



UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
<http://www.dit.unitn.it>

CHANNEL ASSIGNMENT WITH SEPARATION ON TREES
AND INTERVAL GRAPHS

Alan Bertossi, Cristina M. Pinotti and Romeo Rizzi

October 2002

Technical Report # DIT-02-0075

Channel Assignment with Separation on Trees and Interval Graphs *

Alan A. Bertossi[†] Cristina M. Pinotti[‡] Romeo Rizzi[§]

Abstract

Given a vector $(\delta_1, \delta_2, \dots, \delta_t)$ of non increasing positive integers, and an undirected graph $G = (V, E)$, an $L(\delta_1, \delta_2, \dots, \delta_t)$ -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 $L(\delta_1, \delta_2, \dots, \delta_t)$ -coloring for G is one minimizing the largest used integer over all such colorings. This coloring problem has relevant application in channel assignment for interference avoidance in wireless networks, where channels (i.e. colors) assigned to interfering stations (i.e. vertices) at distance i must be at least δ_i apart, while the same channel can be reused only at stations whose distance is larger than t . This paper presents efficient algorithms for finding optimal $L(1, \dots, 1)$ -colorings of trees and interval graphs as well as optimal $L(2, 1, 1)$ -colorings of complete binary trees. Moreover, efficient algorithms are also provided for finding approximate $L(\delta_1, 1, \dots, 1)$ -colorings of trees and interval graphs as well as approximate $L(\delta_1, \delta_2)$ -colorings of unit interval graphs.

Key Words: Wireless Networks, Channel Assignment, Interferences, Interval Graphs, Unit Interval Graphs, Trees, Complete Binary Trees, $L(1, \dots, 1)$ -coloring, $L(\delta_1, 1, \dots, 1)$ -coloring, $L(2, 1, 1)$ -coloring, $L(\delta_1, \delta_2)$ -coloring, perfect elimination scheme.

1 Introduction

In the *channel assignment* problem for wireless communication 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* $(\delta_1, \delta_2, \dots, \delta_t)$ of non increasing positive integers is introduced in such a way that channels assigned to interfering stations at distance i 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 [12]. The purpose of channel assignment algorithms is,

*This work is partially supported by MIUR-RealWine Research Program.

[†]Department of Computer Science, Mura Anteo Zamboni, 7, University of Bologna, 40127 Bologna, ITALY, bertossi@cs.unibo.it

[‡]Department of Computer Science and Telecommunications, University of Trento, 38050 Povo, Trento, ITALY, pinotti@science.unitn.it

[§]Department of Computer Science and Telecommunications, University of Trento, 38050 Povo, Trento, ITALY, rrizzi@science.unitn.it

therefore, to assign channels to the stations so that the above channel separations are verified 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 in V correspond to stations, and edges in E correspond to pairs of stations that can hear each other transmission. Specifically, given a vector $(\delta_1, \delta_2, \dots, \delta_t)$ of non increasing positive integers, and an undirected graph $G = (V, E)$, an $L(\delta_1, \delta_2, \dots, \delta_t)$ -coloring of G is a function f from the vertex set V to the set of nonnegative integers $\{0, \dots, \lambda\}$ 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* $L(\delta_1, \delta_2, \dots, \delta_t)$ -coloring for G is one minimizing λ over all such colorings. Note that, since the set of colors includes 0, the overall number of colors involved by an optimal coloring f is in fact $\lambda + 1$ (although, due to the channel separation constraint, some colors in $\{1, \dots, \lambda - 1\}$ might not be actually assigned to any vertex). Thus, the channel assignment problem consists of finding an optimal $L(\delta_1, \delta_2, \dots, \delta_t)$ -coloring for G .

When the separation vector $(\delta_1, \delta_2, \dots, \delta_t)$ is equal to $(1, 1, \dots, 1)$, the channel assignment problem has been widely studied in the past [1, 2, 9, 13, 15]. In particular, the intractability of optimal $L(1, \dots, 1)$ -coloring, for any positive integer t , has been proved by McCormick [13]. Optimal $L(1, \dots, 1)$ -colorings, for any positive integer t , have been proposed in [2] for rings, complete trees, and bidimensional grids. Moreover, when the separation vector is equal to $(\delta_1, 1, \dots, 1)$, optimal $L(\delta_1, 1, \dots, 1)$ -colorings have been proposed in [4] for rings and bidimensional 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. [16], who provided also an optimal $L(2, 1, 1)$ -coloring for bidimensional grids. The $L(2, 1, 1)$ -coloring problem has been also optimally solved for cellular grids, hexagonal grids, and rings in [3]. Finally, the $L(2, 1)$ -coloring has been investigated in [5, 8, 11, 14]. In particular, solutions for unit interval graphs and trees have been found, respectively, by Sakai [14] and by Chang and Kuo [8]. Approximate solutions for outerplanar, permutation and split graphs have been also presented by Bodlaender et al [5].

This paper further investigates the channel assignment problem for particular separation vectors and two 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 (mountains, lakes,...). In the rest of this paper, the following results will be proposed. First, the notions of t -simplicial and strongly-simplicial vertices of a graph will be introduced. When the separation vector $(\delta_1, \dots, \delta_t)$ is equal to $(1, \dots, 1)$, two algorithms will be devised to optimally solve the $L(1, \dots, 1)$ -coloring problem on trees and interval graphs, which run in $O(nt)$ time, where n is the number of vertices. Such algorithms will then be generalized to find approximate solutions for the $L(\delta_1, 1, \dots, 1)$ -coloring problem on the same classes of graphs. Further particular cases will be also dealt with. An approximate solution for $L(\delta_1, \delta_2)$ -coloring of unit interval graphs will be given, which generalizes that proposed in [14] for the $L(2, 1)$ -coloring problem. Moreover, an algorithm for finding an optimal $L(2, 1, 1)$ -coloring of complete binary trees will be devised which runs in optimal $O(n)$ time.

2 Preliminaries

Throughout this paper, it is assumed that G is a connected undirected graph with at least 2 vertices and that the separations verify $\delta_1 \geq \delta_2 \geq \dots \geq \delta_t$.

When the separation vector is $(1, \dots, 1)$, the $L(1, \dots, 1)$ -coloring problem can be reduced to a classical coloring problem on an *augmented* graph $A_{G,t}$ obtained as follows. The vertex set of $A_{G,t}$ is the same as the vertex set of G , while an edge uv belongs to the edge set of $A_{G,t}$ iff the distance $d(u, v)$ between the vertices u and v in G satisfies $d(u, v) \leq t$. Now, colors must be assigned to the vertices of $A_{G,t}$ so that every pair of vertices connected by an edge is assigned a couple of different colors and the minimum number of colors is used. Hence, the role of *maximum clique* in $A_{G,t}$ is apparent for deriving lower bounds on the minimum number of channels for the $L(1, \dots, 1)$ -coloring problem on G . A *clique* K for $A_{G,t}$ is a subset of vertices of $A_{G,t}$ such that for each pair of vertices in K there is an edge. Clearly, a clique of size k in the augmented graph $A_{G,t}$ implies that at least k different colors are needed to color $A_{G,t}$. In other words, the size of the largest clique in $A_{G,t}$ is a lower bound for the $L(1, \dots, 1)$ -coloring problem. Clearly, a lower bound for the $L(1, \dots, 1)$ -coloring problem is also a lower bound for the $L(\delta_1, 1, \dots, 1)$ -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 $L(1, \dots, 1)$ -colorings $f : V \rightarrow \{0, \dots, \lambda\}$ of $G = (V, E)$. Note that $\lambda_{G,1}^* \geq 1$ since G is assumed to be connected and has at least 2 vertices, and that $\lambda_{G,t}^* + 1$ is at least as large as the size of the largest clique of the augmented graph $A_{G,t}$.

Lemma 1 *The largest color needed by any $L(\delta_1, \delta_2, \dots, \delta_t)$ -coloring is at least $\max_{1 \leq i \leq t} \{\delta_i \lambda_{G,i}^*\}$.*

Proof Since $\delta_1 \geq \delta_2 \geq \dots \geq \delta_t$, any $L(\delta_1, \delta_2, \dots, \delta_i)$ -coloring, for any value of $i \leq t$, requires at least as many colors as any $L(\delta_i, \delta_i, \dots, \delta_i)$ -coloring, which, in its turn, requires at least $\delta_i \lambda_{G,i}^*$ colors. \square

Given $G = (V, E)$, let S be a subset of V . Then $G[S]$ denotes the subgraph of G induced by S , i.e. $G[S] = (S, \{uv \in E : u, v \in S\})$. 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 .

