A Linear Time Algorithm for Computing Longest Paths in 2-Trees

Minko Markov* Tzvetalin S. Vassilev†
Krassimir Manev‡

Abstract

We propose a practical linear time algorithm for the LONGEST PATH problem on 2-trees.

Keywords: *algorithmic graph theory, longest path, treewidth, 2-tree*

1 Introduction

To compute the length of a path of maximum length in an undirected graph is a problem that arises naturally. The computational problem LONGEST PATH is formulated both on unweighted and weighted graphs. Both versions are known to be $\mathcal{NP}$-complete [8]. It is known LONGEST PATH is fixed parameter tractable [11]. Recent research shows there is a subexponential in the parameter algorithm if certain restrictions are imposed on the graph [6]. From the approximation perspective, the problem is not approximable in polynomial time within a multiplicative constant unless $\mathcal{P} = \mathcal{NP}$ [9]. The approximation algorithm with best approximation ratio so far has approximation ratio that is, asymptotically, close to linear [2]. Other relevant results are [1], [14], [7], and [13].

A possible way to tackle with $\mathcal{NP}$-completeness on graphs is to construct polynomial time algorithms on restricted graph classes. It is known there is an $O(k!\,2^k\,n)$ algorithm for graphs of treewidth $\leq k$ [3]. Theoretically speaking, that means a linear time algorithm for LONGEST PATH

*Department of Computing Systems, Faculty of Mathematics and Informatics, "St. Kliment Ohridski" University of Sofia, 5 J. Bourchier Blvd, P.O. Box 48, BG-1164 Sofia, Bulgaria. Email: minkom@fmi.uni-sofia.bg

†Department of Computer Science and Mathematics, Nipissing University, 100 College Drive, Box 5002 North Bay, Ontario P1B 8L7, Canada. Email: tzvetalv@nipissingu.ca

‡Department of Computing Systems, Faculty of Mathematics and Informatics, "St. Kliment Ohridski" University of Sofia, 5 J. Bourchier Blvd, P.O. Box 48, BG-1164 Sofia, Bulgaria. Email: manev@fmi.uni-sofia.bg

on 2-trees has been known since the 90's. However, the said algorithm belongs to a class of algorithms that are extremely impractical, being designed and verified from very general, highly theoretical considerations such as the theory of Robertson and Seymour of graph minor testing. The algorithms based on minor testing have colossal hidden constants and are therefore "linear time" in purely abstract sense. Furthermore, the algorithm for LONGEST PATH in [3] is not written down in pseudocode and is not verified in details. The said algorithm works on non-weighted graphs only. It does not construct a longest path and the author makes no claim the algorithm can be modified so that a longest path is output as well (besides the numerical answer).

A practical linear time algorithm for LONGEST PATH on edge weighted trees was constructed by Dijkstra around 1960 (see [5] for description and formal verification). Uehara and Uno [12] designed polynomial time algorithms for the LONGEST PATH problem on cacti and block graphs, and showed it can be solved efficiently on graphs with interval representation. A practical linear time algorithm for LONGEST PATH on weighted cactus graphs was designed by Andreica and Ţăpuş, and independently by Markov and Manev (see [10] for details).

In this paper we present a practical linear time algorithm that solves LONGEST PATH on 2-trees. It is trivial to modify the algorithm to output a longest path as well without violating the linear time complexity. Furthermore, it is easy to modify the algorithm to work on weighted 2-trees. Our algorithm assumes the 2-tree is rooted at an edge and then works bottom-up in the following manner. With every edge we associate a list of constant length of natural numbers called label. One of those numbers is the length of a longest path in the 2-tree rooted below the egde. At every edge further up we compute its label only from the labels of the sub 2-trees below.

2 Background

We consider undirected graphs without multiple edges or self loops. Let $G = (V, E)$ be a graph. To *delete* a vertex u from G means to transform G into $G' = (V \setminus u, E \setminus \{e \in E \mid u \in e\})$. We denote the vertex deletion by $G' = G - u$. To remove an edge e from G means to transform G into $G'' = (V, E \setminus \{e\})$. We use the minus sign to denote the edge removal as well: $G'' = G - e$. If the vertex set of a graph G is not named explicitly we denote it by $V(G)$. Likewise, $E(G)$ is the edge set. K_n is the complete graph with n vertices.

A path in G is a sequence $p = u_1, e_1, u_2, e_2, \ldots, e_{n-1}, u_n$, for some $n \geq 1$, of alternating distinct vertices $u_1, u_2, \ldots, u_n$ and edges $e_1, e_2, \ldots, e_{n-1}$ such that for $1 \leq i < n$, $e_i = (u_i, u_{i+1})$. u_1 and u_n are called *the endpoints of*

p, and the remaining vertices are *the internal vertices of* p. We abuse the set-theoretical notation "$\in$" by writing "$u_1 \in p$" and "$e_1 \in p$" to denote the facts that u_1 and e_1, respectively, are in p, *etc.* If $n \geq 2$, by $p - u_1$ we denote the path $u_2, e_2, \ldots, e_{n-1}, u_n$. *The length* of a path p is the number of edges in it and is denoted by $|p|$. *A subpath* of p is a contiguous subsequence of p that is a path. Suppose $q_1, q_2, \ldots, q_s$ are paths in G. We say that they *cover* p in that order if:

- q_i is a subpath of p for $1 \leq i \leq s$, and
- one endpoint of q_1 coincides with one endpoint of p, the other endpoint of q_1 coincides with with one endpoint of q_2, the other endpoint of q_2 coincides with one endpoint of q_3, *etc.*, the other endpoint of q_{s-1} coincides with one endpoint of q_s, the other endpoint of q_s coincides with the other endpoint of p.

We denote the fact that $q_1, q_2, \ldots, q_s$ cover p in that order by writing $p = q_1 \oplus q_2 \oplus \ldots \oplus q_s$. Clearly, $|p| = \sum_{i=1}^{s} |q_i|$. When the covering of p is understood, the paths that take part in it—in the current naming scheme these are $q_1, q_2, \ldots, q_s$—are called *the constituents of* p.

Notation 1. *Let G be a graph. Let u and v be any distinct vertices in G.*

- *"u-path" means path with one endpoint u.*
- *"u-to-v-path" means path with endpoints u and v.*
- *"u-in-v-path" means path with one endpoint u in which v is an internal vertex.*
- *"in-u-in-v-path" means path in which u and v are internal vertices.*
- *"u-not-v-path" means path with one endpoint u and not containing v.*
- *For any two paths p and q in G, "$p \perp q$" is an abbreviation for "p and q are vertex disjoint".*
- *"max path" is an abbreviation for "path of maximum length", "max u-path" is an abbreviation for "u-path of maximum length",* etc.
- *"$\langle p, q \rangle$ are max-sum $\langle u, v \rangle$-paths" means p and q are paths in G such that:*
  - *p is a u-path, q is a v-path, $p \perp q$, and*
  - *$|p| + |q|$ is maximum over all such pairs of paths.*

  The angle brackets in the latter notation denote ordered pairs.

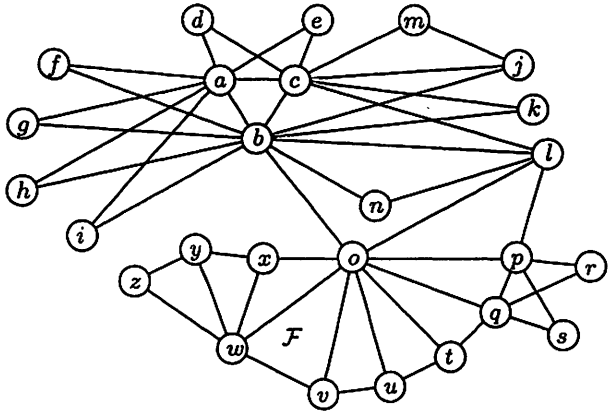


Figure 1: An example of a 2-tree.

