored ones from previous periods (since the removal of some old vertices possibly makes others colorable). Four recoloring strategies are compared in Table 4 (LF, LF+SR, LF+SRP, and LF+10TS) and the results are averages taken over 18 random graphs. We consider traffic rates r equal to $1, \ldots, 15$. For each r we indicate the average number of colored vertices per period (column #). We also indicate the average total number of color changes (column ch), and the average total computing time in seconds (column t), not taking into account the time needed to generate the initial colorings of the random graphs.

We observe that if color changes are forbidden (algorithm LF), 48.2 vertices are colored, in average, while the other strategies allow at least one extra vertex to be colored. The cost of coloring this extra vertex (in terms of number of color changes) depends on the considered algorithm. More specifically, the LF-SR algorithm performs 34.5 color changes, in average, while adding the possibility of permuting the color classes decreases the number of color changes by one to two units. LF-10TS is able to maintain an average of almost 50 colored vertices at the expense of only 4 changes, in average.

The evolution of the total number of color changes with respect to the traffic rate r is illustrated in Figure 9. If ch color changes are required in total for a given r, this corresonds to $100\frac{ch}{15(50-r)}\%$ of the upper bound on the total number of possible color changes (since $(50 - r)$ of the 50 vertices remain in the network at each of the 15 time periods. So, for example, the total of 48.5 color changes for LF+SRP with $r = 15$ corresponds to $100\frac{48.5}{35\cdot 15} = 9.2\%$ of the upper bound on the total number of possible color changes. We observe that when r increases from 1 to 11, this percentage increases for LF+SR and LF+SRP to about10%, while LF+10TS manages to maintain a color changes rate close to 1%. Furthermore, the computing time of LF+10TS in about 30 seconds, which corresponds to approximately 2 seconds per time period. These experiments show that the algorithms based on a a sequential recoloring are more sensitive to the traffic rate r than those that use the tabu search recoloring strategy: the higher the traffic, the harder it is to maintain a maximum number of colored nodes.

7 Conclusion

The problem of coloring as many vertices as possible with k available colors is a famous $\mathcal{NP}$-hard problem. We have studied its online version, where vertices are revealed by clusters of one or more vertices. We have first proved upper and lower bounds on the competitive ratio of online algorithms. We have then described the smallest graphs for which a large family of sequential online coloring algorithms fail to produce an optimal solution.

We have described several online algorithms, where all decisions are irrevocable. These algorithms are based on two ways of ordering the new revealed vertices (k-LF and k-Dsatur) and three coloring strategies (Firs-Fit, Next-Fit and Best-Fit). The winner strategies seem to be k-LF with Best-Fit and k-Dsatur with First-Fit and Best-Fit. We have observed that the Next-Fit strategy, which aims to produce balanced color classes, should be avoided.

We have then considered the case where it is allowed to change the color of colored vertices, as long as they remain colored. We have proposed two recoloring strategies. The LF+SRP algorithm, based on a sequential recoloring and a permutation of color classes is very fast, but typically asks for more color changes than the LF+10TS and LF+50TS algorithms, which are based on a tabu search. Also, LF+SRP typically colors less vertices than LF+10TS and LF+50TS. As mentioned in Section 6.2, the tabu search used in LF+10TS and

LF+50TS has the advantage to be flexible, in that sense that its execution time can easily be controlled by defining an appropriate stopping criterion, and a higher priority can easily be given to one of the two objectives (i.e., maximizing the number of colored vertices, or minimizing the number of color changes).

We have finally considered the case where vertices not only appear in clusters, but also disappear after a while. We have observed that the online algorithms based on a sequential recoloring have a tendency to ask for many color changes when the traffic rate increases, while recolorings based on tabu search are less sensitive to such an increase.

To conclude, we find it important to mention that throughout this paper, we have assumed that all (or almost all) vertices of the considered graphs can be colored with the k available colors. It is obvious that if this is not the case, new strategies would be necessary. For example, when using a sequential coloring algorithm, instead of giving priority to vertices with a high degree, it appears as more natural to reject those nodes, as they are the hardest to color.

