A linear time algorithm for L(2,1)-labeling of trees

Hasunuma, Toru

Ishii, Toshimasa

Ono, Hirotaka

Uno, Yushi

A linear time 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 $L(2,1)$-labeling $f : V(G) \to \{0,\ldots,k\}$, 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, and tree is one of very few classes for which the problem is polynomially solvable. The running time of the best known algorithm for trees had been $O(\Delta^{4.5}n)$ for more than a decade, and an $O(\min\{n^{1.75}, \Delta^{1.5}n\})$-time algorithm has appeared recently, where Δ is the maximum degree of T and $n = |V(T)|$, however, it has been open if it is solvable in linear time. In this paper, we finally settle this problem for $L(2,1)$-labeling of trees by establishing a linear time algorithm.

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 $L(2,1)$-labeling $f : V(G) \to \{0,\ldots,k\}$, 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 the context of frequency 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. From the graph theoretical point of view, since this is a kind of vertex coloring problem, it has attracted a lot of interest [4, 10, 13, 16]. In this context, $L(2,1)$-labeling is generalized into $L(p,q)$-labeling for arbitrary nonnegative integers p and q, and in fact, we can see that $L(1,0)$-labeling ($L(p,0)$-labeling, actually) is equivalent to the classical vertex coloring. We can find a lot of related results on $L(p,q)$-labelings in comprehensive surveys by Calamoneri [2] and by Yeh [17].

Related Work: There are also a number of studies on the $L(2,1)$-labeling problem from the algorithmic point of view [1, 8, 15]. It is known to be NP-hard for general graphs [10], and it still remains NP-hard for some restricted classes of graphs, such as planar graphs, bipartite graphs, chordal graphs [1], and 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, e.g., we can determine the $L(2,1)$-labeling number of paths, cycles, wheels within polynomial time [10].

As for trees, Griggs and Yeh [10] showed that $\lambda(T)$ is either $\Delta + 1$ or $\Delta + 2$ for any tree T, and also conjectured that determining $\lambda(T)$ is NP-hard, however, Chang and Kuo [4] disproved this by presenting a polynomial time algorithm for computing $\lambda(T)$. Their algorithm exploits the fact that $\lambda(T)$ is either $\Delta + 1$ or $\Delta + 2$ for any tree T. Its running time is $O(\Delta^{4.5}n)$, where Δ is the maximum degree of a tree T and $n = |V(T)|$. This result has a great importance because it initiates to cultivate polynomially solvable classes of graphs for the $L(2,1)$-labeling problem and related problems. For example, Fiala et al. showed that $L(2,1)$-labeling of t-almost trees can be solved in $O(\lambda^{2t+4.5}n)$ time for λ given as an input, where a t-almost tree is a graph that can be a tree by eliminating t edges [8]. Also, it was shown that the $L(p,1)$-labeling problem for trees can be solved in $O((p + \Delta)^{5.5}n) = O(\lambda^{5.5}n)$ time [3]. Both results are based on Chang and Kuo's algorithm, which is called as a subroutine in the algorithms. Moreover, the polynomially solvable result for trees holds for more general settings. The notion of $L(p,1)$-labeling is generalized as $H(p,1)$-labeling, in which graph H defines the metric space of distances between two labels, whereas labels in $L(p,1)$-labeling (that is, in $L(p,q)$-labeling) take nonnegative integers; i.e., it is a special case that H is a path graph. In [6], it has been shown that the $H(p,1)$-labeling problem of trees for arbitrary graph H can be solved in polynomial time, which is also based on Chang and Kuo's idea. In passing, these results are unfortunately not applicable for $L(p,q)$-labeling problems for general p and q. Recently, Fiala et al. [7] showed that the $L(p,q)$-labeling problem for trees is NP-hard if q is not a divisor of p, which is contrasting to the positive results mentioned above.

As for $L(2,1)$-labeling of trees again, Chang and Kuo's $O(\Delta^{4.5}n)$ algorithm is the first polynomial time one. It is based on dynamic programming (DP) approach, and it checks whether $(\Delta + 1)$-$L(2,1)$-labeling is possible or not from leaf vertices to a root vertex in the original tree structure. The principle of optimality requires to solve at each vertex of the tree the assignments of labels to subtrees, and the assignments are formulated as the maximum matching in a certain bipartite graph. Recently, an $O(\min\{n^{1.75}, \Delta^{1.5}n\})$ time algorithm has been proposed [11]. It is based on the similar DP framework to Chang and Kuo's algorithm, but achieves its efficiency by reducing heavy computation of bipartite matching in Chang and Kuo's and by using an amortized analysis. We give a concise review of these two algorithms in Subsection 2.2.

Our Contributions: Although there have been a few polynomial time algorithms for $L(2,1)$-labeling of trees, it has been open if it can be improved to linear time [2]. In this paper, we present a linear time algorithm for $L(2,1)$-labeling of trees, which finally settles this problem. It is based on the similar DP approach to the preceding two polynomial time algorithms [4, 11]. In our new algorithm, besides using their ideas, we introduce the notion of "label compatibility", which indicates how we flexibly change labels with preserving its $(\varDelta + 1)$-$L(2,1)$-labeling. Interestingly, we can show that only $\mathrm{O}(\log_{\varDelta} n)$ labels are essential for $L(2,1)$-labeling in any input tree by using this notion. By utilizing this fact, we can replace the bipartite matching of graphs with the maximum flow of much smaller networks as an engine to find the assignments. Consequently, our algorithm finally achieves its linear running time.

Organization of this Paper: The rest of this paper is organized as follows. Section 2 gives basic definitions and introduces as a warm-up the ideas of Chang and Kuo's $\mathrm{O}(\varDelta^{4.5} n)$ time algorithm and its improvement into $\mathrm{O}(n^{1.75})$ time. Section 3 introduces the crucial notion of label compatibility that can bundle a set of compatible vertices and reduce the size of the graph constructed for computing bipartite matchings. Moreover, this allows to use maximum-flow based computation for them. In Section 4, we give precise analyses to achieve linear running time. Some parts of the detailed analyses are omitted due to space limitation. Interested readers can find them in the technical report version of this paper [12].

