A linear time algorithm for edge coloring of binomial trees

M. Kubale*, K. Piwakowski

Technical University of Gdańsk, Foundations of Informatics Department, Narutowicza 11/12,
80-952 Gdańsk, Poland

Received 29 September 1993; revised 15 July 1994

Abstract

We consider the problem of efficient coloring of the edges of a so-called binomial tree \( T \), i.e. an acyclic graph containing two kinds of edges: those which must have a single color and those which are to be colored with \( L \) consecutive colors, where \( L \) is an arbitrary integer greater than 1. We give an \( O(n) \) time algorithm for optimal coloring of such a tree, where \( n \) is the number of vertices of \( T \). Also, we give simple bounds on the chromatic index of \( T \) and a division of all binomial trees into two classes depending on their chromaticity.

1. Introduction

The interval edge-coloring of a weighted graph is the problem of coloring the edges of an edge-weighted graph in such a way that each edge must get an interval of integers of size of its weight and no vertex has two incident edges containing the same integer. The general problem for minimizing the largest number used in such a coloring is known to be NP-hard, as it is already NP-complete to determine the chromatic index of a simple graph [3]. However, unlike the usual problem, the interval edge coloring remains NP-hard even for some restrictive families of graphs for which polynomial-time algorithms for the classical problem are known, e.g. bipartite graphs [2], generalized caterpillars [5], and trees [6]. Nonetheless, in view of potential applications in scheduling and timetabling, it would be useful to have efficient algorithms for polynomially solvable subproblems, and such a subproblem concerning trees is considered in this article.

The interval edge-coloring problem has many applications in scheduling 2-processor tasks on dedicated processors, e.g. scheduling file transfers in a distributed...
network [1] or scheduling diagnostic tests in a multicomputer system [4], as well as scheduling tasks in an open shop system, e.g. constructing class-teacher timetables [7]. In both cases, if the system is duoprocessor, i.e. it consists of two kinds of processors (slow and fast) and overwhelming majority of tasks are zero-time jobs, then it is plausible that the underlying graph is acyclic, i.e. a tree or forest, with two kinds of weights on the edges.

It is well known that deciding interval edge-colorability of a weighted tree $T$ is NP-complete [1] and remains so even if $T$ is a caterpillar with a hair of length two [5]. Herein we consider a simplified problem of coloring the edges of what we call binomial trees, i.e. connected acyclic graphs containing two kinds of edges: the edges of weight 1 and those of weight $L$, where $L$ is an arbitrary integer greater than 1.

The remainder of this paper is organized as follows. In Section 2 we introduce some basic definitions and properties concerning binomial trees. Section 3 is devoted to a lower bound on the minimum number of colors required for such a tree $T$. The main results of this paper are given in Section 4, where we give an $O(n)$ time algorithm for optimal coloring of a binomial tree of $n$ vertices. We also give an upper bound on the chromatic index of $T$ which is at worst one higher than the lower bound. Finally, we classify all binomial trees into two classes depending on their chromaticity.

2. Basic definitions and properties

Let $T = (V, E_1 \cup E_L)$ be a binomial tree. The vertex set of $T$ contains $|V| = n$ vertices. The edge set of $T$ contains $n - 1$ edges and consists of two kinds of elements: $E_1$ — the edges of weight 1, called 1-edges, $E_L$ — the edges of weight $L$, where $L \in \mathbb{N} - \{1\}$, called $L$-edges. Our aim is to find an optimal coloring of $T$ in which each 1-edge gets one color, each $L$-edge gets $L$ consecutive colors, and no two colors of adjacent edges are the same. The minimum possible number of colors used in such a coloring will be called the (binomial) chromatic index of $T$ and denoted by $\chi(T)$.

For a given binomial tree $T = (V, E_1 \cup E_L)$ the symbols: $\rho(v)$, $\rho_1 (v)$, $\rho_L(v)$ denote the degree of vertex $v$ in graph $T$, in graph $(V, E_1)$, and in graph $(V, E_L)$, respectively. The value $\delta(v) = \rho_1 (v) + L \rho_L(v)$ is called the weighted degree of $v$. $D$ stands for the maximum degree $\rho_L$ and $A$ for the maximum weighted degree $\delta$. In addition, symbol $\rho_{1D}(v)$ denotes the number of vertices with degree $\rho_L = D$, which are adjacent to $v$ by 1-edges.

