Algorithms for Finding Generalized Coloring of Trees

TANVEER AWAL[1]
M. MAHBUBUZZAMAN[2]
MD. ABUL KASHEM[3]

[(1,3)]Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka -1000, Bangladesh
[2]American Megatrends Inc.
Oakbrook Parkway Norcross, GA 30093, USA
[1]tanveerawal@cse.buet.ac.bd
[2]tachyon77@gmail.com
[3]kashem@cse.buet.ac.bd

Abstract. Let l be a positive integer, and G be a graph with nonnegative integer weights on edges. Then a generalized vertex-coloring, called an l-vertex-coloring of G, is an assignment of colors to the vertices in such a way that any two vertices u and v get different colors if the distance between u and v in G is at most l. A coloring is optimal if it uses minimum number of distinct colors. The l-vertex-coloring problem is to find an optimal l-vertex-coloring of a graph G. In this paper we present an $O(n^2 + n\Delta^{l+1})$ time algorithm to find an l-vertex-coloring of a tree T, where Δ is the maximum degree of T. The algorithm takes $O(n^2)$ time if both l and Δ are bounded integers. We compute the upper bound of colors to be $1 + \Delta\frac{(\Delta-1)^{\lceil l/2 \rceil}-1}{(\Delta-2)}$. We also present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the l-edge-coloring problem of trees. If both l and Δ are bounded integers, this algorithm also takes $O(n^2)$ time.

Keywords: Algorithm, Chordal Graph, l-chromatic-number, l-edge-coloring, l-vertex-coloring, Graph, Tree.

(Received January 25, 2011 / Accepted May 09, 2011)

1 Introduction

A *vertex coloring* of a graph G is an assignment of colors to the vertices in such a way that any adjacent vertices get different colors [14]. Let l be a positive integer, and G be a graph with nonnegative integer weights on edges. Then an *l-vertex-coloring* of G, is an assignment of colors to the vertices in such a way that any two vertices u and v get different colors if $\text{dist}(u, v) \leq l$, where $dist(u, v)$ denotes the distance between u and v in G, that is the length of the shortest path between u and v in G. Clearly an ordinary vertex coloring is a 1-vertex-coloring of a graph when weight of each edge is one. The *chromatic number* $\chi(G)$ of a graph G is the minimum number of colors needed to color the vertices of G. The *l-chromatic-number* or the *l-chromatic-index* $\chi_l(G)$ of a graph G is the minimum number of distinct colors needed to perform an l-vertex-coloring of G. The *l-vertex-coloring problem* or the *distance-vertex-coloring problem* is to compute the l-chromatic-index $\chi_l(G)$ of a given graph G. For example, Figure 1 depicts a 4-vertex-coloring of a graph G using four colors, where a number next to an edge is its weight and a number next to a vertex is its color. One can easily observe that $\chi_4(G) = 4$ for G in Figure 1.

Vertex coloring has diverse applications in problems such as time tabling and scheduling, frequency assignment for spectrum, register allocation in compiler, pattern matching, analysis of biological and archeological

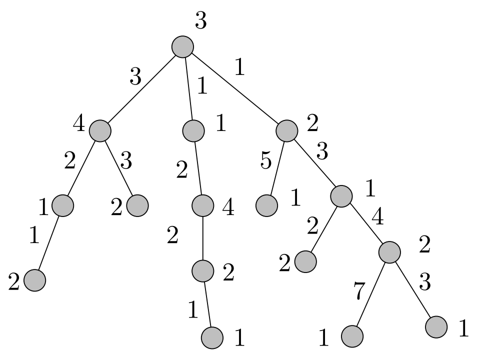

Figure 1: A 4-vertex-coloring of a graph with four colors

data, etc. In a university we may want to assign time slots for final examinations so that two courses with a common student have different time slots. The minimum number of slots needed to schedule examinations without conflict is the chromatic number of the graph in which two courses are adjacent if they have a common student. Compiler optimization is the canonical application for coloring, where we seek to schedule the use of a finite numbers of registers. In a program fragment to be optimized, each variable has a range of times during which its value must be kept intact, in particular, after it is initialized and before its final use. Any two variables whose life spans intersect cannot be associated with the same register. Edge between any two vertices representing variables indicates that the variable life spans intersect. A coloring of the vertices of this graph assigns the variables to classes such that variables with the same color do not clash and so can be assigned to the same register.