2 Preliminaries

2.1 Definitions and Notations

A graph G is an ordered pair 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 $(u, v) \in E(G)$. 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 $\varDelta(G)$ to denote the maximum degree of a graph G. A vertex whose degree is $\varDelta(G)$ 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*.

When we describe algorithms, it is convenient to regard the input tree to be rooted at a leaf vertex r. Then we can define the parent-child relationship on vertices in the usual way. For a rooted tree, its *height* is the length of the longest path from the root to a leaf. For any vertex v, the set of its children is denoted by $C(v)$. For a vertex v, define $d'(v) = |C(v)|$.

2.2 Chang and Kuo's Algorithm and its Improvement

Before explaining algorithms, we give some significant properties on $L(2,1)$-labeling of graphs or trees that have been used so far for designing $L(2,1)$-labeling algorithms. We can see that $\lambda(G) \geq \varDelta + 1$ holds for any graph G. Griggs and Yeh [10] observed that any major vertex in G must be labeled 0 or $\varDelta + 1$ when $\lambda(G) = \varDelta + 1$, and that if $\lambda(G) = \varDelta + 1$, then $N_G[v]$ contains at most two major vertices for any $v \in V(G)$. Furthermore, they showed that $\lambda(T)$ is either $\varDelta + 1$ or $\varDelta + 2$ for any tree T. By using this fact, Chang and Kuo [4] presented an $O(\varDelta^{4.5}n)$ time algorithm for computing $\lambda(T)$.

Chang and Kuo's Algorithm Now, we first review the idea of Chang and Kuo's dynamic programming algorithm (CK algorithm) for the $L(2,1)$-labeling problem of trees, since our linear time algorithm also depends on the same formula of the principle of optimality. The algorithm determines if $\lambda(T) = \varDelta + 1$, and if so, we can easily construct the labeling with $\lambda(T) = \varDelta + 1$.

To describe the idea, we introduce some notations. We assume for explanation that T is rooted at some leaf vertex r. 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 determined by $T(v)$. For $T(u, v)$, we define

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

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 condition that $f(u) = a$ and $f(v) = b$, i.e., the minimum k of k-$L(2,1)$-labelings on $T(u, v)$ satisfying $f(u) = a$ and $f(v) = b$. This δ function satisfies the following formula:

$$\delta((u,v),(a,b)) = \begin{cases} 1, & \text{if there is an injective assignment } g\colon C(v) \to \{0,1,\dots,\varDelta+1\} - \{a, \\ & \quad b-1, b, b+1\} \text{ such that } \delta((v,w),(b,g(w))) = 1 \text{ for each } w \in C(v), \\ 0, & \text{otherwise.} \end{cases}$$

The existence of such an injective assignment g is formalized as the maximum matching problem: For a bipartite graph $G(u, v, a, b) = (C(v), X, E(u, v, a, b))$, where $X = \{0, 1, \dots, \varDelta, \varDelta + 1\}$ and $E(u, v, a, b) = \{(w, c) \mid \delta((v, w), (b, c)) = 1, c \in X - \{a\}, w \in C(v)\}$, we can see that there is an injective assignment $g\colon C(v) \to \{0, 1, \dots, \varDelta+1\}-\{a, b-1, b, b+1\}$ if there exists a matching of size $d'(v)$ in $G(u, v, a, b)$. 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 δ function for $T(v, w), w \in C(v)$ and any two pairs of labels are given. Now let $t(v)$ be the time for calculating $\delta((u, v), (*, *))$ for vertex v. CK algorithm solves the bipartite matching problems of $O(\varDelta)$ vertices and $O(\varDelta^2)$ edges $O(\varDelta^2)$ times for each v, in order to obtain δ-values for all combinations of labels a and b. This amounts $t(v) = O(\varDelta^{2.5}) \times O(\varDelta^2) = O(\varDelta^{4.5})$, where the first $O(\varDelta^{2.5})$ is the time complexity of the bipartite matching problem [14]. Thus the total running time is $\sum_{v \in V} t(v) = O(\varDelta^{4.5}n)$.

An $O(n^{1.75})$-time Algorithm Next, we review the $O(n^{1.75})$-time algorithm proposed in [11]. The running time $O(n^{1.75})$ is roughly achieved by two strategies. One is that the problem can be solved by a simple linear time algorithm if $\varDelta = \varOmega(\sqrt{n})$, and the other is that it can be solved in $O(\varDelta^{1.5}n)$ time for any input tree.

The first idea of the speedup is that for computing $\delta((u, v), (*, b))$, the algorithm does not solve the bipartite matching problems every time from scratch, but reuse the obtained matching structure. More precisely, the bipartite matching problem is solved for $G(u, v, -, b) = (C(v), X, E(u, v, -, b))$ instead of $G(u, v, a, b)$ for a specific a, where

$E(u, v, -, b) = \{(w, c) \mid \delta((v, w), (b, c)) = 1, c \in X, w \in C(v)\}$. A maximum matching of $G(u, v, -, b)$ is observed to satisfy the following properties:

Property 1. If $G(u, v, -, b)$ has no matching of size $d'(v)$, then $\delta((u, v), (i, b)) = 0$ for any label i. □

Property 2. $\delta((u, v), (i, b)) = 1$ if and only if vertex i can be reached by an M-alternating path from some vertex in X unmatched by M in $G(u, v, -, b)$, where M denotes a maximum matching of $G(u, v, -, b)$ (of size $d'(v)$). □

From these properties, $\delta((u, v), (*, b))$ can be computed by a single bipartite matching and a single graph search, and its total running time is $O(\Delta^{1.5} d'(v)) + O(\Delta d'(v)) = O(\Delta^{1.5} d'(v))$ (for solving the bipartite matching of $G(u, v, -, b)$, which has $O(\Delta)$ vertices and $O(\Delta d'(v))$ edges, and for a single graph search). Since this calculation is done for all b, we have $t(v) = O(\Delta^{2.5} d'(v))$.