Appendix

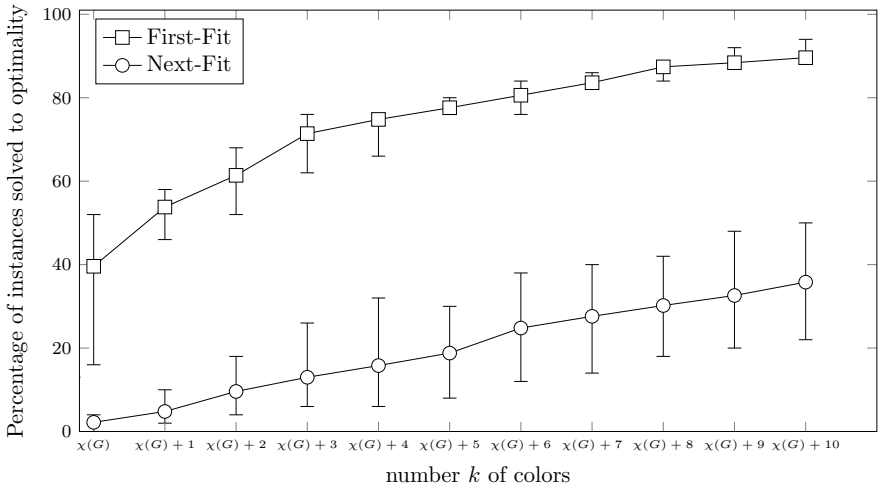

Figure 4: Performance profile when the vertices are revealed one by one

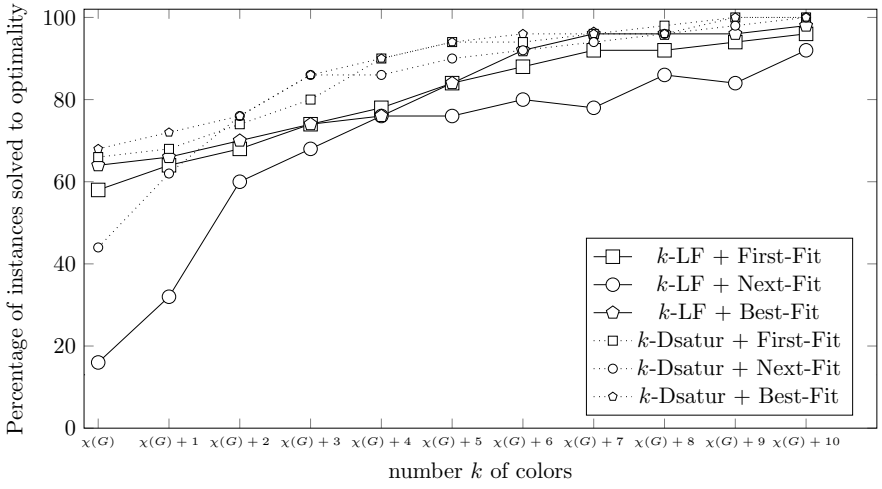

Figure 5: Performance profile when the vertices are revealed in one cluster

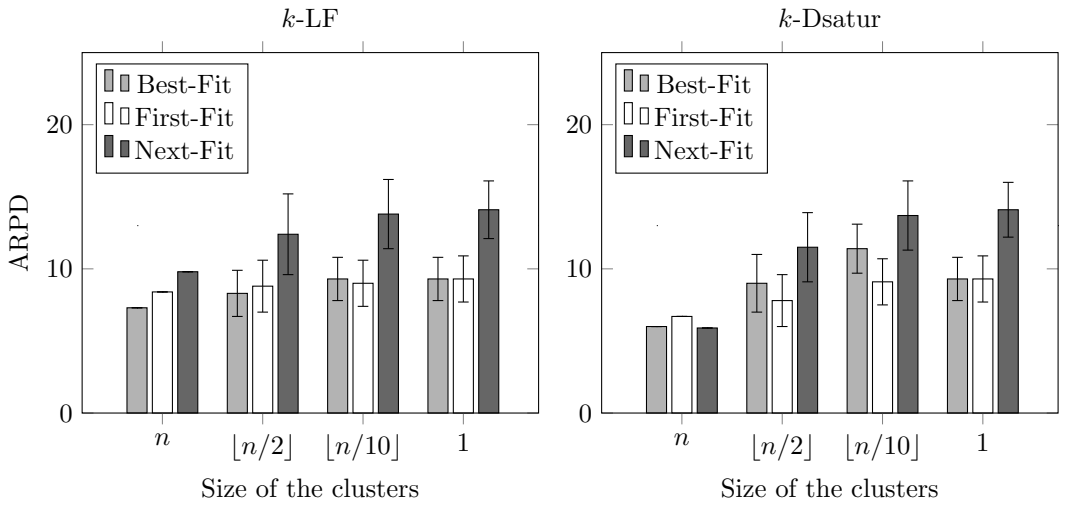

Figure 6: ARPDs for k-LF and k-Dsatur with various coloring strategies

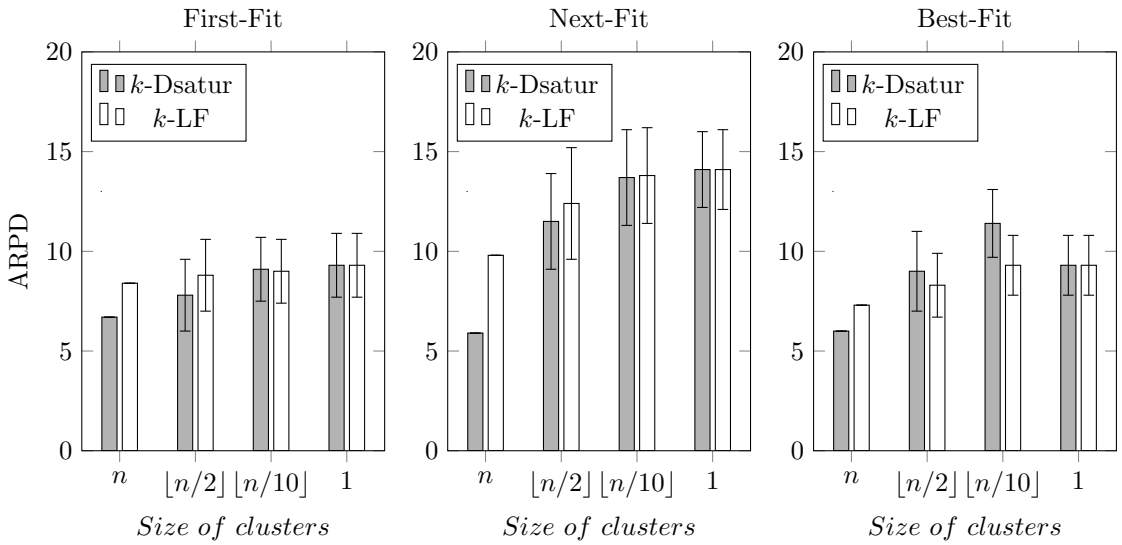

Figure 7: ARPDs for First-Fit, Next-Fit and Best-Fit with k-LF and k-Dsatur

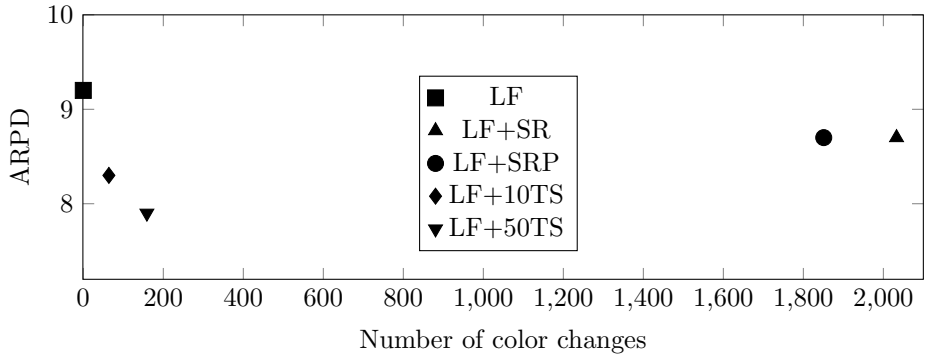

Figure 8: ARPDs and number of color changes

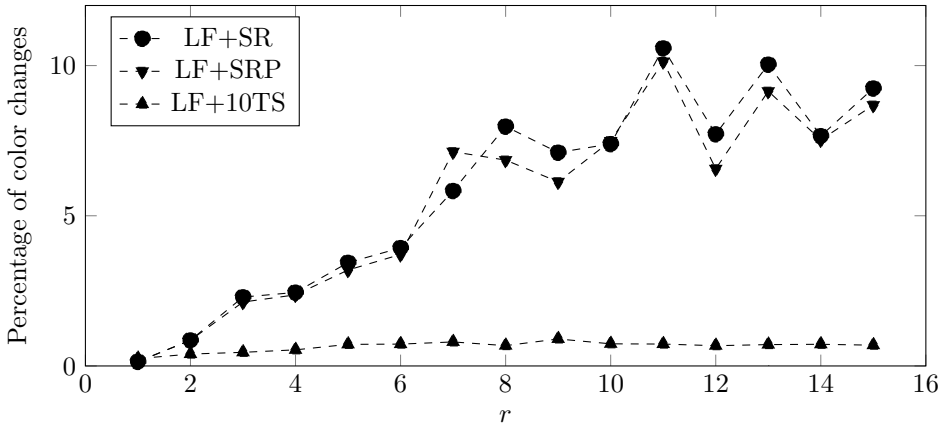

Figure 9: Evolution of the number of color changes in a permanent regime

Table 1: First-Fit versus Next-Fit when the vertices are revealed one by one

	number k of available colors																					
	$\chi(G)$		$\chi(G)+1$		$\chi(G)+2$		$\chi(G)+3$		$\chi(G)+4$		$\chi(G)+5$		$\chi(G)+6$		$\chi(G)+7$		$\chi(G)+8$		$\chi(G)+9$		$\chi(G)+10$	
Graph $(n, \chi(G))$	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t
`fpsol2.i.1` (269, 65)																						
First-Fit	**269.0**	1.4	**269.0**	1.5	**269.0**	1.4	**269.0**	1.5	**269.0**	1.5	**269.0**	1.5	**269.0**	1.4	**269.0**	1.4	**269.0**	1.5	**269.0**	1.5	**269.0**	1.4
Next-Fit	245.3	1.4	244.9	1.4	247.2	1.4	245.5	1.4	247.7	1.4	247.2	1.4	250.8	1.4	247.5	1.4	248.2	1.4	249.3	1.4	250.8	1.4
`fpsol2.i.2` (363, 30)																						
First-Fit	362.2	1.9	**363.0**	1.8	362.9	1.9	**363.0**	1.9	**363.0**	1.9	**363.0**	1.9	**363.0**	1.9	**363.0**	1.8	**363.0**	1.8	**363.0**	1.9	**363.0**	1.9
Next-Fit	337.0	1.7	339.4	1.7	340.7	1.6	342.2	1.7	342.0	1.7	342.7	1.7	343.1	1.7	344.6	1.7	344.5	1.7	345.7	1.7	347.0	1.7
`fpsol2.i.3` (363, 30)																						
First-Fit	362.6	1.9	**363.0**	1.8	**363.0**	1.8	**363.0**	1.8	**363.0**	1.8	**363.0**	1.9	**363.0**	1.9	**363.0**	1.9	**363.0**	1.9	**363.0**	1.8	**363.0**	1.9
Next-Fit	336.9	1.6	339.7	1.7	341.1	1.6	341.3	1.6	342.8	1.7	343.3	1.7	344.9	1.8	345.2	1.7	346.4	1.8	346.0	1.7	346.2	1.7
`inithx.i.1` (519, 54)																						
