An $O(n^{1.75})$ Algorithm for L(2,1)-labeling of Trees

Hasunuma, Toru
Department of Mathematical and Natural Sciences, The University of Tokushima

Ishii, Toshimasa
Department of Information and Management Science, Otaru University of Commerce

Ono, Hirotaka
Department of Computer Science and Communication Engineering, Kyushu University

Uno, Yushi
Department of Mathematics and Information Sciences, Graduate School of Science, Osaka Prefecture University

http://hdl.handle.net/2324/14762
An $O(n^{1.75})$ Algorithm for $L(2, 1)$-labeling of Trees

Toru Hasunuma$^1$, Toshimasa Ishii$^2$, Hirotaka Ono$^3$ and Yushi Uno$^4$

1 Department of Mathematical and Natural Sciences, The University of Tokushima, Tokushima 770–8502 Japan. hasunuma@ias.tokushima-u.ac.jp
2 Department of Information and Management Science, Otaru University of Commerce, Otaru 047-8501. Japan. ishii@res.otaru-uc.ac.jp
3 Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka 812-8581. Japan. ono@csce.kyushu-u.ac.jp
4 Department of Mathematics and Information Sciences, Graduate School of Science, Osaka Prefecture University, Sakai 599-8531, Japan. uno@mi.s.osakafu-u.ac.jp

Abstract. An $L(2,1)$-labeling of a graph $G$ is an assignment $f$ from the vertex set $V(G)$ to the set of nonnegative integers such that $|f(x) - f(y)| \geq 2$ if $x$ and $y$ are adjacent and $|f(x) - f(y)| \geq 1$ if $x$ and $y$ are at distance 2 for all $x$ and $y$ in $V(G)$. A $k$-$L(2,1)$-labeling is an assignment $f: V(G) \to \{0, \ldots, k\}$, and the $L(2,1)$-labeling problem asks the minimum $k$, which we denote by $\lambda(G)$, among all possible assignments. It is known that this problem is NP-hard even for graphs of treewidth 2. Tree is one of a few classes for which the problem is polynomially solvable, but still only an $O(n \sqrt{n})$ time algorithm for a tree $T$ has been known so far, where $\Delta$ is the maximum degree of $T$ and $n = |V(T)|$. In this paper, we first show that an existent necessary condition for $\lambda(T) = \Delta + 1$ is also sufficient for a tree $T$ with $\Delta = O(\sqrt{n})$, which leads a linear time algorithm for computing $\lambda(T)$ under this condition. We then show that $\lambda(T)$ can be computed in $O(n^{1.75})$ time for any tree $T$. Combining these, we finally obtain an $O(n^{1.75})$ time algorithm, which greatly improves the currently best known result.

Keywords. frequency/channel assignment, graph algorithm, $L(2,1)$-labeling, vertex coloring.

1 Introduction

Let $G$ be an undirected graph. An $L(2,1)$-labeling of a graph $G$ is an assignment $f$ from the vertex set $V(G)$ to the set of nonnegative integers such that $|f(x) - f(y)| \geq 2$ if $x$ and $y$ are adjacent and $|f(x) - f(y)| \geq 1$ if $x$ and $y$ are at distance 2 for all $x$ and $y$ in $V(G)$. A $k$-$L(2,1)$-labeling is an assignment $f: V(G) \to \{0, \ldots, k\}$, and the $L(2,1)$-labeling problem asks the minimum $k$ among all possible assignments. We call this invariant, the minimum value $k$, the $L(2,1)$-labeling number and is denoted by $\lambda(G)$. Notice that we can use $k + 1$ different labels when $\lambda(G) = k$ since we can use 0 as a label for conventional reasons.

The original notion of $L(2,1)$-labeling can be seen in Hale [8] and Roberts [10] in the context of frequency/channel assignment, where ‘close’ transmitters must receive different frequencies and ‘very close’ transmitters must receive frequencies that are at least two frequencies apart so that they can avoid interference. Due to its practical importance, the $L(2,1)$-labeling problem has been widely studied. On the other hand, this problem is also attractive from the graph theoretical point of view since it is a kind of vertex coloring problem. In this context, $L(2,1)$-labeling is generalized into $L(h,k)$-labeling for arbitrary nonnegative integers $h$ and $k$,
and in fact, we can see that \( L(h, 0) \)-labeling is equivalent to the classical vertex coloring problem.

**Related Work:** There are also a number of studies about the \( L(2, 1) \)-labeling problem from the algorithmic point of view. It is known to be NP-hard for general graphs [7], and it still remains NP-hard for some restricted classes of graphs, such as planar, bipartite, chordal graphs [1], and recently it turned out to be NP-hard even for graphs of treewidth 2 [5]. In contrast, only a few graph classes are known to have polynomial time algorithms for this problem. Among those, Chang and Kuo [4] established a polynomial time algorithm for the \( L(2, 1) \)-labeling problem for trees. Their polynomial time algorithm fully exploits the fact that \( \lambda(T) \) is either \( A + 1 \) or \( A + 2 \) for any tree \( T \). It is based on dynamic programming, and runs in \( O(A^{4.5} n) \) time, where \( A \) is the maximum degree of a tree \( T \) and \( n = |V(T)| \).

**Our Contributions:** In this paper, we first show that an existent necessary condition for \( \lambda(T) = A + 1 \) for a tree \( T \) is also sufficient for trees with \( A = \Omega(\sqrt{n}) \), which leads a linear time algorithm for computing \( \lambda(T) \) under this condition. Then we show that the \( L(2, 1) \)-labeling problem can be solved in \( O(A^{1.5} n) \) time for any input tree. Our approach is based on dynamic programming similar to Chang and Kuo’s \( O(A^{4.5} n) \)-time algorithm [4], where its \( A^{2.5} \)-factor comes from the complexity of solving the bipartite matching problem of a graph with order \( A \), and its \( A^{2} \)-factor from the number of iterations for solving bipartite matchings. In spite that our algorithm is also under the same framework, the running time \( O(A^{1.5} n) \) is attained by reducing the number of the matching problems to be solved, together with detailed analyses of the algorithm. As a result, our algorithm achieves \( O(n^{1.75}) \) running time, and greatly improves the best known result \( O(A^{4.5} n) \) time, which could be \( O(n^{2.5}) \) in its worst case.