The other technique of the speedup introduced in [11] is based on preprocessing operations for amortized analysis. By some preprocessing operations, the shape of input trees can be restricted while preserving $L(2, 1)$-labeling number, and the input trees can be assumed to satisfy the following two properties.

Property 3. All vertices connected to a leaf vertex are major vertices. □

Property 4. The size of any path component of T is at most 3. □

Here, a sequence of vertices $v_1, v_2, \ldots, v_\ell$ is called 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 ℓ is called the *size* of the path component.

Furthermore, this preprocessing operations enable the following amortized analysis. Let V_L and V_Q be the set of leaf vertices and the set of major vertices whose children are all leaf vertices, respectively. Also, let $d''(v) = |C(v) - V_L|$ for $v \in V$. (Note that $d''(v) = 0$ for $v \in V_L \cup V_Q$.)

By Property 3, if we go down the resulting tree from a root, then we reach a major vertex in V_Q. Then, the following facts are observed: (i) for $v \in V_Q$ $\delta((u, v), (a, b)) = 1$ if and only if $b = 0$ or $\Delta + 1$ and $|a - b| \geq 2$, (ii) $|V_Q| \leq n/\Delta$. Note that (i) implies that it is not required to solve the bipartite matching to obtain δ-values. Also (ii) and Property 4 imply that $|V - V_Q - V_L| = O(n/\Delta)$ (this can be obtained by pruning leaf vertices and regarding V_Q vertices as new leaves). Since it is not necessary to compute bipartite matchings for $v \in V_L \cup V_Q$, and this implies that the total time to obtain δ-values for all v's is $\sum_{v \in V} t(v) = O(\sum_{v \in V - V_L - V_Q} t(v))$, which turned out to be $O(\Delta^{2.5} \sum_{v \in V - V_L - V_Q} d''(v))$. Since $\sum_{v \in V - V_L - V_Q} d''(v) = |V - V_L - V_Q| + |V_Q| - 1 = O(n/\Delta)$, we obtain $\sum_{v \in V - V_L - V_Q} t(v) = O(\Delta^{1.5} n)$. Since we have a linear time algorithm if $\Delta = \Omega(\sqrt{n})$ as mentioned above, we can solve the problem in $O(n^{1.75})$ time in total.

3 Label Compatibility and Flow-based Computation of δ

As reviewed in Subsection 2.2, one of keys of an efficient computation of δ-values is reusing the matching structures. In this section, for a further speedup of the computation of δ-values, we introduce a new novel notion, which we call ‘label compatibility’, that enables to treat several labels equivalently under the computation of δ-values. Then, the faster computation of δ-values is achieved on a maximum flow algorithm instead of a maximum matching algorithm. Seemingly, this sounds a bit strange, because the time complexity of the maximum flow problem is larger than the one of the bipartite matching problem. The trick is that the new flow-based computation uses a smaller network (graph) by this notion than the graph $G(u, v, -, b)$ used in the bipartite matching.

3.1 Label Compatibility and Neck/Head Levels

Let $L_h = \{h, h+1, \ldots, \varDelta - h, \varDelta - h + 1\}$. Let T be a tree rooted at v, and $u \notin V(T)$. We say that T is *head-L_h-compatible* if $\delta((u, v), (a, b)) = \delta((u, v), (a', b))$ for all $a, a' \in L_h$ and $b \in L_0$ with $|a - b| \geq 2$ and $|a' - b| \geq 2$. Analogously, we say that T is neck-L_h-compatible if $\delta((u, v), (a, b)) = \delta((u, v), (a, b'))$ for all $a \in L_0$ and $b, b' \in L_h$ with $|a - b| \geq 2$ and $|a - b'| \geq 2$. The neck and head levels of T are defined as follows:

Definition 1. *Let T be a tree rooted at v, and $u \notin V(T)$.*
(i) *The* neck level (*resp.,* head level) *of T is* 0 *if T is neck-L_0-compatible* (*resp., head-L_0-compatible*). (ii) *The* neck level (*resp.,* head level) *of T is h* (≥ 1) *if T is not neck-L_{h-1}-compatible* (*resp., head-L_{h-1}-compatible*) *but neck-L_h-compatible* (*resp., head-L_h-compatible*).

An intuitive explanation of neck-L_h-compatibility (resp., head-L_h-compatibility) of T is that if for $T(u, v)$, a label in L_h is assigned to v (resp., u) under $(\varDelta + 1)$-$L(2, 1)$-labeling of $T(u, v)$, the label can be replaced with another label in L_h without violating a proper $(\varDelta + 1)$-$L(2, 1)$-labeling; labels in L_h are compatible. The neck and head levels of T represent the bounds of L_h-compatibility of T. Thus, a trivial bound on neck and head levels is $(\varDelta + 1)/2$.

For the relationship between the neck/head levels and the tree size, we can show the following lemma, whose proof can be found in the technical report version [12]:

Lemma 1. *Let T' be a subtree of T. If $|V(T')| \leq (\varDelta - 3 - 2h)^{h/2} - 1$ and $\varDelta - 2h \geq 10$, then the head level and neck level of T' are both at most h.*

By this lemma, we obtain the following theorem:

Theorem 1. For a tree T, both the head and neck levels of T are $O(\log |V(T)| / \log \varDelta)$.

3.2 Flow-based Computation of δ

We are ready to explain the faster computation of δ-values. Recall that $\delta((u, v), (a, b)) = 1$ holds if there exists a matching of $G(u, v, a, b)$ in which all $C(v)$ vertices are just matched; which vertex is matched to a vertex in X does not matter. From this fact, we can treat vertices in X corresponding to L_h equally in computing δ, if T is neck- and head-L_h-compatible. The idea of the fast computation of δ-values is that, by bundling compatible vertices in X of G, we reduce the size of a graph (or a network) to compute the assignments of labels, which is no longer the maximum matching; the maximum flow.