Since the ordinary vertex coloring problem is NP-hard [6], the l-vertex-coloring problem is NP-hard in general [15]. So it is very unlikely that there exists an efficient algorithm to solve the l-vertex-coloring problem for general graphs. However, Zhou et al. presented an $O(n^3 + n(\alpha+1)^{2^{2(k+1)(l+2)+1}})$ time algorithm to solve the l-vertex-coloring problem for partial k-trees, that is, the class of graphs of treewidth bounded by a fixed constant k [15]. Here a partial k-tree has an l-vertex-coloring: $V \to C$, where C is a set of colors and $|C| = \alpha$ and n is the number of vertices in G. If both k and l are bounded integers, then their algorithm runs in polynomial time. Note that a tree is a partial 1-tree, whereas a series-parallel graph is a partial 2-tree. Thus puttting $k = 1$ and 2 in their algorithm, we obtain an l-vertex-coloring algorithm for trees having time complexity of $O(n^3 + n(\alpha+1)^{2^{4l+9}})$ and an l-vertex-coloring algorithm for series-parallel graphs having time complexity of $O(n^3 + n(\alpha+1)^{2^{6l+13}})$ respectively. There is a polynomial-time 2-approximation algorithm for the l-vertex-coloring problem on planar graphs [1]. The algorithm for l-vertex-coloring of a tree T, presented in [9], uses a greedy strategy to assign colors to the nodes in a post order fashion and runs in $O(n\frac{\Delta^{l+1}-1)}{\Delta-1})$ time, where Δ is the maximum degree of T. Though this algorithm runs in linear time if both l and Δ are bounded integers, it is not correct and does not guarantee an optimal solution. Kashem et. al. showed that the l-vertex-coloring problem for series-parallel graphs can be solved in $O(n^3 + n\alpha(l+2)^{4\alpha}log_2\alpha)$ time [10]. The l-vertex-coloring problem on a weighted graph $G = (V, E)$ can be easily reduced to the ordinary vertex coloring problem on a new non-weighted graph $G_l = (V, E_l)$ such that $(u, v) \in E_l$ for any two vertices u and v in V if and only if dist$(u, v) \leq l$ in G [15]. Therefore, one may expect that the l-vertex-coloring problem for a tree can be solved by applying a linear-time algorithm to solve an ordinary vertex coloring problem for a tree [4]. However, it is not the case because G_l obtained for a tree is not always a tree.

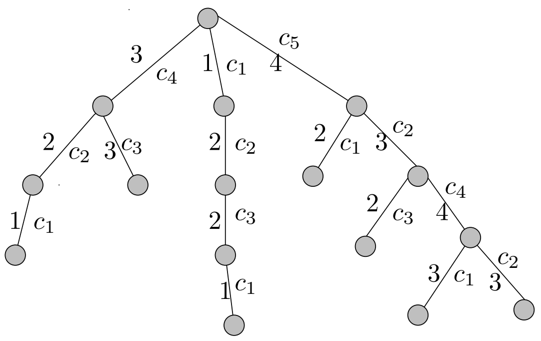

Figure 2: A 3-edge-coloring of a tree with five colors.

An edge version of the l-vertex-coloring problem has been studied for partial k-trees and planar graphs. An ordinary *edge-coloring* of a graph G is to color all edges of G so that any adjacent edges have different colors. For two edges $e = (u, v)$ and $e' = (u', v')$, the *distance between* e *and* e' in G is defined as follows, where $min(w, x, y, z)$ denotes the minimum of w, x, y and z.

$dist(e, e')$ = min{dist(u, u'), dist(u, v'), dist(v, u'), dist(v, v')}.

For a given nonnegative integer l, we wish to color all edges of G so that any two edges e and e' with dist$(e, e') \leq l$ have different colors. Such a coloring is called an *l-edge-coloring* or a *distance-edge-coloring* of G. Thus a 0-edge-coloring is merely an ordinary edge-coloring, and a 1-edge-coloring is a “strong edge-coloring” [11, 13]. The *l-chromatic-index* $\chi_l^{'}(G)$ of a graph G is the minimum number of distinct colors required for an l-edge-coloring of G. The *l-edge-coloring problem* or the *distance-edge-coloring problem* is to compute the l-chromatic-index $\chi_l^{'}(G)$ of a given graph G. For example, Figure 2 depicts a 3-edge-coloring of a tree using five colors c_1, c_2, c_3, c_4, and c_5. One can easily observe that $\chi_3^{'}(G) = 5$ for G in Figure 2.

The edge-coloring problem arises in many applications, including various scheduling and partitioning problems [5]. Since the edge-coloring problem is NP-hard [7], the l-edge-coloring problem is NP-hard in general [8] and hence it is very unlikely that the l-edge-coloring problem can be efficiently solved for general graphs. However, the following results have been known. First, the ordinary edge-coloring problem can be solved in linear time for partial k-trees [16]. Second, the 1-edge-coloring problem can be solved for partial k-trees in $O(n(\alpha+1)^{2^{4(k+1)+1}})$ time [13]. This algorithm takes linear time if k is a bounded integer. Putting $k = 1$ and $k = 2$ in their algorithm, we obtain a 1-edge-coloring algorithm for trees having time complexity of $O(n(\alpha+1)^{2^9})$ and a 1-edge-coloring algorithm for series-parallel graphs having time complexity of $O(n(\alpha+1)^{2^{13}})$ respectively. Third, there is an $O(n(\alpha+1)^{2^{2(k+1)(l+1)+1}})$ time exact algorithm that determines whether a partial k-tree has an l-edge-coloring with a given number of α colors [8]. This algorithm takes linear time if both l, k are bounded integers. Putting $k = 1$ in their algorithm, we obtain an $O(n(\alpha+1)^{2^{4(l+1)+1}})$ time exact algorithm that determines whether a tree has an l-edge-coloring with a given number of α colors. Putting $k = 2$ in their algorithm, we obtain an $O(n(\alpha+1)^{2^{6(l+1)+1}})$ time exact algorithm that determines whether a series-parallel graph has an l-edge-coloring with a given number of α colors. There is a polynomial-time 2-approximation algorithm for the l-edge-coloring problem on planar graphs [8]. The l-edge-coloring problem for a graph G can be reduced to an ordinary vertex coloring problem for a new graph $G^{'}$ obtained from G by some operations. However, $G^{'}$ is not always a partial k-tree or a planar graph even if G is a partial k-tree or a planar graph.