**Organization of this Paper:** The rest of this paper is organized as follows. Section 2 gives basic definitions and related results. Section 3 shows that a necessary condition that \( \lambda(T) = A + 1 \) for a tree \( T \) is also sufficient for trees with \( A = \Omega(\sqrt{n}) \). In Section 4, after introducing fundamental ideas of dynamic programming for solving this problem, then we show that \( \lambda(T) \) can be computed in \( O(A^{1.5} n) \) time for a tree \( T \). Combining the results in Sections 3 and 4, Section 5 presents the overall \( O(n^{1.75}) \) time algorithm for any input tree. Finally, Section 6 gives some concluding remarks.

## 2 Preliminaries

### 2.1 Definitions and Notations

A graph \( G \) is an ordered set of its vertex set \( V(G) \) and edge set \( E(G) \) and is denoted by \( G = (V(G), E(G)) \). We assume throughout this paper that all graphs are undirected, simple and connected, unless otherwise stated. Therefore, an edge \( e \in E(G) \) is an unordered pair of vertices \( u \) and \( v \), which are end vertices of \( e \), and we often denote it by \( e = (u, v) \). Two vertices \( u \) and \( v \) are adjacent if \( (u, v) \in E(G) \), and two edges are adjacent if they share one of their end vertices. A graph \( G = (V(G), E(G)) \) is called bipartite if the vertex set \( V(G) \) can be divided into two disjoint sets \( V_1 \) and \( V_2 \) such that every edge in \( E(G) \) connects a vertex in \( V_1 \) and one in \( V_2 \); such \( G \) is denoted by \( (V_1, V_2, E) \).

For a graph \( G \), the (open) neighborhood of a vertex \( v \in V(G) \) is the set \( N_G(v) = \{ u \in V(G) \mid (u, v) \in E(G) \} \), and the closed neighborhood of \( v \) is the set \( N_G[v] = N_G(v) \cup \{ v \} \). The degree of a vertex \( v \) is \( |N_G(v)| \), and is denoted by \( d_G(v) \). We use \( \Delta(G) \) to denote the maximum degree of a
graph $G$. A vertex whose degree is $\lambda(G)$ is called major. We often drop $G$ in these notations if there are no confusions. A vertex whose degree is 1 is called a leaf vertex, or simply a leaf. A path in $G$ is a sequence $v_1, v_2, \ldots, v_\ell$ of vertices such that $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, \ell - 1$, or equivalently, a sequence $(v_1, v_2), (v_2, v_3), \ldots, (v_{\ell-1}, v_\ell)$ of edges $(v_i, v_{i+1})$ for $i = 1, 2, \ldots, \ell - 1$. The length of a path is the number of edges on it. The distance between two vertices $u$ and $v$ is the minimum length of paths connecting $u$ and $v$. A path $v_1, v_2, \ldots, v_\ell$ is a cycle if $v_1 = v_\ell$. A graph is a tree if it is connected and has no cycle.

In computing $L(2, 1)$-labelings of trees, it is convenient to regard the input tree to be rooted at an arbitrary vertex $r$ of degree 1 by regarding $r$ as a root. Then we can define the parent-child relationship on vertices in the usual way. For any vertex $v$, the sets of its children and grandchildren are denoted by $C(v)$ and $C^2(v)$, respectively. For a vertex $v$, define $d'(v) = |C(v)|$.

### 2.2 Related Results and Basic Properties

In general, $L(h, k)$-labelings of a graph $G$ are defined for arbitrary nonnegative integers $h$ and $k$, as an assignment of nonnegative integers to $V(G)$ such that adjacent vertices receive labels at least $h$ apart and vertices connected by a 2-length path receive labels at least $k$ apart. This problem is one of the generalizations of the vertex coloring problem since $L(h, 0)$-labeling problem is equivalent to it. Therefore, we can hardly expect that the $L(h, k)$-labeling problem is tractable, and in fact, $L(0, 1)$- and $L(1, 1)$-labeling problems are known to be NP-hard, for example. We can find a lot of related results on $L(h, k)$-labelings in comprehensive surveys by Calamoneri [2] and Yeh [12].

As for the $L(2, 1)$-labeling problem, it is also known to be NP-hard for general graphs [7]. It remains NP-hard for planar graphs, bipartite graphs, chordal graphs [1], and even for graphs of treewidth 2 [5]. In contrast, very few affirmative results are known, e.g., we can decide the $L(2, 1)$-labeling number, and along this direction, a conjecture that $\lambda(G) \leq \Delta^2$ for any graph $G$ with $\Delta \geq 2$ is well known and is still open [7].

We here review some significant results on $L(2, 1)$-labeling of graphs or trees that will become relevant later in this paper. We can see that $\lambda(G) \geq \Delta + 1$ holds for any graph $G$. Griggs and Yeh [7] showed a necessary condition for $\lambda(G) = \Delta + 1$ on any graph $G$, by observing that any major vertex in $G$ must be labeled 0 or $\Delta$ when $\lambda(G) = \Delta + 1$.

**Lemma 1.** [7] If $\lambda(G) = \Delta + 1$, then for any $v \in V(G)$, $N_G[v]$ contains at most two major vertices.

**Lemma 2.** [7] For any tree $T$, $\lambda(T)$ is either $\Delta + 1$ or $\Delta + 2$.

Concerning the latter lemma, they also conjectured the problem of determining if $\lambda(T)$ is $\Delta + 1$ or $\Delta + 2$ is NP-hard. Chang and Kuo [4] disproved this by presenting a polynomial time algorithm for computing $\lambda(T)$, whose running time is $O(\lambda^4 n)$. Since tree is a natural and one of the most basic graph classes, this result yields several affirmative results for more general graph classes, e.g., $p$-almost trees, for which the problem can be solved in $O(\lambda^{2p+4} n)$ time [6].