First-Fit	**519.0**	5.6	**519.0**	5.5	**519.0**	5.4	**519.0**	5.5	**519.0**	5.6	**519.0**	5.5	**519.0**	5.5	**519.0**	5.6	**519.0**	5.6	**519.0**	5.5	**519.0**	5.5
Next-Fit	483.1	5.1	482.3	5.1	484.7	5.1	482.5	5.0	481.7	5.0	483.7	5.1	486.6	5.2	484.9	5.1	487.2	5.1	486.9	5.1	488.1	5.1
`inithx.i.2` (558, 31)																						
First-Fit	557.7	5.1	**558.0**	5.0	**558.0**	5.0	**558.0**	4.9	**558.0**	4.9	**558.0**	5.0	**558.0**	5.0	**558.0**	5.0	**558.0**	5.0	**558.0**	5.0	**558.0**	5.0
Next-Fit	524.8	4.4	524.4	4.5	524.3	4.5	526.5	4.5	527.4	4.5	527.0	4.6	527.8	4.5	527.8	4.5	528.8	4.6	530.3	4.5	530.8	4.6
`inithx.i.3` (559, 31)																						
First-Fit	558.9	5.0	**559.0**	5.0	**559.0**	5.0	**559.0**	5.2	**559.0**	5.0	**559.0**	5.1	**559.0**	5.2	**559.0**	5.0	**559.0**	5.0	**559.0**	5.1	**559.0**	5.2
Next-Fit	527.5	4.6	527.0	4.5	526.1	4.5	526.2	4.5	527.4	4.5	528.8	4.7	528.7	4.7	528.7	4.7	530.1	4.7	528.5	4.6	529.5	4.7
`le450-15a` (450, 15)																						
First-Fit	403.6	2.5	417.1	2.6	425.3	2.6	435.0	2.6	442.4	2.7	446.1	2.6	448.7	2.7	449.5	2.6	**450.0**	2.6	**450.0**	2.7	**450.0**	2.6
Next-Fit	386.1	2.5	399.6	2.6	408.0	2.6	416.1	2.6	420.9	2.6	427.1	2.6	430.9	2.6	436.5	2.6	439.3	2.6	442.8	2.6	443.5	2.6
`le450-15b` (450, 15)																						
First-Fit	404.6	2.6	417.4	2.6	427.2	2.6	436.2	2.6	441.9	2.6	446.8	2.7	449.2	2.6	**450.0**	2.6	**450.0**	2.7	**450.0**	2.6	**450.0**	2.6
Next-Fit	388.7	2.5	399.2	2.6	409.8	2.5	417.0	2.6	422.6	2.6	427.8	2.6	433.6	2.6	436.1	2.6	438.9	2.6	442.2	2.6	442.7	2.6
`le450-15c` (450, 15)																						
First-Fit	302.3	3.7	320.0	3.8	334.5	3.7	345.2	3.8	359.5	3.9	372.7	4.0	384.5	4.0	395.4	4.0	406.1	4.1	416.8	4.2	423.5	4.1
Next-Fit	296.9	3.6	308.2	3.7	323.6	3.7	336.7	3.7	347.8	3.8	355.6	3.8	366.7	3.9	373.9	3.9	383.3	3.9	390.2	4.0	400.9	4.1
`le450-15d` (450, 15)																						
First-Fit	300.7	3.7	316.5	3.8	331.0	3.8	345.6	3.9	356.7	3.9	369.3	3.9	383.1	4.1	391.5	4.0	404.5	4.0	414.5	4.1	422.4	4.2
Next-Fit	294.0	3.6	310.8	3.7	322.8	3.7	332.8	3.7	343.1	3.8	357.2	3.8	366.7	3.9	373.1	3.9	383.5	3.9	389.6	4.0	398.7	4.0
`le450-25a` (450, 25)																						