In this paper we give an $O(n^2 + n\Delta^{l+1})$ time algorithm to find an l-vertex-coloring of a tree. The algorithm runs in $O(n^2)$ time if both l and Δ are bounded integers. We compute the upper bound of colors to be $1+\Delta\frac{(\Delta-1)^{\lceil l/2\rceil}-1}{(\Delta-2)}$. We also present an $O(n^2+n\Delta^{l+1})$ time algorithm for solving the l-edge-coloring problem of trees. If both l and Δ are bounded integers, then this algorithm also takes $O(n^2)$ time. Early versions of this paper have been presented at [3] and [2].

The rest of the paper is organized as follows. Section 2 gives some definitions and preliminary ideas. In Section 3, we present an $O(n^2)$ time algorithm for l-vertex-coloring of trees. In Section 4, we present an $O(n^2)$ time algorithm for l-edge-coloring of trees. Finally, Section 5 is a conclusion.

2 Preliminaries

In this section we define several graph theoretical terms used in this paper and prove that the constraint graph for the l-vertex-coloring of a tree is a chordal graph.

Let $G(V, E)$ be a connected simple graph with vertex set $V(G)$ and edge set $E(G)$. We denote by n the number of vertices in G and by m the number of edges in G. Thus $n = |V(G)|, m = |E(G)|$. An edge joining vertices u, v is denoted by (u, v). The *degree* of a vertex v in a graph G, denoted by $d(v)$, is the number of edges incident to v in G. The maximum degree of G is denoted by Δ. We call a graph a *weighted graph* if each of the edges has a positive weight associated with it. Now if N is the set of all positive integers then we can define the weight function for the edges as $w : E \rightarrow N$. A *walk*, $v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l$, in a graph G is an alternating sequence of vertices and edges of G, beginning and ending with a vertex, in which each edge is incident to two vertices immediately preceding and following it. If the vertices $v_0, v_1, \ldots, v_l$ are distinct (except possibly v_0, v_l), then the walk is called a *path* and usually denoted either by the sequence of vertices $v_0, \ldots, v_l$ or by the sequence of edges $e_1, e_2 \ldots, e_l$. The *length* of a path is l which is one less than the number of vertices on the path. A path or walk is *closed* if $v_0 = v_l$. A closed path containing at least one edge is called a *cycle*. A *clique* in a graph is a set of pairwise adjacent vertices. The *clique number* of a graph G, denoted by $\omega(G)$, is the maximum size of a set of pairwise adjacent vertices (clique) in G. By $N_G(v_i)$, we denote the set of vertices adjacent to v_i in G.

A *free tree* is a connected acyclic undirected graph. We often omit the term “free" when we say that a graph is a tree. A *rooted tree* is a free tree in which one of the nodes is distinguished from others. This distinguished node is called the *root* that is drawn generally at the top. If a rooted tree is regarded as a directed graph in which each edge is directed from top to bottom, then every node u other than the root is connected by an edge from some other node p, called the *parent* of u. We also call u a *child* of p. A *leaf* is a node of a tree that has no child. An *internal node* is a node that has one or more children. Every node of a tree is either a leaf or an internal node. Consider a node x in a rooted tree T with root r. Any node y on the unique path from r to x is called an *ancestor* of x. If y is an ancestor of x, then x is a *descendent* of y. The *subtree rooted at* x is the tree induced by descendents of x. The depth of a node x in a tree is the length of the path from the root to x.

Let l be a positive integer, C be a set of colors and the number of colors used by an l-vertex-coloring of a tree T is denoted by $\#\phi$. Then a function $\phi : V \rightarrow C$ is an l-vertex-coloring of T if $\phi(u) \neq \phi(v)$ for any two vertices u and v such that $\text{dist}(u, v) \leq l$. Clearly $\chi_l(T) \leq \#\phi$. We can assume without loss of generality that the consecutive integers $1, 2, \ldots, \#\phi$ are used as the colors. Then C is the color set having colors $1, 2, \ldots, \#\phi$. The *greedy coloring* relative to a vertex ordering $v_1, v_2, \ldots v_n$ of $V(G)$ is obtained by coloring vertices in the order $v_1, v_2, \ldots v_n$ assigning to v_i the smallest indexed color not already used on its lower-indexed neighbors.

A *chord* of a cycle C is an edge, not in C, whose end-points lie in C. A *chordless cycle* in a graph G is a cycle of length at least four in G that has no chord (that is the cycle is an induced subgraph). A graph G is *chordal* if it is simple and has no chordless cycle. A graph G is *perfect* if the chromatic number of every induced subgraph equals the size of the largest clique of that subgraph, i.e. $\chi(H) = \omega(H)$ for every induced subgraph $H \subseteq G$.

A vertex of a graph G is *simplicial* if its neighborhood in G is a clique. A *simplicial elimination ordering* is an ordering $v_n, \ldots, v_1$ for deletion of vertices so that each vertex v_i is a simplicial vertex of the remaining graph induced by $\{v_1, \ldots v_i\}$. These orderings are called *perfect elimination ordering*. We say that the vertex ordering $\delta = (v_1, v_2, \ldots, v_n)$ is a *maximum cardinality ordering* if for every $i \in \{1, 2, \ldots, n-1\}$ and $j \in \{1, \ldots, i\}$, $|N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\}| \geq |N_G(v_j) \cap \{v_{i+1}, \ldots, v_n\}|$. In Figure 3 a, b, c, d, e, f is a maximum cardinality ordering of the vertices of G.