The algorithm introduced in Subsection 2.2 computes δ-values not by solving the maximum matchings of $G(u, v, a, b)$ for all pairs of a and b but by finding a maximum matching M of $G(u, v, -, b)$ once and then searching M-alternating paths. In the new flow-based computation, we adopt the same strategy; for a tree $T(v)$ whose head and neck levels are at most $h(v)$, we do not prepare a network for a specific pair (a, b), say $\mathcal{N}(u, v, a, b)$, but a general network $\mathcal{N}(u, v, -, b) = (\{s, t\} \cup C(v) \cup X_{h(v)}, E(v) \cup E_X \cup E_\delta, cap)$, where $X_{h(v)} = (L_0 - L_{h(v)}) \cup \{h(v)\}$, $E(v) = \{(s, w) \mid w \in C(v)\}$, $E_X = \{(c, t) \mid c \in X_{h(v)}\}$, $E_\delta = \{(w, c) \mid w \in C(v), c \in X_{h(v)}\}$, and $cap(e)$ function is defined as follows: $\forall e \in E(v)$, $cap(e) = 1$, for $e = (w, c) \in E_\delta$, $cap(e) = 1$ if $\delta((v, w), (b, c)) = 1$, $cap(e) = 0$ otherwise, and for $e = (c, t) \in E_X$, $cap(e) = 1$ if $c \neq h(v)$, $cap(e) = |L_{h(v)} - \{b, b+1, b-1\}|$ if $c = h(v)$.

For a maximum flow $\psi : e \to R^+$, we define X' as $\{c \in X_h \mid cap((c, t)) - \psi((c, t)) \geq 1\}$. By the flow integrality and arguments similarly to Properties 1 and 2, we can obtain the following properties:

Lemma 2. *If $\mathcal{N}(u, v, -, b)$ has no flow of size $d'(v)$, then $\delta((u, v), (i, b)) = 0$ for any label i.* □

Lemma 3. *$\delta((u, v), (i, b)) = 1$ if and only if vertex i can be reached by a ψ-alternating path from some vertex in X' in $\mathcal{N}(u, v, -, b)$.* □

Here, a ψ-alternating path is defined as follows: Given a flow ψ, a path in E_δ is called *ψ-alternating* if its edges alternately satisfy $cap(e) - \psi(e) \geq 1$ and $\psi(e) \geq 1$. By these lemmas, we can obtain $\delta((u, v), (*, b))$-values for b by solving the maximum flow of $\mathcal{N}(u, v, -, b)$ once and then applying a single graph search.

The current fastest maximum flow algorithm runs in $O(\min\{m^{1/2}, n^{2/3}\}\, m \log(n^2/m) \log U) = O(n^{2/3} m \log n \log U)$ time, where U, n and m are the maximum capacity of edges, the number of vertices and edges, respectively [9]. Thus the running time of calculating $\delta((u, v), (a, b))$ for a pair (a, b) is

$$O((h(v) + d''(v))^{2/3}(h(v)d''(v)) \log(h(v) + d''(v)) \log \Delta) = O(\Delta^{2/3}(h(v)d''(v)) \log^2 \Delta),$$

since $h(v) \leq \Delta$ and $d''(v) \leq \Delta$ (recall that $d''(v) = |C(v) - V_L|$). By using a similar technique of updating matching structures introduced in [11], we can obtain $\delta((u, v), (*, b))$ in $O(\Delta^{2/3}(h(v)d''(v)) \log^2 \Delta) + O(h(v)d''(v)) = O(\Delta^{2/3}(h(v)d''(v)) \log^2 \Delta)$ time. Since the number of candidates for b is also bounded by $h(v)$ from the neck/head level property, we have the following lemma.

Lemma 4. *$\delta((u, v), (*, *))$ can be computed in $O(\Delta^{2/3}(h(v))^2 d''(v) \log^2 \Delta)$ time, that is, $t(v) = O(\Delta^{2/3}(h(v))^2\, d''(v) \log^2 \Delta)$.* □

Combining this with $\sum_{v \in V - V_L - V_Q} d''(v) = O(n/\Delta)$ shown in Subsection 2.2, we can show the total running time for the $L(2, 1)$-labeling is $O(n(\max\{h(v)\})^2(\Delta^{-1/3} \log^2 \Delta))$. By applying Theorem 1, we have the following theorem:

Theorem 2. *For trees, the $L(2, 1)$-labeling problem can be solved in $O(\min\{n \log^2 n, \Delta^{1.5} n\})$ time. Furthermore, if $n = O(\Delta^{poly(\log \Delta)})$, it can be solved in $O(n)$ time.* □

Corollary 1. *For a vertex v in a tree T, we have $\sum_{w \in V(T(v))} t(w) = O(|T(v)|)$ if $|T(v)| = O(\Delta^{poly(\log \Delta)})$.* □

Only by directly applying Theorem 1 (actually Lemma 1), we obtain much faster running time than the previous one. In the following section, we present a linear time algorithm, in which Lemma 1 is used in a different way.

4 Proof of Linear Running Time

As mentioned in Subsection 2.2, one of keys for achieving the running time $\mathrm{O}(\varDelta^{1.5}n) = \mathrm{O}(n^{1.75})$ is equation $\sum_{v\in V_\delta} d''(v) = \mathrm{O}(n/\varDelta)$, where V_δ is the set of vertices in which δ-values should be computed via the matching-based algorithm; since the computation of δ-values for each v is done in $\mathrm{O}(\varDelta^{2.5}d''(v))$ time, it takes $\sum_{v\in V_\delta} \mathrm{O}(\varDelta^{2.5}d''(v)) = \mathrm{O}(\varDelta^{1.5}n)$ time in total. This equation is derived from the fact that in leaf vertices we do not need to solve the matching to compute δ-values, and any vertex with height 1 has $\varDelta - 1$ leaves as its children after the preprocessing operation.