First-Fit	442.5	2.6	445.7	2.7	448.6	2.6	449.6	2.7	449.9	2.7	**450.0**	2.7	**450.0**	2.7	**450.0**	2.6	**450.0**	2.6	**450.0**	2.7	**450.0**	2.6
Next-Fit	427.4	2.6	431.1	2.6	432.7	2.6	436.8	2.6	437.8	2.7	439.7	2.6	442.4	2.6	444.8	2.6	445.5	2.6	446.1	2.6	447.6	2.7
`le450-25b` (450, 25)																						
First-Fit	442.8	2.7	446.2	2.7	448.7	2.7	**450.0**	2.7	**450.0**	2.6	**450.0**	2.7	**450.0**	2.7	**450.0**	2.7	**450.0**	2.8	**450.0**	2.6	**450.0**	2.6
Next-Fit	429.8	2.7	432.3	2.6	437.8	2.6	438.6	2.7	441.4	2.6	442.4	2.7	444.7	2.7	445.4	2.7	446.6	2.7	447.8	2.6	447.3	2.7
`le450-25c` (450, 25)																						
First-Fit	389.8	4.1	396.8	4.1	405.5	4.0	412.9	4.2	419.1	4.1	426.6	4.2	430.7	4.3	438.8	4.3	442.5	4.3	446.1	4.2	448.6	4.3
Next-Fit	371.2	4.0	379.5	4.0	385.2	4.0	393.1	4.0	396.4	4.0	405.4	4.1	408.9	4.2	413.4	4.1	415.7	4.2	423.6	4.2	426.7	4.2
`le450-25d` (450, 25)																						
First-Fit	388.4	4.2	397.6	4.1	404.7	4.2	413.6	4.1	418.2	4.2	427.0	4.2	433.2	4.2	439.3	4.3	443.4	4.2	445.8	4.3	448.9	4.2
Next-Fit	372.7	4.0	379.0	4.0	386.0	4.1	395.2	4.1	398.9	4.1	405.0	4.2	408.9	4.1	413.0	4.1	418.8	4.1	422.2	4.3	424.3	4.2
`le450-5a` (450, 5)																						
First-Fit	256.8	1.9	297.1	2.0	341.2	2.1	371.4	2.1	399.9	2.1	421.4	2.2	439.4	2.2	447.2	2.2	449.8	2.2	**450.0**	2.1	**450.0**	2.2
Next-Fit	249.3	1.9	287.0	2.0	323.6	2.0	353.8	2.1	377.9	2.1	399.8	2.2	415.3	2.1	427.0	2.2	434.9	2.1	441.5	2.1	446.3	2.2
`le450-5b` (450, 5)																						
First-Fit	255.0	1.9	294.3	2.0	333.0	2.1	370.7	2.2	397.3	2.1	423.0	2.2	437.1	2.2	446.5	2.2	449.9	2.2	**450.0**	2.2	**450.0**	2.2
Next-Fit	245.8	1.9	291.3	2.0	324.3	2.1	351.7	2.1	377.3	2.1	394.6	2.2	413.9	2.2	428.2	2.2	435.4	2.2	443.5	2.2	445.7	2.2
`le450-5c` (450, 5)																						
First-Fit	197.2	2.3	235.0	2.5	270.4	2.6	307.8	2.7	335.9	2.8	358.8	2.8	391.3	2.8	400.3	2.9	427.1	2.9	433.9	2.8	444.4	2.9
Next-Fit	198.8	2.3	226.4	2.5	255.8	2.5	282.1	2.6	310.4	2.6	327.1	2.7	356.8	2.8	381.7	2.8	390.3	2.8	404.1	2.8	414.2	2.9
`le450-5d` (450, 5)																						
First-Fit	208.4	2.4	236.0	2.4	272.9	2.6	296.1	2.6	323.5	2.7	357.3	2.8	378.2	2.8	403.6	2.8	419.7	2.9	433.7	2.9	444.0	2.9
