WILEY InterScience® DISCOVER SOMETHING GREAT

Approximate $L(\delta_1, \delta_2, \ldots, \delta_t)$-Coloring of Trees and Interval Graphs

Alan A. Bertossi
Department of Computer Science, University of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy

Cristina M. Pinotti
Department of Computer Science and Mathematics, University of Perugia, Via Vanvitelli 1, 06123 Perugia, Italy

Given a vector $(\delta_1, \delta_2, \ldots, \delta_t)$ of nonincreasing positive integers, and an undirected graph $G = (V, E)$, an $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring of G is a function f from the vertex set V to a set of nonnegative integers such that $|f(u) - f(v)| \geq \delta_i$, if $d(u, v) = i$, $1 \leq i \leq t$, where $d(u, v)$ is the distance (i.e., the minimum number of edges) between the vertices u and v. An optimal $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring for G is one minimizing the largest integer used over all such colorings. Such a coloring problem has relevant applications in channel assignment for interference avoidance in wireless networks. This article presents efficient approximation algorithms for $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring of two relevant classes of graphs—trees, and interval graphs. Specifically, based on the notion of strongly simplicial vertices, $O(n(t+\delta_1))$ and $O(nt^2\delta_1)$ time algorithms are proposed to find α-approximate colorings on interval graphs and trees, respectively, where n is the number of vertices and α is a constant depending on t and $\delta_1, \ldots, \delta_t$. Moreover, an $O(n)$ time algorithm is given for the $L(\delta_1, \delta_2)$-coloring of unit interval graphs, which provides a 3-approximation. © 2007 Wiley Periodicals, Inc.
NETWORKS, Vol. 49(3), 204–216 2007

Keywords: wireless networks; channel assignment; $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring; interval graphs; trees; approximation algorithms

1. INTRODUCTION

In the *channel assignment* problem for wireless networks, the scarce radio spectrum is partitioned into a set of disjoint channels that can be used simultaneously by transmitting stations while maintaining acceptable radio signals. The same channel can be reused by two stations at the same time provided that no interference arises. However, the interference phenomena are so strong that even different channels assigned to two near stations must be sufficiently apart in the radio spectrum. To avoid such interference, a *separation vector* $(\delta_1, \delta_2, \ldots, \delta_t)$ of nonincreasing positive integers is introduced in such a way that channels assigned to interfering stations at distance i should be at least δ_i apart, with $1 \leq i \leq t$, while the same channel can be reused only at stations whose distance is larger than t [20]. The purpose of channel assignment algorithms is, therefore, to assign channels to the stations so that the above channel separations are satisfied and the difference between the highest and lowest channels assigned is kept as small as possible.

Formally, the channel assignment problem can be modeled as an appropriate coloring problem on an undirected graph $G = (V, E)$, representing the wireless network topology, whose vertices V correspond to stations, and edges E correspond to pairs of stations whose transmission areas intersect. Specifically, given a vector $(\delta_1, \delta_2, \ldots, \delta_t)$ of nonincreasing positive integers, and an undirected graph $G = (V, E)$, an $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring of G is a function f from the vertex set V to the set of nonnegative integers $\{0, \ldots, \lambda\}$ such that $|f(u) - f(v)| \geq \delta_i$ whenever $d(u, v) = i$, $1 \leq i \leq t$, where $d(u, v)$ is the distance (i.e., the minimum number of edges) between the vertices u and v. An *optimal* $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring for G is one minimizing the largest used color λ over all such colorings. Note that, because the set of colors includes 0, the overall number of colors involved in an optimal coloring f is, in fact, $\lambda + 1$ (although, due to the channel separation constraint, some colors in $\{1, \ldots, \lambda - 1\}$ might not be actually assigned to any vertex). Thus, the channel assignment problem consists of finding an optimal $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring for G.

When the separation vector $(\delta_1, \delta_2, \ldots, \delta_t)$ is equal to $(1, 1, \ldots, 1)$, the channel assignment problem has been widely studied in the past [2, 3, 14, 21, 23]. In particular, the intractability of optimal $L(1, \ldots, 1)$-coloring, for any positive integer t, has been proved by McCormick [21].

Received February 2006; accepted August 2006
Correspondence to: C. M. Pinotti; e-mail: pinotti@unipg.it
DOI 10.1002/net.20154
Published online in Wiley InterScience (www.interscience.wiley.com).
© 2007 Wiley Periodicals, Inc.

Optimal $L(1,\ldots,1)$-colorings, for any positive integer t, have been proposed in [3, 5] for rings, bidimensional grids, and honeycomb grids, and in [1] for trees and interval graphs. Moreover, when the separation vector is equal to $(\delta_1, 1, \ldots, 1)$, optimal $L(\delta_1, 1, \ldots, 1)$-colorings have been proposed in [6, 24] for rings, bidimensional grids, and cellular grids. Optimal solutions for the $L(\delta_1, \delta_2)$-coloring problem on bidimensional grids, and cellular grids have been given by Van Den Heuvel et al. [25], who also provided an optimal $L(2, 1, 1)$-coloring for bidimensional grids. The $L(2, 1, 1)$-coloring problem has also been optimally solved for cellular grids, honeycomb grids, and rings in [6]. The $L(2, 1)$-coloring has been investigated in [7, 13, 19, 22]. In particular, solutions for the $L(2, 1)$-coloring of unit interval graphs and trees have been found, respectively, by Sakai [22] and by Chang and Kuo [13]. Approximate solutions for outerplanar, permutation, and split graphs have also been presented by Bodlaender et al. [7]. Zhou et al. [26] provided polynomial time algorithms for optimally $L(1, 1)$-coloring graphs with bounded treewidth. Georges and Mauro [16] gave bounds on the $L(\delta_1, \delta_2)$-coloring of some trees with maximum degree no larger than $\frac{\delta_1}{\delta_2}$ or no smaller than three [17], while Calamoneri et al. [11] considered the $L(\delta_1, \delta_2)$-coloring of trees in the case that $\delta_1 < \delta_2$. Moreover, bounds for the $L(\delta_1, 1)$-coloring of k-trees were provided by Chang et al. [12]. Finally, a recent annotated bibliography on the $L(\delta_1, \delta_2)$-coloring problem for several special classes of graphs can be found in [10].

This article investigates the channel assignment problem for general separation vectors on some specific classes of graphs—trees and interval graphs—which occur in modeling realistic wireless network topologies. Indeed, trees model hierarchical topologies, while interval graphs model wireless networks serving narrow surfaces, like highways or valleys confined by natural barriers (e.g., mountains or lakes). It is still unknown whether finding optimal $L(\delta_1, \ldots, \delta_t)$-colorings of trees or interval graphs is polynomially time solvable or not. Some authors conjecture that even the $L(\delta_1, \delta_2)$-coloring problem with $\delta_2 > 1$ is *NP*-hard for trees and unit interval graphs [10]. In the rest of this article, the following original results will be proposed, which extend those previously presented in [4] for the $L(1,\ldots,1)$-coloring problem of trees and interval graphs. First, the notions of t-simplicial and strongly simplicial vertices of a graph are recalled in Section 2. Then, Sections 3 and 4 present two polynomial time algorithms to approximate the $L(\delta_1, \ldots, \delta_t)$-coloring problem on interval graphs and trees, respectively. Such algorithms run in $O(n(t + \delta_1))$ and $O(nt^2\delta_1)$ time, respectively, where n is the number of vertices. A better approximate solution for $L(\delta_1, \delta_2)$-coloring of unit interval graphs is also given in Subsection 3.2, which generalizes that proposed in [22] for the $L(2, 1)$-coloring problem. Finally, conclusions are offered in Section 5.

2. PRELIMINARIES

Throughout this article, it is assumed that G is a connected undirected graph with at least two vertices, and that the separations satisfy $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_t$. When $\delta_1 = \delta_2 = \cdots = \delta_t = 1$, the $L(1,\ldots,1)$-coloring problem reduces to the classical vertex coloring problem on the t-th *power* G^t of the graph G. The vertex set of G^t is the same as the vertex set of G, while the edge uv belongs to the edge set of G^t if and only if the distance $d(u, v)$ between the vertices u and v in G is at most t. Now, colors must be assigned to the vertices of G^t so that every pair of vertices connected by an edge are assigned different colors and the minimum number of colors is used. Hence, the role of *maximum cliques* in G^t is apparent for deriving lower bounds on the minimum number of channels for the $L(1,\ldots,1)$-coloring problem on G. A *clique* K for G^t is a subset of vertices of G^t such that for each pair of vertices in K there is an edge. Clearly, a clique of size k in the power graph G^t implies that at least k different colors are needed to color G^t. In other words, the size of the largest clique in G^t is a lower bound for the $L(1,\ldots,1)$-coloring problem on G. Of course, a lower bound for the $L(1,\ldots,1)$-coloring problem is also a lower bound for the $L(\delta_1, 1, \ldots, 1)$-coloring problem, with $\delta_1 \geq 1$.

For any value of $t \leq |V|$, let $\lambda^*_{G,t}$ denote the minimum value of λ over all the $L(1,\ldots,1)$-colorings $f : V \to \{0, \ldots, \lambda\}$ of $G = (V, E)$. Note that: (i) $\lambda^*_{G,1} \geq 1$, because G is assumed to be connected and to have at least two vertices; (ii) $\lambda^*_{G,t} = \lambda^*_{G',1}$; and (iii) $\lambda^*_{G,t} + 1$ is at least as large as the size $\omega_{G'}$ of the largest clique in G^t.