Suppose G is a connected graph. *A separator* of G is any nonempty vertex set $U \subset V$, such that $V \backslash U$ can be partitioned into two nonempty sets X and Y, such that any path in G with one endpoint from X and the other one, from Y, contains a vertex from U. *To split* a separator U means the following. Let G' be the subgraph of G induced by U. We first delete U from G, which clearly leads to the appearance of some connected components H_1, $H_2, \ldots, H_k$ for some $k \geq 2$, and transform H_i into $\big(V(H_i) \cup U, E(H_i) \cup E(G') \cup E_i''\big)$, where $E_i'' = \{(u,v) \in E(G) \,|\, u \in V(H_i) \text{ and } v \in U\}$. We denote the splitting operation by $G \ominus U$. In case $U = \{u\}$ we may write $G \ominus u$.

Definition 1 (2-tree). *A graph is a 2-tree if and only if it can be constructed according to the following rules.*

1. *K_2 is a 2-tree.*
2. *If G' is a 2-tree, $e = (v_i, v_j)$ is any edge in $E(G')$, and u is a vertex not in $V(G')$, then $G = (V(G') \cup \{u\}, E(G') \cup \{(u, v_i), (u, v_j)\})$ is a 2-tree.* □

Figure 1 shows a 2-tree. Clearly, it can be constructed according to Definition 1: start with the edge (a, b) and then add the remaining vertices in the alphabetical order. Let G be any 2-tree. If G has precisely two vertices we say it is *trivial*, otherwise it is *non-trivial*. The edges of G are classified into *peripheral edges* and *interior edges* as follows. If G is K_2 or K_3 then all its edges are peripheral edges. Otherwise, G is obtained from some smaller

2-tree G' as in Definition 1. Using the naming convention of the Definition, the edge $e = (v_i, v_j)$ becomes interior regardless of its status before, and the newly added edges (u, v_i) and (u, v_j) become peripheral edges. For example, in the 2-tree on Figure 1, the edges (a, b), (c, j), and (w, y) are interior, while (a, d), (b, n), and (y, z) are peripheral.

Let G be a non-trivial 2-tree. We call the induced K_3's of G, *the faces of* G. Every face is identified by its three vertices, *e.g.* $\mathcal{F} = (o, v, w)$ on Figure 1. Clearly, every peripheral edge in G belongs to precisely one face, and for every interior edge $e = (v_i, v_j)$, $\{v_i, v_j\}$ is a separator of G whose splitting results in the appearance of at least two 2-trees. We say shortly we split e, rather than its vertices, and write $G \ominus e$. We emphasise $G \ominus e$ is a set of non-trivial 2-trees.

A rooted 2-tree G is 2-tree in which one edge is chosen to be special and is called *the root.* We denote the fact that e is the root of G by writing $e = \mathrm{root}(G)$. If the root is a peripheral edge we say G is *simple*, and if G is non-trivial as well we define its *root face* and that is the face that contains the root. If the root is an interior edge we do not define any root face and we say the rooted tree is *complex.* In the latter case, $G \ominus e$ is a set of rooted simple non-trivial 2-trees, each one with root e. We call those, *the folios of* G. Clearly, the following inductive definition of rooted two tree is equivalent to the one we just mentioned.

Let G be a rooted simple non-trivial 2-tree with root (u, v) and root face $\mathcal{F} = (u, v, w)$. Clearly, $(G - e) \ominus w$ consists of two connected components which we call, *the branches of* G. Each of them is considered to be a rooted 2-tree with root the edge from $\mathcal{F}$ that is in it. Figure 2 shows a simple rooted 2-tree and a complex one schematically; the former one (Figure 2(a)) in terms of its branches and the latter one (Figure 2(b)), in terms of its folios.

It follows in every rooted 2-tree, every interior edge has one or more simple 2-trees rooted at it, and every edge is the root of some rooted 2-tree. *The leaves* of any rooted non-trivial 2-tree are the peripheral edges that are distinct from the root. The leaf of any rooted trivial 2-tree is its only edge, *i.e.* the root.

Definition 2. *Let G be any rooted 2-tree and $(u, v) = root(G)$. Then,*

$$\begin{aligned}
\mathsf{L}_G &= \{p \mid p \textit{ is a max path in } G\} \\
\mathsf{L}_G(u \cdots v) &= \{p \mid p \textit{ is a max } u\textit{-to-}v\textit{-path in } G\} \\
\mathsf{L}_G(u \cdots v \cdots) &= \{p \mid p \textit{ is a max } u\textit{-in-}v\textit{-path in } G\} \\
\mathsf{L}_G(u, \neg v) &= \{p \mid p \textit{ is a max } u\textit{-not-}v\textit{-path in } G\} \\
\mathsf{L}_G(u \perp v) &= \{\langle p, q\rangle \mid \langle p, q\rangle \textit{ are max-sum } \langle u, v\rangle\textit{-paths}\}.
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\ell_G &= |p|, \text{ for any } p \in \mathsf{L}_G \\
\ell_G(u\cdots v) &= |p|, \text{ for any } p \in \mathsf{L}_G(u\cdots v) \\
\ell_G(u\cdots v\cdots) &= |p|, \text{ for any } p \in \mathsf{L}_G(u\cdots v\cdots) \\
\ell_G(u, \neg v) &= |p|, \text{ for any } p \in \mathsf{L}_G(u, \neg v) \\
\ell_G(u \perp v) &= |p| + |q|, \text{ for any } \langle p, q\rangle \in \mathsf{L}_G(u \perp v).
\end{aligned}$$ □

Definition 3 (label). *Let G be a rooted 2-tree with root $e = (u, v)$.* The label of e *is the ordered septuple $\widetilde{\lambda}(e) = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)$, where:*

$$\begin{aligned}
\lambda_1 &= \ell_G, \\
\lambda_2 &= \ell_G(u\cdots v), \\
\lambda_3 &= \ell_G(u\cdots v\cdots), \\
\lambda_4 &= \ell_G(u, \neg v), \\
\lambda_5 &= \ell_G(v\cdots u\cdots), \\
\lambda_6 &= \ell_G(v, \neg u), \\
\lambda_7 &= \ell_G(u \perp v)
\end{aligned}$$ □

The graphs we consider are not oriented, therefore both (u, v) and (v, u) are equivalent desctiptions of the same edge. However, some of the elements of $\widetilde{\lambda}(e)$, namely λ_3, λ_4, λ_5, and λ_6, are not invariant under interchanging u and v. For instance, it may be the case that $\ell_G(u\cdots v\cdots) \neq \ell_G(v\cdots u\cdots)$, *etc.* We resolve that issue by postulating the following. If the root is described as (u, v) then the elements of the label are constructed according to that description and the mentioned definition. It follows that if the root of the same 2-tree was described as (v, u) the label would be $(\lambda_1, \lambda_2, \lambda_5, \lambda_6, \lambda_3, \lambda_4, \lambda_7)$. In other words, we impose an order on the two vertices in the root edge when defining the label.