Next-Fit	197.7	2.4	232.1	2.4	263.7	2.5	288.4	2.6	308.6	2.6	341.6	2.7	357.2	2.8	381.9	2.8	388.8	2.8	403.1	2.8	411.7	2.8
`mulsol.i.1` (138, 49)																						
First-Fit	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3	**138.0**	0.3
Next-Fit	127.3	0.3	129.1	0.3	129.8	0.3	131.1	0.3	131.5	0.3	133.7	0.3	134.0	0.3	135.1	0.3	135.9	0.3	135.9	0.3	136.1	0.3
`mulsol.i.2` (173, 31)																						
First-Fit	172.9	0.3	**173.0**	0.3	**173.0**	0.4	**173.0**	0.3	**173.0**	0.4	**173.0**	0.3	**173.0**	0.4	**173.0**	0.4	**173.0**	0.4	**173.0**	0.4	**173.0**	0.3
Next-Fit	153.3	0.4	152.3	0.3	155.1	0.3	155.1	0.3	157.5	0.3	157.4	0.4	158.1	0.3	158.6	0.3	159.9	0.3	161.3	0.3	162.7	0.3
`mulsol.i.3` (174, 31)																						
First-Fit	**174.0**	0.4	**174.0**	0.4	**174.0**	0.4	**174.0**	0.4	**174.0**	0.4	**174.0**	0.4	**174.0**	0.4	**174.0**	0.3	**174.0**	0.3	**174.0**	0.4	**174.0**	0.4
Next-Fit	153.6	0.3	153.8	0.3	155.4	0.3	157.7	0.3	157.8	0.4	158.1	0.3	159.8	0.3	161.8	0.3	160.8	0.3	161.3	0.4	161.8	0.3
`mulsol.i.4` (175, 31)																						
First-Fit	174.9	0.3	**175.0**	0.3	**175.0**	0.3	**175.0**	0.4	**175.0**	0.4	**175.0**	0.3	**175.0**	0.3	**175.0**	0.4	**175.0**	0.4	**175.0**	0.4	**175.0**	0.3
Next-Fit	153.4	0.3	155.5	0.3	156.6	0.3	157.7	0.3	158.1	0.3	159.3	0.3	160.7	0.3	161.4	0.3	161.2	0.4	162.2	0.4	164.2	0.4
`mulsol.i.5` (176, 31)																						
First-Fit	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4	**176.0**	0.4
Next-Fit	155.3	0.4	155.5	0.3	158.3	0.3	159.1	0.4	160.1	0.4	161.5	0.4	161.3	0.4	160.2	0.4	162.5	0.4	165.2	0.4	163.6	0.3
`zeroin.i.1` (126, 49)																						
First-Fit	125.4	0.2	**126.0**	0.3	**126.0**	0.3	**126.0**	0.2	**126.0**	0.3	**126.0**	0.2	**126.0**	0.2	**126.0**	0.3	**126.0**	0.2	**126.0**	0.2	**126.0**	0.3
Next-Fit	110.1	0.2	112.1	0.2	111.3	0.2	112.2	0.2	113.3	0.2	112.9	0.3	113.3	0.2	114.4	0.2	114.5	0.3	115.0	0.2	116.2	0.3
`zeroin.i.2` (157, 30)																						
First-Fit	156.5	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3
Next-Fit	143.5	0.3	143.2	0.3	144.0	0.3	144.4	0.3	143.9	0.3	143.4	0.3	146.2	0.3	145.3	0.3	145.8	0.3	147.3	0.3	146.2	0.3