### 3 A Linear Time Algorithm for Trees with $\Delta = \Omega(\sqrt{n})$

From this section, we concentrate ourselves on the $L(2, 1)$-labeling problem on trees. Obviously, Chang and Kuo’s algorithm [4] runs in linear time if $\Delta = O(1)$. In this section, we show that the $L(2, 1)$-labeling problem for trees can be also solved in linear time if $\Delta > \sqrt{n} + \frac{65}{16} + \frac{1}{4}$. 


Let $T$ be a tree. As shown in Lemmas 1 and 2, we have a necessary condition for $\lambda(T) = A+1$ (or a sufficient condition for $\lambda(T) = A+2$), but no simple necessary and sufficient condition is known although some research such as [11] gave a sufficient condition for $\lambda(T) = A + 1$. Here, we present another sufficient condition for $\lambda(T) = A + 1$, which implies that the necessary condition for $\lambda(T) = A + 1$ of Lemma 1 is also sufficient for large $A$.

Let $N[v]$ denote the set of vertices whose distance from $v$ is at most three.

**Theorem 1.** If for any $v \in V(T)$, $N^3[v]$ contains at most $A-6$ major vertices and $N[v]$ contains at most two major vertices, then $\lambda(T) = A + 1$.

**Proof:** Suppose that for any $v \in V(T)$, $N^3[v]$ contains at most $A - 6$ major vertices and $N[v]$ contains at most two major vertices.

At first, label every major vertex with 0 or $A + 1$ so that two major vertices within distance two do not have the same label. Since for any $v \in V(T)$, $N[v]$ contains at most two major vertices, this labeling can be correctly done.

Next, regard $T$ as a rooted tree by choosing one vertex as the root. Following the definition of the $L(2,1)$-labeling, we will label each vertex in the rooted tree in the breadth-first-search order. Suppose that a vertex $v$ is labeled $b$ and the parent of $v$ is labeled $a$, where $|a-b| \geq 2$.

Divide the set $C(v)$ of children of $v$ into $C'(v), C''(v)$ and $R(v)$ as follows:

- $C'(v) = \{w \in C(v) | w$ is not a major vertex and has a major vertex in $C(w) \cup C^2(w)\}$,
- $C''(v) = \{w \in C(v) | w$ is a major vertex $\}$,
- $R(v) = C(v) - C'(v) - C''(v)$.

Note that $|C'(v)| \leq A - 6, |C''(v)| \leq 2$, and if $d(v) = A$ then $|C'(v)| \leq A - 7$ and $|C''(v)| \leq 1$.

Case 1: $d(v) < A$. Let $U(a,b) = \{a,b-1,b,b+1\} \cup \{0,1,A,A+1\}$, and $\bar{U}(a,b) = \{0,1,\ldots,A+1\} - U(a,b)$. Assign injectively labels in $\bar{U}(a,b)$ to vertices in $C'(v)$. Since $|C'(v)| \leq A - 6$ and $|\bar{U}(a,b)| = A + 2 - |U(a,b)| \geq A - 6$, such a labeling is possible. Let $L(v)$ be the set of labels in $\bar{U}(a,b)$ which are not used in the labeling of $C'(v)$.

Case 1-1: $|C''(v)| = 0$. Assign injectively labels in $L(v) \cup ((0,1,A,A+1) - \{a,b-1,b,b+1\})$ to vertices in $R(v)$.

Case 1-2: $|C''(v)| = 1$. Assign injectively labels in $L(v) \cup ((1,A,A+1) - \{a,b-1,b,b+1\})$ (respectively, $L(v) \cup ((0,1,A) - \{a,b-1,b,b+1\})$) to vertices in $R(v)$, if the major vertex in $C(v)$ is labeled 0 (respectively, $A + 1$).

Case 1-3: $|C''(v)| = 2$. Assign injectively labels in $L(v) \cup ((1,A) - \{b,a-1,a,a+1\})$ to vertices in $R(v)$.

Here, $|L(v)| = |\bar{U}(a,b)| - |C'(v)| = A + 2 - |U(a,b)| - |C'(v)|$. Also,
- $|\{0,1,A,A+1\} - \{a,b-1,b,b+1\}| \geq |U(a,b)| - 4,
- $|\{0,1,A,A+1\} - \{a,b-1,b,b+1\}| \geq |U(a,b)| - 5,$
- $|\{0,1,A\} - \{a,b+1\}| \geq |U(a,b)| - 5,
- $|\{1,A\} - \{a,b-1,b,b+1\}| \geq |U(a,b)| - 6.$

Since $|R(v)| \leq A - 2 - |C'(v)| - |C''(v)|$, each labeling in Cases 1-1,1-2, and 1-3 is possible.

Case 2: $d(v) = A$. Let $U(a) = \{a\} \cup (0,1,A,A+1)$ and $\bar{U}(a) = \{0,1,\ldots,A+1\} - U(a)$. Assign injectively labels in $\bar{U}(a)$ to vertices in $C'(v)$. Since $|C'(v)| \leq A - 7$ and $|\bar{U}(a)| = A + 2 - |U(a)| \geq A - 3$, such a labeling is possible. Let $L(v)$ be the set of labels in $\bar{U}(a)$ which are not used in the labeling of $C'(v)$.

Case 2-1: $|C'(v)| = 0$. Assign injectively labels in $L(v) \cup (\{A,A+1\} - \{a\})$ (respectively, $L(v) \cup ((0,1) - \{a\})$) to vertices in $R(v)$, if $v$ is labeled 0 (respectively, $A + 1$).
Case 2-2: \(|C'(v)| = 1\). Assign injectively labels in \(L(v) \cup \{(a) - \{a\}\}\) (respectively, \(L(v) \cup \{(1) - \{a\}\}\)) to vertices in \(R(v)\), if \(v\) is labeled 0 (respectively, \(A + 1\)).

Here, \(|L(v)| = |\bar{U}(a)| - |C'(v)| = \Delta + 2 - |U(a)| - |C'(v)|\). Also,
- \(|\Delta, A + 1 - \{a\}| \geq |U(a)| - 3,\)
- \(|\{0, 1\} - \{a\}| \geq |U(a)| - 4,\)
- \(|\{1\} - \{a\}| \geq |U(a)| - 4.\)

Since \(|R(v)| = \Delta - 1 - |C'(v)| - |C''(v)|\), each labeling in Cases 2-1 and 2-2 is possible.