Lemma 1. *Any $L(\delta_1, \delta_2, \cdots, \delta_t)$-coloring requires at least $max_{1\leq j\leq t}\{\delta_j\lambda^*_{G,j}\}$ as the largest color.*

Proof. Because $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_t$, any $L(\delta_1, \delta_2, \ldots, \delta_j)$-coloring, for any value of $j \leq t$, requires at least as many colors as any $L(\delta_j, \delta_j, \ldots, \delta_j)$-coloring, which, in turn, requires at least $\delta_j\lambda^*_{G,j}$ as the largest color. ∎

Given $G = (V, E)$, let S be a subset of V, and let $G[S]$ denote the subgraph of G induced by S, that is, $G[S] = (S, \{uv \in E : u, v \in S\})$. A vertex x of G is called *t-simplicial* when, for every pair of vertices u and v such that $d(x, u) \leq t$ and $d(x, v) \leq t$, it holds also that $d(u, v) \leq t$. A vertex x is called *strongly simplicial* when x is t-simplicial for any value of t. Moreover, let $N_t(v)$ denote all the vertices at distance at most t from v, that is, $N_t(v) = \{u \in V : d(u, v) \leq t\}$.

Lemma 2 ([4]). *Given $G = (V, E)$ and an integer t, let v be a t-simplicial vertex of G. Consider $G' = G[V - \{v\}]$ and let f' be an optimal $L(1,\ldots,1)$-coloring of G' using $\lambda^*_{G',t}$ as the largest color. Define an $L(1,\ldots,1)$-coloring f of V extending f' to v so that*

$$f(x) = \begin{cases} \min\{i : i \neq f'(u) \text{ for each } u \in G' \\ \quad \text{with } d(u, v) \leq t\} & \text{if } x = v, \\ f'(x) & \text{if } x \in V - \{v\}. \end{cases}$$

Then f is an optimal $L(1,\ldots,1)$-coloring for G.

Proof. Clearly, $\lambda^*_{G,t} \geq \lambda^*_{G',t}$. If $f(v) \leq \lambda^*_{G',t}$, then f is trivially optimal. Assume, therefore, that $f(v) > \lambda^*_{G',t}$. Then

$\{f'(u): u \in N_t(v) - \{v\}\} = \{0, \ldots, \lambda^*_{G',t}\}$. Because v is t-simplicial, any two vertices in $N_t(v)$ are at distance at most t, and hence, $N_t(v)$ is a clique of G^t. Because $|N_t(v)| = \lambda^*_{G',t} + 2$ and $f(v) = \lambda^*_{G',t} + 1$, then f is optimal. ∎

Note that verifying whether a vertex is t-simplicial or not can be done in polynomial time [1]. Therefore, Lemma 2 implies the existence of an algorithm that optimally solves in polynomial time the $L(1, \ldots, 1)$-coloring problem using exactly $\omega_{G'}$ colors for any class of graphs closed under taking induced subgraphs and with the property that every graph of that class has a t-simplicial vertex. The next lemma shows that there is always an $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring for such a class of graphs where the largest used color is bounded from above by a function of the clique sizes $\omega_{G'}$ and of the separations δ_j, with $1 \leq j \leq t$.

Lemma 3. *Given $G = (V, E)$ and t, let v be a strongly simplicial vertex of G, and consider $G' = G[V - \{v\}]$. Then there is an $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring such that $f(v) = c$, where $c \in \{0, 1, \ldots, \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}\}$.*

Proof. Because v is strongly simplicial, any two vertices in $N_j(v)$ are at distance at most j, for every $1 \leq j \leq t$. Hence, $N_t(v)$ is a clique of G^t, and thus $|N_t(v)| \leq \omega_{G'}$. Therefore, at most, $|N_t(v)| - 1 \leq \lambda^*_{G,t}$ colors have been used for $N_t(v) - \{v\}$. Each of them forbids $2(\delta_t - 1)$ colors due to the δ_t-separation constraint, and overall $2(\delta_t - 1)\lambda^*_{G,t}$ colors are forbidden. Moreover, for any $1 \leq j \leq t-1$, v is j-simplicial, and hence, $N_j(v)$ is a clique of G^j, and $|N_j(v)| - 1 \leq \lambda^*_{G,j}$. Because each color assigned to a vertex of $N_j(v) - \{v\}$ forbids $2(\delta_j - \delta_{j+1})$ colors, $2(\delta_j - \delta_{j+1})\lambda^*_{G,j}$ colors are forbidden due to the δ_j-separation constraint. Before coloring v, the total number of used and forbidden colors is $\lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}$. Therefore, there is at least an available color c in $\{0, 1, \ldots, \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}\}$ that can be assigned to v. ∎

In the next two sections, the above properties will be exploited to derive approximate solutions for the $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring problem on two classes of graphs: interval graphs and trees.

3. INTERVAL GRAPHS

A graph $G = (V, E)$ is termed an *interval graph* if it has an *interval representation*, namely, if each vertex of V can be represented by an interval of the real line such that there is an edge $uv \in E$ if and only if the intervals corresponding to u and v intersect. More formally, let the graph $G = (V, E)$ have n vertices. Two integers l_v and r_v, with $l_v < r_v$, (the *interval endpoints*) are associated with every vertex v of G, and there is an edge $uv \in E$ if and only if $l_u < l_v < r_u$ or $l_u < r_v < r_u$. Without loss of generality, one can assume that all the $2n$ interval endpoints are distinct and are indexed from 1 to $2n$.

Several alternative characterizations of interval graphs have been proposed so far in the literature [9]. Polynomial time algorithms to recognize interval graphs and compute their interval representations are known [8, 15]. Polynomial time algorithms are also known for the classical vertex coloring problem on interval graphs [9]. Because it is known that a power of an interval graph is also an interval graph, the $L(1, \ldots, 1)$-coloring problem for an interval graph G can be solved in polynomial time [1] by coloring the interval graph G^t.

The following lemma, which is useful to find an approximate $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring, shows how to locate a strongly simplicial vertex of an interval graph.

Lemma 4 ([4])**.** *In an interval graph, the vertex with maximum left endpoint is strongly simplicial.*

Proof. Let $G = (V, E)$ be an interval graph with n vertices, and consider its interval representation. Let x be the vertex of G whose left endpoint l_x is maximum. Consider two vertices u and v such that $d(u, x) \leq t$ and $d(v, x) \leq t$. Without loss of generality, let $l_u < l_v < l_x$. Because there is a shortest path $sp(u, x)$ between u and x, there must be a vertex $w \in sp(u, x) - \{x\}$ such that $l_w \leq l_v < r_w$. Therefore, $d(u, v) \leq d(u, w) + 1 \leq d(u, x) \leq t$, and vertex x is t-simplicial. Because such a condition holds for any $t \leq n$, vertex x is strongly simplicial. ∎

Lemma 4 suggests that one can scan the vertices of an interval graph by increasing left endpoints, because in this way, a t-simplicial vertex v of the induced subgraph $G[\{1, \ldots, v\}]$ is processed at each step, for $1 \leq v \leq n$.

3.1. Approximate $L(\delta_1, \ldots, \delta_t)$-Coloring of Interval Graphs

In this subsection, an $O(n(t + \delta_1))$ time algorithm is proposed to find an approximate $L(\delta_1, \ldots, \delta_t)$-coloring of interval graphs.

Consider the interval representation of G, and assume that the intervals (vertices) are indexed by increasing left endpoints, namely $l_1 < l_2 < \cdots < l_n$. For each endpoint k, an interval v is called *open* if $l_v \leq k < r_v$ and *deepest* if it is open and its right endpoint is maximum.

The algorithm maintains a family of $t + 1$ sets of colors, called *palettes*, denoted by $P_0, P_1, \ldots, P_t$. The palette P_0 is initialized to the set of colors $\{0, 1, \ldots, U\}$, where $U = \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}$. As shown by Lemma 3, such a color set is sufficiently large to obtain a legal $L(\delta_1, \ldots, \delta_t)$-coloring for G. The palette P_0 contains the readily usable colors. A color can leave P_0 because it is assigned to an interval. In such a case, the color is inserted into P_t, and it will go downward through all the previous palettes before being reusable. Precisely, the palette P_t includes the colors used for the currently open intervals, while the generic palette P_i, with $1 \leq i \leq t-1$, contains the colors that could be reused as soon as all the next i consecutive deepest intervals will be ended. A color can leave P_0 without being assigned to any interval just because another used color forbids it.

Algorithm Interval-Coloring $(G = (V, E), t, \delta_1, \ldots, \delta_t)$;

$U := \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}$;
$L_v := \emptyset$ for every vertex $v = 1, \ldots, n$;
$P_0 := \{0, 1, \ldots, U\}$ and $P_j := \emptyset$ for $j = 1, \ldots, t$;
TABOO$[\gamma] := 0$ for $\gamma = 0, \ldots, U$;
MAX $:= 0$;
$\delta_{t+1} := 0$;
for $k := 1$ to $2n$ do
  if $k = l_v$ for some v, then
    extract a color c from P_0;
    $f(v) := c$;
    for each color $\max\{0, c - \delta_1 + 1\} \le \gamma \le \min\{c + \delta_1 - 1, U\}$ do
      if $\gamma \in P_0$ then extract γ from P_0;
      TABOO$[\gamma]$:= TABOO$[\gamma]$+1;
    insert color c into both L_v and P_t;
    if $r_v >$ MAX then set MAX $:= r_v$ and DEEP $:= v$;
  otherwise, if $k = r_v$ for some v, then
    for each color c in L_v do
      let j be such that $c \in P_j$;
      extract c from P_j;
      for each color $\max\{0, c - \delta_{t-j+1} + 1\} \le \gamma \le \max\{0, c - \delta_{t-j+2}\}$
      or $\min\{c + \delta_{t-j+2}, U\} \le \gamma \le \min\{0, c + \delta_{t-j+1} - 1, U\}$ do
        if $\gamma \neq c$ then TABOO$[\gamma]$:= TABOO$[\gamma]-1$;
        if TABOO$[\gamma] = 0$ then insert γ into P_0;
      if $j > 1$ then
        insert c into both P_{j-1} and L_{DEEP}
      else
        TABOO$[c]$:= TABOO$[c]-1$;
        if TABOO$[c]$=0 then insert c into P_0;

FIG. 1. The approximate algorithm for $L(\delta_1, \ldots, \delta_t)$-coloring of interval graphs.