Table 2: First-Fit versus Next-Fit when the vertices are revealed one by one (continued)

	number k of available colors																					
	$\chi(G)$		$\chi(G)+1$		$\chi(G)+2$		$\chi(G)+3$		$\chi(G)+4$		$\chi(G)+5$		$\chi(G)+6$		$\chi(G)+7$		$\chi(G)+8$		$\chi(G)+9$		$\chi(G)+10$	
Graph $(n, \chi(G))$	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t	#	t
`zeroin.i.3` (157, 30)																						
First-Fit	156.9	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3	**157.0**	0.3
NextFit	143.4	0.3	143.4	0.3	142.8	0.3	144.6	0.3	144.1	0.3	145.1	0.3	145.9	0.3	145.9	0.3	145.4	0.3	145.6	0.3	147.7	0.3
`anna` (138, 11)																						
First-Fit	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1	**138.0**	0.1
NextFit	134.4	0.1	134.7	0.1	135.5	0.1	136.1	0.1	136.3	0.1	136.5	0.1	136.7	0.1	137.2	0.1	137.5	0.1	137.2	0.1	137.6	0.1
`david` (87, 11)																						
First-Fit	86.5	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0	**87.0**	0.0
NextFit	83.9	0.0	85.2	0.0	85.7	0.0	85.5	0.0	86.1	0.0	85.8	0.0	86.2	0.0	85.8	0.0	86.2	0.0	86.3	0.0	86.4	0.0
`homer` (556, 13)																						
First-Fit	554.4	2.3	555.8	2.3	**556.0**	2.4	**556.0**	2.3	**556.0**	2.4	**556.0**	2.3	**556.0**	2.3	**556.0**	2.3	**556.0**	2.3	**556.0**	2.3	**556.0**	2.3
NextFit	548.9	2.3	550.8	2.4	551.8	2.3	553.4	2.3	552.7	2.3	553.6	2.3	554.2	2.3	554.9	2.3	554.7	2.3	555.2	2.3	555.0	2.2
`huck` (74, 11)																						
First-Fit	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0	**74.0**	0.0
NextFit	72.3	0.0	72.9	0.0	73.0	0.0	73.3	0.0	73.4	0.0	73.8	0.0	73.7	0.0	73.7	0.0	73.7	0.0	73.7	0.0	73.6	0.0
`jean` (77, 10)																						
First-Fit	76.8	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0
NextFit	75.0	0.0	75.8	0.0	76.5	0.0	76.7	0.0	76.7	0.0	76.9	0.0	**77.0**	0.0	76.9	0.0	**77.0**	0.0	**77.0**	0.0	**77.0**	0.0
`games120` (120, 9)																						
First-Fit	119.9	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1
NextFit	117.9	0.1	119.4	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1	**120.0**	0.1
`miles1000` (128, 42)																						
First-Fit	124.8	0.2	126.1	0.2	127.5	0.2	127.9	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2
NextFit	120.2	0.2	121.3	0.2	121.8	0.2	122.6	0.2	123.7	0.2	123.9	0.2	125.2	0.2	124.8	0.2	125.4	0.2	125.0	0.2	126.1	0.2
`miles1500` (128, 73)																						
First-Fit	127.1	0.3	127.7	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3	**128.0**	0.3
NextFit	121.2	0.3	121.4	0.3	122.4	0.3	122.7	0.3	122.8	0.3	124.6	0.3	124.9	0.3	124.1	0.3	125.2	0.3	125.9	0.3	126.0	0.3
`miles250` (125, 8)																						
First-Fit	123.1	0.1	124.6	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1
NextFit	122.4	0.1	123.3	0.1	124.3	0.1	124.5	0.1	124.9	0.1	124.9	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1	**125.0**	0.1
`miles500` (128, 20)																						
First-Fit	126.0	0.1	127.1	0.1	127.8	0.1	**128.0**	0.1	**128.0**	0.1	**128.0**	0.1	**128.0**	0.1	**128.0**	0.1	**128.0**	0.1	**128.0**	0.1	**128.0**	0.1
NextFit	123.8	0.1	124.5	0.1	125.8	0.1	126.0	0.1	126.6	0.1	127.3	0.1	127.4	0.1	**128.0**	0.1	127.6	0.1	127.9	0.1	127.9	0.1
`miles750` (128, 31)																						
First-Fit	125.2	0.1	125.9	0.2	127.5	0.2	**128.0**	0.2	127.8	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2	**128.0**	0.2
NextFit	122.3	0.2	123.1	0.2	124.6	0.1	124.5	0.1	125.6	0.2	125.9	0.1	126.9	0.1	126.1	0.1	126.8	0.1	127.4	0.2	127.5	0.2
`queen11-11` (121, 11)																						
First-Fit	95.1	0.1	100.7	0.1	107.7	0.1	112.5	0.1	117.9	0.1	120.4	0.1	120.9	0.1	**121.0**	0.1	**121.0**	0.1	**121.0**	0.1	**121.0**	0.1
NextFit	91.4	0.1	96.6	0.1	102.4	0.1	106.4	0.1	111.1	0.1	114.3	0.1	116.7	0.1	118.5	0.1	119.5	0.1	119.9	0.1	120.7	0.1
`queen13-13` (169, 13)																						
First-Fit	132.7	0.3	142.5	0.3	149.4	0.3	157.7	0.3	162.1	0.3	165.7	0.3	168.5	0.3	**169.0**	0.3	**169.0**	0.3	**169.0**	0.3	**169.0**	0.3
NextFit	128.4	0.3	134.8	0.3	142.7	0.3	148.2	0.3	152.9	0.3	157.4	0.3	160.2	0.3	163.8	0.3	164.9	0.3	167.9	0.3	167.9	0.3
`queen5-5` (25, 5)																						
First-Fit	19.7	0.0	23.0	0.0	24.6	0.0	24.8	0.0	24.9	0.0	**25.0**	0.0	**25.0**	0.0	**25.0**	0.0	**25.0**	0.0	**25.0**	0.0	**25.0**	0.0
NextFit	17.5	0.0	20.5	0.0	22.1	0.0	23.8	0.0	24.3	0.0	24.6	0.0	24.9	0.0	**25.0**	0.0	**25.0**	0.0	**25.0**	0.0	**25.0**	0.0
`queen6-6` (36, 7)																						
First-Fit	30.6	0.0	33.6	0.0	35.4	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0
NextFit	29.6	0.0	31.5	0.0	33.4	0.0	35.0	0.0	35.6	0.0	35.7	0.0	35.8	0.0	35.9	0.0	**36.0**	0.0	**36.0**	0.0	**36.0**	0.0
`queen7-7` (49, 7)																						
First-Fit	37.4	0.0	41.5	0.0	45.9	0.0	48.0	0.0	**49.0**	0.0	**49.0**	0.0	**49.0**	0.0	**49.0**	0.0	**49.0**	0.0	**49.0**	0.0	**49.0**	0.0
NextFit	35.9	0.0	39.1	0.0	42.7	0.0	45.5	0.0	46.5	0.0	47.4	0.0	48.3	0.0	48.8	0.0	48.9	0.0	**49.0**	0.0	**49.0**	0.0
`queen8-12` (96, 12)																						
First-Fit	85.9	0.1	90.9	0.1	93.2	0.1	95.9	0.1	**96.0**	0.1	**96.0**	0.1	**96.0**	0.1	**96.0**	0.1	**96.0**	0.1	**96.0**	0.1	**96.0**	0.1
NextFit	80.8	0.1	85.9	0.1	89.0	0.1	92.6	0.1	93.9	0.1	94.3	0.1	95.4	0.1	95.8	0.1	95.8	0.1	95.8	0.1	**96.0**	0.1
`queen8-8` (64, 9)																						
First-Fit	54.6	0.0	57.3	0.0	61.6	0.0	63.5	0.0	63.9	0.0	**64.0**	0.0	**64.0**	0.0	**64.0**	0.0	**64.0**	0.0	**64.0**	0.0	**64.0**	0.0
NextFit	51.0	0.0	53.8	0.0	57.5	0.0	60.7	0.0	61.9	0.0	62.9	0.0	63.5	0.0	63.8	0.0	**64.0**	0.0	**64.0**	0.0	63.9	0.0
`queen9-9` (81, 10)																						
First-Fit	68.4	0.1	73.3	0.1	76.0	0.1	79.8	0.1	80.9	0.1	**81.0**	0.1	**81.0**	0.1	**81.0**	0.1	**81.0**	0.1	**81.0**	0.1	**81.0**	0.1
NextFit	64.9	0.1	69.4	0.1	72.5	0.1	76.4	0.1	76.8	0.1	78.6	0.1	79.9	0.1	80.4	0.1	80.7	0.1	80.9	0.1	**81.0**	0.1
`myciel3` (11, 4)																						
First-Fit	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0
NextFit	10.8	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0	**11.0**	0.0
`myciel4` (23, 5)																						
First-Fit	22.9	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0
NextFit	22.1	0.0	22.7	0.0	22.7	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0	**23.0**	0.0
`myciel5` (47, 6)																						
First-Fit	46.8	0.0	46.9	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0
NextFit	45.4	0.0	45.9	0.0	46.3	0.0	46.8	0.0	46.9	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0	**47.0**	0.0
`myciel6` (95, 7)																						
First-Fit	94.9	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1	**95.0**	0.1
NextFit	88.6	0.1	91.4	0.1	92.1	0.1	92.1	0.1	93.3	0.1	93.9	0.1	94.1	0.1	94.6	0.0	94.4	0.1	94.7	0.1	94.9	0.1
`myciel7` (191, 8)																						
First-Fit	189.9	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3	**191.0**	0.3
NextFit	176.4	0.3	178.8	0.3	180.2	0.3	184.1	0.3	184.7	0.3	185.0	0.3	187.8	0.3	188.1	0.3	188.2	0.3	189.2	0.3	188.4	0.3