The crucial point of the method is to distinguish the colors that can be safely used for 1-edges incident with vertices having $\rho_L = D$. To this end we define the notion of good colors. Given an integer $k \in \mathbb{N}$ and a tree $T$, a color $c \leq k$ is said to be good for $T$ with respect to $k$, if one can choose $D$ pairwise disjoint intervals of length $L$ among $\{1, 2, \ldots, c - 1, c + 1, \ldots, k - 1, k\}$. Otherwise, $c$ is called a bad color for $T$ with respect to $k$. For simplicity, by $\text{GOOD}(k)$ we mean the set of good colors with respect to $k$ and by $\text{BAD}(k)$, the set of bad colors.

Let us notice some important properties of good colors.
Proposition 2.1. For any binomial tree $T$ the number of good colors with respect to $k$ is an increasing function of $k$ for $k \geq DL$.

Proposition 2.2. For any binomial tree $T$ and integer $k$ every interval of length $L$ contained in $\{1, 2, \ldots, k\}$ contains a fixed number of good colors.

Let us denote the number of good colors mentioned in Proposition 2.2 by $\xi(T, k)$, or shortly $\xi(k)$.

3. A lower bound on the chromatic index

One trivial lower bound on the number of colors needed is the maximum weighted degree $\Delta$. Herein we give another bound.

If a vertex has $D$ incident $L$-edges, each of the incident 1-edges must get a good color. Call these edges critical. The number of critical edges incident with arbitrary vertex $v$ is not less than $\rho_{1D}(v)$ (may be greater if $\rho_L(v) = D$). Thus, on the basis of Proposition 2.2, one can say that vertex $v$ requires at least $\xi(k)\rho_L(v) + \rho_{1D}(v)$ good colors. Let us define $\gamma(v)$ as the minimum value of $k$ such that $|GOOD(k)|$ satisfies this requirement, and let $\Gamma$ be the maximum value of $\gamma(v)$ among all vertices. Hence we have the following.

Theorem 3.1. For any binomial tree $T$ $\max\{\Gamma, \Delta\} \leq \chi(T)$.

In Section 5 we shall show how this lower bound can be found in linear time. From here on we denote $\Omega = \max\{\Gamma, \Delta\}$.

4. Algorithm

In the following we assume that the tree is rooted with root $v_1$.

4.1. A 1-absolute approximation algorithm

We start by showing that the lower bound of Theorem 3.1 is at worst one less than the chromatic index.

Assume that $k > \Gamma$ and $k \geq \Delta$. In this case we color the edges top-down, from the root to the leaves. We use a recursive procedure, which colors all uncolored edges incident to a vertex given as a parameter and then calls itself with each child vertex as a parameter. Program starts with calling this procedure with $v_1$. During one pass of the procedure at first all $L$-edges are colored, then the remaining 1-edges. Each $L$-edge receives the first available interval of type $[c + 1, \ldots, c + L]$ such that
0 \leq c \mod L \leq k - DL. The 1-edges are colored subject to the restriction that critical edges receive only good colors. Below a control abstraction of the described algorithm is presented.

**Procedure** Color1(v: vertex, k: integer);

begin
for every uncolored L-edge incident with v assign the first available interval of type 
\[ [c + 1, \ldots, c + L] \text{ such that } 0 \leq c \mod L \leq k - DL; \]
for every uncolored critical 1-edge incident with v assign the first available good

color with respect to k;

for all uncolored remaining 1-edges incident with v assign the first available color;

for every vertex w which is a child of v call Color1(w, k);

end;

For a given k the program starts with calling Color1(v1, k).

**Theorem 4.1.** If \( k > \Gamma \) and \( k \geq \Delta \) then algorithm Color1 realizes coloring successfully.

**Proof** (sketch). During each pass of procedure Color1, on entry to the algorithm in the moment when L-edges have to be colored, at most one adjacent edge has already received a color. Thus, by the assumptions \( k > \Gamma \), \( k \geq \Delta \) and by the definition of \( \Gamma \) and \( \Delta \) (also \( \gamma, \delta \)), for an arbitrary chosen pass of procedure Color1, assuming inductively that all previous passes finished successfully (that is, all actions could be done, e.g. there were enough good colors; no one interval or color exceeds k), we prove successful termination of the chosen pass. A precise proof is long and tedious and has rather technical character. Many cases should be considered, e.g. vertex v satisfies \( \rho_L(v) = D \) or not, colored adjacent edge (from the father) is 1-edge or L-edge, etc.