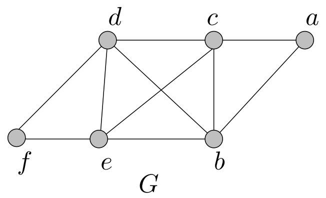

Figure 3: Maximum cardinality ordering.

We call $G_c = (V_c, E_c)$ the constraint graph for the l-vertex-coloring of a tree $T = (V, E)$, if $V_c = V$ and for any two vertices u and v with $\text{dist}(u, v) \leq l$ in T, there is an edge $(u, v) \in E_c$. Let $C =< v_1, v_2, \ldots, v_n, v_1 >$ be a cycle in G_c and $p_m =< v_i, v_{i+1}, \ldots, v_k >$ be a path in C such that the path between v_i and v_k in T also includes the vertices v_{i+1} through v_{k-1} in the same order. Then we call p_m an *original path*. If p_m is not a subpath of any original path in C, we call p_m a *maximal original path* in C. For every original path $p_m =< v_i, v_{i+1}, \ldots, v_k >$ in $C =< v_1, v_2, \ldots, v_n, v_1 >$, we term the path $< v_{k+1}, v_{k+2} \ldots, v_n >$ as the return path p_r. We have the following lemma and theorem.

Lemma 2.1 *The l-vertex-coloring problem for a tree T reduces to the ordinary vertex coloring problem for the constraint graph G_c.*

Theorem 2.2 *The constraint graph G_c for the l-vertex-coloring of a tree T is a chordal graph.*

Proof. Let $C =< v_1, v_2, \ldots v_n, v_1 >$ be a cycle in G_c, p_m be a maximal original path and p_r be its return path in C. Without any loss of generality we can order the vertices in C so that $p_m =< v_1, v_2, \ldots, v_k >$ and $p_r =< v_{k+1}, v_{k+2}, \ldots, v_n >$. In the following figures the vertices of p_m are shown as shaded circles. If we can show that there exists at least one chord in C, we shall be able to conclude that as long as the length of C remains greater than three, a chord will be found and the lengths of the newly formed cycles will be less than the original cycle. Eventually there will be no chordless cycle of length greater than three. According to the definition of maximal original path, p_m has at least two vertices. We have the following cases to consider.

Case 1: p_m has exactly two vertices and p_r has at least two vertices.
In this case $p_m =< v_1, v_2 >$ and $p_r =< v_3, \ldots, v_n >$. There are two possible subcases.

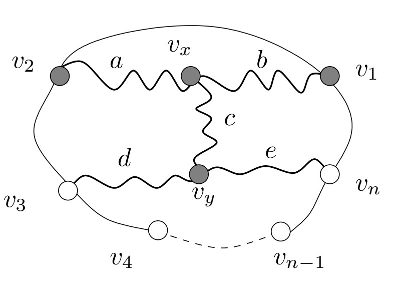

Figure 4: The original path does not have a common subpath

Case 1a: The original path between v_3 and v_n does not have any common subpath with the original path between v_1 and v_2 (see Figure 4). If the original path between v_3 and v_n shares only one vertex in common with the original path between v_1 and v_2, c will be 0 and that will not affect the following proof. Now if C is a chordless cycle, there could be no edges between v_2 and v_n or between v_1 and v_3. So the following two conditions must hold.

$$a + c + e > l \tag{1}$$

$$b + c + d > l \tag{2}$$

But for the presence of edges (v_2, v_3) and (v_1, v_n) in C,

$$a + c + d \leq l \tag{3}$$

$$b + c + e \leq l \tag{4}$$

From equation 1 and 4 we get $a > b$. But from equation 2 and 3 we get $a < b$. So the conditions contradict. So either dist$(v_1, v_3) \leq l$ or dist$(v_2, v_n) \leq l$. Hence there exists at least one chord in C.

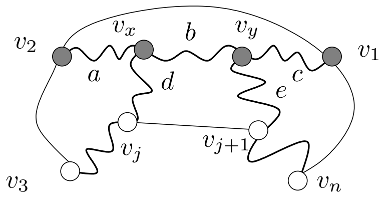

Figure 5: The common subpath lies between the vertices of the same edge

Case 1b: The original path between v_3 and v_n shares a common subpath $< v_x, \ldots, v_y >$ with the original path between v_1 and v_2 (see Figure 5). If any vertex in the return path p_r lies in the common subpath, then it eventually falls in the actual path between v_1 and v_2 resulting the existence of two chords between that vertex and v_1 and v_2. So let no vertex in p_r lies in the original path between v_1 and v_2. Now, as the original path between v_3 and v_n has some part lying in p_m, and no vertices of p_r are allowed to be lying on the common subpath either, there exists at least one edge (v_j, v_{j+1}) in p_r such that the original path between v_j and v_{j+1} includes the common subpath.

To be chordless, C cannot have any edge either between v_2 and v_{j+1} or between v_1 and v_j. So at least the following two conditions must hold.

$$a + b + e > l \tag{5}$$

$$b + c + d > l \tag{6}$$

But for the presence of edges (v_1, v_2) and (v_j, v_{j+1}) in C,

$$a + b + c \leq l \tag{7}$$

$$b + d + e \leq l \tag{8}$$

From equation 5 and 7 we get $e > c$. But from equation 6 and 8 we get $c > e$. So the conditions contradict. So either dist$(v_1, v_j) \leq l$ or dist$(v_2, v_{j+1}) \leq l$. Hence there exists at least one chord in C.