Every edge in a rooted 2-tree can be assigned a label because, as we observed, every edge is the root of some rooted 2-tree. Obviously, the label of any leaf is $(1, 1, 0, 0, 0, 0, 0)$. We place a tilde sign above the names of ordered tuples, *e.g.* $\widetilde{\lambda}(e)$ or $\widetilde{\lambda}$. The same name but with an index i, $1 \leq i \leq 7$, and without the tilde above it, refers to the i-th elements of the tuple, *e.g.* $\lambda_2(e)$ is the second element of $\widetilde{\lambda}(e)$ and λ_2 is the second element of $\widetilde{\lambda}$, *etc.*

3 Our algorithm, its verification and time complexity analysis

3.1 The algorithm

LONGEST PATH(G: rooted 2-tree, e: the root of G)
1 ▷ Computes the length of a longest path
2 $\widetilde{\lambda} \leftarrow$ COMPUTE LABEL(G, e)
3 **return** λ_1

COMPUTE LABEL(G: rooted 2-tree, e: the root)
1 ▷ Computes the label of the root
2 **if** G is trivial
3 $\quad \widetilde{\lambda} \leftarrow (1,1,0,0,0,0,0)$
4 **else**
5 $\quad$ let $e = (u, v)$
6 $\quad$ let $G \ominus e$ be $\{G^1, G^2, \ldots, G^k\}$
7 $\quad$ **for** $i \leftarrow 1$ **to** k **do**
8 $\quad\quad$ let the root face of G^i be (u, v, w_i)
9 $\quad\quad$ let $(G^i - e) \ominus w_i$ be $\{H^1, H^2\}$ where $u \in H^1$ and $v \in H^2$
10 $\quad\quad \widetilde{\mu}^1 \leftarrow$ COMPUTE LABEL(H^1, (u, w_i))
11 $\quad\quad \widetilde{\mu}^2 \leftarrow$ COMPUTE LABEL(H^2, (w_i, v))
12 $\quad\quad \widetilde{\nu}^i \leftarrow$ COMBINE ON FACE($\widetilde{\mu}^1$, $\widetilde{\mu}^2$)
13 $\quad \widetilde{\lambda} \leftarrow$ COMBINE ON EDGE($\widetilde{\nu}^1$, $\widetilde{\nu}^2$, $\ldots$, $\widetilde{\nu}^k$)
14 **return** $\widetilde{\lambda}$

COMBINE ON EDGE($\widetilde{\lambda^1}$, $\widetilde{\lambda^2}$, $\ldots$, $\widetilde{\lambda^k}$: label)
1 ▷ Computes $\widetilde{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)$, the label of rooted
2 ▷ non-trivial 2-tree whose folios have labels $\widetilde{\lambda^1}$, $\widetilde{\lambda^2}$, $\ldots$, $\widetilde{\lambda^k}$.
3 **if** $k = 1$
4 $\quad \widetilde{\lambda} \leftarrow \widetilde{\lambda^1}$
5 **else**
6 $\quad x \leftarrow \max\{\lambda_7^i \mid 1 \le i \le k\}$
7 $\quad y \leftarrow \max\{\lambda_4^i + \lambda_6^j \mid 1 \le i, j \le k, i \ne j\}$
8 $\quad \lambda_7 \leftarrow \max\{x, y\}$
9 $\quad \lambda_2 \leftarrow \max\{\lambda_2^i \mid 1 \le i \le k\}$
10 $\quad \lambda_4 \leftarrow \max\{\lambda_4^i \mid 1 \le i \le k\}$
11 $\quad \lambda_6 \leftarrow \max\{\lambda_6^i \mid 1 \le i \le k\}$
12 $\quad x \leftarrow \max\{\lambda_3^i \mid 1 \le i \le k\}$

13 $y \leftarrow \max\{\lambda_2^i + \lambda_6^j \mid 1 \le i,j \le k, i \ne j\}$
14 $\lambda_3 \leftarrow \max\{x, y\}$
15 $x \leftarrow \max\{\lambda_5^i \mid 1 \le i \le k\}$
16 $y \leftarrow \max\{\lambda_2^i + \lambda_4^j \mid 1 \le i,j \le k, i \ne j\}$
17 $\lambda_5 \leftarrow \max\{x, y\}$
18 $x \leftarrow \max\{\lambda_1^i \mid 1 \le i \le k\}$
19 $x_1 \leftarrow \max\{\lambda_3^i + \lambda_4^j \mid 1 \le i,j \le k, i \ne j\}$
20 $x_2 \leftarrow \max\{\lambda_5^i + \lambda_6^j \mid 1 \le i,j \le k, i \ne j\}$
21 $x_3 \leftarrow \max\{\lambda_2^i + \lambda_7^j \mid 1 \le i,j \le k, i \ne j\}$
22 $y_1 \leftarrow \max\{\lambda_4^i + \lambda_4^j \mid 1 \le i,j \le k, i \ne j\}$
23 $y_2 \leftarrow \max\{\lambda_6^i + \lambda_6^j \mid 1 \le i,j \le k, i \ne j\}$
24 $z \leftarrow 0$
25 **if** $k \ge 3$
26 $\quad z \leftarrow \max\{\lambda_2^i + \lambda_4^j + \lambda_6^t \mid 1 \le i,j,t \le k, i \ne j \ne t \ne i\}$
27 $\lambda_1 \leftarrow \max\{x, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, x_1, x_2, x_3, y_1, y_2, z\}$
28 $\widetilde{\lambda} \leftarrow (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)$
29 **return** $\widetilde{\lambda}$

Let ψ be a function of several nonnegative variables, one of which is x. The function positive(ψ, x) is defined as follows:

$$\text{positive}(\psi, x) = \begin{cases} 0, & \text{if } x = 0 \\ \psi, & \text{else} \end{cases}$$

COMBINE ON FACE($\widetilde{\lambda^1}$, $\widetilde{\lambda^2}$: label)
1 ▷ Computes $\widetilde{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)$, the label of a rooted
2 ▷ simple non-trivial 2-tree with root (u, v) whose branches have
3 ▷ roots (u, w) and (w, v) and labels $\widetilde{\lambda^1}$ and $\widetilde{\lambda^2}$, respectively.
4 $\lambda_2 \leftarrow \lambda_2^1 + \lambda_2^2$
5 $x_1 \leftarrow \text{positive}(1 + \lambda_6^2, \lambda_6^2)$
6 $x_2 \leftarrow \text{positive}(\lambda_2^1 + \lambda_3^2, \lambda_3^2)$
7 $x_3 \leftarrow \text{positive}(1 + \lambda_5^2, \lambda_5^2)$
8 $x_4 \leftarrow \text{positive}(1 + \lambda_2^2 + \lambda_6^1, \lambda_6^1)$
9 $\lambda_3 \leftarrow \max\{x_1, x_2, 1 + \lambda_2^2, x_3, x_4\}$
10 $\lambda_4 \leftarrow \max\{\lambda_3^1, \lambda_4^1, \lambda_2^1 + \lambda_4^2\}$
11 $x_1 \leftarrow \text{positive}(1 + \lambda_4^1, \lambda_4^1)$
12 $x_2 \leftarrow \text{positive}(\lambda_2^2 + \lambda_5^1, \lambda_5^1)$
13 $x_3 \leftarrow \text{positive}(1 + \lambda_3^1, \lambda_3^1)$
14 $x_4 \leftarrow \text{positive}(1 + \lambda_2^1 + \lambda_4^2, \lambda_4^2)$
15 $\lambda_5 \leftarrow \max\{x_1, x_2, 1 + \lambda_2^1, x_3, x_4\}$
16 $\lambda_6 \leftarrow \max\{\lambda_5^2, \lambda_6^2, \lambda_2^2 + \lambda_6^1\}$