Table 3: Recoloring strategies with $k = \chi(G)$, and clusters of size $\lfloor \frac{n}{10} \rfloor$

	LF			LF+SR			LF+SRP			LF+10TS			LF+50TS		
INSTANCE (n, χ)	#	ch	t	#	ch	t	#	ch	t	#	ch	t	#	ch	t
fpsol2.i.1 (269, 65)	269	0	0.1	269	0	0.1	269	0	0.1	269	0	0.1	269	0	0.1
fpsol2.i.2 (363, 30)	363	0	0.1	363	0	0.1	363	0	0.1	363	0	0.1	363	0	0.1
fpsol2.i.3 (363, 30)	363	0	0.1	363	0	0.1	363	0	0.1	363	0	0.1	363	0	0.1
inithx.i.1 (519, 54)	519	0	0.2	519	0	0.2	519	0	0.2	519	0	0.2	519	0	0.2
inithx.i.2 (558, 31)	558	0	0.2	558	0	0.2	558	0	0.2	558	0	0.2	558	0	0.2
inithx.i.3 (559, 31)	559	0	0.2	559	0	0.2	559	0	0.2	559	0	0.2	559	0	0.2
le450-15a (450, 15)	400	0	0.1	400	0	0.2	400	0	0.4	409	20	661.3	400	2	476.6
le450-15b (450, 15)	405	0	0.1	413	230	0.1	413	230	0.4	407	6	571.8	407	6	540.3
le450-15c (450, 15)	306	0	0.1	310	156	0.3	308	141	0.5	298	10	2164.8	306	0	1849.1
le450-15d (450, 15)	307	0	0.1	308	234	0.3	309	214	0.6	302	5	2152.3	306	5	1957.7
le450-25a (450, 25)	443	0	0.1	443	0	0.1	443	0	0.3	443	0	69.9	443	0	50.6
le450-25b (450, 25)	443	0	0.1	450	357	0.1	450	327	0.3	443	0	62.2	443	0	30
le450-25c (450, 25)	389	0	0.1	399	222	0.3	399	210	0.8	393	8	994.4	389	5	953.6
le450-25d (450, 25)	389	0	0.1	404	224	0.3	402	216	0.8	391	4	805.5	390	1	743.1
le450-5a (450, 5)	261	0	0.1	261	0	0.1	261	0	0.2	262	4	1303.5	264	4	1238.7
le450-5b (450, 5)	257	0	0.1	257	0	0.1	257	0	0.2	258	22	1491.4	254	2	1285.9
le450-5c (450, 5)	196	0	0.1	196	0	0.1	196	0	0.2	213	4	2025.8	198	4	2141.4
le450-5d (450, 5)	195	0	0.1	195	0	0.1	195	0	0.2	216	6	1916.9	194	6	2115.8
mulsol.i.1 (138, 49)	138	0	0.1	138	0	0.1	138	0	0.1	138	0	0.1	138	0	0.1
mulsol.i.2 (173, 31)	173	0	0.1	173	0	0.1	173	0	0.1	173	0	0.1	173	0	0.1
mulsol.i.3 (174, 31)	174	0	0.1	174	0	0.1	174	0	0.1	174	0	0.1	174	0	0.1
mulsol.i.4 (175, 31)	175	0	0.1	175	0	0.1	175	0	0.1	175	0	0.1	175	0	0.1
mulsol.i.5 (176, 31)	176	0	0.1	176	0	0.1	176	0	0.1	176	0	0.1	176	0	0.1
zeroin.i.1 (126, 49)	126	0	0.1	126	0	0.1	126	0	0.1	126	0	0.1	126	0	0.1
zeroin.i.2 (157, 30)	157	0	0.1	157	0	0.1	157	0	0.1	157	0	0.1	157	0	0.1
zeroin.i.3 (157, 30)	157	0	0.1	157	0	0.1	157	0	0.1	157	0	0.1	157	0	0.1
anna (138, 11)	137	0	0.1	138	59	0.1	138	59	0.1	137	0	0.2	137	0	0.1
david (87, 11)	86	0	0.1	87	70	0.1	87	44	0.1	87	2	5.7	86	0	0.1
homer (556, 13)	554	0	0.1	556	207	0.1	556	207	0.2	556	6	35	555	1	17.8
huck (74, 11)	74	0	0.1	74	0	0.1	74	0	0.1	74	0	0.1	74	0	0.1
jean (77, 10)	77	0	0.1	77	0	0.1	77	0	0.1	77	0	0.1	77	0	0.1
games120 (120, 9)	120	0	0.1	120	0	0.1	120	0	0.1	120	0.1	0	120	0	0.1
miles1000 (128, 42)	126	0	0.1	126	0	0.1	126	0	0.1	126	0.1	5.9	126	0	2.8
miles1500 (128, 73)	127	0	0.1	128	106	0.1	128	53	0.3	127	0.1	7	127	0	3.2
miles250 (125, 8)	122	0	0.1	123	55	0.1	123	55	0.1	125	4	18.4	124	2	5.8
miles500 (128, 20)	127	0	0.1	128	100	0.1	128	84	0.1	128	1	12.7	127	0	0.3
miles750 (128, 31)	126	0	0.1	126	0	0.1	126	0	0.1	127	1	17.9	126	0	1.5
queen11-11 (121, 11)	91	0	0.1	94	5	0.1	94	5	0.1	95	2	57.2	100	6	29.5
queen13-13 (169, 13)	134	0	0.1	138	3	0.1	138	1	0.2	141	5	175.6	141	3	86
queen5-5 (25, 5)	20	0	0.1	20	0	0	20	0	0.1	24	7	0.1	24	4	0.2
queen6-6 (36, 7)	31	0	0.1	32	1	0	32	1	0.1	33	13	2.4	32	2	0.6
queen7-7 (49, 7)	40	0	0.1	41	1	0	41	1	0.1	42	5	2	42	1	0.6
queen8-12 (96, 12)	88	0	0.1	87	2	0	87	2	0.1	92	3	41.4	91	5	22
queen8-8 (64, 9)	55	0	0.1	55	0	0	55	0	0.1	58	11	13.2	54	2	4
queen9-9 (81, 10)	66	0	0.1	67	1	0	67	1	0.1	72	10	34.5	73	3	9.2
myciel3 (11, 4)	11	0	0.1	11	0	0.1	11	0	0.1	11	0	0.1	11	0	0.1
myciel4 (23, 5)	23	0	0.1	23	0	0.1	23	0	0.1	23	0	0.1	23	0	0.1
myciel5 (47, 6)	47	0	0.1	47	0	0.1	47	0	0.1	47	0	0.1	47	0	0.1
myciel6 (95, 7)	95	0	0.1	95	0	0.1	95	0	0.1	95	0	0.1	95	0	0.1
myciel7 (191, 8)	191	0	0.1	191	0	0.1	191	0	0.1	191	0	0.1	191	0	0.1
TOTAL	10466	0	2.3	10527	2033	3.6	10524	1851	7.1	10550	159	14650.1	10504	64	13567.6