In our new algorithm, we generalize this idea: By replacing leaf vertices with subtrees with size at least $\varDelta^4$ in the above argument, we can obtain $\sum_{v\in V_\delta} d''(v) = \mathrm{O}(n/\varDelta^4)$, and in total, the running time $\sum_{v\in V_\delta} \mathrm{O}(\varDelta^{2.5}d''(v)) = \mathrm{O}(n)$ is roughly achieved. Actually, this argument contains a cheating, because a subtree with size at most $\varDelta^4$ is not always connected to a major vertex, whereas a leaf is, which is well utilized to obtain $\sum_{v\in V_\delta} d''(v) = \mathrm{O}(n/\varDelta)$. Also, whereas we can neglect leaves to compute δ-values, we cannot neglect such subtrees. We resolve these problems by best utilizing the properties of neck/head levels and the maximum flow techniques introduced in Section 3.

4.1 Efficient Assignment of Labels for Computing δ

In this section, by compiling observations and techniques for assigning labels in the computation of $\delta((u, v), (*, *))$ for $v \in V$, given in Sections 2 and 3, we will design an algorithm to run in linear time within the DP framework. Throughout this section, we assume that an input tree T satisfies Properties 3 and 4. Below, we first partition the vertex set V into five types of subsets defined later, and give a linear time algorithm for computing the value of δ functions, specified for each type.

We here start with defining such five types of subsets V_i ($i = 1, \dots, 5$). Throughout this section, for a tree T', we may denote $|V(T')|$ simply by $|T'|$. Let V_M be the set of vertices $v \in V$ such that $T(v)$ is a "maximal" subtree of T with $|T(v)| \le \varDelta^5$; i.e., for the parent u of v, $|T(u)| > \varDelta^5$. Divide V_M into two sets $V_M^{(1)} := \{v \in V_M \mid |T(v)| \ge (\varDelta - 19)^4\}$ and $V_M^{(2)} := \{v \in V_M \mid |T(v)| < (\varDelta - 19)^4\}$ (notice that $V_L \subseteq \cup_{v\in V_M} V(T(v))$). Define $\tilde{d}(v) := |C(v) - V_M^{(2)}|$ $(= d'(v) - |C(v) \cap V_M^{(2)}|)$. Let

$$\begin{aligned}
V_1 &:= \cup_{v\in V_M} V(T(v)),\\
V_2 &:= \{v \in V - V_1 \mid \tilde{d}(v) \ge 2\},\\
V_3 &:= \{v \in V - V_1 \mid \tilde{d}(v) = 1, C(v) \cap (V_M^{(2)} - V_L) = \emptyset\},\\
V_4 &:= \{v \in V - (V_1 \cup V_3) \mid \tilde{d}(v) = 1, \textstyle\sum_{w\in C(v)\cap(V_M^{(2)}-V_L)} |T(w)| \le \varDelta(\varDelta - 19)\},\\
V_5 &:= \{v \in V - (V_1 \cup V_3) \mid \tilde{d}(v) = 1, \textstyle\sum_{w\in C(v)\cap(V_M^{(2)}-V_L)} |T(w)| > \varDelta(\varDelta - 19)\}.
\end{aligned}$$

Notice that $V = V_1 \cup V_2 \cup V_3 \cup V_4 \cup V_5$, and $V_i \cap V_j = \emptyset$ for each i, j with $i \neq j$ (see Figure 1).

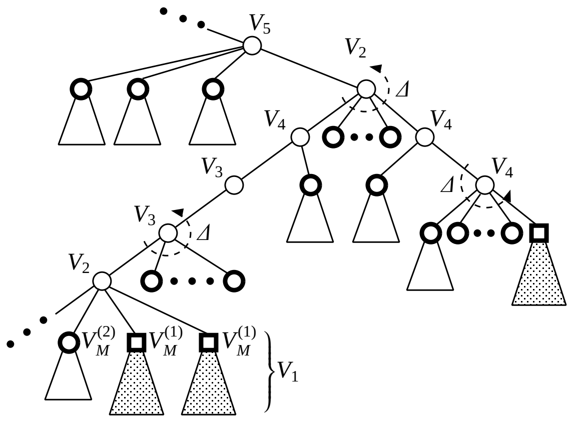

Fig. 1. Partition of V into V_i's ($i = 1, \ldots, 5$). Bold circles are leaves (V_L) or pseudo-leaves ($V_M^{(2)} - V_L$) with their subtrees, while bold squares are vertices in $V_M^{(1)}$ with their subtrees.

Here we describe an outline of the algorithm for computing $\delta((u,v),(*,*))$, $v \in V$, named Compute-$\delta(v)$ (Algorithm 1), which can be regarded as a subroutine of the DP framework. Below, we show that for each V_i, $\delta((u,v),(*,*))$, $v \in V_i$ can be computed in linear time in total; i.e., $O(\sum_{v \in V_i} t(v)) = O(n)$. Namely, we have the following theorem.