17 $\lambda_7 \leftarrow \max\{\lambda_4^1 + \lambda_6^2, \lambda_2^1 + \lambda_6^2, \lambda_3^1 + \lambda_6^2, \lambda_2^1 + \lambda_7^2, \lambda_4^1 + \lambda_2^2, \lambda_4^1 + \lambda_5^2, \lambda_7^1 + \lambda_2^2\}$
18 $x_1 \leftarrow \lambda_6^1 + \lambda_4^2$
19 $x_2 \leftarrow \lambda_5^1 + \lambda_4^2$
20 $x_3 \leftarrow \lambda_6^1 + \lambda_3^2$
21 $x_4 \leftarrow \lambda_5^1 + \lambda_3^2$
22 $\lambda_1 \leftarrow \max\{\lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_1^1, \lambda_1^2, x_1, x_2, x_3, x_4, \lambda_7 + 1\}$
23 $\widetilde{\lambda} \leftarrow (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7)$
24 **return** $\widetilde{\lambda}$

3.2 Verification

Lemma 1. *Algorithm* COMBINE ON FACE *is correct under the assumption that* $\widetilde{\lambda^1}$ *and* $\widetilde{\lambda^2}$ *are the correct labels of the branches.*

Proof: Assume G is a simple nontrivial rooted 2-tree as shown on Figure 2(a) on page 6, the edge (u, v) is called e, and $\widetilde{\lambda^1}$ and $\widetilde{\lambda^2}$ are the labels of G_1 and G_2, respectively. We prove the correctness of the assignments to λ_i in the order they appear in the algorithm. Let q be the path $q = u, v$.

Consider any $p \in \mathsf{L}_G(u \cdots v)$. p is covered by two paths such that one is in $\mathsf{L}_{G_1}(u \cdots w)$ and the other one, in $\mathsf{L}_{G_2}(w \cdots v)$. To see why, assume $w \notin p$. Then p must coincide with q, so $|p| = 1$. But there is a length 2, u-to-v-path in G, namely u, w, v. It follows $w \in p$. Further, w is an internal vertex in p, therefore $e \notin p$. Now it is obvious there exist paths p' and q', such that $p = p' \oplus q'$ and p' is a u-to-w-path in G_1 and q' is a w-to-v-path in G_2. Furthermore, $p' \in \mathsf{L}_{G_1}(u \cdots w)$ and $q' \in \mathsf{L}_{G_2}(w \cdots v)$ because $|p| = |p'| + |q'|$ and $|p'|$ and $|q'|$ are maximised independently. By the inductive assumption, $\lambda_2^1 = |p'|$ and $\lambda_2^2 = |q'|$. So the assignment at line 4 is correct.

Now we argue about the assignment at line 9. Consider any $p \in \mathsf{L}_G(u \cdots v \cdots)$. The following five subcases are exhaustive .

1. $w \notin p$. Then $p = q \oplus p'$ for some path p' in G_2. Clearly, $p' \in \mathsf{L}_{G_2}(v, \neg w)$, so $|p'| = \ell_{G_2}(v, \neg w)$. Recall that $\lambda_6^2 = \ell_{G_2}(w, \neg v)$. By the inductive assumptions, $\lambda_6^2 = \ell_{G_2}(v, \neg w)$, therefore $|p| = 1 + \lambda_6^2$.

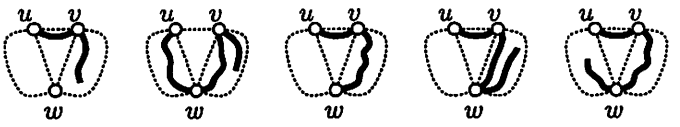
2. $w \in p$ and $(u, v) \notin p$. It must be the case that $p = p' \oplus q'$, such that $p' \in \mathsf{L}_{G_1}(u \cdots w)$ and $q' \in \mathsf{L}_{G_2}(w \cdots v \cdots)$. So, $|p| = \lambda_2^1 + \lambda_3^2$.

3. $w \in p$, $(u, v) \in p$, and w is an endpoint of p. Then $|p| = 1 + \lambda_2^2$.

4. $w \in p$, $(u, v) \in p$, w is an internal vertex in p, and the endpoint of p that is not u is in G_2. Then $|p| = 1 + \lambda_5^2$.

5. $w \in p$, $(u, v) \in p$, w is an internal vertex in p, and the endpoint of p that is not u is in G_1. Then $|p| = 1 + \lambda_2^2 + \lambda_6^1$.

At line 9 we assign to λ_3 the maximum of those five quantities. Figure 3 illustrates the said five subcases in order and for each subcase, how $|p|$ is computed from the applicable λ_j^i's.



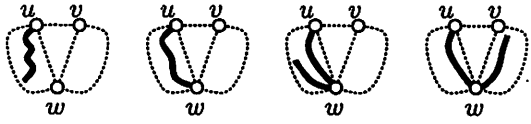
(a) $1+\lambda_6^2$ (b) $\lambda_2^1+\lambda_3^2$ (c) $1+\lambda_2^2$ (d) $1+\lambda_5^2$ (e) $1+\lambda_2^2+\lambda_6^1$

Figure 3: The five subcases when $p \in \mathsf{L}_G(u \cdots v \cdots)$ (line 9 of COMBINE ON FACE).

Note that the right-hand sides of lines 5–8 are wrapped in the function positive( ). The reason is that we want to discard one or more subcases among 1, 2, 4, and 5, if one constituent of p is a zero length path. For instance, consider subcase 2, illustrated on Figure 3(b). Assume G_2 is trivial. We must discard subcase 2 because v cannot be an internal vertex in p when G_2 is trivial and w is between u and v, which is implied by subcase 2. Discarding subcase 2 is equivalent to assigning $x_2 \leftarrow 0$ at line 6. Suppose we do not use the function positive( ) and perform the direct assignment $x_2 \leftarrow \lambda_2^1+\lambda_3^2$ at line 6. What will hapen is $x_2 \leftarrow \lambda_2^1$ since $\lambda_3^2 = 0$. However, λ_2^1 can be arbitrarily large so we may assign incorrectly a nonzero value to x_2. Likewise, if G_2 is trivial then we have to discard subcase 1 (see Figure 3(a)) by assigning $x_1 \leftarrow 0$ rather than $x_1 \leftarrow 1+0$, which is what will happen unless positive( ) is used at line 5. And so on.

Now we argue about the assignment at line 10. Consider any $p \in \mathsf{L}_G(u, \neg v)$. If $w \notin p$ then $p \in \mathsf{L}_{G_1}(u, \neg w)$, therefore $|p| = \lambda_4^1$. Suppose $w \in p$. If w is an endpoint of p then $p \in \mathsf{L}_{G_1}(u \cdots w)$, therefore $|p| = \lambda_2^1$. Suppose w is an internal vertex in p. Let the endpoint of p that is not u be z. If $z \in G_1$ then $p \in \mathsf{L}_{G_1}(u \cdots w \cdots)$, therefore $|p| = \lambda_3^1$. If $z \in G_2$ then p is covered by two paths $p_1 \in \mathsf{L}_{G_1}(u \cdots w)$ and $p_2 \in \mathsf{L}_{G_2}(w, \neg v)$ therefore $|p| = \lambda_2^1+\lambda_4^2$. Those four cases yield only three possibilities because $\lambda_2^1+\lambda_4^2 \geq \lambda_2^1$ and consequently it suffices to compute the maximum of three, not four, quantities. Figure 4 illustrates the mentioned four cases for p with respect to line 10 and for each possibility, how $|p|$ is computed from the applicable λ_j^i's.