Table 4: Permanent regime with traffic rate r and $n = 50$ users at each time period

	LF			LF+SR			LF+SRP			LF+10TS		
r	#	ch	t	#	ch	t	#	ch	t	#	ch	t
1	46.6	0	0	46.8	1.1	0.1	46.8	1.1	0.2	49.5	1.8	19.4
2	47.0	0	0	47.9	6.2	0.1	47.9	6.2	0.2	49.9	2.9	22.8
3	47.1	0	0	48.6	16.2	0.1	48.6	15.0	0.2	49.7	3.2	25.5
4	47.9	0	0	48.8	16.9	0.1	48.8	16.3	0.2	50.0	3.7	27.3
5	47.7	0	0	48.8	23.2	0.1	48.8	21.6	0.2	49.9	4.8	33.4
6	48.2	0	0	49.5	25.9	0.1	49.5	24.5	0.1	49.8	4.8	28.6
7	48.1	0	0	49.8	37.6	0.1	49.8	46.0	0.1	50.0	5.2	31.2
8	48.0	0	0	49.8	50.2	0.1	49.8	43.2	0.1	50.0	4.3	28.6
9	48.7	0	0	49.9	43.7	0.1	49.9	37.7	0.1	50.0	5.5	32.3
10	48.5	0	0	50.0	44.4	0	50.0	44.9	0.1	50.0	4.4	24.8
11	49.3	0	0	50.0	61.9	0	50.0	59.3	0.1	50.0	4.3	24.7
12	48.9	0	0	50.0	44.0	0	50.0	37.4	0.1	50.0	3.8	21.4
13	48.7	0	0	50.0	55.7	0.1	49.9	50.8	0.1	50.0	3.9	24.3
14	48.6	0	0	50.0	41.4	0	50.0	40.6	0.1	50.0	3.9	20.4
15	49.1	0	0	50.0	48.5	0	50.0	45.6	0.1	50.0	3.6	21.5
Mean	48.2	0	0	49.3	34.5	0.1	49.3	32.7	0.1	49.9	4.0	25.7

References

[1] J. M. Lewis and M. Yannakakis, "The node-deletion problem for hereditary properties is np-complete," Journal of Computer and System Sciences, 20: 219–230, 1980.

[2] C. Lund and M. Yannakakis, "The approximation of maximum subgraph problems," in Automata, Languages and Programming, 20nd International Colloquium, ICALP93, Lund, Sweden, July 5-9, 1993, Proceedings, 40–51, 1993.

[3] G. Narasimhan, "The maximum k-colorable subgraph problem," Ph.D. dissertation, University of Wisconsin, Madison, 1989.

[4] T. Januschowski and M. Pfetsch, "The maximum k-colorable subgraph problem and orbitopes," Discrete Optimization, 8(3): 478–494, 2011.

[5] A. Hertz, R. Montagné, and F. Gagnon, "Constructive algorithms for the partial directed weighted improper coloring problem," Journal of Graph Algorithms and Applications, 20: 159–188, 2016.

[6] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. S. Naor, "A general approach to online network optimization problems," ACM Transactions on Algorithms (TALG), 2(4): 640–660, 2006.

[7] A. Gyárfás and J. Lehel, "On-line and first fit colorings of graphs." Journal of Graph theory, 12(2): 217–227, 1988.

[8] L. Lovász, M. Saks, and W. T. Trotter, "An on-line graph coloring algorithm with sublinear performance ratio," Discsrete Mathematics, 75(1-3): 319–325, 1989.

[9] B. Escoffier and P. Thomas, "On-line models and algorithms for max independent set," ANNALES DU LAMSADE N 2 Juin 2004, p. 219, 2004.

[10] L. M. Favrholdt and J. W. Mikkelsen, "Online dual edge coloring of paths and trees," in Approximation and Online Algorithms. Springer, 181–192, 2014.

[11] "Dimacs website." [Online]. Available: http://mat.gsia.cmu.edu/COLOR/instances.html

[12] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," The Computer Journal, 10(1): 85–86, 1967.

[13] D. Brélaz, "New methods to color the vertices of a graph," Communications of the ACM, 22(4): 251–256, 1979.

[14] F. T. Leighton, "A graph coloring algorithm for large scheduling problems," Journal of research of the national bureau of standards, 84(6): 489–506, 1979.

[15] G. Palubeckis, "On the recursive largest first algorithm for graph colouring," International Journal of Computer Mathematics, 85(2): 191–200, 2008.

[16] P. Hansen and J. Kuplinsky, "The smallest hard-to-color graph," Discrete mathematics, 96(3): 199–212, 1991.

[17] M. A. Zafer, "Channel assignment algorithms and blocking probability analysis for connection-oriented traffic in wireless networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[18] P. Sankowski, "Weighted bipartite matching in matrix multiplication time," in Automata, Languages and Programming. Springer, 274–285, 2006.

[19] F. Glover, "Future paths for integer programming and links to artificial intelligence," Computers and Operations Research, 13: 533–549, 1986.

[20] F. Glover and M. Laguna, Tabu Search. Springer, 2013.

[21] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," Computing, 39(4): 345–351, 1987.