It can easily be checked that the labeling of \(C(v)\) is a valid \(L(2, 1)\)-labeling. Therefore, \(\lambda(T) = \Delta + 1.\)

From Theorem 1, we can see that the necessary condition for \(\lambda(T) = \Delta + 1\) in Lemma 1 is also sufficient if the number of major vertices is at most \(\Delta - 6\).

**Corollary 1.** If the number of major vertices is at most \(\Delta - 6\), then \(\lambda(T) = \Delta + 1\) if and only if for any \(v \in V(T)\), \(N[v]\) contains at most two major vertices. □

**Corollary 2.** If \(\Delta > \sqrt{n + \frac{65}{16} + \frac{11}{4}}\), then \(\lambda(T) = \Delta + 1\) if and only if for any \(v \in V(T)\), \(N[v]\) contains at most two major vertices.

**Proof:** Suppose that for any \(v \in V(T)\), \(N[v]\) contains at most two major vertices. Assume that there are \(\Delta - 5\) major vertices \(w_i, (1 \leq i \leq \Delta - 5)\). Since any two edges joining two major vertices are not adjacent, the number of edges joining two major vertices is at most \(\frac{\Delta}{2} - 5\). Therefore,

\(|\bigcup_{1 \leq i \leq 5} E(w_i)| \geq \Delta(A - 5) - \frac{\Delta - 5}{2}|,\)

where \(E(w_i)\) denotes the set of edges incident to \(w_i\) in \(T\). Hence, \(n - 1 \geq \frac{(\Delta - 5)^2}{2}\). From this inequality, we obtain \(\Delta \leq \sqrt{n + \frac{65}{16} + \frac{11}{4}}\). Therefore, if \(\Delta > \sqrt{n + \frac{65}{16} + \frac{11}{4}}\), then the number of major vertices is at most \(\Delta - 6\). Hence, from Corollary 1, this corollary follows. □

Clearly, the condition that \(N[v]\) contains at most two major vertices for any \(v \in V(T)\) can be checked in linear time. Thus, when \(\Delta > \sqrt{n + \frac{65}{16} + \frac{11}{4}}\), we can decide \(\lambda(T)\) in linear time, and if \(\lambda(T) = \Delta + 1\), then a \((\Delta + 1)\)-\(L(2, 1)\)-labeling of \(T\) can be obtained by an algorithm based on the proof of Theorem 1, which runs in linear time. Otherwise, we can obtain \((\Delta + 2)\)-\(L(2, 1)\)-labeling by Algorithm **Greedy:** traverse \(T\) in the breadth-first order, and if reached vertex \(v\) where \(f(v) = a\) and \(f(u) = b\) for its parent \(u\), label vertices in \(C(v)\) from \([0, 1, \ldots, \Delta + 2]\) - \([b, a - 1, a, a + 1]\). This is always possible since \(|C(v)| \leq |\{0, 1, \ldots, \Delta + 2\} - [b, a - 1, a, a + 1]|\) for any \(v\), and it gives \((\Delta + 2)\)-\(L(2, 1)\)-labeling.

## 4 An \(O(\Delta^{1.5} n)\)-time Algorithm

### 4.1 Chang and Kuo's Algorithm

In this subsection, we review a dynamic programming algorithm for the \(L(2, 1)\)-labeling problem of trees, which is proposed by Chang and Kuo [4], since our algorithm also utilizes the same formula of the principle of optimality. For a tree \(T\) with maximum degree \(\Delta\), Griggs and
Yeh [7] proved that $\lambda(T) = A + 1$ or $A + 2$. The algorithm determines if $\lambda(T) = A + 1$, and if so, we can easily construct the labeling with $\lambda(T) = A + 1$.

Before describing the algorithm, we introduce some notations. We assume that $T$ is rooted at some leaf vertex $r$ for explanation. Given a vertex $v$, we denote the subtree of $T$ rooted at $v$ by $T(v)$. Let $T(u, v)$ be a tree rooted at $u$ that forms $T(u, v) = ([u] \cup V(T(v)), ([u, v]) \cup E(T(v)))$. Note that this $u$ is just a virtual vertex for explanation and $T(u, v)$ is uniquely decided for $T(v)$ in a sense. For a rooted tree, we call the length of the longest path from the root to a leaf its height. For $T(u, v)$, we define

$$\delta(u, v, (a, b)) = \begin{cases} 1 & \text{if } \lambda(T(u, v)) \mid f(u) = a, f(v) = b \leq A + 1, \\ 0 & \text{otherwise}, \end{cases}$$

where $\lambda(T(u, v)) \mid f(u) = a, f(v) = b$ denotes the $L(2, 1)$-labeling number on $T(u, v)$ under the assumption that $f(u) = a$ and $f(v) = b$, that is, the minimum $k$ of $k$-$L(2, 1)$-labeling on $T(u, v)$ satisfying $f(u) = a$ and $f(v) = b$. This $\delta$ function satisfies the following:

$$\delta((u, v), (a, b)) = \begin{cases} 1 & \text{if there is a distinct assignment } c_1, c_2, \ldots, c_{d'(v)} \text{ on } w_1, w_2, \ldots, w_{d'(v)}, \text{ where } c_i \text{ is different from } a, b, \\ b - 1, b + 1, \text{ and } \delta((v, w_i), (b, c_i)) = 1 \text{ for each } i, \\ 0 & \text{otherwise}, \end{cases}$$