Lemma 2 *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 $L(1, \dots, 1)$ -coloring of G' using $\lambda_{G',t}^*$ as the largest color. Define an $L(1, \dots, 1)$ -coloring f of V extending f' to v so that*

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

Then f is an optimal $L(1, \dots, 1)$ -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 $f(v) > \lambda_{G',t}^*$, and let $N_t(v) = \{u \in V - \{v\} : d(u, v) \leq t\}$. Then, $\{f'(u) : u \in N_t(v)\} = \{0, \dots, \lambda_{G',t}^*\}$. Since v is *t-simplicial*, any two vertices in $N_t(v) \cup \{v\}$ are at distance at most t ,

and hence $N_t(v) \cup \{v\}$ is a clique of $A_{G,t}$. Since $|N_t(v) \cup \{v\}| = \lambda_{G',t}^* + 2$, and $f(v) = \lambda_{G',t}^* + 1$, then f is optimal. \square

Note that verifying whether a vertex is t -simplicial or not can be done in polynomial time. Therefore, Lemma 2 implies the existence of an algorithm that optimally solves in polynomial time the $L(1, \dots, 1)$ -coloring problem on any class of graphs closed under taking induced subgraphs and with the property that every graph of that class has a t -simplicial vertex. In this paper, we will look more closely at two classes of graphs with this property: trees and interval graphs.

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 to 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 $2n$.

Polynomial time algorithms to recognize interval graphs and compute their interval representations are known [6, 10]. Moreover, several alternative characterizations of interval graphs have been proposed so far in the literature [7].

The following lemma shows that the $L(1, \dots, 1)$ -coloring problem can be optimally solved in polynomial time on interval graphs.

Lemma 3 *Every interval graph has a strongly-simplicial vertex.*

Proof Let $G = (V, E)$ be an interval graph with n vertices, and consider its interval representation. It will be now shown that x is t -simplicial for any value of t . 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 $l_u < l_v < l_x$. Since 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. Since such a condition holds for any $t \leq n$, x is strongly-simplicial. \square

Lemmas 2 and 3 suggest to scan the vertices of an interval graph by increasing left endpoints since, in this way, the t -simplicial vertex v of the induced subgraph $G[\{1, \dots, v\}]$ is processed at every time, for $1 \leq v \leq n$.

3.1 Optimal $L(1, \dots, 1)$ -coloring

In this subsection, an $O(nt)$ time algorithm is proposed to find an optimal $L(1, \dots, 1)$ -coloring of interval graphs, which exploits the properties given in Lemmas 2 and 3.

Algorithm Interval- $L(1, \dots, 1)$ -coloring ($G = (V, E), t$);

```

set  $L_v := \emptyset$  for every  $v \in V$ ;
set  $P_i := \emptyset$  for  $i = 0, \dots, t$ ;  $\text{MAX-}r := 0$ ;  $\lambda_{G,t}^* := -1$ ;
for  $k := 1$  to  $2n$  do
  if  $k = l_v$  for some  $v$ , then
    if  $P_0 = \emptyset$  then
       $\lambda_{G,t}^* := \lambda_{G,t}^* + 1$ ;
      insert  $\lambda_{G,t}^*$  in  $P_0$ ;
    extract a color  $c$  from  $P_0$  and set  $f(v) := c$ ;
    insert color  $c$  into both  $L_v$  and  $P_t$ ;
    if  $r_v > \text{MAX-}r$  then
       $\text{MAX-}r := r_v$ ;
       $\text{DEEP} := v$ ;
  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  $P_j$  and insert  $c$  into  $P_{j-1}$ ;
      if  $j > 1$  then insert  $c$  into  $L_{\text{DEEP}}$ ;
```

Figure 1: The algorithm for optimal $L(1, \dots, 1)$ -coloring of interval graphs.

Consider the interval representation of G , and assume that the intervals (vertices) are indexed by increasing left endpoints, namely $l_1 < l_2 < \dots < l_n$. 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 scans the $2n$ interval endpoints from left to right, and it maintains a family of $t + 1$ sets of colors, called *palettes*, denoted by P_0, P_1, \dots, P_t . For each endpoint k , the palette P_0 contains the readily usable colors, while the palette P_t includes the colors used for the currently open intervals. Moreover, the generic palette P_i , with $1 \leq i \leq t - 1$, contains the colors that can be reused as soon as all the next i deepest intervals will be ended.

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. Moreover, the used color c is put both in the palette P_t and in the set L_v of colors depending on vertex v .

Whenever an interval v ends, that is a right endpoint r_v is encountered, every color c belonging to L_v is moved from its current palette, say P_j , to the previous palette P_{j-1} and it is inserted in the set L_{DEEP} of the colors depending on the current deepest interval DEEP .

Figure 1 illustrates the algorithm for optimal $L(1, \dots, 1)$ -coloring of interval graphs.

Lemma 4 *Consider to be at the beginning of iteration k of the algorithm. Let u be any vertex yet uncolored and let w be any vertex colored with some color c . Assume $c \in P_j$. Let z be the vertex with r_z maximum and such that $c \in L_z$. Then the following holds:*

(i) $d(u, w) > t - j$;

(ii) if $l_u > r_z$, then $d(u, w) > t - j + 1$;

(iii) if $l_u < r_z$, then $d(u, w) = t - j + 1$.

(iv) the minimum distance from z to a vertex colored c is $t - j$.

Proof The statement is vacuously true before the first iteration, when every vertex is still uncolored. Assume that the statement holds before iteration k , and let us show that it holds also after iteration k . There are two cases to consider, depending on whether k corresponds to a left or right interval endpoint.

Case 1: At iteration k , $k = l_v$ for some vertex v .

In this case, it is enough to check what happens for $w = v$. Indeed, during iteration k , v is the only vertex which gets assigned a color, namely c , which is inserted both into P_t and L_v . Moreover, no other color moves to a palette of lower index. Hence, $j = t$ and $z = v = w$. Since u is uncolored whereas v has just been colored, then $u \neq v$ which accounts for (i): $d(u, v) > 0 = t - t$. Clearly, if $l_u > r_z$, or in other words $l_u > r_w$, then $d(u, w) > 1$, which gives (ii). Moreover, if $l_u < r_z$, or in other words $l_u < r_w$, then $d(u, w) = 1$, which gives (iii). Finally, (iv) is trivial.

Case 2: At iteration k , $k = r_v$ for some vertex v .

Here one needs only to check what happens when $c \in L_v$. Now, since $c \in L_v$ and $l_u > r_v$, then $d(u, w) > t - j$ follows since (ii) was true at the beginning of iteration k . Furthermore, in case $j > 0$, then $c \in L_{\text{DEEP}}$. If $l_u > r_{\text{DEEP}}$, then $d(u, v) > t - j + 1$, since in any path from u to v the vertex closest to u must be uncolored. Moreover, (ii) follows from the fact that, clearly, $z = \text{DEEP}$. Besides, if $l_u < r_z$, or equivalently $l_u < r_{\text{DEEP}}$, then $d(u, w) = t - j + 1$ follows since (iv) was true at the beginning of iteration k . Finally, $d(\text{DEEP}, v) = 1$, which combined with (i) gives (iv). \square

Theorem 1 *The Interval- $L(1, \dots, 1)$ -coloring algorithm gives an optimal coloring and runs in $O(nt)$ time.*

Proof The correctness follows from Lemmas 2, 3, and 4. Indeed, according to Lemma 3, the algorithm scans the vertices by increasing left endpoints, coloring at each iteration the strongly-simplicial vertex v of the induced subgraph $G[\{1, \dots, v\}]$. Moreover, the algorithm assigns to v either a minimum brand new color or a color that belongs to P_0 . In the former case, the coloring of $G[\{1, \dots, v\}]$ is trivially legal and its optimality follows from Lemma 2. In the latter case, the coloring is optimal since no new color is added, and its legality follows from the fact that a color c comes back to P_0 only when all the uncolored vertices are at distance larger than t from the last vertex colored c by Lemma 4.