A counter is used to keep track of how many used colors currently forbid it.

Figure 1 illustrates the algorithm for $L(\delta_1, \ldots, \delta_t)$-coloring of interval graphs, called *Interval-Coloring*. The algorithm scans the $2n$ interval endpoints from left to right. Whenever a new interval v begins (that is, a left endpoint l_v is encountered), v is colored by a color c extracted from the palette P_0 and, if needed, the deepest interval is updated. The used color c is put both in the palette P_t and in the set L_v of colors that depend on vertex v. Moreover, all the colors γ with $|\gamma - c| < \delta_1$ are forbidden by c, and thus their counters are incremented. Whenever an interval v ends (that is, a right endpoint r_v is encountered), every color c belonging to L_v is deleted from its current palette, say P_j. Because the δ_{t-j+1}-separation constraint due to c does not hold any more, all the colors γ with $\delta_{t-j+2} \le |\gamma - c| < \delta_{t-j+1}$ are no longer forbidden by c, and their counters are decremented. A color γ becomes available whenever its counter reaches 0, and in such a case it is reinserted in P_0. Whenever j is larger than 1, the color c, previously extracted from P_j, is moved to palette P_{j-1} and inserted in the set L_{DEEP} of the colors which depend on the current deepest interval, which is maintained in the DEEP variable. If $j = 1$, the color c becomes reusable and is inserted into P_0, provided that its counter becomes 0.

Lemma 5. *Consider the Interval-Coloring algorithm at the beginning of iteration k with $k = l_v$ for some interval v to be colored. Consider any color $c \in P_j$, and let w be the rightmost interval colored by c. Then v is at distance $t - j + 1$ from w; that is, $d(w, v) = t - j + 1$.*

Proof. Note that an interval u is colored as soon as its left endpoint is reached and $f(u)$ is set. If $j = t$, the interval w is still open, because otherwise c would have been extracted from P_t when $k = r_w$. Therefore, $l_w < l_v < r_w$ and $d(w, v) = 1 = t - t + 1$. If $0 \le j < t$, then w has ended, and c has been moved through the lists L_{DEEP} of $t - j$ consecutive deepest intervals, because the deepest intervals form a shortest path from w to v, $d(w, v) = t - j + 1$. ∎

Lemma 6. *Consider the Interval-Coloring algorithm at the beginning of iteration k, and let c be any color.*

- *If $c \in P_0$, then c is readily usable, and it does not forbid any other color;*

- *If $c \in P_j$ with $j > 0$, then c cannot be used, and it forbids all the colors γ such that $c - \delta_{t-j+1} + 1 \le \gamma \le c + \delta_{t-j+1} - 1$;*
- *If $c \notin P_0 \cup \cdots \cup P_t$, then it is forbidden, but it does not forbid any other color.*

Proof. If $c \in P_0$, then TABOO$[c] = 0$. The algorithm can work on c only when $k = l_v$. By Lemma 5, $d(w, v) = t + 1$, and thus c can be assigned to v and it does not forbid any other color.

If $c \in P_j$ with $j > 0$, then $d(w, v) = t - j + 1$ by Lemma 5, and thus c cannot be yet assigned to any interval. When c was inserted into P_t, each color γ with $|\gamma - c| < \delta_1$ was forbidden by incrementing its counter TABOO$[\gamma]$. Moving c downward through the palettes, the distance between w and the yet uncolored intervals increases by Lemma 5, and thus the counters of the farthest colors are decremented. Precisely, when c moves from P_{j+1} to P_j, the counter TABOO$[\gamma]$ is decremented for each color γ with $\delta_{t-j+1} \le |\gamma - c| < \delta_{t-j}$. Hence, only the counters for the colors in $[c - \delta_{t-j+1} + 1, \ldots, c + \delta_{t-j+1} - 1]$ still remain to be decremented.

Finally, if c does not belong to any palette, then two cases may arise. In the first case, c was extracted from P_1, the counters of all its remaining forbidden colors were decremented, but c was not inserted into P_0 because its counter was greater than zero. In the second case, c was extracted from P_0 because its counter was incremented by another color. In both cases, by Lemma 5, the distance between w, and the yet uncolored intervals is larger than t and c does not forbid any other color. Assume that TABOO$[c] = h$. Then, there are h colors, each belonging to some palette P_j, with $j > 0$, that incremented TABOO$[c]$. Each of these colors will decrement TABOO$[c]$ no later than when it will be extracted from P_1. Hence, as soon as TABOO$[c]$ becomes 0, c will be reinserted into P_0. ∎

Lemma 7. *The largest color used by the Interval-Coloring algorithm is at most $U = \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}$.*

Proof. By Lemma 3, there is an $L(\delta_1, \ldots, \delta_t)$-coloring of G using the colors $\{0, \ldots, U\}$. To show that the Interval-Coloring algorithm succeeds in coloring using so many colors, one needs to prove that there is an available color in P_0 whenever an interval v has to be colored. For such a goal, by Lemma 6, observe that each color belonging to P_j, with $j > 0$, forbids $2(\delta_{t-j+1} - 1) + 1$ colors, itself included. Note that each forbidden color not included in any palette is already taken into account because it is forbidden by some color in the palettes. Hence, at any iteration k of the algorithm, the overall number of forbidden and used colors is at most $\sum_{j=1}^{t}(2(\delta_{t-j+1} - 1) + 1)|P_j|$. The above formula can be rewritten as

$$\begin{aligned}
&\sum_{j=1}^{t} |P_j| + \sum_{j=1}^{t} 2(\delta_{t-j+1} - 1)|P_j| \\
&\quad \le \lambda^*_{G,t} + \sum_{j=1}^{t} 2(\delta_j - 1)|P_{t-j+1}| \\
&\quad = \lambda^*_{G,t} + 2(\delta_t - 1)\sum_{j=1}^{t} |P_j| + \sum_{j=2}^{t} 2(\delta_{t-j+1} - \delta_t)|P_j| \\
&\quad = \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + 2(\delta_{t-1} - \delta_t)\sum_{j=2}^{t} |P_j| \\
&\qquad + \sum_{j=3}^{t} 2(\delta_{t-j+1} - \delta_{t-1})|P_j| \\
&\qquad \cdots \\
&\quad = \lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j} = U.
\end{aligned}$$

Because there are $U + 1$ colors overall, out of which at most U are used or forbidden, there is always an available color in P_0 that can be assigned to v. ∎

Theorem 1. *Let $\delta_m \lambda^*_{G,m} = max_{1 \le j \le t}\{\delta_j \lambda^*_{G,j}\}$, and recall that $\delta_{t+1} = 0$. The Interval-Coloring algorithm gives an α-approximation with $\alpha = min\ \{2t, \frac{2\delta_{m+1}-1}{\delta_t} + \frac{2(\delta_1 - \delta_{m+1})}{\delta_m}\}$.*

Proof. To find the approximation factor, the ratio $\alpha = \frac{U}{L}$ between the upper bound U given by Lemma 7 and the lower bound L given by Lemma 1 has to be evaluated. Precisely, such a ratio is

$$\frac{U}{L} = \frac{\lambda^*_{G,t} + 2(\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j}}{\delta_m \lambda^*_{G,m}}$$

Although the ratio can be evaluated exactly because all the values of $\lambda^*_{G,j} = \lambda^*_{G^j,1}$ can be computed in polynomial time [1], U/L can be bounded from above by a constant, which depends only on t, δ_1, δ_m, δ_{m+1}, and δ_t. Because $L = \delta_m \lambda^*_{G,m}$, one has $\frac{\lambda^*_{G,j}}{\delta_m \lambda^*_{G,m}} \le \frac{1}{\delta_j}$. Then

$$\begin{aligned}
\frac{U}{L} &\le 2\sum_{j=1}^{t-1} \frac{\delta_j - \delta_{j+1}}{\delta_j} + \frac{(2\delta_t - 1)\lambda^*_{G,t}}{\delta_m \lambda^*_{G,m}} \\
&\le 2\sum_{j=1}^{t-1} \left(1 - \frac{\delta_{j+1}}{\delta_j}\right) + \frac{2\delta_t \lambda^*_{G,t}}{\delta_m \lambda^*_{G,m}} \\
&\le 2(t-1) + 2 \\
&= 2t.
\end{aligned}$$