where $w_1, w_2, \ldots, w_{d'(v)}$ are children of $v$. The existence of an assignment $c_1, c_2, \ldots, c_{d'(v)}$ on $w_1, w_2, \ldots, w_{d'(v)}$ as above is formalized as the maximum bipartite matching problem; we consider a bipartite graph $G(u, v, a, b) = (V(v), X, E(u, v, a, b))$, where $V(v) = \{w_1, w_2, \ldots, w_{d'(v)} \in C(v)\}$, $X = \{0, 1, \ldots, A, A + 1\}$ and $E(u, v, a, b) = \{(w, c) \mid \delta((v, w), (b, c)) = 1, c \in X - \{a\}, w \in V(v)\}$. (Analogously, we also define $G(u, v, -b)$ by $E(u, v, -b) = \{(w, c) \mid \delta((v, w), (b, c)) = 1, c \in X, w \in V(v)\}$, which will be used in Subsection 4.3.) We can see that an assignment $c_1, c_2, \ldots, c_{d'(v)}$ on $w_1, w_2, \ldots, w_{d'(v)}$ is feasible if there exists a matching with size $d'(v)$ of $G(u, v, a, b)$. Namely, for $T(u, v)$ and two labels $a$ and $b$, we can easily (i.e., in polynomial time) determine the value of $\delta((u, v), (a, b))$ if the values of $\delta$ function for $T(v, w_i)$ and any two pairs of labels are given. According to these observations, Chang and Kuo proposed the following dynamic programming algorithm:

**Algorithm CK**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Let $\delta((u, v), (\ast, \ast)) := 1$ for all $T(u, v)$ of height 1, where $(\ast, \ast)$ means all pairs of labels $a$ and $b$, where $</td>
</tr>
<tr>
<td>1</td>
<td>For all $T(u, v)$ of height $h$, compute $\delta((u, v), (\ast, \ast))$.</td>
</tr>
<tr>
<td>2</td>
<td>If $h = h'$ where $h'$ is the height of root $r$ of $T$, then goto Step 3. Otherwise let $h := h + 1$ and goto Step 1.</td>
</tr>
<tr>
<td>3</td>
<td>If $\delta((r, v), (a, b)) = 1$ for some $(a, b)$, then output “Yes”. Otherwise output “No”. Halt.</td>
</tr>
</tbody>
</table>

Since Steps 0, 2, and 3 can be done just by looking up the table of $\delta$, the running time is dominated by Step 1; the total running time of the algorithm is $O(\sum_{v \in V} t(v))$, where $t(v)$ denotes the time for calculating $\delta((u, v), (\ast, \ast))$. Each calculation of $\delta((u, v), (a, b))$ in Step 1 can be
executed in $O(|V(v)\cup X|^2) = O(d^2)$ time, because an $O(n^{2.5})$ time algorithm is known for the maximum matching of a bipartite graph with $n$ vertices [9]. Since the number of pairs $(a, b)$ is at most $(d + 2) \times (d + 2)$, we obtain $t(v) \leq (d + 2) \times O(d^2) = O(d^4)$. Thus the total running time of the algorithm is $\sum_{v \in V} t(v) = O(d^5 n)$. \footnote{By a careful analysis, this running time is reduced to $O(d^{4.5} n)$.}

In the following subsections, we propose a more efficient algorithm. It is also based on the formula (1) as the principle of optimality, but it computes $\delta((u, v), (v_1, v_2))$ more efficiently.

### 4.2 Preprocessing Operations for Input Trees

In this subsection, we introduce preprocessing operations in our algorithm. Let $T$ be an original input tree. These preprocessing operations are carried out for the purpose that (1) remove inessential vertices from $T$, where “inessential” means that they do not affect the $L(2, 1)$-labeling number of $T$, and (2) divide $T$ into several subtrees that preserves the $L(2, 1)$-labeling number of $T$. Obviously, these operations enable to reduce the input size to solve and we may expect some speedup. However, the effect for speedup is not important actually, because the preprocessing operations may do nothing for some instances. Instead, a more important effect is that we can restrict the shape of input trees, which enables to reduce the running time of our algorithm shown later.

First, we describe how to remove inessential vertices.

1. Check if there is a leaf $v$ whose unique neighbor $u$ has degree less than $d$. If so, remove $v$ and edge $(u, v)$ from $T$ until such a leaf does not exist.

   This operation does not affect the $L(2, 1)$-labeling number of $T$, that is, $\lambda(T) = \lambda(T')$ where $T$ is the original tree and $T'$ is the resulting tree. This is because, in $T$, such leaf vertex $v$ can be properly labeled by some number in $\{0, 1, \ldots, d + 1\}$ if $u$ and any other neighbor vertices of $u$ are properly labeled by numbers among $\{0, 1, \ldots, d + 1\}$. Also, the operation does not change the maximum degree $d$. Since this can be done in linear time, the labeling problem for $T$ is equivalent to the one for $T'$ in terms of linear time computation. Thus, from now on, we assume that an input tree $T$ has the following property.

   **Property 1.** All vertices connected to a leaf vertex are major vertices.

   We define $V_L$ as the set of all leaf vertices in $T$. Also we define $V_Q$ as the set of major vertices whose children are all leaves.

   Next, we explain how to divide $T$ into subtrees. We call a sequence of consecutive vertices $v_1, v_2, \ldots, v_\ell$ a path component if $(v_i, v_{i+1}) \in E$ for all $i = 1, 2, \ldots, \ell - 1$ and $d(v_i) = 2$ for all $i = 1, 2, \ldots, \ell$, and we call $\ell$ the size of the path component. For example, consider vertices $v_1, v_2, v_3$ and $v_4$ of $T$ where each $v_i$ is connected to $v_{i+1}$ for $i = 1, 2, 3$. If $d(v_1) = \cdots = d(v_4) = 2$ holds, then $v_1, \ldots, v_4$ is a path component with size 4.

2. Check if there is a path component whose size is at least 4, say $v_1, v_2, \ldots, v_\ell$, and let $v_0$ and $v_{\ell+1}$ be the unique adjacent vertices of $v_1$ and $v_\ell$ other than $v_2$ and $v_{\ell-1}$, respectively.