Case 2: p_m has three or more vertices and p_r has one or more vertex.
In this case $p_m =< v_1, v_2, \ldots, v_k >$,$p_r =< v_{k+1}, \ldots,$ $v_n >$ and $k > 2$. The path p_m, from v_1 to v_k in G_c indicates that the path between v_1 and v_k in T must also go through every vertex in p_m. So to make a cycle, the path $< v_{k+1}, v_{k+2}, \ldots, v_n, v_1 >$ in G_c must take a different path in the original tree other than the shortest path between v_1 and v_k as covered by p_m. Now, for a tree, there exists exactly one simple path between each pair of vertices. So p_r must have gone along a complex path and this path must visit all the vertices lying in the original path between v_1 and v_k. So there exists at least one edge e_j between v_j and $v_{j+1}, j \in \{k + 1, \ldots, n\}$ in p_r for every $v_i, i \in \{2, \ldots, k - 1\}$ in p_m such that v_i lies in the original path between v_j and v_{j+1}. This along with the presence of e_j implies that the distance between v_j or v_{j+1} and v_i is not more than l. Hence there must be a chord in C.

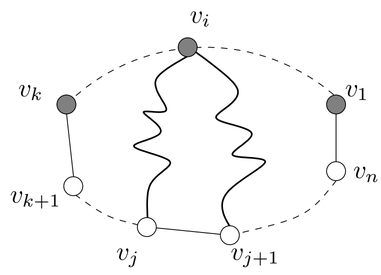

Figure 6: The vertices of p_m fall in original paths between vertices of p_r

Case 3: p_m has four or more vertices and p_r has no vertex.
In this case, $p_m =< v_1, v_2, \ldots, v_k >$,$p_r = \phi$ and $k > 3$. This is the simplest of all cases because the edge (v_k, v_1) in G_c implies dist$(v_k, v_1) \leq l$. But from the definition of maximal original path, all the vertices of p_m must lie in the original path between v_1 and v_k. So dist$(v_k, v_1) \leq l$ implies dist$(v_k, v_i) \leq l$ for all $i \in \{2, \ldots, k - 1\}$. Each of these inequalities induces a chord in C. $\mathcal{Q.E.D.}$

In Section 3, we provide an $O(n^2)$ time algorithm for solving the l-vertex-coloring problem on trees.

3 l-vertex-coloring

In this section we present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the l-vertex-coloring problem on trees.

Given a tree T, we first transform T into its constraint graph G_c, then find maximum cardinality ordering of the vertices of G_c and perform greedy coloring in the reverse of the obtained perfect elimination ordering to obtain the required l-vertex-coloring. The following algorithm Generate_Constraint_Graph generates the constraint graph for the l-vertex-coloring of a tree.

Algorithm 1: Generate_Constraint_Graph

```
Input  : A weighted tree $T = (V, E)$ and $l$.
Output: A constraint graph $G_c = (V_c, E_c)$
        for the $l$-vertex-coloring of a tree $T$.

begin
    $G_c \leftarrow T$
    for each vertex $u \in V$ do
        BFS-Traverse $(T, u, l)$
end

Procedure BFS-Traverse $(G, s, l)$

begin
    for each vertex $u \in V[G]$ do
        $color[u] \leftarrow$ white
        $d[u] \leftarrow \infty$
    $d[s] \leftarrow 0$
    $Q \leftarrow \{s\}$
    while $Q \neq \phi$ do
        $u \leftarrow head[Q]$
        for each vertex $v \in Adj[u]$ do
            if $color[v] = white$ then
                $d[v] \leftarrow d[u] + w(u, v)$
                if $d[v] \leq l$ then
                    Enqueue$(Q, v)$
                    if $v \notin Adj[s]$ then
                        $E_c \leftarrow E_c \cup \{(s, v)\}$
                else $color[v] \leftarrow$ black
        Dequeue$(Q)$
        $color[u] \leftarrow$ black
end
```

We have the following lemmas and theorem.

Lemma 3.1 *Let $\chi_l(T)$ be the l-chromatic-index of a tree T with maximum degree Δ. Then*
$\chi_l(T) \leq 1 + \Delta\frac{(\Delta-1)^{\lceil l/2 \rceil}-1}{(\Delta-2)}$.

Proof. Since the l-vertex-coloring problem for a tree T reduces to the ordinary vertex coloring for the chordal graph G_c, $\chi(G_c)$ will be the same as $\chi_l(T)$. Now a chordal graph is a perfect graph and for a perfect graph $G, \omega(G) = \chi(G)$. So to determine the upper bound of $\chi_l(T)$, we have to know the size of the maximum clique in G_c. We need to know how many vertices could be at maximum such that any two of them remain at max l distance apart in T. We assume the weights of each edges as minimum which is one.