Moreover, assuming $m \le t - 1$, one can bound U as follows:

$$\begin{aligned}
U &= (2\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{m} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j} \\
&\qquad + \sum_{j=m+1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,j} \\
&\le (2\delta_t - 1)\lambda^*_{G,t} + \lambda^*_{G,m} \sum_{j=1}^{m} 2(\delta_j - \delta_{j+1}) \\
&\qquad + \lambda^*_{G,t} \sum_{j=m+1}^{t-1} 2(\delta_j - \delta_{j+1})
\end{aligned}$$

$$= (2\delta_t - 1)\lambda^*_{G,t} + \lambda^*_{G,m}2(\delta_1 - \delta_{m+1}) + \lambda^*_{G,t}2(\delta_{m+1} - \delta_t)$$
$$= (2\delta_{m+1} - 1)\lambda^*_{G,t} + 2(\delta_1 - \delta_{m+1})\lambda^*_{G,m}.$$

Therefore, recalling that $\frac{\lambda^*_{G,t}}{\delta_m \lambda^*_{G,m}} \leq \frac{1}{\delta_t}$, it follows that

$$\frac{U}{L} \leq \frac{(2\delta_{m+1} - 1)\lambda^*_{G,t} + 2(\delta_1 - \delta_{m+1})\lambda^*_{G,m}}{\delta_m \lambda^*_{G,m}}$$
$$\leq \frac{2\delta_{m+1} - 1}{\delta_t} + \frac{2(\delta_1 - \delta_{m+1})}{\delta_m}.$$

Finally, if $m = t$, then

$$\frac{U}{L} \leq \frac{(2\delta_t - 1)\lambda^*_{G,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{G,t-1}}{\delta_t \lambda^*_{G,t}}$$
$$\leq \frac{(2\delta_t - 1)\lambda^*_{G,t} + 2(\delta_1 - \delta_t)\lambda^*_{G,t}}{\delta_t \lambda^*_{G,t}}$$
$$= \frac{2\delta_1 - 1}{\delta_t} = \frac{2\delta_{t+1} - 1}{\delta_t} + \frac{2(\delta_1 - \delta_{t+1})}{\delta_t}. \quad \blacksquare$$

It is worth noting that the Interval-Coloring algorithm provides a 4-approximation for the $L(\delta_1, \delta_2)$-coloring problem, as one can easily check by setting $t = 2$ in the formula for α given by Theorem 1. However, a better approximation can be found, even for arbitrary t, when δ_1 can be the only separation greater than 1.

Corollary 1 ([4]). *When $\delta_2 = \cdots = \delta_t = 1$, the Interval-Coloring algorithm yields a 3-approximate $L(\delta_1, 1, \ldots, 1)$-coloring.*

Proof. When the separation vector is $(\delta_1, 1, \ldots, 1)$, the ratio between the upper bound and the lower bound simplifies as follows:

$$\frac{U}{L} = \frac{\lambda^*_{G,t} + 2(\delta_1 - 1)\lambda^*_{G,1}}{\max\left\{\delta_1 \lambda^*_{G,1}, \lambda^*_{G,t}\right\}}.$$

If $\delta_1 \lambda^*_{G,1} \geq \lambda^*_{G,t}$, the above ratio becomes

$$\frac{U}{L} = \frac{\lambda^*_{G,t} + 2(\delta_1 - 1)\lambda^*_{G,1}}{\delta_1 \lambda^*_{G,1}}$$
$$\leq \frac{3\delta_1 \lambda^*_{G,1} - 2\lambda^*_{G,1}}{\delta_1 \lambda^*_{G,1}} \leq 3 - \frac{2}{\delta_1} \leq 3.$$

If $\delta_1 \lambda^*_{G,1} < \lambda^*_{G,t}$, the ratio is

$$\frac{U}{L} = \frac{\lambda^*_{G,t} + 2(\delta_1 - 1)\lambda^*_{G,1}}{\lambda^*_{G,t}}$$
$$\leq \frac{3\lambda^*_{G,t} - 2\lambda^*_{G,1}}{\lambda^*_{G,t}} \leq 3 - 2\frac{\lambda^*_{G,1}}{\lambda^*_{G,t}} \leq 3. \quad \blacksquare$$

Theorem 2. *The Interval-Coloring algorithm runs in $O(n(t + \delta_1))$ time.*

Proof. The algorithm starts by computing the upper bound U, which depends on $\lambda^*_{G,j}$, for $1 \leq j \leq t$. Recalling that $\lambda^*_{G^j,1} = \lambda^*_{G,j}$, one needs to compute all the power graphs G^j, with $1 \leq j \leq t$, which can be done in $O(nt)$ time [1]. Given G^j, $\lambda^*_{G^j,1}$ can be computed in $O(n)$ time [18]. Therefore, the computation of all the $\lambda^*_{G,j}$, and hence, of U, needs $O(nt)$ time. Because $U = O(n\delta_1)$, the initialization of P_0 and TABOO takes $O(n\delta_1)$ time.

To perform insertions and extractions in constant time, all the palettes P_i, $0 \leq i \leq t$ and all the sets L_v, with $1 \leq v \leq n$, are implemented by doubly linked lists. At any iteration, any color c can belong at most to one palette and to one set L_{DEEP}. Thus, because no more than U colors can be used, a vector C, indexed by colors, can be maintained. Each vector entry $C[c]$ stores the current palette index j, to which c belongs, along with a pointer to the position of c within the doubly linked list P_j. When a color c is inserted in either a palette P_j or a set of colors L_{DEEP}, such a color is added at the front of the list and the vector entry $C[c]$ is accordingly updated. Overall, an insertion takes constant time. The extraction of color c from a palette P_j is performed in constant time by retrieving from $C[c]$ the pointer to c within the list P_j. In the latter case, the vector entry $C[c]$ is also accordingly updated in constant time.

To evaluate the overall time complexity, observe that the algorithm consists of $2n$ iterations and, at every iteration k, each step takes $O(\delta_1)$ time, except the scan of list L_v whenever interval v ends. Each color c, after being assigned to a vertex, goes through t lists L_{DEEP} before being reassigned to another vertex. In fact, every time c moves from palette P_j to P_{j-1}, the distance between the last vertex colored c and the uncolored vertices increases by one as shown in Lemma 5. Hence, between two consecutive assignments of the same color c, there are at most $t + 1$ moves, each performed in a different iteration. Overall such $t + 1$ moves take $O(t)$ time for insertions and extractions, and $O(\delta_1)$ time for updating the TABOO counters. Let m_c be the overall number of vertices of G colored c. Therefore, the total time required by color c is $O((t+\delta_1)m_c)$. Summing up over all the colors assigned to the intervals, the overall time is $\sum_c O((t+\delta_1)m_c) = O(n(t+\delta_1))$, because $\sum_c m_c = n$. In conclusion, the algorithm takes $O(n(t + \delta_1))$ time provided that the interval representation of G is available and the $2n$ interval endpoints are sorted. $\blacksquare$

3.2. Approximate $L(\delta_1, \delta_2)$-Coloring of Unit Interval Graphs

This subsection deals with the $L(\delta_1, \delta_2)$-coloring problem on the class of *unit interval graphs*. This is a subclass of interval graphs in which all the intervals have the same length, or equivalently, for which no interval is properly contained within another. Recalling that vertices are assumed to be indexed by increasing left endpoints, the main property of unit interval graphs is that whenever $v < u$ and $vu \in E$, then the vertex set $\{v, v + 1, \ldots, u - 1, u\}$ forms a clique and $u \leq v + \lambda^*_{G,1}$ (as a consequence, the maximum vertex w at distance 2 from v satisfies $w \leq v + 2\lambda^*_{G,1}$).

Algorithm Unit-Interval-Coloring $(G = (V, E), \delta_1, \delta_2)$;

if $\delta_1 > 2\delta_2$ then
  for $v := 1$ to n do
    $p := (v - 1) \bmod (2\lambda^*_{G,1} + 2)$;
    $$f(v) := \begin{cases} \delta_1(\lambda^*_{G,1} - p) & \text{if } 0 \le p \le \lambda^*_{G,1} \\ \delta_1(\lambda^*_{G,1} - p) + \delta_2 & \text{if } \lambda^*_{G,1} + 1 \le p \le 2\lambda^*_{G,1} + 1 \end{cases}$$
if $\delta_1 \le 2\delta_2$ then
  for $v := 1$ to n do
    $f(v) := (2\delta_2(v - 1)) \bmod (2\delta_2\lambda^*_{G,1} + 3\delta_2)$;

FIG. 2. The $L(\delta_1, \delta_2)$-coloring algorithm for unit interval graphs.

In this subsection, it is assumed that the unit interval graph to be colored is not a path, because otherwise the optimal $L(\delta_1, \delta_2)$-coloring algorithm in [25] can be applied. In Figure 2, a linear time algorithm, called Unit-Interval-Coloring, is presented. The algorithm distinguishes two cases and uses either at most δ_2 additional colors with respect to the optimum when $\delta_1 > 2\delta_2$, or at most $2\delta_2$ additional colors when $\delta_1 \le 2\delta_2$.

Theorem 3. *The Unit-Interval-Coloring algorithm finds a* 3*-approximate* $L(\delta_1, \delta_2)$*-coloring.*

Proof. Let us start by showing that the δ_1- and δ_2-separation constraints are satisfied.