In order to evaluate the time complexity, all the palettes P_i , $0 \leq i \leq t$ and all the sets L_u , with $1 \leq u \leq n$, are implemented by double linked lists, so that insertions and extractions can be performed in constant time as follows.

At any iteration, any already introduced color c can belong either to P_0 or to exactly a palette P_j ($j > 0$) and exactly a set L_z . Thus, since no more than n 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 double linked list P_j .

When a color c is inserted in either a palette P_j or a set of colors L_z , such a color is added in front of the list and the vector entry $C[c]$ is accordingly updated. Overall an insertion takes constant time.

On the other hand, while an extraction from palette P_0 is always performed in constant time from the front of P_0 , the extraction of color c from palette P_j ($j > 0$) 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.

In order to evaluate the overall time complexity, observe that the algorithm consists of $2n$ iterations and, at every iteration k , each step takes $O(1)$ 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 to be 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 4. Hence, between two consecutive assignments of the same color c , there are at most $t + 1$ moves, each performed in a different iteration and each taking constant time. Let m_c be the overall number of vertices of G colored c . Therefore, the total number of moves for color c is at most $(t + 1)m_c$. Summing up over all the used colors, the overall number of moves is at most $\sum_c (t + 1)m_c = O(nt)$, since $\sum_c m_c = n$. In conclusion, the algorithm takes $O(nt)$ time provided that the interval representation of G is available and the $2n$ interval endpoints are sorted. \square

3.2 Approximate $L(\delta_1, 1, \dots, 1)$ -coloring

In this subsection, an algorithm is proposed to find an approximate $L(\delta_1, 1, \dots, 1)$ -coloring of interval graphs, which is a generalization of the Interval- $L(1, \dots, 1)$ -coloring algorithm presented in the previous subsection.

At first, the algorithm computes $\lambda_{G,1}^*$ and $\lambda_{G,t}^*$ invoking twice the Interval- $L(1, \dots, 1)$ -coloring algorithm. As before, the palettes P_0, P_1, \dots, P_t are maintained, but P_0 is initialized to the set of colors $\{0, 1, \dots, U\}$, where $U = \lambda_{G,t}^* + 2(\delta_1 - 1)\lambda_{G,1}^*$. As it will be shown later, such a color set is sufficiently large to obtain a legal $L(\delta_1, 1, \dots, 1)$ -coloring for G .

The main difference, with respect to the Interval- $L(1, \dots, 1)$ -coloring algorithm, relies on the fact that when a color c is extracted from P_0 and assigned to vertex v , the δ_1 -separation constraint must be guaranteed. Therefore, all the colors in $\{\max\{0, c - \delta_1 + 1\}, \dots, \min\{c + \delta_1 - 1, U\}\}$, but c itself, are inserted in P_1 . In this way, such colors cannot be reused until the interval v ends.

Figure 2 illustrates the algorithm for $L(\delta_1, 1, \dots, 1)$ -coloring of interval graphs. The overall time complexity of the algorithm is $O(n(t + \delta_1))$. Indeed, the δ_1 -separation constraint can be guaranteed in $O(\delta_1)$ time by storing also in the vector entry $C[c]$ the position of c within the double linked list P_0 , for each color c .

Theorem 2 *The Interval- $L(\delta_1, 1, \dots, 1)$ -coloring algorithm gives an approximate coloring using $\lambda_{G,t}^* + 2(\delta_1 - 1)\lambda_{G,1}^*$ as the largest color.*

Proof The correctness follows from Theorem 1 and from the fact that, once an interval is

Algorithm Interval- $L(\delta_1, 1, \dots, 1)$ -coloring ($G = (V, E), \delta_1, t$);

set $\lambda_{G,1}^*$ to the maximum color employed by the Interval- $L(1)$ -coloring algorithm;
 set $\lambda_{G,t}^*$ to the maximum color employed by the Interval- $L(1, \dots, 1)$ -coloring algorithm;
 set $U := \lambda_{G,t}^* + 2(\delta_1 - 1)\lambda_{G,1}^*$;
 set $L_v := \emptyset$, for $v \in V$;
 set $P_0 := \{0, 1, \dots, U\}$ and $P_i := \emptyset$ for $i = 1, \dots, t$;
 set $\text{MAX-}r := 0$;
 for $k := 1$ to $2n$ do
 if $k = l_v$ for some v , then
 extract a color c from P_0 and set $f(v) := c$;
 insert color c into both L_v and P_t ;
 for each color $\max\{0, c - \delta_1 + 1\} \leq \gamma \leq \min\{c + \delta_1 - 1, U\}$ do
 if $\gamma \in P_0$ then
 extract γ from P_0 and insert γ into P_1 ;
 if $r_v > \text{MAX-}r$ then
 $\text{MAX-}r := r_v$;
 $\text{DEEP} := v$;
 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 and insert c into P_{j-1} ;
 if $j > 1$ then insert c into L_{DEEP} ;

Figure 2: The $L(\delta_1, 1, \dots, 1)$ -coloring algorithm for interval graphs.

colored c , the δ_1 -separation constraint is achieved by inserting in P_1 the $2(\delta_1 - 1)$ closest colors to c .

One also needs to show that the colors initially in P_0 are enough to obtain a legal $L(\delta_1, 1, \dots, 1)$ -coloring of G . When a color c is assigned to an interval v , all intervals in $G[\{1, \dots, v\}]$ at distance smaller than or equal to t from v must get a different color since v is t -simplicial. Hence, all their colors, which are at most $\lambda_{G,t}^*$, must belong to $P_1 \cup P_2 \cup \dots \cup P_t$. Moreover, due to the δ_1 separation-constraint, at most $2(\delta_1 - 1)\lambda_{G,1}^*$ colors have been forced into P_1 . Hence, $|P_1 \cup P_2 \cup \dots \cup P_t| \leq \lambda_{G,t}^* + 2(\delta_1 - 1)\lambda_{G,1}^*$ and since initially there were $\lambda_{G,t}^* + 2(\delta_1 - 1)\lambda_{G,1}^* + 1$ colors in P_0 , it holds $|P_0| - |P_1 \cup P_2 \cup \dots \cup P_t| \geq 1$. Thus, there is always an available color that can be assigned to v .

In order to find the approximation factor, the ratio between the upper bound U on the maximum color used by the above algorithm and the lower bound, $L = \max\{\delta_1 \lambda_{G,1}^*, \lambda_{G,t}^*\}$ on the maximum color needed, given by Lemma 1, is

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

Note that, since the values of $\lambda_{G,1}^*$ and $\lambda_{G,t}^*$ can be computed in polynomial time invoking the Interval- $L(1, \dots, 1)$ -coloring algorithm, for any interval graph G , the ratio U/L can be computed exactly. However, as it is shown below, the worst value U/L can be bounded from above by a constant, independent of G , δ_1 and t . Indeed, two cases may arise.

If $\delta_1 \lambda_{G,1}^* \geq \lambda_{G,t}^*$, the above ratio U/L 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 U/L is

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

In conclusion, the Interval- $L(\delta_1, 1, \dots, 1)$ -coloring algorithm gives an approximate solution which uses at most 3 times the number of colors needed by any $L(\delta_1, 1, \dots, 1)$ -coloring. \square

3.3 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 the interval graphs for which all the intervals are of 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 $\{v, v+1, \dots, u-1, u\}$ forms a clique and $u \leq v + \lambda_{G,1}^*$ (as a consequence, the maximum vertex w at distance 2 from v verifies $w \leq v + 2\lambda_{G,1}^*$).

In this subsection, it is assumed that the unit interval graph to be colored is not a path, since otherwise the optimal $L(\delta_1, \delta_2)$ -coloring algorithm in [16] can be applied.