   If it exists, assume $T$ is rooted at $v_1$, divide $T$ into $T_1 := T(v_1, v_0)$ and $T_2 := T(v_4, v_5)$, and remove $v_2$ and $v_3$. Continue this operation until such a path component does not exist.
We assume \( A \geq 7 \), because otherwise the original algorithm \( \text{CK} \) is already a linear time algorithm. Here, we show that \( \lambda(T) = A + 1 \) if and only if \( \lambda(T_1) = \lambda(T_2) = A + 1 \). The only-if part is obvious, and we show the if part. Suppose that \( f(v_1) = a \) and \( f(v_0) = b \) in a \((A + 1)\)-\( L(2, 1) \)-labeling of \( T_1 \), and \( f(v_2) = a' \) and \( f(v_3) = b' \) in a \((A + 1)\)-\( L(2, 1) \)-labeling of \( T_2 \). Then set \( f(v_2) = c \) where \( |c - a| \geq 2 \) and \( c \) is neither \( b \) nor \( a' \), and set \( f(v_3) = c' \) where \( |c' - c| \geq 2 \), \( |c' - a'| \geq 2 \) and \( c' \) is neither \( a \) nor \( b' \). This gives a \((A + 1)\)-\( L(2, 1) \)-labeling of \( T \) and is always possible since \( A \geq 7 \). Namely, we can find an \( L(2, 1) \)-labeling of \( T \) by finding \( L(2, 1) \)-labelings of \( T_1 \) and \( T_2 \) independently, which guarantees that this preprocessing preserves \((A + 1)\)-\( L(2, 1) \)-labeling of \( T \) if it exists. Clearly, this operation can be done in linear time. Thus, from now on, we assume that an input tree \( T \) has the following property.

**Property 2.** The size of any path component of \( T \) is at most 3.

### 4.3 Efficient Search for Augmenting Paths

As observed in Subsection 4.1, the running time of algorithm \( \text{CK} \) is dominated by Step 1. Step 1 of algorithm \( \text{CK} \) computes the maximum bipartite matching \( O(A^2) \) times for calculating \( \delta(u, v, (*, *)) \) for \( T(u, v) \), which takes \( O(A^{4.5}) \) time. In this subsection, we show that for \( T(u, v) \), \( \delta(u, v, (*, *)) \) can be calculated more efficiently; for a fixed label \( b \), \( \delta(u, v, (i, b)) \mid i \in \{0, 1, \ldots, A \} \) can be obtained in \( O(A^{1.5}d'(v)) \) time by computing a single maximum bipartite matching and a single graph search, where \( d'(v) \) is the number of children of \( v \). This shows that \( t(v) = O(A^{2.5}d'(v)) \).

Let \( G(u, v, -b) = (V(v), X, E(u, v, -b)) \) be the bipartite graph defined in Subsection 4.1, where \( V(v) = \{v_1, v_2, \ldots, v_{d'(v)}\} \) and \( X = \{0, 1, \ldots, A + 1\} \). In this subsection, we may refer to \( i \in X \) as a label \( i \). It is not difficult to see that the following property holds.

**Lemma 3.** If \( G(u, v, -b) \) has no matching of size \( d'(v) \), then \( \delta(u, v, (i, b)) = 0 \) for any label \( i \).

Below, consider the case where \( G(u, v, -b) \) has a matching of size \( d'(v) \); without loss of generality, let \( M = \{(w_{i+1}, i) \mid i \in \{0, 1, \ldots, d'(v) - 1\}\} \) be such a matching in \( G(u, v, -b) \) (note that by \( d'(v) \leq A \), each vertex in \( V(v) \) is matched). Recall, as mentioned in Subsection 4.1, that for each label \( i \in \{0, 1, \ldots, A + 1\}, \delta(u, v, (i, b)) = 1 \) if and only if \( G(u, v, i, b) \) has a matching of size \( d'(v) \). Clearly, \( \delta(u, v, (i, b)) = 1 \) for each \( i \in \{d'(v), d'(v) + 1, \ldots, A + 1\} \).

Next consider the value of \( \delta(u, v, (i, b)) \) for \( i \in \{0, 1, \ldots, d'(v) - 1\} \). Let \( i \in \{0, 1, \ldots, d'(v) - 1\} \). Note that \( G(u, v, i, b) \) has the matching \( M - \{(w_{i+1}, i)\} \) of size \( d'(v) - 1 \). Given a matching \( M' \), a path is called \( M' \)-alternating if its edges are alternately in and not in \( M' \). In particular, an \( M' \)-alternating path is called \( M' \)-augmenting if the end vertices of the path are both unmatched by \( M' \). It is well-known that \( M' \) is a maximum matching if and only if there is no \( M' \)-augmenting path.

Hence, \( G(u, v, i, b) \) has a matching of size \( d'(v) \) if and only if \( G(u, v, i, b) \) has an \((M - \{(w_{i+1}, i)\})\)-augmenting path; \( G(u, v, -b) \) has an \((M - \{(w_{i+1}, i)\})\)-augmenting path not passing through vertex \( i \). It follows that for each label \( i \in \{0, 1, \ldots, d'(v) - 1\} \), we can decide the value of \( \delta(u, v, (i, b)) \) by checking whether there exists an \((M - \{(w_{i+1}, i)\})\)-augmenting path not passing through vertex \( i \) in \( G(u, v, -b) \). Notice that for any label \( i \), if such an augmenting path \( P \) exists, then one of two end vertices of \( P \) is always included in \( X' \), where \( X' = \{d'(v), d'(v) + 1, \ldots, A + 1\} \subseteq X \) (note that the other end vertex is \( w_{i+1} \)). Moreover, by the following Lemma 4, we can decide the value of \( \delta(u, v, (i, b)) \), \( i \in \{0, 1, \ldots, A + 1\} \) simultaneously by traversing all vertices which can be reached by an \( M' \)-alternating path from some vertex in \( X' \) in \( G(u, v, -b) \).
Lemma 4. \( \delta((u, v), (i, b)) = 1 \) if and only if vertex \( i \) can be reached by an \( M \)-alternating path from some vertex in \( X' \) in \( G(u, v, -b) \).

Proof: Assume that \( \delta((u, v), (i, b)) = 1 \). Then, there exists an \((M - \{w_{i+1}, i\})\)-augmenting path \( P \) not passing through vertex \( i \). Note that two end vertices of \( P \) are \( w_{i+1} \) and some vertex \( u' \in X' \). Hence, it follows that vertex \( i \) can be reached by the \( M \)-alternating path \( P \cup \{w_{i+1}, i\} \) from \( u \in X' \).