When $\delta_1 > 2\delta_2$, the algorithm colors the vertices by repeating the following sequence of colors of length $2\lambda^*_{G,1}+2$:

$$0, \delta_1, 2\delta_1, \ldots, \lambda^*_{G,1}\delta_1, \delta_2, \delta_1 + \delta_2, \delta_1 + 2\delta_2, \ldots, \lambda^*_{G,1}\delta_1 + \delta_2.$$

Consider a vertex v colored $c = j\delta_1$, with $0 \le j \le \lambda^*_{G,1}$ (analogous reasoning holds when $c = j\delta_1 + \delta_2$). First of all, the color c is used exactly once within the sequence. Thus, if c is assigned to vertex v, then it is reused at vertex $v+2\lambda^*_{G,1}+2$, which is at distance at least 3 from v, because otherwise $\lambda^*_{G,1}$ would not be optimal. To satisfy the δ_1-separation constraint, it remains to check that all the colors $c - \delta_1 + 1, \ldots, c - 1$, $c + 1, \ldots, c + \delta_1 - 1$ cannot be reused for any vertex at distance 1 from v. Among such colors, only the color $c + \delta_2$ is used in the sequence, and it is assigned to the vertices $v \pm (\lambda^*_{G,1} + 1)$, as one can easily check by inspecting the sequence above. The vertices v and $v \pm (\lambda^*_{G,1} + 1)$ cannot be adjacent, because otherwise there would be a clique of size $\lambda^*_{G,1} + 2$, including vertices $v, v \pm 1, \ldots, v \pm (\lambda^*_{G,1} + 1)$, which contradicts the optimality of $\lambda^*_{G,1}$. Moreover, the δ_2-separation constraint trivially follows from the fact that the colors $c - \delta_2 + 1, \ldots, c - 1, c + 1, \ldots, c + \delta_2 - 1$ are never used.

When $\delta_1 \le 2\delta_2$, the algorithm colors the vertices by repeating the following sequence of colors of length $2\lambda^*_{G,1}+3$:

$$0, 2\delta_2, 4\delta_2, \ldots, 2(\lambda^*_{G,1} + 1)\delta_2, \delta_2, 3\delta_2, 5\delta_2, \ldots, 2\lambda^*_{G,1}\delta_2 + \delta_2.$$

Consider, again, a vertex v colored $f(v) = c$. As in the previous case, the color c is used exactly once within the sequence. Thus, if c is assigned to vertex v, then it is reused at vertex $v + 2\lambda^*_{G,1} + 3$, which cannot be at distance 1 or 2 from v. With regard to the δ_1-separation constraint, note that, other than c, only the colors $c \pm \delta_2$ are used in the sequence. They are assigned to vertices $v \mp (\lambda^*_{G,1} + 1)$, as one can easily check by computing $f(v \mp (\lambda^*_{G,1} + 1))$. Those vertices cannot be adjacent because otherwise the vertices $v, v \mp 1, \ldots, v \mp (\lambda^*_{G,1} + 1)$ would form a clique, contradicting the optimality of $\lambda^*_{G,1}$. Furthermore, the colors $c - \delta_2 + 1, \ldots, c - 1, c + 1, \ldots, c + \delta_2 - 1$ are never used, and hence, the δ_2-separation constraint holds too.

To find the approximation, observe that, by Lemma 1, the largest color used by any $L(\delta_1, \delta_2)$-coloring is at least $L = \max\{\delta_1\lambda^*_{G,1}, \delta_2\lambda^*_{G,2}\}$. When $\delta_1 > 2\delta_2$, L becomes $\delta_1\lambda^*_{G,1}$ because $\delta_2\lambda^*_{G,2} \le 2\delta_2\lambda^*_{G,1} < \delta_1\lambda^*_{G,1}$. In contrast, when $\delta_1 \le 2\delta_2$, L can be either $\delta_1\lambda^*_{G,1}$ or $\delta_2\lambda^*_{G,2}$. On the other hand, the maximum color U used by the algorithm is $\delta_1\lambda^*_{G,1} + \delta_2$ when $\delta_1 > 2\delta_2$, and $2\delta_2\lambda^*_{G,1} + 3\delta_2$ when $\delta_1 \le 2\delta_2$. Therefore

$$\frac{U}{L} = \begin{cases} \frac{\delta_1\lambda^*_{G,1} + \delta_2}{\delta_1\lambda^*_{G,1}} & \text{if } \delta_1 > 2\delta_2, \\ \frac{2\delta_2\lambda^*_{G,1} + 2\delta_2}{\max\{\delta_1\lambda^*_{G,1}, \delta_2\lambda^*_{G,2}\}} & \text{if } \delta_1 \le 2\delta_2. \end{cases}$$

Although the ratio U/L can be evaluated exactly, because the values of $\lambda^*_{G,1}$ and $\lambda^*_{G,2}$ can be computed in polynomial time [1], U/L can be bounded from above by a constant, independent of G, δ_1, and δ_2.

Specifically, when $\delta_1 > 2\delta_2$, the ratio is

$$\frac{\delta_1\lambda^*_{G,1} + \delta_2}{\delta_1\lambda^*_{G,1}} = 1 + \frac{\delta_2}{\delta_1}\frac{1}{\lambda^*_{G,1}} \le \frac{3}{2},$$

because

$$\frac{\delta_2}{\delta_1}\frac{1}{\lambda^*_{G,1}} \le \frac{1}{2}$$

from the assumption that the unit interval graph is connected.

Moreover, when $\delta_1 \le 2\delta_2$ and $L = \delta_1\lambda^*_{G,1}$, the above ratio becomes

$$\frac{2\delta_2\lambda^*_{G,1} + 2\delta_2}{\delta_1\lambda^*_{G,1}} = 2\frac{\delta_2}{\delta_1}\left(1 + \frac{1}{\lambda^*_{G,1}}\right) \le 3,$$

because

$$\frac{\delta_2}{\delta_1} \leq 1 \text{ and } \frac{1}{\lambda^*_{G,1}} \leq \frac{1}{2}$$

from the assumption that the unit interval graph is not a path.

Finally, when $\delta_1 \leq 2\delta_2$ and $L = \delta_2\lambda^*_{G,2}$,

$$\frac{2\delta_2\lambda^*_{G,1} + 2\delta_2}{\delta_2\lambda^*_{G,2}} = 2\frac{\lambda^*_{G,1}}{\lambda^*_{G,2}}\left(1 + \frac{1}{\lambda^*_{G,1}}\right) \leq 3,$$

because

$$\frac{\lambda^*_{G,1}}{\lambda^*_{G,2}} \leq 1 \text{ and } \frac{1}{\lambda^*_{G,1}} \leq \frac{1}{2}$$

as in the previous case. ∎

It is worth noting that, when $\delta_1 = 2$ and $\delta_2 = 1$, the ratio U/L becomes $\frac{2\lambda^*_{G,1}+2}{2\lambda^*_{G,1}}$, namely the same derived in [22] for the $L(2,1)$-coloring problem on unit interval graphs.

4. TREES

An undirected graph $T = (V, E)$ is a *free tree* when it is connected and it has exactly $|V| - 1$ edges. Given a vertex v of a free tree T, $Adj(v)$ denotes the set of vertices adjacent to v. Given also an integer t, $N_t(v)$ denotes the set of vertices at distance at most t from v. Clearly, $Adj(v) = N_1(v) - \{v\}$. A *rooted tree* is a free tree in which a vertex r is identified as a *root* and all the other vertices are ordered by levels, where the level $\ell(v)$ of a vertex v is equal to the distance $d(r, v)$. Thus, all the vertices adjacent to v are partitioned into its *father*, denoted by $father(v)$, which is at level $\ell(v) - 1$, and into its children, which are at level $\ell(v) + 1$. The *height* h of a tree T is the maximum level of its vertices. For each vertex v of T, let $anc_i(v)$ denote the ancestor of v at distance i from v (which clearly is at level $\ell(v) - i$). Of course, $anc_1(v) = father(v)$ and $anc_0(v) = v$. Moreover, $lca(u, v)$ denotes the *lowest common ancestor* of u and v, that is, the vertex with maximum level among all the common ancestors of both u and v. Finally, given a vertex v of T, T_v denotes the induced subtree rooted at v consisting of all the vertices having v as an ancestor.

To derive an approximate $L(\delta_1, \ldots, \delta_t)$-coloring of a rooted tree, the following lemma is useful because it shows how to locate a strongly simplicial vertex.

Lemma 8 ([4]). *In a rooted tree of height h, any vertex at level h is strongly simplicial.*

Proof. Let T be a rooted tree and consider a vertex x with $\ell(x) = h$. Let t be any arbitrary integer not larger than $2h$. Consider two vertices u and v such that $d(u, x) \leq t$ and $d(v, x) \leq t$. Consider also the shortest paths $sp(x, v)$, $sp(x, u)$, and $sp(x, w)$, where w is the vertex of smallest level belonging to both $sp(x, v)$ and $sp(x, u)$. Because $\ell(x) \geq \max\{\ell(u), \ell(v)\}$, then $\min\{d(u, w), d(v, w)\} \leq d(x, w)$. Assume w.l.o.g. $d(u, w)$ to be minimum. Then $d(u, v) = d(u, w) + d(w, v) \leq d(x, w) + d(w, v) \leq d(x, v) \leq t$. Therefore, vertex x is t-simplicial. Because such a condition holds for any $t \leq 2h$, x is strongly simplicial. ∎

Lemma 8 suggests visiting the tree in breadth-first-search order, namely scanning the vertices by increasing levels. Hereafter, it is assumed that the vertices are numbered according to the breadth-first search order, obtained by starting the visit from the root. Precisely, the vertices are numbered level by level, and those at the same level from left to right. In this way, when a vertex v is considered, v is a t-simplicial vertex of the subtree $T[\{1, 2, \ldots, v\}]$ induced by the first v vertices of T, for $1 \leq v \leq n$.

4.1. Approximate $L(\delta_1, \ldots, \delta_t)$-Coloring of Trees

In this subsection, an $O(nt^2\delta_1)$ time algorithm is exhibited to find an approximate $L(\delta_1, \ldots, \delta_t)$-coloring of trees.