Theorem 3. *For trees, the* $L(2,1)$*-labeling problem can be solved in linear time.*

Algorithm 1 Compute-$\delta(v)$

```
1: /** Assume that the head and neck levels of $T(v)$ are at most $h$. **/
2: If $v \in V_1 \cup V_2$, then for each $b \in (L_0 - L_h) \cup \{h\}$, compute $\delta((u,v),(*,b))$ by the max-flow
   computation in the network $\mathcal{N}(u,v,-,b)$ defined in Subsection 3.2.
3: If $v \in V_3$, execute the following procedure for each $b \in L_0$ in the case of $C(v) \cap V_L = \emptyset$, and
   for each $b \in \{0, \varDelta + 1\}$ in the case of $C(v) \cap V_L \neq \emptyset$.
   /** Let $w^*$ denote the unique child of $v$ not in $V_M^{(2)}$.**/
  3-1: If $|\{c \mid \delta((v,w^*),(b,c)) = 1\}| \geq 2$, then let $\delta((u,v),(*,b)) := 1$.
  3-2: If $\{c \mid \delta((v,w^*),(b,c)) = 1\} = \{c^*\}$, then let $\delta((u,v),(c^*,b)) := 0$ and $\delta((u,v),(a,b)) := 1$ for
   all other labels $a \notin \{b-1, b, b+1\}$.
  3-3: If $|\{c \mid \delta((v,w^*),(b,c)) = 1\}| = 0$, then let $\delta((u,v),(*,b)) := 0$.
4: If $v \in V_4 \cup V_5$, then similarly to the case of $v \in V_1 \cup V_2$, compute $\delta((u,v),(*,*))$ by the
   max-flow computation in a network such as $\mathcal{N}(u,v,-,b)$ specified for this case (details will
   be described in Subsection 4.3).
```

We first show $O(\sum_{v \in V_1} t(v)) = O(|V_1|)$. For each $v \in V_M$, we have $O(\sum_{w \in V(T(v))} t(w)) = O(|T(v)|)$, by Corollary 1 and $|T(v)| = O(\varDelta^5)$. Hence, we have $O(\sum_{v \in V_1} t(v)) = O(\sum_{v \in V_M} \sum_{w \in V(T(v))} t(w)) = O(\sum_{v \in V_M} |T(v)|) = O(|V_1|)$.

The sketch of proofs for V_2, V_3, V_4 and V_5 are given in the subsequent subsections, where some proofs of lemmas are omitted. See [12] for details.

4.2 Computation of δ-value for V_2

By Lemma 4, we can see that $\sum_{v\in V_2} t(v) = \mathrm{O}(\sum_{v\in V_2} \Delta^{2/3} d'(v) h^2 \log^2 \Delta) = \mathrm{O}(\Delta^{8/3} \log^2 \Delta \sum_{v\in V_2} d'(v))$ (note that $h \le \Delta$ and $d''(v) \le d'(v)$). Now, we have $d'(v) \le \tilde{d}(v) + \Delta \le \Delta\tilde{d}(v)$. It follows that $\sum_{v\in V_2} t(v) = \mathrm{O}(\Delta^{11/3} \log^2 \Delta \sum_{v\in V_2} \tilde{d}(v))$. Below, in order to show that $\sum_{v\in V_2} t(v) = \mathrm{O}(n)$, we prove that $\sum_{v\in V_2} \tilde{d}(v) = \mathrm{O}(n/\Delta^4)$.

By definition, there is no vertex whose all children are vertices in $V_M^{(2)}$, since if there is such a vertex v, then for each $w \in C(v)$, we have $|T(w)| < (\Delta - 19)^4$ and hence $|T(v)| < \Delta^5$, which contradicts the maximality of $T(w)$. It follows that in the tree T' obtained from T by deleting all vertices in $V_1 - V_M^{(1)}$, each leaf vertex belongs to $V_M^{(1)}$ (note that $V(T') = V_M^{(1)} \cup V_2 \cup V_3 \cup V_4 \cup V_5$). Hence,

$$\begin{aligned}
|V(T')| - 1 = |E(T')| &= \tfrac{1}{2} \textstyle\sum_{v\in V(T')} d_{T'}(v) \\
&= \tfrac{1}{2}(|V_M^{(1)}| + \textstyle\sum_{v\in V_2\cup V_3\cup V_4\cup V_5}(\tilde{d}(v) + 1) - 1) \\
&= \tfrac{1}{2}(|V_M^{(1)}| + \textstyle\sum_{v\in V_2}(\tilde{d}(v) + 1) + 2|V_3| + 2|V_4| + 2|V_5| - 1) \\
&\ge \tfrac{1}{2}|V_M^{(1)}| + \tfrac{3}{2}|V_2| + |V_3| + |V_4| + |V_5| - \tfrac{1}{2}
\end{aligned}$$

(the last inequality follows from $\tilde{d}(v) \ge 2$ for all $v \in V_2$). Thus, $|V_M^{(1)}| - 1 \ge |V_2|$. Therefore, we can observe that $\sum_{v\in V_2} \tilde{d}(v) = |E(T')| - |V_3| - |V_4| - |V_5| = |V_M^{(1)}| + |V_2| - 1 \le 2|V_M^{(1)}| - 2$ (the first equality follows from $|E(T')| = \sum_{v\in V_2\cup V_3\cup V_4\cup V_5} \tilde{d}(v) = \sum_{v\in V_2} \tilde{d}(v) + |V_3| + |V_4| + |V_5|$ and the second equality follows from $|E(T')| = |V(T')| - 1 = |V_M^{(1)}| + |V_2| + |V_3| + |V_4| + |V_5| - 1$). It follows by $|V_M^{(1)}| = \mathrm{O}(n/\Delta^4)$ that $\sum_{v\in V_2} \tilde{d}(v) = \mathrm{O}(n/\Delta^4)$.

4.3 Computation of δ-value for V_3, V_4, and V_5

We sketch proofs for V_3, V_4, and V_5. Since Property 3 indicates that $|T(w)| \ge \Delta$ for each $w \in V_M - V_L$ (resp., $\sum_{w\in C(v)\cap(V_M^{(2)} - V_L)} |T(w)| > \Delta(\Delta - 19)$), we have $|V_4| = \mathrm{O}(n/\Delta)$ (resp., $|V_5| = \mathrm{O}(n/\Delta^2)$). By Property 4, we can observe that $|V_3| = \mathrm{O}(n/\Delta)$. Hence, it suffices to show that for each $v \in V_3 \cup V_4$ (resp., V_5), $\delta((u, v), (*, *))$ can be computed in $\mathrm{O}(\Delta)$ (resp., $\mathrm{O}(\Delta^2)$) time. Now,

$$\text{the head and neck levels of } T(w) \text{ are at most 8 for each } w \in V_M^{(2)} \tag{1}$$

by Lemma 1 and $|T(w)| < (\Delta - 19)^4$ (note that we assume that $\Delta \ge 26$, since otherwise the original CK algorithm is already a linear time algorithm). Let w^* be the unique child of v in $C(v) - V_M^{(2)}$.