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

if $\delta_1 > 2\delta_2$ then
 for every vertex $v \in V$ do
 set $p := (v - 1) \bmod (2\lambda_{G,1}^* + 2)$;
 assign to v the color $f(v) := \begin{cases} \delta_1(\lambda_{G,1}^* - p) & \text{if } 0 \leq p \leq \lambda_{G,1}^* \\ \delta_1(\lambda_{G,1}^* - p) + \delta_2 & \text{if } \lambda_{G,1}^* + 1 \leq p \leq 2\lambda_{G,1}^* + 1 \end{cases}$
if $\delta_1 \leq 2\delta_2$ then
 for every vertex $v \in V$ do
 assign to v the color $f(v) := (2\delta_2(v - 1)) \bmod (2\delta_2\lambda_{G,1}^* + 3\delta_2)$;

Figure 3: The $L(\delta_1, \delta_2)$ -coloring algorithm for unit interval graphs.

In Figure 3, a linear time algorithm, called Unit-Interval- $L(\delta_1, \delta_2)$ -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 $\delta_1 \leq 2\delta_2$.

Theorem 3 *The Unit-Interval- $L(\delta_1, \delta_2)$ -coloring algorithm gives an approximate coloring using as the largest color $\delta_1\lambda_{G,1}^* + \delta_2$, if $\delta_1 > 2\delta_2$, or $2\delta_2\lambda_{G,1}^* + 3\delta_2$, if $\delta_1 \leq 2\delta_2$.*

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

$$0, \delta_1, 2\delta_1, \dots, \lambda_{G,1}^*\delta_1, \delta_2, \delta_1 + \delta_2, \delta_1 + 2\delta_2, \dots, \lambda_{G,1}^*\delta_1 + \delta_2.$$

Consider a vertex v colored $c = j\delta_1$, with $0 \leq j \leq \lambda_{G,1}^*$ (an 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 , since otherwise $\lambda_{G,1}^*$ would not be optimal. In order to verify the δ_1 separation-constraint, it remains to check that all the colors $c - \delta_1 + 1, \dots, c - 1, c + 1, \dots, c + \delta_1 - 1$ 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, since otherwise there would be a clique of size $\lambda_{G,1}^* + 2$, including vertices $v, v \pm 1, \dots, v \pm (\lambda_{G,1}^* + 1)$, which contradicts the optimality of $\lambda_{G,1}^*$. Moreover, the δ_2 separation-constraint trivially follows from the fact that the colors $c - \delta_2 + 1, \dots, c - 1, c + 1, \dots, c + \delta_2 - 1$ are never used.

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

$$0, 2\delta_2, 4\delta_2, \dots, 2(\lambda_{G,1}^* + 1)\delta_2, \delta_2, 3\delta_2, 5\delta_2, \dots, 2\lambda_{G,1}^*\delta_2 + \delta_2.$$

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 . As 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 $v, v \mp 1, \dots, v \mp (\lambda_{G,1}^* + 1)$ would form a clique, contradicting the optimality of $\lambda_{G,1}^*$. Furthermore, the colors $c - \delta_2 + 1, \dots, c - 1, c + 1, \dots, c + \delta_2 - 1$ are never used, and hence the δ_2 separation-constraint holds too.

In order to find the approximation factor, 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 $\delta_1 \lambda_{G,1}^*$ since $\delta_2 \lambda_{G,2}^* \leq 2\delta_2 \lambda_{G,1}^* < \delta_1 \lambda_{G,1}^*$. In contrast, when $\delta_1 \leq 2\delta_2$, 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 $\delta_1 \leq 2\delta_2$.

Therefore, it holds:

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

Although the ratio U/L can be evaluated exactly since the values of $\lambda_{G,1}^*$ and $\lambda_{G,2}^*$ can be computed in polynomial time invoking the Interval- $L(1, \dots, 1)$ -coloring algorithm, the worst value 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

$$\frac{\delta_1 \lambda_{G,1}^* + \delta_2}{\delta_1 \lambda_{G,1}^*} = 1 + \frac{\delta_2}{\delta_1} \frac{1}{\lambda_{G,1}^*} \leq \frac{3}{2},$$

because $\frac{\delta_2}{\delta_1} \frac{1}{\lambda_{G,1}^*} \leq \frac{1}{2}$ from the assumption that the unit interval graph is connected.

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

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

since $\frac{\delta_2}{\delta_1} \leq 1$ 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 $L = \delta_2 \lambda_{G,2}^*$,

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

since $\frac{\lambda_{G,1}^*}{\lambda_{G,2}^*} \leq 1$ and $\frac{1}{\lambda_{G,1}^*} \leq \frac{1}{2}$ as in the previous case.

In conclusion, the Unit-Interval- $L(\delta_1, \delta_2)$ -coloring algorithm gives an approximate solution which, in the worst case, is far from the optimal by a factor of 3. \square

It is worth to note 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 [14] 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. 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 $l(v)$ of a vertex v is equal to the distance $d(r, v)$. Thus, all the vertices adjacent to v are partitioned into its *father*, which is at level $l(v) - 1$, and into its children, which are at level $l(v) + 1$. The *height* h of a tree T is the maximum level of its vertices. An *ordered tree* is a rooted tree in which an ordering is imposed among all the children of v . From now on, for the sake of simplicity, an ordered tree will be called *tree*. Given a tree T of height h and an integer $\ell \leq h$, the *induced subtree* T_ℓ consists of all the vertices v with $l(v) \leq \ell$ as well as all the original edges among them. 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 $l(v) - i$). Besides, let $D_i(v)$ denote the set of the *descendants* of v at distance i from v (which clearly belong to level $l(v) + i$).

The following lemma suggests that the $L(1, \dots, 1)$ -coloring problem can be optimally solved in polynomial time on trees.

Lemma 5 *Every tree has a strongly-simplicial vertex.*

Proof Let T be a tree and consider a vertex x with $l(x) = h$. 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 $sp(x, u)$. Since $l(x) \geq \max\{l(u), l(v)\}$, 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. Since such a condition holds for any $t \leq 2h$, x is strongly-simplicial. \square

Lemmas 2 and 5 suggest to visit the tree in breadth-first-search order, namely scanning the vertices by increasing levels, and those at the same level from left to right. In this way, at every turn, a t -simplicial vertex v at level $l(v)$ of the induced subtree $T_{l(v)}$ is processed, for $1 \leq v \leq n$. For this purpose, hereafter, it is assumed that the vertices are numbered according to the above breadth-first-search order.

4.1 Optimal $L(1, \dots, 1)$ -coloring

In this subsection, an $O(nt)$ time algorithm is exhibited to find an optimal $L(1, \dots, 1)$ -coloring of trees. The algorithm first performs a preprocessing in order to compute, for each vertex x of T , the $t + 1$ lists D_i , $0 \leq i \leq t$. Such a computation can be performed in $O(nt)$ time by visiting the tree in postorder. The corresponding recursive procedure, called *Explore-Descendants*, is shown in Figure 4 and has to be invoked starting from the root r of T .

The Explore-Descendants procedure can be easily modified to compute in the same $O(nt)$ time also all the cardinalities $|D_i(x)|$ for every vertex x simply by substituting the last statement with $|D_i(x)| := |D_i(x)| + |D_{i-1}(x)|$.

The algorithm also uses another function, called *Up-Neighborhood*, illustrated in Figure 5.

The function Up-Neighborhood takes in input a vertex y and a distance *uplevel* and returns the set F of the vertices at distance no larger than *uplevel* from vertex y in the induced subtree

Procedure Explore-Descendents ($x, T = (V, E), t$);

set $D_0(x) := \{x\}$ and $D_i(x) := \emptyset$, for $1 \leq i \leq t$;
if x is not a leaf then
 for every child v of x do
 Explore-Descendents(v, T, t);
 for $i := 1$ to t do
 $D_i(x) := D_i(x) \cup D_{i-1}(v)$;

Figure 4: The recursive procedure to compute the lists of the descendents up to distance t .

Function Up-Neighborhood ($y, uplevel$): vertex-set;

set $F := \emptyset$;
set $anc := y$;
for $i := 1$ to $\lceil \frac{t}{2} \rceil - 1$ do
 set $anc := father(anc)$;
for $i := \lceil \frac{t}{2} \rceil$ to $uplevel$ do
 set $anc := father(anc)$;
 if $i = \lceil \frac{t}{2} \rceil$ then
 $F := F \cup D_{t-i-1}(anc)$;
 if t is odd then $F := F \cup D_{t-i}(anc)$;
 if $\lceil \frac{t}{2} \rceil < i < t$ then
 $F := F \cup D_{t-i-1}(anc) \cup D_{t-i}(anc)$;
 if $i = t$ then
 $F := F \cup D_{t-i}(anc)$;
return F ;

Figure 5: The function to compute the neighborhood of vertex y at distance no larger than $uplevel$ in $T_{l(y)}$ rooted at y .