We can add vertices to the set constituting the clique until we could add no more vertices keeping it within l distance from all the members of the clique. Now if we start with the vertex v_r and add vertices accordingly, we can add vertices to the set till all the leaves of the sub-tree rooted at v_r are within l distance from v_r. The depth of the sub-tree could be no more than $\lceil l/2 \rceil$ as any two leaves must be within l distance from each other and the distance between them is twice the depth of the sub-tree. The number of vertices could be found by adding up the following series, where the i-th term denotes the number of vertices in $(i + 1)$-th level.

$$\chi_l(T)$$
$$= \omega(G_c)$$
$$= 1+\Delta+\Delta(\Delta-1)+\Delta(\Delta-1)^2+\ldots+\Delta(\Delta-1)^{\lceil l/2 \rceil-1}$$
$$= 1 + \Delta\frac{(\Delta-1)^{\lceil l/2 \rceil}-1}{(\Delta-2)}$$

$\mathcal{Q.E.D.}$

Lemma 3.2 *The constraint graph $G_c = (V_c, E_c)$ has $O(n\Delta^l)$ edges.*

Proof. We have to add edges between all the vertices that are at most l distance apart in T. For each vertex the number of vertices within l distance is found by the exact analogy used in lemma 3.1. The same edge is considered twice if the edges for all the vertices are taken separately. Hence
$|E_c| = \frac{n}{2}(\Delta+\Delta^2+\ldots+\Delta^l) = \frac{n\Delta(\Delta^l-1)}{2(\Delta-1)} = O(n\Delta^l)$.
$\mathcal{Q.E.D.}$

Lemma 3.3 *The constraint graph G_c can be generated in $O(n^2 + n\Delta^{l+1})$ time.*

Proof. Algorithm Generate_Constraint_Graph exhaustively adds edges to each pair of vertices u and v with dist$(u, v) \leq l$ in T. The BFS-Traverse part of the algorithm traverses the subtree rooted at any vertex u, up to l level deep at maximum when each edge has a minimum weight of one. The calculation for number of vertices covered in each such subtree is similar to the number of vertices obtained in lemma 3.1 with only going l level deep rather than halting at level $\frac{l}{2}$.

So maximum number of vertices to traverse for BFS-Traverse is $1 + \Delta\frac{(\Delta-1)^l-1}{(\Delta-2)} = O(\Delta^l)$. From lemma 3.2 the maximum number of edges for each vertex in G_c is $O(\Delta^l)$. Since the adjacency list of each vertex is scanned at most once, at most $O(\Delta.\Delta^l) = O(\Delta^{l+1})$ time is spent in total scanning adjacency lists. The overhead for initialization is $O(n)$, and thus the total running time of BFS-Traverse $O(n) + O(\Delta^l) + O(\Delta^{l+1}) = O(n + \Delta^{l+1})$. Hence the run time of Generate_Constraint_Graph would be $O(n^2 + n\Delta^{l+1})$.

$\mathcal{Q.E.D.}$

The following algorithm MCS finds maximum cardinality ordering of the vertices of a chordal graph.

Algorithm 2: MCS

```
Input  : A chordal graph $G_c = (V_c, E_c)$.
Output: A maximum cardinality ordering of $V_c$.
begin
    $currentOrder \leftarrow |V_c|$
    $currentLabel \leftarrow 1$
    for each vertex $v \in V_c$ do
        $label[v] \leftarrow 0$
    for $i = 1$ to $|V_c|$ do
        pick an unordered vertex $u$ with the highest label
        $order[u] \leftarrow currentOrder$
        for each vertex $w \in Adj[u]$ do
            $label[w] \leftarrow currentLabel$
            $E_c \leftarrow E_c - \{(u, w)\}$
        $V_c \leftarrow V_c - \{u\}$
        $currentOrder \leftarrow currentOrder - 1$
        $currentLabel \leftarrow currentLabel + 1$
end
```

We have the following lemmas and theorem.

Lemma 3.4 *Algorithm MCS can be implemented in time $O(n\Delta^l)$.*

Proof. To implement MCS in linear time, we define S_i to be the set of vertices with label i. Then we present every set by a list. Also for each vertex we store its label i and pointer to its position in S_i. When a vertex v receives a number, we remove it from the corresponding list and move each of its neighbors one list upwards. This takes $O(1 + Adj[v])$ time. So MCS can be implemented in $O(|V_c| + |E_c|) = O(n\Delta^l)$ time. $\mathcal{Q.E.D.}$

Lemma 3.5 *A graph is chordal if and only if every ordering obtained by Algorithm MCS is a perfect elimination ordering* [12].

Lemma 3.6 *If greedy coloring is applied in the reverse perfect elimination ordering of a chordal graph, the coloring will be optimal.*

Proof. We prove it by induction on the number of vertices colored.

Base case: v_n can arbitrarily be given color 1.