Theorem 4.1 together with Theorem 3.1 imply immediately the following.

**Theorem 4.2.** For any \( T \max \{ \Gamma, \Delta \} \leq \chi(T) \leq \max \{ \Gamma + 1, \Delta \}. \)

Possibly the reader wonders whether the assumption \( k > \Gamma \) in Theorem 4.1 could be replaced by \( k \geq \Gamma \). In that case \( \Omega \) would be equal to the chromatic index \( \chi(T) \). In the following example we show that this is not the case.

**Example 4.1.** Fig. 1 depicts this example.

Fig. 1 explains why the condition \( k \geq \Gamma \) is not sufficient. It is possible that one of the good colors will be taken for edge \((v, father(v))\) which is not critical and, consequently, one of the critical edges will not be able to get a good color. In the last section we present a simple example of binomial tree \( T \) for which \( \chi(T) > \Omega \).
4.2. Deciding the chromaticity

Now we present an algorithm to determine if $\chi(T) = \Omega$. It assigns one of the following three possible labels to each 1-edge. Label \textit{Good} (\textit{Bad}) is assigned in the case when one can be sure that in every legal coloring of $T$ (for $k = \Omega$) a given edge must get good (bad) color. Otherwise, the edge receives a label \textit{Ind}. We label only 1-edges, so that graph $(V, E_1)$ may be disconnected. In that case we apply this algorithm to each tree in the forest $(V, E_1)$ separately. The final answer as to whether $T$ can be colored using $\Omega$ colors or not, is the conjunction of answers given by the algorithm when applied to all such subtrees.

The tree is traversed bottom-up from the leaves to the root. Each 1-edge $(father(v), v)$ receives a label after labeling all other 1-edges incident with $v$. Let us define:

$$MaxGood(v) = |GOOD(\Omega)| - \xi(\Omega)\rho_L(v),$$

$$MaxBad(v) = |BAD(\Omega)| - (L - \xi(\Omega))\rho_L(v).$$

By Proposition 2.2, for $k = \Omega$, no more than $MaxGood(v)$ ($MaxBad(v)$) edges incident with $v$ can get good (bad) colors. During the process of labeling when the next 1-edge $(father(v), v)$ has to get a label, at first the number of already labeled \textit{Good} and \textit{Bad} 1-edges incident with $v$ is determined. If the number exceeds, respectively, $MaxGood(v)$ or $MaxBad(v)$, then the process is terminated with the following negative answer: 'tree cannot be colored using $\Omega$ colors'. The following specification explains in detail all conditions determining: which label has to be assigned and when the algorithm must stop with a negative answer (return \textit{false}).

Assume that at any state of the program the functions $SumGood(v)$ and $SumBad(v)$ express the number of vertices incident with $v$ which have already been labeled with \textit{Good} and \textit{Bad}, respectively. For any vertex $v$ after all edges joining $v$ with its child vertices have been labeled, fulfilling either of the following two conditions implies program termination and returning \textit{false} (the second one has no sense if $v$ is the root):

$$SumGood(v) > MaxGood(v) \quad \text{or} \quad SumBad(v) > MaxBad(v), \quad (1)$$

$$\text{Edge } (father(v), v) \text{ is critical} \quad \text{and} \quad SumGood(v) = MaxGood(v). \quad (2)$$
The first one says: 'limit of good or bad colors has been exceeded', while the second: 'edge \((father(v), v)\) must get a good color but all of them are used'. After this check, if program has not been terminated and \(v\) is not the root, one assigns a proper label to edge \((father(v), v)\). The following conditions determine this label:

**Label Good:**

- Edge is critical or
- \(\text{SumBad}(v) = \text{MaxBad}(v)\) (no more bad colors are available).

**Label Bad:**

- \(\text{SumGood}(v) = \text{MaxGood}(v)\) (no more good colors are available).

Otherwise, label \(\text{Any}\) is assigned.