The algorithm assumes a double representation of T, considering T both as a free tree and as the rooted tree T_1. Specifically, $Adj(v)$, $father(v)$, and $\ell(v)$ are maintained for each vertex v. As before, the algorithm maintains the palette P_0 of readily usable colors, initialized to the set $\{0, 1, \ldots, U\}$, where U is the upper bound given by Lemma 3. Again, a counter TABOO$[c]$ keeps track of how many used colors forbid color c.

The Tree-Coloring algorithm, illustrated in Figure 3, uses three procedures: Ancestor, Up-Neighborhood-BFS, and Clear-BFS, depicted in Figures 4, 5, and 6, respectively. Tree-Coloring first performs a preprocessing to compute the upper bound U, which depends on $\lambda_{T,j}$, for each j, $1 \leq j \leq t$.

The algorithm scans the n vertices according to the BFS numbering. At each iteration, v represents the vertex to be colored next, while u is the last colored vertex. Hence, $u = v - 1$, for $2 \leq v \leq n$. To color v, one needs to determine the set of colors already used and forbidden in the neighborhood $N'_t(v) = N_t(v) \cap T[\{1, \ldots, v\}]$. Such a set of colors depends only on the distance $d(z, v)$ for each vertex $z \in N'_t(v)$, and it is computed incrementally with respect to the neighborhood $N'_t(u)$ of the last colored vertex u.

Algorithm Tree-Coloring $(T = (V, E), t, \delta_1, \ldots, \delta_t)$;

```
U := λ*_{T,t} + 2(δ_t − 1)λ*_{T,t} + Σ_{j=1}^{t−1} 2(δ_j − δ_{j+1})λ*_{T,j};
P_0 := {0, 1, ..., U};
TABOO[γ] := 0 for γ = 0, ..., U;
u := 1; δ_0 := 0;
for v := 1 to n do
    x := Ancestor(u, v);
    if new then Clear-BFS(u);
    Up-Neighborhood-BFS(v, x);
    extract a color c from P_0;
    f(v) := c;
    TABOO[c] := TABOO[c] + 1;
    u := v;
```

FIG. 3. The tree-coloring algorithm for $L(\delta_1, \ldots, \delta_t)$-coloring of trees.

Function Ancestor (u, v);

$w := u$; $y := u$; $up := \min\{t, \ell(u)\}$;
if $\ell(v) = \ell(u) + 1$
  then $i := 1$; $x := father(v)$
  else $i := 0$; $x := v$;
while $x \neq w$ and $i < up$ do
  $y := w$; $w := father(y)$;
  $x := father(x)$; $i := i + 1$;
if $x \neq w$ or $\ell(v) = \ell(u) + 1$
  then $new :=$ true
  else $new :=$ false;
return(x)

FIG. 4. The Ancestor procedure to find the vertex x with maximum level between $lca(u, v)$ and $anc_t(v)$. If $x = lca(u, v)$ then y is the child of x on the path between u and x.

The behavior of the Tree-Coloring algorithm depends on whether $N'_t(v)$ and $N'_t(u)$ intersect or not, and on whether u and v are at the same level or not. When $\ell(v) = \ell(u) + 1$ or when $\ell(v) = \ell(u)$ and $N'_t(v) \cap N'_t(u) = \emptyset$, the variable *new* is set to true by the Ancestor function, the counters of all the used and forbidden colors in the old neighborhood $N'_t(u)$ are decremented by the Clear-BFS procedure, while the distances and the forbidden colors in $N'_t(v)$ are computed from scratch by the Up-Neighborhood-BFS procedure. Indeed, although the neighborhoods $N'_t(v)$ and $N'_t(u)$ may intersect when $\ell(v) = \ell(u) + 1$, each node z belonging to the intersection has $d(z, v) \neq d(z, u)$, and thus its distance and its forbidden colors have to be recomputed. In particular, the empty intersection between $N'_t(v)$ and $N'_t(u)$ is recognized by the Ancestor function when $x = anc_t(v)$ is deeper than $lca(u, v)$, the least common ancestor between u and v. When $\ell(v) = \ell(u)$ and $N'_t(v) \cap N'_t(u) \neq \emptyset$, $x = lca(u, v)$ is deeper than $anc_t(v)$, and x belongs to the shortest path $sp(u, v)$ between u and v. Consider the vertices y and y', which are the children of x on the paths $sp(u, x)$ and $sp(v, x)$, respectively. In this case, the Ancestor function sets *new* to false and returns $x = lca(u, v)$ along with the vertex y. Indeed, y and y' are the roots of the subtrees $T_y \cap N'_t(v)$ and $T_{y'} \cap N'_t(u)$ containing each vertex $z \in N'_t(v) \cap N'_t(u)$ such that $d(z, u) \neq d(z, v)$. The Up-Neighborhood-BFS procedure updates only the colors already used and forbidden by vertices in $T_y \cap N'_t(v)$ and in $T_{y'} \cap N'_t(u)$, leaving unchanged those colors used and forbidden by vertices in $(N'_t(v) \cap N'_t(u)) - (T_y \cup T_{y'})$ because for each vertex z in such a subset $d(z, v) = d(z, u)$ holds. Moreover, the procedure introduces new forbidden colors due to the vertices in $N'_t(v) - N'_t(u)$, and finally frees the colors no longer used or forbidden by vertices in $N'_t(u) - N'_t(v)$.

Procedure Up-Neighborhood-BFS (v, x);

$dist(v) := 0$; $M := \emptyset$;
Enqueue(Q, v);
while $Q \neq \emptyset$ do
  $w :=$ Dequeue(Q);
  if $w = x$ and not *new* then $S := \{y\}$ else $S := Adj(w)$;
(1) for each $z \in S$ do
    if $\ell(x) \leq \ell(z) \leq \ell(v)$ and $z < v$ and z is not marked then
      if $w = v$ or $0 < dist(w) < t$ then
(2)   for each $f(z) - \delta_{dist(z)} + 1 \leq \gamma \leq f(z) + \delta_{dist(z)} - 1$ do
          if $\gamma \neq f(z)$ then
            TABOO$[\gamma]$:= TABOO$[\gamma] - 1$;
            if TABOO$[\gamma] = 0$ then insert γ into P_0;
        $dist(z) := dist(w) + 1$; mark z;
(3)   for each $f(z) - \delta_{dist(z)} + 1 \leq \gamma \leq f(z) + \delta_{dist(z)} - 1$ do
          if $\gamma \neq f(z)$ then
            if TABOO$[\gamma] = 0$ then extract γ from P_0;
            TABOO$[\gamma]$:= TABOO$[\gamma] + 1$;
      if not *new* and $w \neq v$ and ($dist(w) = 0$ or $dist(w) = t$) and $dist(z) > 0$ then
(4)   for each $f(z) - \delta_{dist(z)} + 1 \leq \gamma \leq f(z) + \delta_{dist(z)} - 1$ do
          TABOO$[\gamma]$:= TABOO$[\gamma] - 1$;
          if TABOO$[\gamma] = 0$ then insert γ into P_0;
        $dist(z) := 0$; mark z;
      Enqueue(Q, z);
    insert w into M;
  for each $z \in M$ do unmark z;

FIG. 5. The modified BFS procedure to compute distances and forbid colors for vertices in $N'_t(v)$ and clear distances and colors for vertices in $N'_t(u) - N'_t(v)$.

Procedure Clear-BFS (u);

```
Enqueue(Q, u);
while Q ≠ ∅ do
  w := Dequeue(Q);
  for each z ∈ Adj(w) do
    if dist(z) > 0 then
      for each f(z) − δ_dist(z) + 1 ≤ γ ≤ f(z) + δ_dist(z) − 1 do
        TABOO[γ] := TABOO[γ] − 1;
        if TABOO[γ] = 0 then insert γ into P_0;
      dist(z) := 0;
      Enqueue (Q, z);
```

FIG. 6. The procedure to clear distances and colors for vertices in $N'_t(u)$ when $\ell(v) = \ell(u) + 1$ or $N'_t(v) \cap N'_t(u) = \emptyset$.

Precisely, the Up-Neighborhood-BFS procedure is invoked with the aim of computing the distances from v of each vertex z in $N'_t(v)$, and accordingly changing the TABOO counters. This is done during a breadth-first search starting from vertex v in which the label $dist(z)$ is set to $d(z, v)$, and each separation constraint is updated by first decrementing the counters of the colors γ with $0 < |f(z) - \gamma| < \delta_{d(z,u)}$ and then incrementing those of the colors with $0 < |f(z) - \gamma| < \delta_{d(z,v)}$. Summarizing, when *new* is true, the procedure computes from scratch all the distances and forbidden colors for all vertices in $N'_t(v)$. In contrast, when *new* is false, distances and forbidden colors are computed from scratch only for vertices in $N'_t(v) - N'_t(u)$, they are updated only for a subset of vertices in $N'_t(v) \cap N'_t(u)$, and they are cleared for those in $N'_t(u) - N'_t(v)$.

Lemma 9. *Let the Tree-Coloring algorithm be at iteration v just before coloring vertex v, and consider any vertex z in $T[\{1, \ldots, v\}]$. Then*

$$dist(z) = \begin{cases} d(z, v) & \text{if } z \in N'_t(v), \\ 0 & \text{otherwise.} \end{cases}$$

Proof. The proof is by induction. If $v = 1$, $N'_t(v) = v$ and $dist(v) = d(v, v) = 0$. Thus, assume as the inductive hypothesis that the lemma holds up to $u = v - 1$, and consider a vertex $z \leq v$.