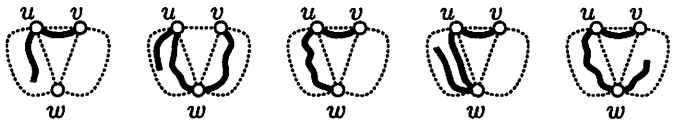
Now we argue about the assignment at line 15. Consider any $p \in \mathsf{L}_G(v \cdots u \cdots)$. The argument is identical to the argument about the correctness of the assignment at line 9 after the following change of names: swap G_1 with G_2, swap u with v, swap λ_2^1 with λ_2^2, substitute λ_3^2 with λ_5^1, substitute λ_5^2 with



(a) λ_4^1 (b) λ_2^1 (c) λ_3^1 (d) $\lambda_2^1+\lambda_4^2$

Figure 4: The four subcases when $p \in \mathsf{L}_G(u, \neg v)$ (line 10 of COMBINE ON FACE). To compute $|p|$ we consider only 4(a), 4(c), and 4(d).

λ_3^1, substitute λ_6^2 with λ_4^1, and substitute λ_6^1 with λ_4^2. Figure 5 illustrates the five distinct possibilities for p with respect to line 15. Subfigures 5(a)–5(e) correspond to subfigures 3(a)–3(e) under the said name swaps and substitutions.

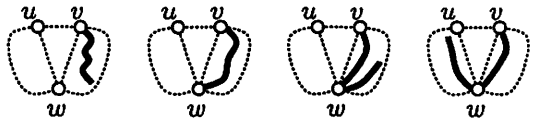


(a) $1+\lambda_4^1$ (b) $\lambda_5^1+\lambda_2^2$ (c) $1+\lambda_2^1$ (d) $1+\lambda_3^1$ (e) $1+\lambda_2^1+\lambda_4^2$

Figure 5: The five subcases when $p \in \mathsf{L}_G(v \cdots u \cdots)$ (line 15 of COMBINE ON FACE).

Now we argue about the assignment at line 16. Conside any $p \in \mathsf{L}_G(v, \neg u)$. The argument is identical to the argument about the correctness of the assignment at line 10 after the following change of names: swap G_1 with G_2, swap u with v, substitute λ_3^1 with λ_5^2, substitute λ_4^1 with λ_6^2, substitute λ_2^1 with λ_2^2, and substitute λ_4^2 with λ_6^1. Figure 6 illustrates the four cases for p with respect to line 16 and for each possibility, how $|p|$ is computed from the applicable λ_j^i's.

Now we argue about the assignment at line 17. Consider any $\langle p_1, p_2 \rangle \in \mathsf{L}_G(u \perp v)$. Note that $e \notin p_1$ and $e \notin p_2$. Furthermore, it cannot be the case that simultaneously $w \in p_1$ and $w \in p_2$. Furthermore, $w \notin p_1$ implies $p_1 \in \mathsf{L}_{G_1}(u, \neg w)$ and $w \notin p_2$ implies $p_2 \in \mathsf{L}_{G_2}(w, \neg v)$. So, if $w \notin p_1$ and $w \notin p_2$ then $|p_1| + |p_2| = \lambda_4^1 + \lambda_6^2$. Suppose $w \in p_1$. If w is an endpoint of p_1 then $|p_1| + |p_2| = \lambda_2^1 + \lambda_6^2$. Suppose w is an internal vertex in p_1. Let the endpoint of p_1 that is not u be z. If $z \in G_1$ then $|p_1| + |p_2| = \lambda_3^1 + \lambda_6^2$. If $z \in G_2$ then p_1 is covered by two paths p_1' and p_1'', such that p_1 is in G_1 and with endpoints u and w, and p_1'' is in G_2 with one endpoint w and is vertex

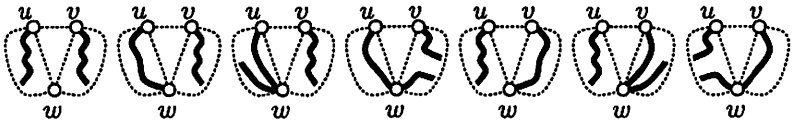


(a) λ_4^1 (b) λ_2^1 (c) λ_3^1 (d) $\lambda_2^1+\lambda_4^2$

Figure 6: The four cases for $p \in \mathsf{L}_G(v, \neg u)$ (line 16 of COMBINE ON FACE). To compute $|p|$ we consider only 6(a), 6(c), and 6(d).

disjoint with p_2. Since $|p_1| + |p_2| = |p_1'| + |p_1''| + |p_2|$ and because of the way these three paths are situated in G_1 and G_2, it is clear that the sum is maximised when $|p_1'|$, on one hand, and $|p_1''| + |p_2|$, on the other hand, are maximised independently. In other words, when $p_1' \in \mathsf{L}_{G_1}(u \cdots w)$ and $\langle p_1'', p_2 \rangle \in \mathsf{L}_{G_2}(w \perp v)$. Therefore, $|p_1| + |p_2| = \lambda_2^1 + \lambda_7^2$.

If $w \in p_2$ we use completely analogous arguments to prove that $|p_1| + |p_2| \in \{\lambda_4^1 + \lambda_2^2, \lambda_4^1 + \lambda_5^2, \lambda_7^1 + \lambda_2^2\}$. Figure 7 illustrates the seven possibilities for p_1 and p_2, and how $|p_1| + |p_2|$ is computed for each possibility.



(a) $\lambda_4^1+\lambda_6^2$ (b) $\lambda_2^1+\lambda_6^2$ (c) $\lambda_3^1+\lambda_6^2$ (d) $\lambda_2^1+\lambda_7^2$ (e) $\lambda_4^1+\lambda_2^2$ (f) $\lambda_4^1+\lambda_5^2$ (g) $\lambda_7^1+\lambda_2^2$

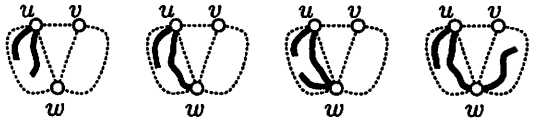
Figure 7: The seven possibilities for $\langle p_1, p_2 \rangle \in \mathsf{L}_G(u \perp v)$ (line 17 of COMBINE ON FACE).

Now we argue about the assignment at line 22. Consider any longest path p in G. Each of u and v can be, independently from the other one, an endpoint, an internal vertex, or not be at all, in p. There are nine possibilities altogether and we have already covered five of them, in that order: p being u-to-v-path, u-in-v-path, u-not-v-path, v-in-u-path, v-not-u-path. The results are stored in λ_2, λ_3, λ_4, λ_5, λ_6, respectively. The other four possibilities do not have corresponding elements in $\widetilde{\lambda}$ and they are covered in the remainder of the proof.

First suppose $u, v \notin p$. If $w \notin p$ either, then p lies either entirely in G_1, or in G_2. Assume p is in G_1. Obviously, p is a longest path in G_1, so by the inductive hypothesis $|p| = \lambda_1^1$. Likewise, if p is in G_2 then $|p| = \lambda_1^2$. If $w \in p$ it must be the case that $|p| = \lambda_6^1 + \lambda_4^2$.