$T_{l(y)}$, rooted at y . Conceptually, F corresponds to the set $F_{uplevel}(y)$ of vertices up to distance $uplevel$ traversed by a Breadth-First-Search starting from y in $T_{l(y)}$. However, computing F in such a way would imply to change the root of the subtree at every time the function is invoked. To avoid this, the tree is maintained with the original root, and the set F is computed from the sets of the descendants which have been obtained once for all by the Explore-Descendants procedure.

In practice, when $uplevel = t$, the function visits the vertices along the path from $anc_{\lceil \frac{t}{2} \rceil}(y)$ up to $anc_t(y)$, building the required neighborhood set $F_t(y)$ including the following descendent sets of the vertices on such a path:

$$F_t(y) = \begin{cases} D_0(anc_t(y)) \cup S \cup D_{\lfloor \frac{t}{2} \rfloor - 1}(anc_{\lceil \frac{t}{2} \rceil}(y)) & \text{if } t \text{ is even} \\ D_0(anc_t(y)) \cup S \cup D_{\lfloor \frac{t}{2} \rfloor - 1}(anc_{\lceil \frac{t}{2} \rceil}(y)) \cup D_{\lfloor \frac{t}{2} \rfloor}(anc_{\lceil \frac{t}{2} \rceil}(y)) & \text{if } t \text{ is odd} \end{cases}$$

and

$$S = D_0(anc_{t-1}(y)) \cup D_1(anc_{t-1}(y)) \cup \dots \cup D_{\lfloor \frac{t}{2} \rfloor - 2}(anc_{\lceil \frac{t}{2} \rceil + 1}(y)) \cup D_{\lfloor \frac{t}{2} \rfloor - 1}(anc_{\lceil \frac{t}{2} \rceil + 1}(y)).$$

Note that if $uplevel < t$, the path from $anc_{\lceil \frac{t}{2} \rceil}(y)$ to $anc_{uplevel}(y)$ ends at the root of T and only the sets of descendants associated with those vertices are included.

Clearly, the largest color $\lambda_{T,t}^*$ needed by any $L(1, \dots, 1)$ -coloring of T is given by the size of the largest neighborhood set returned by the Up-Neighborhood function. Therefore, $\lambda_{T,t}^* = \max_{y \in V} \{|F_t(y)|\}$, which can be computed in $O(nt)$ time simply modifying the Up-Neighborhood function so as to manipulate the set cardinalities in place of the sets themselves.

The Tree- $L(1, \dots, 1)$ -coloring algorithm, illustrated in Figure 6, starts invoking the Explore-Descendants procedure to compute the sets of the descendants and the largest color $\lambda_{T,t}^*$, and henceforth initializes accordingly the palette P . Then, observed that all the vertices in the uppermost $\lfloor \frac{t}{2} \rfloor + 1$ levels are all mutually at distance at most t , the algorithm colors them with all different colors extracted from P . The rest of the tree is colored level by level. For each level $\ell > \lfloor \frac{t}{2} \rfloor$, P is set to $\{0, \dots, \lambda_{T,t}^*\}$. The vertices in level ℓ are then colored from left to right, identifying groups of consecutive vertices which share the same palette. Each group contains all the vertices such that the level of their lowest common ancestor lca is larger than or equal to $\ell - \lfloor \frac{t}{2} \rfloor + 1$. Each group is identified by its leftmost vertex x and its rightmost vertex $last_x$. When a new level ℓ starts, x is simply the leftmost vertex in level ℓ . Otherwise, two consecutive groups $\{old_x, \dots, last_{old_x}\}$ and $\{x, \dots, last_x\}$ at the same level ℓ verify $x = last_{old_x} + 1$. To color the leftmost group, the Up-Neighborhood function is invoked with $uplevel = \min\{t, \ell\}$. Such a procedure returns in F the vertices whose colors are no longer available, and which are then removed from the palette P . Then the colors in P are used to color all the vertices in $\{x, \dots, last_x\}$. For each of the remaining groups at the same level, the palette is updated by invoking twice the Up-Neighborhood function. The first call $\text{Up-Neighborhood}(old_x, uplevel)$ returns in F the set $F_{uplevel}(old_x)$ of the vertices whose colors are again available. Such colors are then added to the palette P . The second call $\text{Up-Neighborhood}(x, uplevel)$ returns in F the set $F_{uplevel}(x)$ of the vertices whose colors are forbidden, which are then removed from the palette P . As before the colors currently in P are then used to color the vertices of the group.

Theorem 4 *The Tree- $L(1, \dots, 1)$ -coloring algorithm gives an optimal coloring and runs in $O(nt)$ time.*

Algorithm Tree- $L(1, \dots, 1)$ -coloring ($T = (V, E), t$);

Explore-Descendants(r, T, t);
 set $\lambda_{T,t}^* := \max_{x \in V} \{|F_t(x)|\}$;
 set $P := \{0, \dots, \lambda_{T,t}^*\}$;
 for $\ell := 0$ to $\lfloor \frac{t}{2} \rfloor$ do
 for each vertex x with $l(x) = \ell$ do
 extract a color c from P and set $f(x) := c$;
 for $\ell := \lfloor \frac{t}{2} \rfloor + 1$ to h do
 set $P := \{0, \dots, \lambda_{T,t}^*\}$;
 set x to the leftmost vertex with $l(x) = \ell$;
 set $last_x$ to the rightmost vertex in $D_{\lceil \frac{t}{2} \rceil}(anc_{\lceil \frac{t}{2} \rceil}(y))$;
 set $uplevel$ to $\min\{t, \ell\}$;
 set $F := \text{Up-Neighborhood}(x, uplevel)$;
 set $P := P - \{c : c = f(u), u \in F\}$;
 for each vertex u with $x \leq u \leq last_x$ do
 extract a color c from P and set $f(u) := c$;
 while $l(last_x + 1) = \ell$ do
 set $old_x := x, x := last_x + 1$;
 set lca to the lowest common ancestor between x and old_x ;
 set $uplevel$ to $\min\{t, \ell - l(lca) - 1\}$;
 set $F := \text{Up-Neighborhood}(old_x, uplevel)$;
 set $P := P \cup \{c : c = f(u), u \in F\}$;
 set $F := \text{Up-Neighborhood}(x, uplevel)$;
 set $P := P - \{c : c = f(u), u \in F\}$;
 for each vertex u with $x \leq u \leq last_x$ do
 extract a color c from P and set $f(u) := c$;

Figure 6: The algorithm for optimal $L(1, \dots, 1)$ -coloring of trees.

Proof The correctness follows from Lemma 5. Indeed, the algorithm scans the vertices in breadth-first order, and therefore colors at every turn a strongly-simplicial vertex. This guarantees that, to find the vertices which are at mutual distance no more than t (and therefore those vertices which must get all different colors), it is enough to compute the set $F_t(x)$ of all the vertices at distance at most t from the current strongly-simplicial vertex x . Such a set $F_t(x)$ is computed from scratch by function Up-Neighborhood every time it is invoked on the leftmost uncolored vertex x of a new level ℓ . Note that $F_t(x) = F_t(u)$ for all vertices $u \in \{x, \dots, last_x\}$. That is, such vertices have all the same neighborhood in T_ℓ , which is computed only once in correspondence of the leftmost vertex x of each group. Moreover, consider the leftmost vertices old_x and x of two consecutive groups at level ℓ . Then, the sets $F_t(old_x)$ and $F_t(x)$ differ only for the vertices at levels $\ell-1, \dots, l(lca)$, where lca is the lowest common ancestor between x and old_x . Therefore, the palette P is accordingly updated by invoking twice the function Up-Neighborhood with $uplevel = \min\{t, \ell - l(lca) - 1\}$.