First consider the case where $v \in V_3$ (i.e., Step 3 in algorithm Compute-$\delta(v)$). Let b be a label such that $b \in L_0$ if $v \in V_3^{(1)} := \{v \in V_3 \mid C(v) \cap V_L = \emptyset\}$, and $b \in \{0, \Delta + 1\}$ if $v \in V_3^{(2)} := V_3 - V_3^{(1)}$. Notice that if $v \in V_3^{(2)}$ (i.e., $C(v) \cap V_L \ne \emptyset$), then by Property 3, v is major and hence $\delta((u, v), (a, b)) = 1$, $a \in L_0$ indicates that $b = 0$ or $b = \Delta + 1$. Observe that if there is a label $c \in L_0 - \{b - 1, b, b + 1\}$ such that $\delta((v, w^*), (b, c)) = 1$, then for all $a \in L_0 - \{b - 1, b, b + 1, c\}$, we have $\delta((u, v), (a, b)) = 1$. It is not difficult to see that this shows the correctness of the procedure in this case. Obviously, for each $v \in V_3$, we can check which case of 3-1, 3-2, or 3-3 in algorithm Compute-$\delta(v)$ holds, and determine the values of $\delta((u, v), (*, b))$, in $\mathrm{O}(1)$ time. Therefore, the values of $\delta((u, v), (*, *))$ can be determined in $\mathrm{O}(\Delta)$ time.

Next consider the case where $v \in V_4$. For a label b, we divide $C(v) \cap (V_M^{(2)} - V_L)$ into two subsets $C_1(b) := \{w \in C(v) \cap (V_M^{(2)} - V_L) \mid \delta((v,w),(b,c)) = 1$ for all $c \in L_8 - \{b-1, b, b+1\}\}$ and $C_2(b) := \{w \in C(v) \cap (V_M^{(2)} - V_L) \mid \delta((v,w),(b,c)) = 0$ for all $c \in L_8 - \{b-1, b, b+1\}\}$. By the following property, we only have to consider the assignments for $\{w^*\} \cup C_2(b)$.

Lemma 5. *Let $v \in V_4$ and a and b be labels with $|b - a| \geq 2$ such that $b \in L_0$ if $C(v) \cap V_L = \emptyset$ and $b \in \{0, \varDelta + 1\}$ otherwise. Then, $\delta((u,v),(a,b)) = 1$ if and only if there exists an injective assignment $g : \{w^*\} \cup C_2(b) \to L_0 - \{a, b-1, b, b+1\}$ such that $\delta((v,w),(b,g(w))) = 1$ for each $w \in \{w^*\} \cup C_2(b)$.*

Below, we will show how to compute $\delta((u,v),(*,b))$ in O(1) time for a fixed b, where $b \in L_0$ if $C(v) \cap V_L = \emptyset$ and $b \in \{0, \varDelta + 1\}$ otherwise. If $|C_2(b)| \geq 17$, then $\delta((u,v),(*, b)) = 0$ because in this case, there exists some $w \in C_2(b)$ to which no label in $L_0 - L_8$ can be assigned since $|L_0 - L_8| = 16$. Assume that $|C_2(b)| \leq 16$. There are the following three possible cases: (Case-1) $\delta((v,w^*),(b,c_i)) = 1$ for at least two labels $c_1, c_2 \in L_8$, (Case-2) $\delta((v,w^*),(b,c_1)) = 1$, for exactly one label $c_1 \in L_8$, and (Case-3) otherwise.

(Case-1) By assumption, for any a, $\delta((v,w^*),(b,c)) = 1$ for some $c \in L_8 - \{a\}$. By Lemma 5, we only have to check whether there exists an injective assignment $g : C_2(b) \to L_0 - L_8 - \{a, b-1, b, b+1\}$ such that $\delta((v,w),(b,g(w))) = 1$ for each $w \in C_2(b)$. According to Subsection 3.2, this can be done by utilizing the maximum flow computation on the subgraph $\mathcal{N}'$ of $\mathcal{N}(u,v,-,b)$ induced by $\{s,t\} \cup C_2(b) \cup X'$ where $X' = \{0, 1, \ldots, 7, \varDelta-6, \varDelta-5, \ldots, \varDelta+1\}$. Obviously, the size of $\mathcal{N}'$ is O(1) and it follows that its time complexity is O(1).

(Case-2) For all $a \neq c_1$, the value of $\delta((u,v),(a,b))$ can be computed similarly to Case-1. Consider the case where $a = c_1$. In this case, if $\delta((v,w^*),(b,c)) = 1$ holds, then it turns out that $c \in L_0 - L_8$. Hence, by Lemma 5, it suffices to check whether there exists an injective assignment $g : \{w^*\} \cup C_2(b) \to L_0 - L_8 - \{b-1, b, b+1\}$ such that $\delta((v,w),(b,g(w))) = 1$ for each $w \in \{w^*\} \cup C_2(b)$. Similarly to Case-1, this can be done in O(1) time, by utilizing the subgraph $\mathcal{N}''$ of $\mathcal{N}(u,v,-,b)$ induced by $\{s,t\} \cup (C_2(b) \cup \{w^*\}) \cup X'$.

(Case-3) By assumption, if $\delta((v,w^*),(b,c)) = 1$ holds, then it turns out that $c \in L_0 - L_8$. Similarly to the case of $a = c_1$ in Case-2, by using $\mathcal{N}''$, we can compute the values of $\delta((u,v),(*,b))$ in O(1) time.