Induction step: Assume that $v_n, v_{n-1}, \ldots, v_{i+1}$ have been colored optimally. We want to show that choosing the color for v_i in a greedy manner will yield an optimal coloring for $v_n, v_{n-1}, \ldots, v_{i+1}, v_i$. We choose an arbitrary color for v_i that has not been used by any of its neighbors. If any color that has already been used is available, we use that for v_i keeping the number of colors as before. Otherwise, if all the already colored neighbors of v_i form a clique, there is no way they (including v_i) could be colored by any fewer colors than the size of this clique. If there is no other color than the colors used by the neighbors, we have no choice but to use a new color for v_i and this coloring is optimal. Since the coloring was optimal for $v_n, v_{n-1}, \ldots, v_{i+1}$, it must also be optimal for the subgraph $v_n, v_{n-1}, \ldots, v_{i+1}, v_i$ and thus we have a new optimal sub-solution. This completes the inductive step. $\mathcal{Q.E.D.}$

Theorem 3.7 *The l-vertex-coloring of a tree T can be solved in time $O(n^2 + n\Delta^{l+1})$, where l is a positive integer and Δ is the maximum degree of T. If both l and Δ are bounded integers, then an l-vertex-coloring of T can be found in $O(n^2)$ time.*

Proof. At first the input tree T is transformed into constraint graph G_c. Then we run the MCS algorithm on G_c which is a chordal graph and greedy coloring is applied in the reverse of the obtained perfect elimination ordering to get the required l-vertex-coloring. The simple greedy-coloring algorithm can be implemented in $O(n\Delta^l)$ time. Thus from lemma 3.3 and 3.4, the total running time is $O(n^2 + n\Delta^{l+1}) + O(n\Delta^l) + O(n\Delta^l) = O(n^2 + n\Delta^{l+1})$. Hence if both l and Δ are bounded integers, then our algorithm runs in $O(n^2)$ time.

$\mathcal{Q.E.D.}$

Now we illustrate an example for solving the l-vertex-coloring problem on a weighted tree for a given l.

We first transform T into its constraint graph G_c. Figure 7(b) shows the constraint graph G_c for the l-vertex-coloring of the input weighted tree in Figure 7(a) with $l = 3$. Then using Algorithm MCS, we find the maximum cardinality ordering of the vertices of G_c. Figure 8(a) shows the maximum cardinality ordering

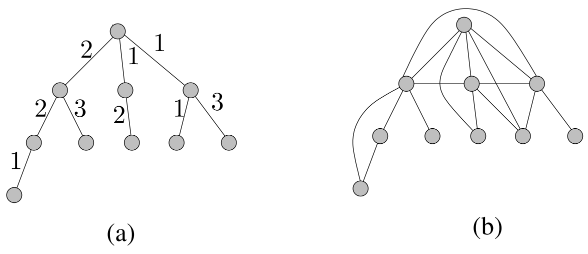

Figure 7: (a) Input weighted tree T with $l = 3$ and (b) its constraint graph G_c

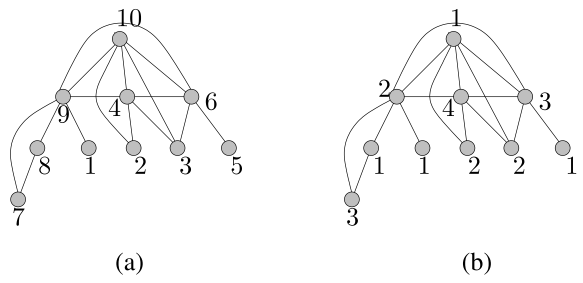

Figure 8: (a) Maximum cardinality ordering and (b) greedy coloring of the vertices of G_c

of the vertices of G_c, where a number next to a vertex is its order. Then we perform greedy coloring in the reverse of the obtained perfect elimination ordering to obtain the required l-vertex-coloring. Figure 8(b) shows the required l-vertex-coloring using four colors, where a number next to a vertex is its color.

In Section 4, we provide an $O(n^2)$ time algorithm for solving l-edge-coloring problem on trees.

4 l-edge-coloring

In this section we present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the l-edge-coloring problem on trees.

The constraint graph $G_c^{'} = (V_c^{'}, E_c^{'})$ for the l-edge-coloring of a tree $T = (V, E)$ is the graph whose vertices are the edges of T, with $(e, f) \in E_c^{'}$ when $e = (u, v), f = (u^{'}, v^{'})$ and dist$(e, f) \leq l$ in T. We have the following lemmas and theorem.

Lemma 4.1 *The l-edge-coloring problem for a tree T reduces to the ordinary vertex coloring problem for the constraint graph $G_c^{'}$.*

Theorem 4.2 *The constraint graph $G_c^{'}$ for the l-edge-coloring of a tree T is a chordal graph.*

Proof. The proof is similar to that of Theorem 2.2.

Lemma 4.3 *The constraint graph $G_c^{'}$ can be generated in $O(n^2 + n\Delta^{l+1})$ time.*

Lemma 4.4 *The constraint graph $G_c^{'}$ has $O(n\Delta^l)$ edges.*

So we have transformed the l-edge-coloring of tree into ordinary vertex coloring of the constraint graph $G_c^{'}$ which is chordal. In Section 3, we have already shown the procedure for coloring a chordal graph in $O(n\Delta^l)$ time. From Lemma 4.4, the number of edges in the chordal graph created in this case is also $O(n\Delta^l)$. Hence the algorithm for solving the l-edge-coloring of tree runs in time $O(n^2 + n\Delta^{l+1})$. Thus we have the following theorem.

Theorem 4.5 *The l-edge-coloring of a tree T can be solved in time $O(n^2 + n\Delta^{l+1})$, where l is a positive integer and Δ is the maximum degree of T. If both l and Δ are bounded integers, then an l-edge-coloring of T can be found in $O(n^2)$ time.*

Now we illustrate an example for solving the l-edge-coloring problem on a weighted tree for a given l.