The optimality follows from the fact that the algorithm uses only the colors in $\{0, \dots, \lambda_{T,t}^*\}$, and $\lambda_{T,t}^*$ is a lower bound for the largest color needed.

In order to evaluate the time complexity, observe first that both the computations of $last_x$ and lca require $O(t)$ time.

The palette P is implemented by means of a double linked list of colors and a vector C indexed by colors. Each entry $C[c]$ stores a pointer to the position of color c within the double linked list P . When a color c is inserted in P , such a color is added in 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 P is performed in constant time by retrieving from $C[c]$ the pointer to c within the list P and accordingly updating both P and C .

In order to count how many times a color is inserted into or extracted from P , let us count how many times a generic vertex v is added to or removed from the neighborhood sets F 's. Then, given a vertex v , consider a level ℓ along with the leftmost group at level ℓ for which v belongs to some neighborhood set. Let x be the leftmost vertex of such a group and $F_t(x)$ be its neighborhood. Vertex v remains in the set $F_t(x)$ for any leftmost vertex y of a group on the right of x as long as the lowest common ancestor lca between x and y verifies $l(lca) \geq l(v)$, because, as said before, $F_t(x)$ and $F_t(y)$ differ only for the vertices at levels $\ell-1, \dots, l(lca)$. Moreover, since the level of the ancestor lca between x and y decreases while y moves rightwards away from x , v will leave the set $F_t(y)$ as soon as $l(lca) < l(v)$, and it will never appear in any neighborhood set for the remaining vertices at level ℓ . Therefore, for the same level ℓ , once v enters in a neighborhood, it remains in the neighborhood of the leftmost vertex for some consecutive groups, and finally it leaves. Thus, v is involved twice by the Up-Neighborhood function: the first time to be inserted in F and the second time to be removed from F .

Since each vertex v can appear in sets F 's only for t consecutive levels $l(v), \dots, l(v) + t$, v is involved in exactly t insertions into the neighborhood F 's and exactly t deletions from the neighborhood F 's while coloring the entire tree.

In conclusion, overall $O(nt)$ operations are performed by the Tree- $L(1, \dots, 1)$ -coloring algorithm. \square

4.2 Approximate $L(\delta_1, 1, \dots, 1)$ -coloring

In this subsection, an algorithm is proposed to find an approximate $L(\delta_1, 1, \dots, 1)$ -coloring of trees, which is a generalization of the Tree- $L(1, \dots, 1)$ -coloring algorithm devised above.

As seen in Subsection 3.2, the Tree- $L(\delta_1, 1, \dots, 1)$ -coloring algorithm computes $\lambda_{T,t}^*$ by means of the Tree- $L(1, \dots, 1)$ -coloring algorithm and uses an enriched palette $P = \{0, 1, \dots, U\}$, where $U = \lambda_{T,t}^* + 2(\delta_1 - 1)$.

The Tree- $L(\delta_1, 1, \dots, 1)$ -coloring algorithm differs from the Tree- $L(1, \dots, 1)$ -coloring algorithm, in addition to the use of the above mentioned enriched palette P , in how a vertex is colored. Specifically, in order to satisfy the δ_1 separation constraint, the instructions to color a vertex u become:

find a color c in P such that $|c - f(\text{anc}_1(u))| \geq \delta_1$;
extract c from P and set $f(u) := c$;

Therefore, to color a single vertex, $O(\delta_1)$ time is required and hence an overall time complexity of $O(n(t + \delta_1))$ results.

Theorem 5 *The Tree- $L(\delta_1, 1, \dots, 1)$ -coloring algorithm gives an approximate coloring using $\lambda_{T,t}^* + 2(\delta_1 - 1)$ as the largest color.*

Proof The proof is similar to that of Theorem 2. In particular, one also needs to show that the colors initially in P are enough to obtain a legal $L(\delta_1, 1, \dots, 1)$ -coloring of T . When a vertex u must be colored, the number of forbidden colors is at most $\lambda_{T,t}^*$ (which can happen when u is the last vertex to be colored in the set F of vertices at distance at most t from u such that $|F| = \lambda_{T,t}^*$) plus additional $2(\delta_1 - 1)$ (due to the δ_1 separation-constraint that must hold between u and its father). Since initially there were $\lambda_{T,t}^* + 2\delta_1 - 1$ colors in P and since at most $\lambda_{T,t}^* + 2\delta_1 - 2$ colors are forbidden when u must be colored, there is an available color that can be assigned to u .

The Tree- $L(\delta_1, 1, \dots, 1)$ -coloring algorithm uses $U = \lambda_{T,t}^* + 2\delta_1 - 2$ as the largest color while the largest color is at least $L = \max\{\lambda_{T,t}^*, \delta_1 \lambda_{T,1}^*\}$. Therefore, the ratio U/L results to be

$$\frac{U}{L} = \frac{\lambda_{T,t}^* + 2\delta_1 - 2}{\max\{\lambda_{T,t}^*, \delta_1 \lambda_{T,1}^*\}}.$$

If $\delta_1 \lambda_{T,1}^* \geq \lambda_{T,t}^*$, the above ratio is

$$\frac{U}{L} = \frac{\lambda_{T,t}^* + 2\delta_1 - 2}{\delta_1 \lambda_{T,1}^*} \leq \frac{2}{\lambda_{T,1}^*} + \frac{\lambda_{T,t}^*}{\delta_1 \lambda_{T,1}^*} \leq 2 + \frac{\lambda_{T,t}^*}{\lambda_{T,t}^*} \leq 3.$$

If $\delta_1 \lambda_{T,1}^* < \lambda_{T,t}^*$, the ratio becomes

$$\frac{U}{L} = \frac{\lambda_{T,t}^* + 2\delta_1 - 2}{\lambda_{T,t}^*} \leq 1 + \frac{2\delta_1}{\lambda_{T,t}^*} \leq 1 + \frac{2}{\lambda_{T,1}^*} \leq 3.$$

Therefore, the algorithm gives an approximate solution which, in the worst case, is far from the optimal by a factor of 3. \square

4.3 Optimal $L(2, 1, 1)$ -coloring of complete binary trees

In this subsection, an $O(n)$ time algorithm for optimal $L(2, 1, 1)$ -coloring of complete binary trees is presented.

A *complete binary tree* T of size n is a rooted tree in which all the leaves are at the same level and each internal vertex has exactly 2 children. The height of T is $h = \lceil \log_2 n \rceil$.

For easy of exposition, in this subsection, the root of T is denoted as $r = (0, 0)$, while the $(j + 1)$ -th vertex at level i is denoted as $u = (i, j)$, where $0 < i < h$ and $0 \leq j = l(u) \leq 2^i - 1$. Note that $u = (i, j)$ is connected to its father $v = (i - 1, \lfloor \frac{j}{2} \rfloor)$, and to its left and right child $z = (i + 1, 2j)$ and $w = (i + 1, 2j + 1)$, respectively.

In order to derive a lower bound on the maximum color needed by any $L(2, 1, 1)$ -coloring of a complete binary tree, the following preliminary result is mentioned.

Lemma 6 [11] *Consider a star graph S_ρ , consisting of a center vertex γ with degree ρ and ρ vertices of degree 1. Let the center γ of S_ρ be already colored. Then, the largest color λ required for an $L(2, 1)$ -coloring of S_ρ is at least:*

$$k = \begin{cases} \rho + 1 & \text{if } f(\gamma) = 0 \text{ or } f(\gamma) = \rho + 1, \\ \rho + 2 & \text{if } 0 < f(\gamma) < \rho + 1, \end{cases}$$

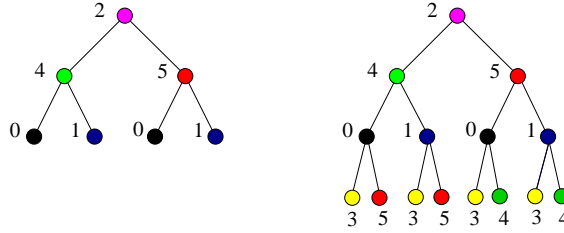


Figure 7: An optimal $L(2, 1, 1)$ -coloring for a complete binary tree of size $n = 7$ and $n = 15$.