If $\ell(v) = \ell(u)$ and $N'_t(v) \cap N'_t(u) \neq \emptyset$, then $x = \ell(lca(u, v)) \geq \ell(anc_t(v))$ holds. In this case, to compute $d(z, v)$, the vertices of $T[\{1, \ldots, v\}]$ are ideally partitioned into five subsets. Recall that y and y' are the children of x on $sp(u, v)$.

(*i*) If $z \in T_y \cap N'_t(v)$, then $d(z, v)$ is different from $d(z, u)$ because $x \in sp(v, z)$ but $x \notin sp(u, z)$. A symmetric consideration also holds for $z \in T_{y'} \cap N'_t(u)$. Therefore, $dist(z)$ is recomputed as $dist(z) = d(z, v)$ at iteration v of the Tree-Coloring algorithm. Specifically, this is done in Loop (1) by the Up-Neighborhood-BFS procedure.

(*ii*) If $z \in N'_t(v) - N'_t(u)$, $dist(z)$ is computed from scratch as $d(z, v)$ also in Loop (1) of the Up-Neighborhood-BFS procedure during iteration v of the algorithm.

(*iii*) If $z \in (N'_t(v) \cap N'_t(u)) - (T_y \cup T_{y'})$, then $x = lca(u, v)$ belongs to both $sp(u, z)$ and $sp(v, z)$. Hence, $d(v, z) = d(u, z)$ because $d(v, z) = d(v, x) + d(x, z) = d(u, x) + d(x, z) = d(u, z)$ and $\ell(v) = \ell(u)$. Thus, because by the inductive hypothesis $dist(z) = d(z, u)$, no action is taken by the algorithm during iteration v.

(*iv*) If $z \in N'_t(u) - N'_t(v)$, then $dist(z)$ is set to zero in Loop (4) by the Up-Neighborhood-BFS procedure. The condition on z is checked in the *if* instruction just before Loop (4). In fact, if $z \in N'_t(u) - \{u\}$ then $dist(z) > 0$ by the inductive hypothesis. Moreover, $z \notin N'_t(v)$ because z is reached during the BFS visit after a vertex whose distance from v is at least t.

(*v*) Finally, if $z \in T[\{1, \ldots, v\}] - (N'_t(v) \cup N'_t(u))$, then $dist(z)$ has been already cleared by the inductive hypothesis in a previous iteration.

If $\ell(v) = \ell(u)$ and $N'_t(v) \cap N'_t(u) = \emptyset$, then $\ell(anc_t(v)) > \ell(lca(u, v))$ holds. Again, $T[\{1, \ldots, v\}]$ is ideally partitioned into three subsets.

(*i*) If $z \in N'_t(v)$, then $dist(z)$ is set to $d(v, z)$ during the current iteration in Loop (1) of Up-Neighborhood-BFS.

(*ii*) If $z \in N'_t(u)$, then $dist(z)$ is set to 0 at iteration v of the algorithm by the Clear-BFS procedure. Such a procedure performs a BFS starting from u visiting all those vertices with $dist(z) = d(u, z) > 0$, which by the inductive hypothesis coincides with $N'_t(u)$.

(*iii*) Finally, if $z \in T[\{1, \ldots, v\}] - ((N'_t(v) \cup N'_t(u))$, then $dist(z)$ has already been cleared in a previous iteration by the inductive hypothesis.

If $\ell(v) = \ell(u) + 1$, although $N'_t(v)$ and $N'_t(u)$ may intersect, $dist(z)$ has to be recomputed also for every vertex $z \in N'_t(u) \cap N'_t(v)$ because $d(v, lca(u, v)) \neq d(u, lca(u, v))$. Because $dist(z)$ has to be recomputed for every vertex in $N'_t(v)$, Tree-Coloring acts as when $\ell(v) = \ell(u)$ and $N'_t(v) \cap N'_t(u) = \emptyset$, and the proof is the same. ∎

Lemma 10. *Let the Tree-Coloring algorithm be at iteration v just before coloring vertex v, and consider any color c.*

- *If c is assigned to a vertex $z \in N'_t(v)$, then $c \notin P_0$ and c forbids only the colors γ such that $c - \delta_{d(z,v)} + 1 \le \gamma \le c + \delta_{d(z,v)} - 1$.*
- *If c is not assigned to any vertex in $N'_t(v)$ and $c \notin P_0$, then c is forbidden by at least a color assigned to a vertex in $N'_t(v)$, but c does not forbid any color.*
- *If $c \in P_0$, then c is readily usable and c does not forbid any color.*

Proof. If $v = 1$, then no color at all has been yet used or forbidden. Thus, all the colors are in P_0 and they are readily usable. Assume, by the inductive hypothesis, that the lemma holds for $u = v - 1$, and consider any color c.

If c is assigned to a vertex $z \in N'_t(v)$, c cannot be reassigned to v, because $d(v, z) \le t$. Indeed, TABOO$[c] > 1$, and hence, $c \notin P_0$. Moreover, the counters of the old colors forbidden by the $\delta_{d(z,u)}$-separation constraint are decremented in Loop (2) where $dist(z) = d(z, u)$ by Lemma 9, while the counters of the new colors forbidden by the $\delta_{d(z,v)}$-separation constraint are incremented in Loop (3) where $dist(z) = d(z, v)$ has just been set. Note that the algorithm properly works even if $z \notin N'_t(u)$, because in such a case $\delta_{dist(z)} = \delta_0 = 0$, and thus no counter is decremented in Loop (2).

If c is not assigned to any vertex of $N'_t(v)$ and $c \notin P_0$, then TABOO$[c] > 0$ because it has been incremented during Loop (3). Hence, c is forbidden by a vertex $z \in N'_t(v)$ such that $c \ne f(z)$ and $f(z) - \delta_{d(z,v)} + 1 \le c \le f(z) + \delta_{d(z,v)} - 1$.

If $c \in P_0$, the statement trivially holds if c has never been assigned or forbidden. Then assume that $c \in P_0$, and that it has been freed either by the Up-Neighborhood-BFS procedure during Loop (2) or Loop (4), or by the Clear-BFS procedure. If c was freed by $f(z)$ during Loop (2), c was not assigned to any vertex in $N'_t(v)$, but forbidden by just the single vertex z. If c was freed by $f(z)$ in Loop (4) or by Clear-BFS, then either c was assigned to z (that is, $c = f(z)$) or c was forbidden by just the single vertex z, with $z \in N'_t(u) - N'_t(v)$. In all cases, TABOO$[c] = 0$ and c is readily usable. ∎

In practice, the above lemma guarantees that a legal $L(\delta_1, \delta_2, \ldots, \delta_t)$-coloring is found, namely, a color c is assigned to a vertex v only when it satisfies all the separation constraints due to colors assigned to vertices at distance at most t from v. In particular, any vertex already colored c is at distance greater than t from v.

Lemma 11. *The largest color used by the Tree-Coloring algorithm is at most $U = \lambda^*_{T,t} + 2(\delta_t - 1)\lambda^*_{T,t} + \sum_{j=1}^{t-1} 2(\delta_j - \delta_{j+1})\lambda^*_{T,j}$.*

Proof. By Lemma 3, there is an $L(\delta_1, \ldots, \delta_t)$-coloring of T using the colors $\{0, \ldots, U\}$. To show that the Tree-Coloring algorithm succeeds in coloring using so many colors, it is enough to observe that $N'_t(v) - \{v\} = \bigcup_{j=1}^{t} D_j(v)$ for each vertex v, where $D_j(v) = N'_j(v) - N'_{j-1}(v)$. When v has to be colored, the overall number of forbidden and used colors is given by $\sum_{j=1}^{t}(2(\delta_j - 1) + 1)|D_j(v)|$. Then the proof is analogous to that of Lemma 7, where P_j is replaced by $D_{t-j+1}(v)$. ∎

Theorem 4. *Let $\delta_m \lambda^*_{G,m} = max_{1 \le j \le t}\{\delta_j \lambda^*_{G,j}\}$, and recall that $\delta_{t+1} = 0$. The Tree-Coloring algorithm gives an α-approximation with $\alpha = \min\left\{2t, \frac{2\delta_{m+1}-1}{\delta_t} + \frac{2(\delta_1 - \delta_{m+1})}{\delta_m}\right\}$.*

Proof. The proof follows from Lemma 11 and Theorem 1. ∎

As a consequence of Theorem 4, the Tree-Coloring algorithm provides a 4-approximate $L(\delta_1, \delta_2)$-coloring. If applied to binary trees, however, a better approximation is attained.

Corollary 2. *Let t be equal to 2 and let T be a binary tree. Then the Tree-Coloring algorithm yields a $\frac{10}{3}$-approximate $L(\delta_1, \delta_2)$-coloring.*

Proof. When $t = 2$ and T is a binary tree, $\lambda^*_{T,2} = 3$ and $\lambda^*_{T,1} = 1$. Therefore, the bounds U and L become, respectively, $U = 3 + 4(\delta_2 - 1) + 2(\delta_1 - 1)$ and $L = \max\{3\delta_2, \delta_1\}$.

If $3\delta_2 > \delta_1$, one has

$$\frac{U}{L} = \frac{4\delta_2 + 2\delta_1 - 3}{3\delta_2} \le \frac{4}{3} + \frac{2\delta_1 - 3}{\delta_1} \le \frac{10}{3}.$$

If $3\delta_2 \le \delta_1$, then

$$\frac{U}{L} = \frac{4\delta_2 + 2\delta_1 - 3}{\delta_1} \le 2 + \frac{4\delta_2 - 3}{3\delta_2} \le \frac{10}{3}.$$ ∎

A better approximation can be reached for arbitrary t, when the separation vector is $(\delta_1, 1, \ldots, 1)$.