Assume that vertex \( i \) can be reached by an \( M \)-alternating path from some vertex in \( X' \) in \( G(u, v, -b) \). Let \( P \) be such an \( M \)-alternating path in which vertex \( i \) appears exactly once. Since \( P \) starts from a vertex in \( X' \), we can observe that the edge which appears immediately before reaching vertex \( i \) in \( P \) is \((w_{i+1}, i) \in M \). Hence, the path \( P - \{w_{i+1}, i\} \) is an \((M - \{w_{i+1}, i\})\)-augmenting path not passing through vertex \( i \), and it follows that \( \delta((u, v), (i, b)) = 1 \). □

All vertices which can be reached by an \( M \)-alternating path from some vertex in \( X' \) in \( G(u, v, -b) \) can be computed in \( O(|E(u, v, -b)| + |X'|) = O(Md(v)) \) time, by using the depth first search from vertex \( s \) in \( G_s \), where \( G_s \) denotes the graph obtained from \( G(u, v, -b) \) by adding a new vertex \( s \) and new edges connecting \( s \) and each vertex in \( X' \).

Consequently, \( [\delta((u, v), (i, b)) \mid i \in \{0, 1, \ldots, A+1\}] \) can be obtained by computing a single bipartite matching and a single depth first search. The time complexity is \( O(A^{1.5}d(v) + Ad'(v)) = O(A^{1.5}d(v)) \). Hence, \( \delta((u, v), (*, *)) \) can be obtained in \( O(A^{2.5}d'(v)) \) by applying the above computation for each label \( b \); \( t(v) = O(A^{2.5}d'(v)) \).

4.4 Efficient Computation of \( \delta \)-values near Leaves

In Subsections 4.1 and 4.3, we have observed that algorithm CK runs in \( O(\sum_{v \in V} t(v)) = O(A^{2.5} \sum_{v \in V} d'(v)) \) time. In this subsection, we show that algorithm CK can be implemented to run in \( O(A^{2.5} \sum_{v \in V - L - V_Q} d''(v)) \) time by avoiding unnecessary bipartite matching computations for vertices incident to leaves, where \( V_L \) and \( V_Q \) are defined in Subsection 4.2 and \( d''(v) = |C(v) - V_L| \).

For a vertex \( v \in V_L \cup V_Q \), we can easily obtain \( \delta((u, v), (*, *)) \) without computing the bipartite matching. Actually, for a leaf \( v \in V_L, \delta((u, v), (a, b)) = 1 \) if and only if \(|a - b| \geq 2 \). For a vertex \( v \in V_Q \), we have \( \delta((u, v), (a, b)) = 1 \) if and only if \( b \in [0, A + 1] \) and \(|a - b| \geq 2 \) (notice that each vertex in \( V_Q \) is major). Thus, the running time of algorithm CK is dominated by Step 1 for vertices \( v \in V - V_L - V_Q; O(\sum_{v \in V} t(v)) = O(\sum_{v \in V - V_L - V_Q} t(v)) \).

Also for a vertex \( v \in V - V_L - V_Q \) incident to some leaf, we can gain some saving of time for computing \( \delta((u, v), (*, *)) \); for a label \( b \), the calculation of \( \delta((u, v), (*, b)) \) can be done in \( O(A^{1.5}d''(v)) \) time, instead of \( O(A^{1.5}d'(v)) \) time. Let \( v \) be a vertex in \( V - V_L - V_Q \) incident to some leaf, \( C(v) \cap V_L \neq \emptyset \). Note that \( v \) is major by Property 1, and that \( \delta((u, v), (*, b)) = 0 \) for each \( b \notin [0, A + 1] \). Thus, we have only to decide the value of \( \delta((u, v), (*, b)) \) for \( b \in [0, A + 1] \).

Then, we can observe that for computing \( \delta((u, v), (*, b)) \), it suffices to check whether there exists a feasible assignment only on \( C(v) - V_L \), instead of \( C(v) \). Actually, if \( b = 0 \) and there exists a feasible assignment on \( C(v) - V_L \), then the number of the remaining labels is \( A + 1 \) since \( v \) is major. The case of \( b = A + 1 \) can also be treated similarly. Therefore, it follows that the calculation of \( \delta((u, v), (*, b)) \) is dominated by the maximum matching computation in the subgraph of \( G(u, v, -b) \) induced by \( (V(v) - V_L) \cup X \); its time complexity is \( O(A^{1.5}d'(v)) \).

\[ O(A^{1.5}d'(v)) \]
Consequently, algorithm CK can be implemented to run in $O(\sum_{v \in V - V_L} t(v)) = O(\Delta^{2.5} \sum_{v \in V - V_L - V_0} d''(v))$ (note that $d''(v) = d'(v)$ for each vertex $v$ with $C(v) \cap V_L = \emptyset$).

### 4.5 Amortized Analysis

In Subsections 4.2–4.4, we have observed that by an efficient implementation of algorithm CK, $\lambda(T)$ can be decided in $O(\sum_{v \in V - V_L - V_0} t(v)) = O(\Delta^{2.5} \sum_{v \in V - V_L - V_0} d''(v))$ time. Below, we show that $O(\Delta^{2.5} \sum_{v \in V - V_L - V_0} d''(v)) = O(\Delta^{1.5} n)$ by amortized analysis; namely, we show the following lemma.

**Lemma 5.** Algorithm CK can be implemented to run in $O(\Delta^{1.5} n)$ time. □

Let $V_B$ be the set of vertices $v \in V - V_L - V_Q$ with $d''(v) \geq 2$, $V_P$ be the set of vertices $v \in V - V_L - V_Q$ with $d'(v) = 1$, and $V'_P = V - (V_L \cup V_Q \cup V_B \cup V_P)$. Note that each vertex in $V_P$ belongs to a certain path component. Also note that each $v \in V_P$ satisfies $d''(v) = 1$ and $C(v) \cap V_L \neq \emptyset$, and hence by Property 1, it is incident to exactly $\Delta$ leaves.