Now suppose p contains u as an internal vertex and does not contain v. Clearly, it is not possible both endpoints of p to be in G_2. The following four possibilities are exhaustive for this subcase. They are illustrated on Figure 8

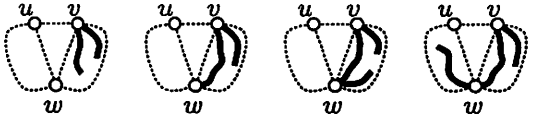
- Both endpoints of p are in G_1 and $w \notin p$. Then $|p| = \lambda_1^1$. See Figure 8(a).
- Both endpoints of p are in G_1 and one of them is w. Then $|p| = \lambda_1^1$. See Figure 8(b).
- Both endpoints of p are in G_1, none of them is w, but $w \in p$. Then $|p| = \lambda_1^1$. See Figure 8(c).
- One endpoint of p is in G_2. Then p is covered by two paths p_1 and p_2 such that $p_1 \in \mathsf{L}_{G_1}(w \cdots u \cdots)$ and $p_2 \in \mathsf{L}_{G_2}(w, \neg v)$. Consequently, $|p| = \lambda_5^1 + \lambda_4^2$. See Figure 8(d).



(a) λ_1^1 (b) λ_1^1 (c) λ_1^1 (d) $\lambda_5^1+\lambda_4^2$

Figure 8: The four cases when p contains u as internal vertex and does not contain v.

Now suppose p contains v as an internal vertex and does not contain u. Clearly, this is a mirror case of the former one. Figure 9 depicts the analogous four possible subcases and in each one says how $|p|$ is computed.



(a) λ_2^2 (b) λ_2^2 (c) λ_2^2 (d) $\lambda_6^1+\lambda_3^2$

Figure 9: The four cases when p contains v as internal vertex and does not contain u.

The only remaining case to consider is that p contains u and v as internal vertices. If $e \notin p$ then p has w as an internal vertex between u and v and consequently p is covered by two paths p_1 and p_2, such that p_1 is in G_1 and has u as internal vertex and w as an endpoint, and p_2 is in G_2 and has v as internal vertex and w as an endpoint. Clearly, $p_1 \in \mathsf{L}_{G_1}(w \cdots u \cdots)$ and $p_2 \in \mathsf{L}_{G_2}(w \cdots v \cdots)$, therefore $|p| = \lambda_5^1 + \lambda_3^2$. Suppose $e \in p$. Suppose we remove e from p, without removing u or v. Clearly, this edge removal leads to the appearance of two paths p_1 and p_2, such that p_1 is a u-path, p_2 is a v-path, $p_1 \perp p_2$, and $|p_1| + |p_2|$ is maximum over all such paths. Then $|p| = |p_1| + |p_2| + 1$. Note that $\langle p_1, p_2 \rangle \in \mathsf{L}_G(u \perp v)$, therefore $|p| = 1 + \lambda_7$. □

Lemma 2. *Algorithm* COMBINE ON EDGE *is correct under the assumption that* $\widetilde{\lambda^1}$, $\widetilde{\lambda^2}$, ..., $\widetilde{\lambda^k}$ *are the correct labels of the folios.*

Proof: Assume G is a rooted 2-tree with folios G_1, G_2, ..., G_k as shown on Figure 2(b) on page 6. Let $\widetilde{\lambda^i}$ be the label of G_i, for $1 \leq i \leq k$. If $k = 1$ then G is the same graph as G_1, so the assignment at line 4 is correct. Assume $k \geq 2$. We prove the correctness of the assignments of values to λ_t in the order they appear in the algorithm. Consider any path p in G such that $u \in p$ or $v \in p$. Let $V' = V(p) \setminus \{u, v\}$. Assume $V' \neq \emptyset$. For any i such that $1 \leq i \leq k$, we say that p is *mostly in* G_i if $V' \subset V(G_i)$. For any i and j such that $1 \leq i \leq k$ and $1 \leq j \leq k$ and $i \neq j$, p *is mostly in* G_i *and* G_j if $V' \cap V(G_i) \neq \emptyset$ and $V' \cap V(G_j) \neq \emptyset$ and $V' \subset V(G_i) \cup V(G_j)$. For any i and j and t such that $1 \leq i \leq k$ and $1 \leq j \leq k$ and $1 \leq t \leq k$ and $i \neq j \neq t \neq i$, p *is mostly in* G_i *and* G_j *and* G_t if $V' \cap V(G_i) \neq \emptyset$ and $V' \cap V(G_j) \neq \emptyset$ and $V' \cap V(G_t) \neq \emptyset$ and $V' \subset V(G_i) \cup V(G_j) \cup V(G_t)$.

Consider any $\langle p_1, p_2 \rangle \in \mathsf{L}_G(u \perp v)$. Either both of them are mostly in the same folio, or not. The former means $\langle p_1, p_2 \rangle \in \mathsf{L}_{G_i}(u \perp v)$ for some $i \in \{1, 2, \ldots, k\}$. The latter means $p_1 \in \mathsf{L}_{G_i}(u, \neg v)$ and $p_2 \in \mathsf{L}_{G_j}(v, \neg u)$ for some $i, j \in \{1, 2, \ldots, k\}$ such that $i \neq j$. Clearly, the assignment at line 6 is corresponds to the former possibility and the one at line 7, to the latter one. Ir follows the assignment at line 8 is correct. The two said possibilities are illustrated schematically on Figure 10.

The verification of lines 9, 10, and 11 is straightforward. Any path from $\mathsf{L}_G(u \cdots v)$ has to be mostly in a single folio, therefore

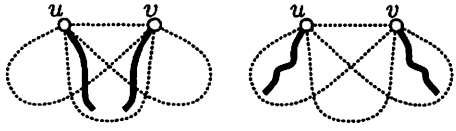
$$\ell_G(u \cdots v) = \max \{\ell_{G_i}(u \cdots v) \mid 1 \leq i \leq k\}$$

Likewise,

$$\ell_G(u, \neg v) = \max \{\ell_{G_i}(u, \neg v) \mid 1 \leq i \leq k\}$$

and

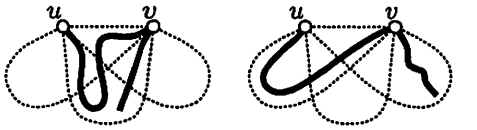
$$\ell_G(v, \neg u) = \max \{\ell_{G_i}(v, \neg u) \mid 1 \leq i \leq k\}$$



(a) $\max\{\lambda_7^i\}$ (b) $\max_{i \neq j}\{\lambda_4^i + \lambda_6^j\}$

Figure 10: The two possibilities for $\langle p_1, p_2 \rangle \in \mathsf{L}_G(u \perp v)$ with respect to line 8 of COMBINE ON EDGE.

Consider any $p \in \mathsf{L}_G(u \cdots v \cdots)$. Clearly, p is covered by two paths p_1 and p_2 such that p_1 has endpoints u and v, and p_2 has one endpoint v. Either both p_1 and p_2 are mostly in the same folio, or not. The former means there is one folio such that p is mostly in it, *i. e.* $p \in \mathsf{L}_{G_i}(u \cdots v \cdots)$ for some $i \in \{1, 2, \ldots, k\}$. The latter means p is mostly in precisely two folios, that is, $p_1 \in \mathsf{L}_{G_i}(u \cdots v)$ and $p_2 \in \mathsf{L}_{G_j}(v, \neg u)$ for some $i, j \in \{1, 2, \ldots, k\}$ such that $i \neq j$. The assignment at line 12 is with respect to the former possibility tje one at line 13, with respect to the latter one. It follows the assignment at line 14 is correct. The two said possibilities are illustrated schematically on Figure 11.