Corollary 3 ([4])**.** *When $\delta_2 = \cdots = \delta_t = 1$, the Tree-Coloring algorithm yields a 3-approximate $L(\delta_1, 1, \ldots, 1)$-coloring.*

Proof. The proof is the same as that of Corollary 1. ∎

Theorem 5. *The Tree-Coloring algorithm runs in $O(nt^2\delta_1)$ time.*

Proof. The preprocessing required to evaluate the upper bound U takes $O(nt^2)$ time, because $O(jn)$ time is needed to compute $\lambda^*_{T^j,1} = \lambda^*_{T,j}$ [1]. Because $U = O(n\delta_1)$, the initialization of P_0 and TABOO takes $O(n\delta_1)$ time. The palette P_0 is implemented by means of a doubly-linked list of colors and a vector C indexed by colors, so that insertions and extractions take $O(1)$ time as already explained in the proof of Theorem 2. Moreover, the Ancestor function takes $O(nt)$ time because it is invoked once for each vertex, and each invocation requires $O(t)$ time.

To determine the overall time complexity of the Tree-Coloring algorithm, consider a generic vertex x and observe that $O(\delta_1)$ time is required whenever the value of $dist(x)$ is changed by the algorithm. Notice that $dist(x)$ is set to 0 for the first time at iteration x, when $N'_t(x)$ is computed by the Up-Neighborhood-BFS procedure. At a generic iteration $v > x$, $dist(x)$ is changed whenever $d(u,x) \neq d(v,x) \leq t$ with $u = v - 1$. Moreover, $dist(x)$ is set again to 0 if $d(u,x) \leq t$ and $d(v,x) > t$ or if $\ell(v) = \ell(u) + 1$.

Consider the vertices at a given level ℓ of T, numbered according to their BFS-ordering, say $v_1, v_2, \ldots, v_m$. Under such an assumption, $d(v_i,x) \neq (v_j,x)$ if and only if $lca(v_i,x) \neq lca(v_j,x)$. Consider also the subtree T_x rooted at x, and let $v_l, \ldots, v_r$ be the vertices at level ℓ of T, which belong to T_x. Observe that $dist(x)$ does not increase while the vertices $v_1, \ldots, v_{l-1}$ are examined, it remains the same while examining $v_l, \ldots, v_r$, and it does not decrease when examining $v_{r+1}, \ldots, v_m$. Observe also that, if $d(v_i,x) = d(v_j,x)$ and $v_1 \leq v_i < v_j < v_l$ or $v_r < v_i < v_j \leq v_m$, then $d(v_i,x) = d(v,x)$ for every vertex v with $v_i \leq v \leq v_j$, because such vertices are numbered in BFS-ordering and are at the same level. This implies that all the vertices at level ℓ, for which $dist(x)$ has a given value, are split into at most two sequences of consecutive vertices and thus $O(\delta_1)$ time is required only at the beginning of each subsequence.

On the other hand, there are $O(t)$ different values, no larger than t, that $dist(x)$ can assume, one for each possible $lca(x,v) = anc_k(x)$ with $0 \leq k \leq \lfloor \frac{t}{2} \rfloor$. Therefore, $O(t\delta_1)$ is required for updating $dist(x)$ for iterations concerning vertices at level ℓ. Because x can be involved in the neighborhoods of vertices of at most $t+1$ levels, namely at levels $\ell(x), \ell(x)+1, \ldots, \ell(x)+t$, the time taken by the algorithm for updating $dist(x)$ is $O(t^2\delta_1)$. Finally, the overall time complexity is $O(nt^2\delta_1)$, because there are n vertices in the tree. ∎

5. CONCLUSION

This article has considered the channel assignment problem for general separation vectors and two specific classes of graphs—trees, and interval graphs. Based on the notion of strongly simplicial vertices, $O(n(t+\delta_1))$ and $O(nt^2\delta_1)$ time algorithms have been proposed to find α-approximate $L(\delta_1, \ldots, \delta_t)$-colorings on interval graphs and trees, respectively, where α is a constant depending on t and $\delta_1, \ldots, \delta_t$. When $t = 2$, such algorithms provide 4-approximate $L(\delta_1, \delta_2)$-colorings, while they yield 3-approximate $L(\delta_1, 1, \ldots, 1)$-colorings when δ_1 is the only separation greater than 1. For $t = 2$ and binary trees, a $\frac{10}{3}$-approximation is achieved. Moreover, a linear time 3-approximation algorithm giving an $L(\delta_1, \delta_2)$-coloring of unit interval graphs has also been presented.

Several questions remain open. For instance, one could devise better approximation algorithms for finding $L(\delta_1, \ldots, \delta_t)$-colorings of interval graphs and trees, or one could determine whether finding optimal $L(\delta_1, \delta_2)$-colorings of unit interval graphs or trees is NP-hard or not.

REFERENCES

[1] G. Agnarsson, R. Greenlaw, and M.M. Halldorson, On powers of chordal graphs and their colorings, Congressus Numer 144 (2000), 41–65.

[2] R. Battiti, A.A. Bertossi, and M.A. Bonuccelli, Assigning codes in wireless networks: Bounds and scaling properties, Wireless Networks 5 (1999), 195–209.

[3] A.A. Bertossi and M.C. Pinotti, Mappings for conflict-free access of paths in bidimensional arrays, circular lists, and complete trees, J Parallel Distrib Comput 62 (2002), 1314–1333.

[4] A.A. Bertossi, M.C. Pinotti, and R. Rizzi, Channel assignment on strongly-simplicial graphs, Proc. 3rd Int'l Workshop on Wireless, Mobile and Ad Hoc Networks (WMAN), IEEE IPDPS, Nice, 2003 (published on CD-ROM).

[5] A.A. Bertossi, M.C. Pinotti, R. Rizzi, and A.M. Shende, Channel assignment for interference avoidance in honeycomb wireless networks, J Parallel Distrib Comput 64 (2004), 1329–1344.

[6] A.A. Bertossi, M.C. Pinotti, and R.B. Tan, Channel assignment with separation for interference avoidance in wireless networks, IEEE Trans Parallel Distrib Syst 14 (2003), 222–235.

[7] H.L. Bodlaender, T. Kloks, R.B. Tan, and J. van Leeuwen, Approximations for λ-coloring of graphs, Comput J 47 (2004), 193–204.

[8] K.S. Booth and G.S. Lueker, Linear algorithms to recognize interval graphs and test for the consecutive ones property, Proc. Seventh Annual ACM Symposium on Theory of Computing, Albuquerque, NM, 1975, pp. 255–265.

[9] A. Brandstädt, V.B. Le, and J.P. Spinrad, Graph classes: A survey, SIAM, Philadelphia, PA, 1999.

[10] T. Calamoneri, The $L(h,k)$-labelling problem: A survey and annotated bibliography, Comput J (2006), to appear.

[11] T. Calamoneri, A. Pelc, and R. Petreschi, Labeling trees with a condition at distance two, Discrete Math 306 (2006), 1534–1539.

[12] G.J. Chang, W.-T. Ke, D. Kuo, D.D.-F. Liu, and R.K. Yeh, On $L(d,1)$-labelings on graphs, Discrete Math 220 (2000), 57–66.

[13] G.J. Chang and D. Kuo, The $L(2,1)$-labeling problem on graphs, SIAM J Discrete Math 9 (1996), 309–316.

[14] I. Chlamtac and S.S. Pinter, Distributed nodes organization algorithm for channel access in a multihop dynamic radio network, IEEE Trans Comput 36 (1987), 728–737.

[15] D.G. Corneil, S. Olariu, and L. Stewart, The ultimate interval graph recognition algorithm?, Proc. of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1998, pp. 175–180.

[16] J.P. Georges and D.W. Mauro, Generalized vertex labelings with a condition at distance two, Congressus Numer 109 (1995), 141–159.

[17] J.P. Georges and D.W. Mauro, Labeling trees with a condition at distance two, Discrete Math 269 (2003), 127–148.

[18] M.C. Golumbic, Algorithmic graph theory and perfect graphs, Academic Press, New York, 1980.

[19] J.R. Griggs and R.K. Yeh, Labelling graphs with a condition at distance 2, SIAM J Discrete Math 5 (1992), 586–595.

[20] W.K. Hale, Frequency assignment: Theory and application, Proceedings of the IEEE 68 (1980), 1497–1514.

[21] S.T. McCormick, Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem, Math Program 26 (1983), 153–171.

[22] D. Sakai, Labeling chordal graphs: Distance two condition, SIAM J Discrete Math 7 (1994), 133–140.

[23] A. Sen, T. Roxborough, and S. Medidi, Upper and lower bounds of a class of channel assignment problems in cellular networks, Proc. of IEEE INFOCOM'98, San Francisco, CA, 1998, Vol. 3, pp. 1284–1291.

[24] A.M. Shende, et al., A characterisation of optimal channel assignments for wireless networks modelled as cellular and square grids, Proc. 3rd Int'l Workshop on Wireless, Mobile and Ad Hoc Networks (WMAN) IEEE IPDPS, Nice, 2003 (published on CD-ROM).

[25] J. Van den Heuvel, R.A Leese, and M.A. Shepherd, Graph labelling and radio channel assignment, J Graph Theory 29 (1998), 263–283.

[26] X. Zhou, Y. Kanari, and T. Nishizeki, Generalized vertex coloring of partial k-trees, IEICE Trans. Fundam Electron Commun Comput Sci E83-A (2000), 671–678.