Now by Property 2, for each vertex $v \in V_P$, there exist the root $r$ or a vertex in $V_B \cup V'_P$ among its ancestors which are at most at distance 3 from $v$. Hence, we have $|V_P| \leq 3 \sum_{v \in V_B \cup V'_P} d'(v) + 3$. By $\sum_{v \in V_P} d'(v) = |V_P|$, it follows that

$$\Delta^{2.5} \sum_{v \in V - V_L - V_0} d''(v) = \Delta^{2.5} \sum_{v \in V_B \cup V'_P} d''(v) + \Delta^{2.5} |V_P| \leq \Delta^{2.5} (\sum_{v \in V_B \cup V'_P} 4d''(v) + 3) = O(\Delta^{2.5} (\sum_{v \in V_B \cup V'_P} d''(v) + 1)) .$$

Thus, for proving $O(\Delta^{2.5} \sum_{v \in V - V_L - V_0} d''(v)) = O(\Delta^{1.5} n)$, it suffices to show that $\sum_{v \in V_B \cup V'_P} d''(v) = O(n/\Delta)$.

**Lemma 6.** $\sum_{v \in V_B \cup V'_P} d''(v) = O(n/\Delta)$.

**Proof:** Let $E'$ be the set of all edges connecting a vertex in $V_B \cup V'_P$ and its non-leaf child. Note that $|E'| = \sum_{v \in V_B \cup V'_P} d''(v)$. Let $E_L$ denote the set of all edges incident to a leaf, and $E_P$ denote the set of all edges connecting a vertex in $V_P$ and its unique child. Also note that $|E_L| = |V_L|$, $|E_P| = |V_P|$, $E_L \cap E_P = \emptyset$, and $(E_L \cup E_P) \cap E' = \emptyset$. Hence, we have $|E'| \leq |E| - |E_L| - |E_P| = n - 1 - |V_L| - |V_P|$. Now, by $V = V_L \cup V_Q \cup V_B \cup V_P$, and that $V_L$, $V_Q$, $V_B$, $V_P$ are disjoint each other, we have $n = |V| = |V_L| + |V_Q| + |V_B| + |V_P|$. Therefore, it follows that $|E'| \leq n - 1 - (n - |V_Q| - |V_B| - |V_Q|) = |V_B| + |V_P| + |V_Q| - 1$.

Now since each vertex $u \in V_B$ has at least two non-leaf children and each leaf not incident to $V_B \cup V'_P$ is incident to a vertex in $V_Q$, we can observe that $|V_Q| \geq |V_B| + 1$ holds. Since each vertex in $V'_P$ (respectively, $V_Q$) is incident to exactly $\Delta$ (respectively, $\Delta - 1$) leaves, we have $|V_L| \geq |V'_P|(\Delta - 2) + |V_Q|(\Delta - 1)$. Consequently, we have $\sum_{v \in V_B \cup V'_P} d''(v) = |E'| \leq |V_B| + |V'_P| + |V_Q| - 1 \leq 2|V_Q| + |V'_P| - 2 \leq 2|V_L|/(\Delta - 2) - 2$. □

### 5 An $O(n^{1.75})$-time Algorithm

Summarizing the arguments given in Sections 3 and 4, we give a description of the overall algorithm named LABEL-TREE, for determining in $O(n^{1.75})$ time whether $\lambda(T) = \Delta + 1$ or not for any input tree $T$. 
Theorem 2. Namely we have the following result.

Finally, we remark that our results can be extended to apply to some wider variations of labeling problems, as well as the $L(2,1)$-labeling problem on trees.

We show that algorithm LABEL-TREE can be implemented to run in $O(n^{1.75})$ time. Clearly, all of the preprocessing, Steps 0 and 1 can be executed in linear time. As observed in Subsection 4.5, Steps 2–5 can be executed in $O(hp)$ time. Moreover, as shown in the proof of Corollary 2, if $N[v]$ contains at most two major vertices for any vertex $v \in V$ and the total number of major vertices is at least $A - 5$, we have $A = O(\sqrt{n})$. Thus, Steps 2–5 take $O(n^{1.75})$ time, and it follows that the running time of algorithm LABEL-TREE is $O(n^{1.75})$.

Moreover, we remark that in both cases of $\lambda(T) = A + 1, A + 2$, we can easily construct a $\lambda(T)$-$L(2,1)$-labeling in the same complexity. Actually, if $\lambda(T) = A + 2$, then a $\lambda(T)$-$L(2,1)$-labeling can be obtained by Algorithm GREEDY in Section 3. If $\lambda(T) = A + 1$ is determined as a result of Step 1, then according to the proof of Theorem 1, a $\lambda(T)$-$L(2,1)$-labeling can be obtained in linear time. Also if $\lambda(T) = A + 1$ is determined as a result of Step 5, then we can obtain the $\lambda(T)$-$L(2,1)$-labeling in $O(n^{1.75})$ time, following the dynamic programming based procedure of Steps 2–5. Namely we have the following result.

Theorem 2. For trees, the $L(2,1)$-labeling problem can be solved in $O(\min\{n^{1.75}, n^{1.5}\})$ time.

□

6 Concluding Remarks

Finally, we remark that our results can be extended to apply to some wider variations of labeling problems, as well as the $L(2,1)$-labeling problem on trees.

It is known that Chang and Kuo’s algorithm [4] can be extended to solve the $L(h,1)$-labeling problem on trees [3] and $p$-almost trees [6], where a $p$-almost tree is a connected graph with $n + p - 1$ edges. By extending the original Chang and Kuo’s algorithm, the $L(h,1)$-labeling problem on trees can be solved in $O((h + A)^{3.5}n) = O(\lambda^{3.5}n)$ time, and $L(2,1)$-labeling on $p$-almost trees can be solved in $O(\lambda^{2p+4.5}n)$ time for $\lambda$ given as an input. Our techniques in Subsection 4.3 can also be applied to speed up those algorithms. In fact, it is easy to show
that our techniques can solve the $L(h, 1)$-labeling problem on trees in $O(n^{3.5})$ time, and the $L(2, 1)$-labeling problem on $p$-almost trees in $O(n^{2p+2.5})$ time. Moreover, if some properties such as Theorem 1 hold, then we may expect some more improvement on these problems.

References