Lemma 7 *An optimal $L(2, 1, 1)$ -coloring for a complete binary tree T of size n uses as the largest color*

$$\lambda^* = \begin{cases} 3 & \text{if } n = 3 \\ 5 & \text{if } n = 7 \text{ or } n = 15 \end{cases}$$

Proof When $n = 3, 7$ or 15 , the claim follows immediately from Lemma 6. Indeed, for $n = 3$ there is a vertex of T with degree 2, while for $n = 7$ and $n = 15$ at least two vertices of T have degree 4. An optimal $L(2, 1, 1)$ -coloring for $n = 7$ and $n = 15$ is provided in Figure 7. \square

Lemma 8 *There is an $L(2, 1, 1)$ -coloring for a complete binary tree T of size $n \geq 31$ only if $\lambda^* \geq 6$.*

Turn	0	1
0	0	1
1	0	2
2	2	1

Shift	0	1
0	1	3
1	1	-1
2	-1	3

	0	1	2	3	4	5
Color	0	6	1	4	2	5

Table 1: The functions **Turn**, **Shift**, **Color** used in the $\text{Tree-}L(2, 1, 1)$ -coloring algorithm.

Proof When $n \geq 31$, given a complete binary tree $T = (V, E)$, consider the augmented graph $A_{T,3} = (V, E')$, the vertex $u = (i + 3, 8j)$, with $0 \leq i \leq h - 3$, $0 \leq j \leq 2^i - 1$, and the subset S_u of vertices at reciprocal distance no more than 3, where $S_u = \{(i + 3, 8j), (i + 3, 8j + 1), (i + 2, 4j), (i + 2, 4j + 1), (i + 1, 2j), (i, j)\}$. Since S_u forms a clique in $A_{T,3}$, there is an $L(2, 1, 1)$ -coloring for a complete binary tree T only if $\lambda^* \geq 5$. However, one more color is necessary to satisfy the 2 separation-constraint as noted below.

First of all, consider S_u and observe that, to color S_u with 6 colors, $|f(p) - f(l)| \geq 3$ must hold for the father-child pair $p = (i + 1, 2j)$ and $l = (i + 2, 4j)$. Now, let $v = (i + 3, 8j + 2)$, and consider $S_v = \{(i + 3, 8j + 2), (i + 3, 8j + 3), (i + 2, 4j + 1), (i + 2, 4j), (i + 1, 2j), (i, j)\}$. Again, for the father-child pair $p = (i + 1, 2j)$ and $r = (i + 2, 4j + 1)$, $|f(p) - f(r)| \geq 3$ must result in order to color S_v with 6 colors. Repeating the same reasoning for S_w and S_z , where $w = (i + 3, 8j + 4)$ and $z = (i + 3, 8j + 6)$, implies that the colors assigned to any father-child pair at levels $i + 1$ and $i + 2$ must be at least 3 apart.

Since the same property holds for any father-child pair at levels $i + 2$ and $i + 3$, all the three vertices adjacent to any vertex v at level $i + 2$ must get a color which is at least 3 apart from that of v . Observing that there must be at least one vertex at level $i + 2$ whose color is different from 0 and 5, from Lemma 6, $\lambda^* \geq 6$ follows. \square

In order to describe the $\text{Tree-}L(2, 1, 1)$ -coloring algorithm for complete binary trees, consider for any vertex $u = (i, j)$ the binary representation of j using i bits. Note that the k -th bit of such representation is 1 (resp., 0) if and only if the k -th vertex on the unique path from the root to u is a right (resp., left) child.

Moreover, let the functions **Turn**: $\{0, 1, 2\} \times \{0, 1\} \rightarrow \{0, 1, 2\}$, **Shift**: $\{0, 1, 2\} \times \{0, 1\} \rightarrow \{-1, 1, 3\}$, and **Color**: $\{0, \dots, 5\} \rightarrow \{0, \dots, 6\}$ be defined as shown in Table 1.

The tree coloring is derived from the $L(2, 1, 1)$ -coloring of an infinite bidimensional grid B with 2 columns, by mapping the entire tree T on B . For the sake of simplicity, the infinite grid can be represented by a 6×2 grid with toroidal columns and its $L(2, 1, 1)$ -coloring is given in Figure 8. As an example, a mapping for a tree of size $n = 31$ is also shown in Figure 8. Once B has been colored, the root $r = (0, 0)$ of T is mapped to a vertex $(i, 1)$ of B colored with 0. The left child a (resp., right child b) of r is mapped to the vertex $(i + 1, 1)$ (resp., $(i - 1, 1)$) which is below (resp., above) $(i, 1)$ on B . Then, the left and right children c and d of the left child a of the root are mapped, respectively, to the vertices $(i - 2, 1)$ and $(i - 1, 0)$. Similarly, the left and right children e and f of b are mapped, respectively, to the vertices $(i + 2, 1)$ and $(i + 1, 0)$. Again, the left and right children of the tree vertex c , mapped into the grid vertex $(i + 2, 1)$, are mapped to the vertices $(i + 3, 1)$ and $(i + 2, 0)$, respectively, and so on. In particular, the left and right children o and p of the tree vertex f , mapped into the grid vertex $(i - 1, 0)$, are mapped to the

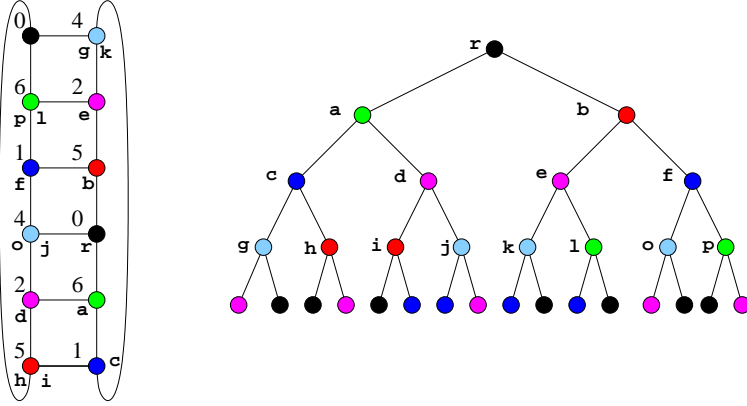


Figure 8: Mapping of a complete binary tree into an infinite bidimensional grid.

vertices $(i - 2, 0)$ and $(i - 1, 1)$, respectively.

The above reasoning can be then summarized as follows. Given a tree vertex u , mapped to the grid vertex $v = (p, q)$, its right child is always mapped to the grid vertex $(p, (q + 1) \bmod 2)$, except when the father of u has already been mapped to such a vertex. In such a case, the right child of v is mapped to the grid vertex above v , that is $(p - 1, q)$.

Similarly, given the tree vertex u , mapped to the grid vertex $v = (p, q)$, its left child is mapped to the grid vertex $(p - 1, q)$ except when the father of u has already been mapped to such a vertex. When this happens, the left child of v is mapped to the grid vertex below v , that is to $(p + 1, q)$. Hence, the left and right children of a vertex u are mapped to different grid vertices depending on where their father has been mapped.

Iterating this argument, it follows that the left and right children are mapped to different grid vertices depending on how many right vertices, or *turns*, there are on the path from the root to u .

In particular, the number of turns $W(i, j)$ on the path from the root $r = (0, 0)$ to the tree vertex $u = (i, j)$ are counted by the function `Turn` as follows:

- if $u = (i, j)$ is a right vertex and there was 0 or 1 turn already counted, the number of turns increases by 1, while if there were 2 turns the number of turns decreases by 1;
- if $u = (i, j)$ is a left vertex, the number of turns remains the same except when there was already 1 turn counted; in this case, the number of turns returns to 0.

Intuitively, for a given vertex $u = (i, j)$ at level i of the path, $W(i, j) = 0$ means that the mapping of the tree path is proceeding southwards on the grid, $W(i, j) = 2$ means that it is proceeding northwards, while $W(i, j) = 1$ means that it is crossing a horizontal edge.

A tree vertex, once mapped to a grid vertex, inherits the same color. By the optimal $L(2, 1, 1)$ -coloring of a grid given in [3], the grid B is colored in a cyclic way by repeating on the two columns the color pattern sequence 0, 6, 1, 4, 2, 5, which is specified by the function `Color`.