If \(v\) is the root, the program terminates with the positive answer (return \(true\)).

One may wonder which label will be assigned if \(\text{SumGood}(v) = \text{MaxGood}(v)\) and \(\text{SumBad}(v) = \text{MaxBad}(v)\). This situation cannot occur because

\[
\text{MaxGood}(v) + \text{MaxBad}(v) = \Omega - \rho_L(v)L \geq \rho_1(v) > \text{SumGood}(v) + \text{SumBad}(v).
\]

The last inequality follows from the fact that the edge \((father(v), v)\) has not been labeled yet.

The idea of using recursion is similar to that used in procedure \(\text{Color1}\), with the only exception that scanning is carried out in the reverse order. Below we show a pseudo-code abstraction of the labeling algorithm.

**procedure AssignLabels**(*v*: vertex);

begin

- for every vertex \(w\) which is a child of \(v\) call \(\text{AssignLabels}(w)\);
- if \(v\) is the root
  - then if condition (1) is satisfied then terminate program and return \(false\)
  - else terminate program and return \(true\)
- else if condition (1) or (2) is satisfied then terminate program and return \(false\)
  - else do case
    - (3) or (4): assign label \(\text{Good}\) to edge \((father(v), v)\);
    - (5) : assign label \(\text{Bad}\) to edge \((father(v), v)\);
    - else : assign label \(\text{Any}\) to edge \((father(v), v)\);
  - endcase;

end;

Program starts with calling \(\text{AssignLabels}(v_1)\).
Now we are ready to prove the following.

**Theorem 4.3.** \( \chi(T) = \Omega \) iff program AssignLabels returns true.

**Proof.** \( (\Rightarrow) \) By conditions (1) and (2), and by the fact that labels \( \text{good} (\text{bad}) \) are assigned only if a given edge must receive good (bad) color (cf. conditions (3)–(5)), one can see that this program cannot return \text{false} if the legal coloring is possible.

\( (\Leftarrow) \) By Theorem 3.1 it is enough to give a method of coloring \( T \) with \( \Omega \) colors in the case where program AssignLabels returns \text{true}. Procedure Color2 described below realizes this task.

### 4.3. Coloring \( \Omega \)-chromatic graphs

To color \( T \) with \( k = \Omega \) colors we use a new procedure Color2 which is a slight modification of procedure Color1. The only difference is that the new one assigns colors to 1-edges with respect to labeling which is obtained from procedure AssignLabels. Its control abstraction is as follows:

```bash
procedure Color2(v: vertex);
begin
  for every uncolored L-edge incident with v assign the first available interval of type 
  \[ [c + 1, \ldots, c + L] \text{ such that } 0 \leq c \mod L \leq \Omega - DL; \]
  for every uncolored 1-edge with label \( \text{good} (\text{bad}) \) incident to v assign the first 
  available good (bad) color with respect to \( \Omega \); 
  for all uncolored remaining 1-edges (with label \( \text{any} \)) incident with v assign the first 
  available color; 
  for every vertex w which is a child of v call Color2(w);
end;
```

The program is initialized by invoking Color2\((v_1)\).

**Theorem 4.4.** If procedure AssignLabels returns true, then procedure Color2, using obtained labeling, gives a legal \( \Omega \)-coloring of \( T \).

**Proof (sketch).** The argument is similar to that used in the proof of Theorem 4.1. Almost all attention is focused on the assurance that at any moment of assigning a color to 1-edge with label \( \text{good} (\text{bad}) \) there will be at least one good (bad) color. Similar to the proof of Theorem 4.1, a number of cases must be considered based on the property of labeling which guarantees that for each vertex \( v \) there are no more than \( \text{MaxGood}(v) (\text{MaxBad}(v)) \) 1-edges incident to \( v \) with label \( \text{good} (\text{bad}) \). In addition, if any vertex has exactly \( \text{MaxGood}(v) (\text{MaxBad}(v)) \) 1-edges labeled \( \text{good} (\text{bad}) \) then all other incident 1-edges have label \( \text{bad} (\text{good}) \). \( \square \)

Let us come back to Example 4.1 and notice that the property of labeling mentioned above guarantees that the edge \((\text{father}(v), v)\) gets label \( \text{bad} \) (clearly, if
procedure AssignLabels returns true) and a good color needed just after that will not be wasted.