(a) $\max\{\lambda_3^i\}$ (b) $\max_{i \neq j}\{\lambda_2^i + \lambda_6^j\}$

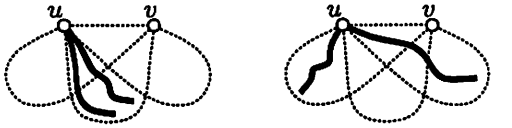
Figure 11: The two possibilities for $p \in \mathsf{L}_G(u \cdots v \cdots)$ with respect to line 14 of COMBINE ON EDGE.

The argument about the correctness of the assignment at line 17 is completely analogous to the argument about line 14.

Now we argue about the assignment at line 27. Consider any longest path p in G. Each of u and v can be, independently from the other one, an endpoint, an internal vertex, or not be at all, in p. There are nine possibilities altogether and we have already covered five of them, in that order: p being u-to-v-path, u-not-v-path, v-in-u-path, u-in-v-path, v-not-u-path. The results are stored in λ_2, λ_4, λ_6, λ_3, λ_5, respectively. The other four possibilities do not have corresponding elements in $\widetilde{\lambda}$ and they are covered in the remainder of the proof.

First suppose $u, v \notin p$. Then $|p| = \max\{\lambda_1^i \mid 1 \leq i \leq k\}$. Clearly, this case is taken care of by the assignment at line 18. Now suppose p contains u as an internal vertex and does not contain v. We consider the following two subcases. Figure 12 illustrates them.

- p is mostly in a single folio. Then $|p| = \max\{\lambda_1^i\}$ where $1 \leq i \leq k$. See Figure 12(a). This subcase is also taken care of by the assignment at line 18.
- p is covered by two paths p_1 and p_2 such that p_1 is mostly in some folio G_i and p_2 is mostly in some folio G_j where $i \neq j$. In this subcase, $|p| = \max\{\lambda_4^i + \lambda_4^j \mid 1 \leq i \leq k, 1 \leq j \leq k, i \neq j\}$, which we compute at line 22. See Figure 12(b).



(a) $\max\{\lambda_1^i\}$ (b) $\max_{i \neq j}\{\lambda_4^i + \lambda_4^j\}$

Figure 12: The two cases when p contains u as internal vertex and does not contain v. In each case, p is drawn with thick line.

Now suppose p contains v as an internal vertex and does not contain u. Clearly, this is a mirror case of the former one and is divided into analogous subcases. In the first one, $|p| = \max\{\lambda_1^i \mid 1 \leq i \leq k\}$ as before. In the second one, $|p| = \max\{\lambda_6^i + \lambda_6^j\}$ where $1 \leq i \leq k, 1 \leq j \leq k, i \neq j$, which we compute at line 23.

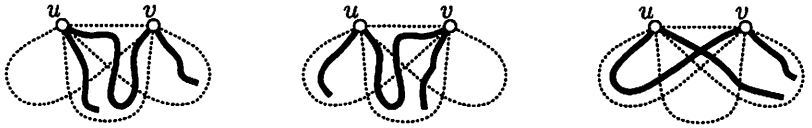
The only remaining case to consider is the one in which p contains both u and v as internal vertices. Clearly, p is covered by three paths p_1, p_2, and p_3, such that p_2 is a u-to-v-path, p_1 is a u-path and p_3 is a v-path.

First suppose that p is mostly in a single folio. Then $|p| = \max\{\lambda_1^i \mid 1 \leq i \leq k\}$, which possibility is covered by the assignment at line 18. Now suppose that p is mostly in exactly two folios G_i and G_j, which includes the possibility that the edge (u, v) is in p. The following subcases, illustrated on Figure 13, are exhaustive.

- For some folio G_i, either p_1 and p_2 are mostly in G_i (Figure 13(a)) or p_2 and p_3 are mostly in G_i (Figure 13(b)). In the former subcase, $p_1 \oplus p_2 \in \mathsf{L}_{G_i}(v \cdots u \cdots)$ and $p_3 \in \mathsf{L}_{G_j}(v, \neg u)$, so $|p| = \max\{\lambda_5^i + \lambda_6^j \mid 1 \leq i \leq k, 1 \leq j \leq k, i \neq j\}$. We compute that value at line 20. In the

latter subcase, $p_2 \oplus p_3 \in \mathsf{L}_{G_i}(u \cdots v \cdots)$ and $p_1 \in \mathsf{L}_{G_j}(u, \neg v)$, therefore $|p| = \max\{\lambda_3^i + \lambda_4^j\}$ where $1 \leq i \leq k, 1 \leq j \leq k, i \neq j$, which value we compute at line 19.

- For some folio G_i, p_1 and p_3 are mostly in G_i (Figure 13(c)). Clearly, $p_2 \in \mathsf{L}_{G_j}(u \cdots v)$ and $\langle p_1, p_3 \rangle \in \mathsf{L}_{G_i}(u \perp v)$, so $|p| = \max\{\lambda_2^i + \lambda_7^j\}$ where $1 \leq i \leq k, 1 \leq j \leq k, i \neq j$, which value we compute at line 21.



(a) $\max_{i \neq j}\{\lambda_5^i + \lambda_6^j\}$ (b) $\max_{i \neq j}\{\lambda_3^i + \lambda_4^j\}$ (c) $\max_{i \neq j}\{\lambda_2^i + \lambda_7^j\}$

Figure 13: The subcases when p is mostly in exactly two folios.

Finally, suppose p is mostly in exactly three folios. That is possible only when $k \geq 3$, that is why z is initialized with zero at line 24 and the assignement at line 26 is executed only if $k \geq 3$. Under the current assumptions, p is covered by three paths p_1, p_2, and p_3, such that p_2 has endpoints u and v, p_1 is a u-path and p_3 is a v-path. Furthermore, every p_i is mostly in a distinct folio. Figure 14 illustrates the three subpaths in G.

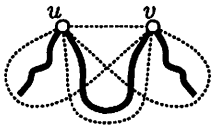


Figure 14: p is mostly in three folios: $|p| = \max\{\lambda_2^i + \lambda_4^j + \lambda_6^t \mid i \neq j \neq t \neq j\}$.

Note that $|p| = |p_1| + |p_2| + |p_3|$ and $p_1 \in \mathsf{L}_{G_j}(u, \neg v)$, $p_2 \in \mathsf{L}_{G_i}(u \cdots v)$, and $p_3 \in \mathsf{L}_{G_t}(v, \neg u)$ for some pairwise distinct i, j, and t from $\{1, 2, \ldots, k\}$. The correctness of the assignment at line 26 is obvious. □

The remainder of the correctness proof of algorithm LONGEST PATH is straightforward. Having in mind the inductive Definition 1, and Lemma 1 and Lemma 2, a proof by structural induction of the correctness of Algorithm COMPUTE LABEL follows immediately. And based on that, the correctness of LONGEST PATH is obvious.

3.3 Time complexity analysis

The complexity of LONGEST PATH obviously equals the complexity of COMPUTE LABEL. In general, for every folio, COMPUTE LABEL (CL) makes two calls to itself and a call to COMBINE ON FACE (CoF). After that, there is a single call to COMBINE ON EDGE (CoE).