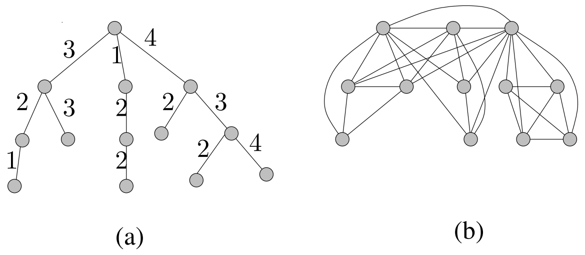

Figure 9: (a) Input weighted tree T with $l = 3$ and (b) its constraint graph $G_c^{'}$

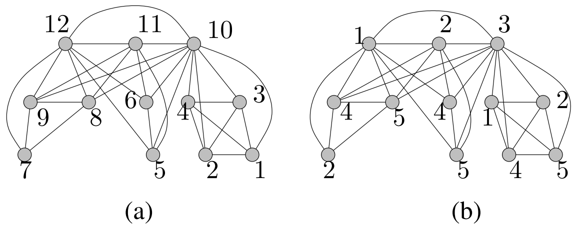

Figure 10: (a) Maximum cardinality ordering and (b) Greedy coloring of the vertices of $G_c^{'}$

We first transform T into its constraint graph $G_c^{'}$. Figure 9(b) shows the constraint graph $G_c^{'}$ for the l-vedge-coloring of the input weighted tree T in Figure 9(a) with $l = 3$. Then using Algorithm MCS, we find

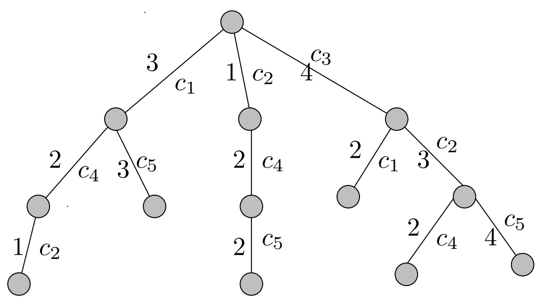

Figure 11: l-edge-coloring of T

the maximum cardinality ordering of the vertices of $G_c^{'}$. Figure 10(a) shows the maximum cardinality ordering of the vertices of $G_c^{'}$, where a number next to a vertex is its order. Then we perform greedy coloring in the reverse of the obtained perfect elimination ordering in Figure 10(a). Figure 10(b) shows the greedy coloring, where a number next to a vertex is its color. Hence Figure 11 shows the required l-edge-coloring using five colors c_1, c_2, c_3, c_4, and c_5.

5 Conclusion

In this paper we present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the l-vertex-coloring problem on trees. If both l and Δ are bounded integers, then our algorithm takes $O(n^2)$ time. We compute the upper bound of colors to be $1 + \Delta\frac{(\Delta-1)^{\lceil l/2 \rceil}-1}{(\Delta-2)}$. We also present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the l-edge-coloring problem on trees. If both l and Δ are bounded integers, then this algorithm also takes $O(n^2)$ time.

References

[1] Agnarsson, G. and Halldórsson, M. Coloring powers of planar graphs. *SIAM J. Discrete Math.*, 16:651–662, 2003.

[2] Awal, T. and Mahbubuzzaman, M. *Coloring of trees*. Vdm Verlag, 2011.

[3] Awal, T., Mahbubuzzaman, M., and Kashem, M. A. Algorithm for l-vertex-coloring of trees. In *Proc. of 6th International Conference on Electrical and Computer Engineering (ICECE)*, pages 534–537, 2010.

[4] Borie, R., Parker, R., and Tovey, C. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.

[5] Fiorini, S. and Wilson, R. J. *Edge-Colourings of Graphs*. Pitman, London, 1977.

[6] Garey, M. and Johnson, D. *Computers and Intractability: A Guide to the theory of NP-completeness*. W.H Freeman and Co., 1979.

[7] Holyer, I. The NP-completeness of edge-colouring. *SIAM J. Comput.*, 10:718–720, 1981.

[8] Ito, T., Kato, A., Zhou, X., and Nishizeki, T. Algorithms for finding distance-edge-colorings of graphs. *Journal of Discrete Algorithms*, 5:304–322, 2007.

[9] Kashem, M. A. and Yasmeen, A. Optimal l-vertex-coloring of trees. In *Proc. of International Conference on Computer and Information Technology (ICCIT)*, pages 151–153, 2000.

[10] Kashem, M. A. and Yasmeen, A. Optimal l-vertex-colorings of series-parallel graphs. In *Proc. of International Conference on Computer and Information Technology (ICCIT)*, pages 103–107, 2001.

[11] Mahdian, M. On the computational complexity of strong edge coloring. *Discrete Appl. Math.*, 118:239–248, 2002.

[12] Rose, D., Lueker, G., and Tarjan, R. Algorithmic aspects of vertex elimination of graph. *SIAM J. Computing*, 5:266–283, 1976.

[13] Salavatipour, M. A polynomial time algorithm for strong edge coloring of partial k-trees. *Discrete Appl. Math.*, 143:285–291, 2004.

[14] West, D. *Introduction to Graph Theory*. Prentice-Hall Inc., 1999.

[15] Zhou, X., Kanari, Y., and Nishizeki, T. Generalized vertex-colorings of partial k-trees. *IEICE Trans. on Fundamentals*, pages 555–581, 2000.

[16] Zhou, X., Nakano, S., and Nishizeki, T. Edge-coloring partial k-trees. *J. Algorithms*, 21:598–617, 1996.