The algorithm `Tree- $L(2, 1, 1)$ -coloring`, instead of computing the grid vertex to which each tree vertex is mapped, directly maps each grid edge to a shift on the color pattern sequence, thus

Algorithm Tree- $L(2, 1, 1)$ -coloring (T, n);

```
set  $f(0, 0) := 0$ ,  $W(0, 0) := 1$ ,  $s(0, 0) := 0$ ;  
for  $i := 1$  to  $\lfloor \log_2 n \rfloor$  do  
  for  $j := 0$  to  $2^i - 1$  do  
    set  $W(i, j) := \text{Turn} \left[ W(i-1, \lfloor \frac{j}{2} \rfloor), j \bmod 2 \right]$ ;  
    set  $s(i, j) := \left( s(i-1, \lfloor \frac{j}{2} \rfloor) + \text{Shift} \left[ W(i-1, \lfloor \frac{j}{2} \rfloor), j \bmod 2 \right] \right) \bmod 6$ ;  
    set  $f(i, j) := \text{Color}[s(i, j)]$ ;
```

Figure 9: The algorithm for optimal $L(2, 1, 1)$ -coloring of complete binary trees.

obtaining the color of a tree vertex as the sum of the shifts along the tree path. In particular, crossing a vertical edge upwards corresponds to one left cyclic shift on the pattern sequence, crossing a vertical edge downwards corresponds to one right cyclic shift, and crossing on a horizontal edge corresponds to a cyclic shift of three positions. In conclusion, given the walking direction W , the function `shift` maps any tree edge into a specific grid edge, and returns the cyclic shift on the color pattern sequence.

Summarizing, the Tree- $L(2, 1, 1)$ -coloring algorithm for complete binary trees of size $n \geq 31$ is given in Figure 9. It stores, for each vertex $u = (i, j)$, two local variables $W(i, j)$ and $s(i, j)$. Moreover, the algorithm uses, as look-up tables, the above mentioned arrays `Shift[0..2, 0..1]`, `Turn[0..2, 0..1]`, and `Color[0..5]`.

Clearly, the entire complete binary tree can be colored in $O(n)$ time.

Theorem 6 *The Tree- $L(2, 1, 1)$ -coloring algorithm is optimal for complete binary trees of size $n \geq 31$.*

Proof Since the $L(2, 1, 1)$ -coloring of the grid with $\lambda^* = 6$ satisfies the 2 separation-constraint, and adjacent tree vertices are mapped to adjacent grid vertices, the Tree- $L(2, 1, 1)$ -coloring algorithm also verifies the 2 separation-constraint.

To show that the same color is reused in vertices at distance at least 4, observe that any two consecutive tree edges are mapped to two different grid edges. In other words, two consecutive tree edges cannot be mapped to the same grid edge. For example, consider a horizontal grid edge. It can be crossed when a right vertex $u = (i, j)$ is encountered on the tree path and the number of turns is either 0 (i.e., $W(i, j) = 0$) or 2 (i.e., $W(i, j) = 2$). However, in the next move, since the number of turns is 1 (i.e., $W(i, j) = 1$), both the left and right vertices are mapped to vertical grid edges, and the horizontal grid edge is not used. Hence, on every grid path that corresponds to a tree path, the minimum distance between two vertices colored the same is 4. The optimality follows immediately from Lemma 8. \square

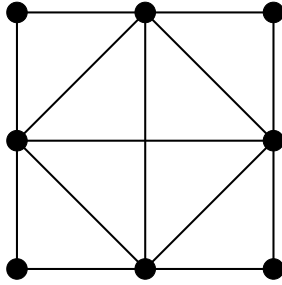


Figure 10: A chordal graph with no 2-simplicial vertex.

5 Conclusion

This paper has considered the channel assignment problem for particular separation vectors and two specific classes of graphs – trees and interval graphs. Based on the notions of t -simplicial and strongly-simplicial vertices, $O(nt)$ time algorithms have been proposed to find optimal $L(1, \dots, 1)$ -colorings on trees and interval graphs. Such algorithms have been generalized to find approximate $L(\delta_1, 1, \dots, 1)$ -colorings on the same classes of graphs. Moreover, an approximate $L(\delta_1, \delta_2)$ -coloring of unit interval graphs as well as an optimal $L(2, 1, 1)$ -coloring of complete binary trees have been presented.

Several questions remain open. For instance, one could devise polynomial time algorithms for finding optimal $L(\delta_1, 1, \dots, 1)$ -colorings of interval graphs and trees, as well as optimal $L(\delta_1, \delta_2)$ -colorings of unit interval graphs.

Moreover, one could search for further classes of graphs that verify the strongly-simplicial property. Good candidates could seem chordal graphs, since they include interval graphs and they verify the 1-simplicial property. Unfortunately, as shown in Figure 10, chordal graphs do not verify already the 2-simplicial property.

References

- [1] R. Battiti, A.A. Bertossi, and M.A. Bonuccelli, “Assigning Codes in Wireless Networks: Bounds and Scaling Properties”, *Wireless Networks*, Vol. 5, 1999, pp. 195-209.
- [2] A.A. Bertossi and M.C. Pinotti, “Mappings for Conflict-Free Access of Paths in Bidimensional Arrays, Circular Lists, and Complete Trees”, *Journal of Parallel and Distributed Computing*, to appear 2002.
- [3] A.A. Bertossi, M.C. Pinotti, and R. Tan, “Efficient Use of Radio Spectrum in Wireless Networks with Channel Separation between Close Stations”, *DIAL M for Mobility; Int’l ACM Workshop on Discrete Algorithms and Methods for Mobile Computing*, Boston, August 2000.
- [4] A.A. Bertossi, M.C. Pinotti, and R.B. Tan, “Channel Assignment with Separation for Special Classes of Wireless Networks: Grids and Rings”, *Computing Issues in Wireless Networks and Mobile Computing*, IPDPS, Fort Lauderdale, Florida, April 2002.
- [5] H.L. Bodlaender, T. Kloks, R.B. Tan, and J. van Leeuwen, “Approximation λ -Coloring on Graphs”, *STACS*, 2000.

- [6] K.S. Booth and G.S. Lueker, "Linear Algorithms to Recognize Interval Graphs and Test for the Consecutive Ones Property", *Seventh Annual ACM Symposium on Theory of Computing*, Albuquerque, New Mexico, 1975, pp. 255–265.
- [7] A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph Classes: a Survey*, SIAM, Philadelphia, PA, 1999.
- [8] G. J. Chang and D. Kuo, "The $L(2, 1)$ -Labeling Problem on Graphs", *SIAM Journal on Discrete Mathematics*, Vol. 9, 1996, pp. 309-316.
- [9] I. Chlamtac and S.S. Pinter, "Distributed Nodes Organizations Algorithm for Channel Access in a Multihop Dynamic Radio Network", *IEEE Transactions on Computers*, Vol. 36, 1987, pp. 728-737.
- [10] D.G. Corneil, S. Olariu, and L. Stewart, "The Ultimate Interval Graph Recognition Algorithm?", *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1998, pp. 175-180.
- [11] J. R. Griggs and R.K. Yeh, "Labelling Graphs with a Condition at Distance 2", *SIAM Journal on Discrete Mathematics*, Vol. 5, 1992, pp. 586-595.
- [12] W.K. Hale, "Frequency Assignment: Theory and Application", *Proceedings of the IEEE*, Vol. 68, 1980, pp. 1497-1514.
- [13] S.T. McCormick, "Optimal Approximation of Sparse Hessians and its Equivalence to a Graph Coloring Problem", *Mathematical Programming*, Vol. 26, 1983, pp. 153–171.
- [14] D. Sakai, "Labeling Chordal Graphs: Distance Two Condition", *SIAM Journal on Discrete Mathematics*, Vol. 7, 1994, pp. 133-140.
- [15] A. Sen, T. Roxborough, and S. Medidi, "Upper and Lower Bounds of a Class of Channel Assignment Problems in Cellular Networks", Technical Report, Arizona State University, 1997
- [16] J. Van den Heuvel, R. A. Leese, and M.A. Shepherd, "Graph Labelling and Radio Channel Assignment", *Journal of Graph Theory*, Vol. 29, 1998, pp. 263-283.