We analyze the time complexity for computing $\delta((u,v), (*,*))$. It is dominated by that for computing $C_1(b)$, $C_2(b)$, and $\delta((u,v),(*,b))$ for each $b \in L_0$. By (1), we have $C_i(b) = C_i(b')$ for all $b, b' \in L_8$ and $i = 1, 2$. It follows that the computation of $C_1(b)$ and $C_2(b)$, $b \in L_0$ can be done in $\mathrm{O}(|C(v) \cap (V_M^{(2)} - V_L)|)$ time. On the other hand, the values of $\delta((u,v),(*,b))$ can be computed in constant time in each case of Cases-1, 2 and 3 for a fixed b. Thus, $\delta((u,v),(*,*))$ can be computed in $\mathrm{O}(\varDelta)$ time.

Finally, we consider the case where $v \in V_5$. We will prove that the values of $\delta((u,v),(*,b))$ can be computed in $\mathrm{O}(\varDelta)$ time for a fixed b. A key is that the children $w \in C(v) \cap V_M^{(2)}$ of v can be classified into 2^{17} (= O(1)) types, depending on its δ-values $(\delta((v,w),(b,i)) \mid i \in (L_0 - L_8) \cup \{\tilde{c}_8\})$ where $\tilde{c}_8$ is some label in $L_8 - \{b-1, b, b+1\}$, since by (1), $\delta((v,w),(b,c)) = \delta((v,w),(b,\tilde{c}_8))$ for any $c \in L_8 - \{b-1, b, b+1\}$. Then, we can construct in $O(d'(v))$ time a network $\mathcal{N}'(u,v,a,b)$ with O(1) vertices, O(1) edges, and

$O(\varDelta)$ units of capacity from $\mathcal{N}(u, v, a, b)$ by letting $X_h := X_8$ and replacing $C(v)$ with a set of 2^{17} vertices corresponding to types of vertices in $C(v) \cap V_M^{(2)}$, and compute in $O(\log \varDelta)$ time the values of $\delta((u, v), (a, b))$ by applying the maximum flow techniques to $\mathcal{N}'(u, v, a, b)$ (see [12] for the details about $\mathcal{N}'(u, v, a, b)$). Furthermore, by the following lemma, we can see that $\delta((u, v), (*, b))$ can be obtained by checking $\delta((u, v), (a, b))$ for $O(1)$ candidates of a; $\delta((u, v), (*, b))$ can be obtained in $O(\varDelta)$ time.

Lemma 6. *If $\delta((u, v), (a_1, b)) \neq \delta((u, v), (a_2, b))$ for some $a_1, a_2 \in L_8 - \{b - 1, b, b + 1\}$ (say, $\delta((u, v), (a_1, b)) = 1$), then we have $\delta((v, w^*), (b, a_2)) = 1$ and $\delta((v, w^*), (b, a)) = 0$ for all $a \in L_8 - \{a_2, b - 1, b, b + 1\}$, and moreover, $\delta((u, v), (a, b)) = 1$ for all $a \in L_8 - \{a_2, b - 1, b, b + 1\}$.*

References

1. H. L. Bodlaender, T. Kloks, R. B. Tan and J. van Leeuwen. Approximations for λ-coloring of graphs. *The Computer Journal* 47, 193–204 (2004).
2. T. Calamoneri. The $L(h, k)$-labelling problem: A survey and annotated bibliography. *The Computer Journal* 49, 585–608 (2006). (`http://www.dsi.uniroma1.it/~calamo/PDF-FILES/survey.pdf`, ver. Jan. 13, 2009.)
3. G. J. Chang, W.-T. Ke, D. Kuo, D. D.-F. Liu and R. K. Yeh. On $L(d, 1)$-labeling of graphs. *Discr. Math.* 220, 57–66 (2000).
4. G. J. Chang and D. Kuo. The $L(2, 1)$-labeling problem on graphs. *SIAM J. Discr. Math.* 9, 309–316 (1996).
5. J. Fiala, P. A. Golovach and J. Kratochvíl. Distance constrained labelings of graphs of bounded treewidth. *Proc. 32th ICALP*, 360–372 (2005).
6. J. Fiala, P. A. Golovach and J. Kratochvíl. Distance constrained labelings of trees, *Proc. 5th TAMC*, 125–135 (2008).
7. J. Fiala, P. A. Golovach and J. Kratochvíl. Computational complexity of the distance constrained labeling problem for trees, *Proc. 35th ICALP*, Part I, 294–305 (2008).
8. J. Fiala, T. Kloks and J. Kratochvíl. Fixed-parameter complexity of λ-labelings. *Discr. Appl. Math.* 113, 59–72 (2001).
9. A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM* 45, pp. 783–797 (1998).
10. J. R. Griggs and R. K. Yeh. Labelling graphs with a condition at distance 2. *SIAM J. Disc. Math.* 5, 586–595 (1992).
11. T. Hasunuma, T. Ishii, H. Ono and Y. Uno. An $O(n^{1.75})$ algorithm for $L(2, 1)$-labeling of trees. *Proc. 11th SWAT*, 185–197 (2008). (Journal version to appear in *Theoretical Comp. Sci.*, `doi:10.1016/j.tcs.2009.04.025`.)
12. T. Hasunuma, T. Ishii, H. Ono and Y. Uno. A linear time algorithm for $L(2, 1)$-labeling of trees. CoRR abs/0810.0906: (2008).
13. F. Havet, B. Reed and J.-S. Sereni. $L(2, 1)$-labelling of graphs. *Proc. 19th SIAM-SODA*, 621–630 (2008).
14. J. E. Hopcroft, R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* 2, 225–231 (1973).
15. J. Kratochvíl, D. Kratsch and M. Liedloff. Exact algorithms for $L(2, 1)$-labeling of graphs. *Proc. 32nd MFCS*, 513–524 (2007).
16. W.-F. Wang. The $L(2, 1)$-labelling of trees. *Discr. Appl. Math.* 154, 598–603 (2006).
17. R. K. Yeh. A survey on labeling graphs with a condition at distance two. *Discr. Math.* 306, 1217–1231 (2006).