4.4. Optimal coloring

Theorems 4.1–4.4 lead to the final algorithm for optimal coloring of binomial trees. Algorithm OptimalColor makes two passes through $T$. First, try to label the edges of $T$ for $k = \max\{\Gamma, \Delta\}$. If the labeling succeeds, then color the edges using procedure Color2. Otherwise, color the edges for $k = \max\{\Gamma + 1, \Delta\} = \Gamma + 1$ using procedure Color1. Below a pseudo-code of the method is presented.

program OptimalColor($T$);
begin
  calculate $\Gamma, \Delta$;
  call AssignLabels;
  if true is returned then call Color2($v_0$) else call Color1($v_1, \Gamma + 1$);
end;

5. Complexity considerations

It is easy to see that during each course of procedures: AssignLabels, Color1 and Color2, every edge is assigned a label or a color (set of colors). Since there are $n - 1$ edges, the total number of these assignments is $O(n)$. Each of these operations is preceded by a number of conditions. By applying appropriate data structures, verifying of these conditions as well as the selection and assignment of a color (or $L$-interval of colors) can be done in $O(1)$ time per edge. For this reason the time needed to assign a suitable set of colors to an edge incident with $v$ should be independent of $L$ and $\rho(v)$. This can be achieved if the colors already assigned are represented as tight intervals of colors occupied by $L$-edges and tight intervals of available colors, both good and bad. A more detailed analysis of the way the colors are assigned indicates that in each of the three cases the total number of intervals is bounded by 3. Hence, the total number of steps of program OptimalColor is $O(n)$.

Finally, we present efficient formulas to calculate the value of $\Gamma$, and $\text{MaxBad}(v)$, $\text{MaxGood}(v)$ for each vertex $v$ of $T$. It is enough to show formulas for $\gamma(v)$ and $\xi(k)$ which can be calculated in $O(1)$ time. We present them without proving. For any $k \geq DL$, $\xi(k) = \min\{L, k - DL\}$ and

$$
\gamma(v) = \begin{cases} 
0 & \text{if } \mu(v) = 0, \\
\mu(v) + DL & \text{if } 0 < \mu(v) \leq L, \\
\rho_{1D}(v) + \rho_{L}(v)L & \text{if } \mu(v) > L,
\end{cases}
$$

where $\mu(v) = \lceil \rho_{1D}(v)/(D + 1 - \rho_{L}(v)) \rceil$.

Thus $\Gamma$ can be easily found in linear time.
6. Final remarks

Theorem 4.2 gives rise to a natural division of all binomial trees into two subclasses. We can say that \( T \) is of Type 1 if \( \chi(T) = \Omega \), and that \( T \) is of Type 2 if \( \chi(T) = \Omega + 1 \). The type of binomial tree depends only on local properties of the tree. However, some highly structured graphs belong to the first subclass irrespective of the value of \( L \) and configuration of \( L \)-edges. Examples of such Type 1 trees are paths, stars and caterpillars.

Now, let us present an example of Type 2 tree proving that the lower bound may not be equal to the chromatic index.

Example 6.1. Fig. 2 depicts this example.

Let \( L = 2 \). Then,

\[
\delta(v_i) = 3 \quad \text{for } i = 2, 3, \ldots, 7; \quad \delta(v_i) = 2 \quad \text{for } i = 1, 8, 9, 10, 11;
\]

\[
\gamma(v_i) = 0 \quad \text{for } i = 1, 4, 5, 6, \ldots, 11; \quad \gamma(v_i) = 3 \quad \text{for } i = 2, 3.
\]

Hence \( \Gamma = \Delta = 3 \).

Fig. 2(a) shows the labeling of edges in the moment of execution of the last pass of procedure \textit{AssignLabels} (with \( v_1 \) as a parameter). Procedure returns \textit{false} since \( \text{SumGood}(v_1) = 2 > 1 = \text{MaxGood}(v_1) \). Thus \( \chi(T) = 4 \). Fig. 2(b) shows the optimal coloring produced by \textit{Color2}.

Acknowledgements

The authors wish to thank two anonymous referees whose suggestions considerably improved the presentation of this paper.

Fig. 2. (a) Labeled tree; (b) Tree colored for \( k = 4 \).
References