The complexity of COMBINE ON FACE is obviously $\Theta(1)$. Now we argue the complexity of COMBINE ON EDGE is $\Theta(k)$ where k is the number of folios. That may not be immediately obvious: for instance, computing the maximum at line 7 requires $\Theta(k^2)$ time if done in a naive straightforward way, and likewise line 26 takes $\Theta(k^3)$ time is implemented naively. We explain how to compute line 7 in $\Theta(k)$:

- Compute in $\Theta(k)$ a maximum m_4 and a second maximum s_4 of λ_4^i, for $1 \leq i \leq k$, and store the index of m_4.
- Compute in $\Theta(k)$ a maximum m_6 and a second maximum s_6 of λ_6^i, for $1 \leq i \leq k$, and store the index of m_6.
- If m_4 and m_6 have different indices then $y \leftarrow m_4 + m_6$.
- Otherwise, $y \leftarrow \max\{m_4 + s_6, m_6 + s_4\}$.

Obviously, lines 13, 16, 19, 20, 21, 22, and 23 can be computed in $\Theta(k)$ likewise. Finally, note that line 26 can also be computed in $\Theta(k)$ in a similar fashion by computing a maximum, second maximum, and third maximum for each list, recording the indices the first and second maxima, and then dealing with a constant number of possible situations, in each one computing the maximum of a constant number of summands.

Therefore, the complexity of COMPUTE LABEL on an arbitrary 2-tree of n vertices is captured by the following recurrence relation:

$$\begin{aligned} T(n) &= \sum_{i=1}^{k} (\underbrace{T(n_i^1)}_{\text{CL}} + \underbrace{T(n_i^2)}_{\text{CL}} + \underbrace{\Theta(1)}_{\text{CoF}}) + \underbrace{\Theta(k)}_{\text{CoE}} \\ &= \sum_{i=1}^{k} (T(n_i^1) + T(n_i^2)) + \Theta(k) \end{aligned} \tag{1}$$

where k is the number of folios and for folio number i, n_i^1 and n_i^2 are the numbers of vertices of its branches. Let n_i be the number of vertices in folio number i. Clearly,

$$\sum_{i=1}^{k} n_i = n + 2(k-1)$$

$$n_i^1 + n_i^2 = n_i + 1$$

therefore

$$\sum_{i=1}^{k}(n_i^1 + n_i^2) = \sum_{i=1}^{k}(n_i + 1) = n + 2(k-1) + k = n + 3k - 2$$

Apply Lemma 3 and conclude that the solution to recurrence (1) is $T(n) = O(n)$. It follows our algorithm is a linear time one.

Lemma 3. *Let the recurrence relation $T(m)$ be defined as follows for $m > 0$:*

$$T(1) = \Theta(1)$$
$$T(m) = \sum_{i=1}^{q} T(m_i) + \Theta(q)$$

where

- $\forall i_{1 \le i \le q}(m_i \in \mathbb{N}^+ \text{ and } 1 \le m_i < m)$, *and*
- $\sum_{i=1}^{q} m_i = m + \Theta(q)$.

Then $T(m) = O(m)$.

Proof:
By induction on m. Assume there are positive constants b and c such that $T(m) \le cm - b$. By the inductive hypothesis,

$$\begin{aligned} T(m) &\le \sum_{i=1}^{q}(c.m_i - b) + \Theta(q) \\ &= c\sum_{i=1}^{q}(m_i) - b.m + \Theta(q) \\ &= c(m + \Theta(q)) - bm + \Theta(q) \\ &= cm - bm + \Theta(q) \qquad (2) \\ &\le cm - bm + pm, \end{aligned}$$

assuming the larger of the two hidden constants in the "$\Theta(\,)$" expression in (2) is p. Clearly, $cm + (p-b)m < cm - b$ for all sufficiently large m if $p - b < 0$. □

3.4 Longest Path on weighted and partial 2-trees

After the following modifications to our algorithm it solves the LONGEST PATH problem on edge-weighted 2-trees. In COMPUTE LABEL, change the

basic case (line 3) to $\widetilde{\lambda} \leftarrow (t, t, 0, 0, 0, 0, 0)$, where t is the weight of the sole edge. In function COMBINE ON FACE, modify lines 5, 7, 8, 9, 11, 13, 14, 15, and 22 so that each "+1" becomes "+t", where t is the weight of the root edge. The verification of the modified algorithm is straightforward and the time complexity analysis is the same.

To solve the LONGEST PATH problem on partial 2-trees in linear time using our algorithm, we proceed as follows. We compute the tree decomposition of the partial 2-tree. There is a linear time algorithm for this due to Bodlaender [4]. Then, from the computed tree decompostion, we obtain a 2-tree in an obvious way, adding dummy edges. The weights of all the dummy edges are set to a large negative constant (any number larger in absolute value than the sum of all edge weights in the initial partial 2-tree will suffice). Since there are only $O(n)$ edges in any 2-tree, the addition of the dummy edges can be done in linear time. Then we run our algorithm to obtain the length of a longest path in the 2-tree. The path is guaranteed not to contain any dummy edges since for any such path we have subpaths with strictly larger weights, namely those not containing any dummy edge. It is evident from our analysis earlier in Section 3 that the algorithm works correctly with negative edge weights. It is worth mentioning that the class of partial 2-trees includes well known and important graph classes such as the outerplanar graphs.

4 Conclusions

We have designed a practical, easy to implement algorithm for the LONGEST PATH problem on 2-trees. Indeed, there has been a linear time algorithm (see [3]) solving that problem on that graph class for two decades. However, Bodlaender's algorithm is utmostly impractical, its hidden constant is enormous, it does not output a maximum path, it does not work on weighted graphs, and it is described not in pseudocode but only as an idea, undoubtedly because of its great descriptional complexity.

In contrast to that, our algorithm is trivial to implement and clearly its hidden constant is small. Furthermore, unlike the former algorithm, ours can run on weighted graphs and actually output a longest path.

5 Acknowledgments

Dr. Vassilev's work is supported by NSERC Discovery Grant.
Preliminary results of this research were presented at the Second Annual Workshop on Algorithmic Graph Theory held at Nipissing University, North Bay, Ontario on May 16–20, 2011. We thank the participants in the work-

shop for their suggestions and useful discussions which helped us improve the algorithm.

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[2] Andreas Björklund and Thore Husfeldt. Finding a path of superlogarithmic length. *SIAM J. Comput.*, 32(6):1395–1402, 2003.

[3] Hans L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, January 1993.

[4] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, December 1996.

[5] R. W. Bulterman, F. W. van der Sommen, G. Zwaan, T. Verhoeff, A. J. M. van Gasteren, and W. H. J. Feijen. On computing a longest path in a tree. *Information Processing Letters*, 81(2):93–96, 2002.

[6] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in h-minor-free graphs. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 631–640, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[7] Harold N. Gabow and Shuxin Nie. Finding long paths, cycles and circuits. In *ISAAC '08: Proceedings of the 19th International Symposium on Algorithms and Computation*, pages 752–763, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Michael Garey and David Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1979.

[9] David R. Karger, Rajeev Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[10] M. Markov, Krassimir Manev, Mugurel Ionuţ Andreica, and Nicolae Ţăpuş. A linear time algorithm for computing longest paths in cactus graphs. *Serdica Journal of Computing*, 6(3):287–298, 2012.

[11] B. Monien. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25:239–254, 1985.

[12] Ryuhei Uehara and Yushi Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(5):911–930, 2007.

[13] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

[14] Zhao Zhang and Hao Li. Algorithms for long paths in graphs. *Theoretical Computer Science*, 377(1-3):25–34, 2007.