A Supplementary File for "Projection onto A Nonnegative Max-Heap"

A.1 Examples for Elaborating the Related Definitions

A.1.1 Root-Tree in Definition 1

For the tree depicted in Figure 1 (a), the following three trees are valid root-trees:

(1) $T_1^t = (V_1^t, E_1^t)$, $V_1^t = \{x_1\}, E_1 = \emptyset$;

(2) $T_2^t = (V_2^t, E_2^t)$, $V_2^t = \{x_1, x_2\}$, $E_2^t = \{(x_1, x_2)\}$;

(3) $T_3^t = (V_3^t, E_3^t)$, $V_3^t = \{x_1, x_3, x_6\}$, $E_3^t = \{(x_1, x_3), (x_3, x_6)\}$.

Similarly, for the tree depicted in Figure 1 (b), the following three trees are valid root-trees:

(1) $T_1^t = (V_1^t, E_1^t)$, $V_1^t = \{x_1\}, E_1^t = \emptyset$;

(2) $T_2^t = (V_2^t, E_2^t)$, $V_2^t = \{x_1, x_2\}$, $E_2^t = \{(x_1, x_2)\}$;

(3) $T_3^t = (V_3^t, E_3^t)$, $V_3^t = \{x_1, x_2, x_3\}$, $E_3^t = \{(x_1, x_2), (x_2, x_3)\}$.

A.1.2 Root-Tree Set in Definition 2

When T is a sequential list of length p, we have $|R(T)| = p$. For example, the sequential list depicted in Figure 1 (b) has 7 root-trees. When T is a full binary tree of depth d (the number of nodes $p = 2^{d+1} - 1$), we have $|R(T)| = 1 - d + \sum_{i=1}^{d} 2^{(2^i)}$. For example, the binary tree depicted in Figure 1 (a) has 19 root-trees. When T is a tree of depth 1 that contains p nodes, we have $|R(T)| = 2^{p-1}$. For example, the tree shown in Figure 1 (c) has $2^6 = 64$ root-trees.

A.1.3 Tree Value in Definition 3

For the tree in Figure 2 (b), its value is 0.8. For the tree depicted in the first row of Figure 4 (b), its value is 1.

A.1.4 Maximal Root-Tree in Definition 4

For the tree T (sequential list) depicted in the first row of Figure 4 (b), we have $m_{\max}(T) = 2$. There are two root-trees of T that achieve the maximal root-tree value:

(1) $T_1 = (V_1, E_1)$ with $V_1 = \{v_1\}$ and $E_1 = \emptyset$;

(2) $T_2 = (V_2, E_2)$ with $V_2 = \{v_1, v_2, x_3\}$ and $E_2 = \{(v_1, v_2), (v_2, v_3)\}$.

According to Definition 4, we have $M_{\max}(T) = T_2$.

For the tree T depicted in Figure 2 (a), we have $m_{\max}(T) = 3$. There are two root-trees of T that achieve the maximal root-tree value:

(1) $T_1 = (V_1, E_1)$, where $V_1 = \{v_1, v_2\}$, $E_1 = \{(v_1, v_2)\}$;

(2) $T_2 = (V_2, E_2)$, where $V_2 = \{v_1, v_2, v_3\}$, $E_2 = \{(v_1, v_2), (v_1, v_3)\}$.

According to Definition 4, we have $M_{\max}(T) = T_2$.

A.1.5 Maximal Root-Tree in Definition 5

When treating each tree enclosed by the dashed frame as a super-node, the plot in the last row of Figure 4 (b) is a super-tree of the plot in the first row of the same figure, and the plot in Figure 2 (d) is a super-tree of the tree shown in Figure 2 (b).

A.2 Justification of Atda for A Sequential List

When the target tree is a sequential list $T^t = (V^t, E^t)$ with $V^t = \{x_i, i = 1, 2, \ldots, p)$ and $E^t = \{(x_{i-1}, x_i), i = 2, 3, \ldots, p\}$, the problem (3) reduces to

$$\min_{x_1 \geq x_2 \geq x_3 \geq \ldots \geq x_p \geq 0} \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|_2^2. \tag{22}$$

With the target tree T^t, we can construct the input tree $T = (V, E)$, where $V = \{v_i, i = 1, 2, \ldots, p)$ and $E = \{(v_{i-1}, v_i), i = 2, 3, \ldots, p\}$.

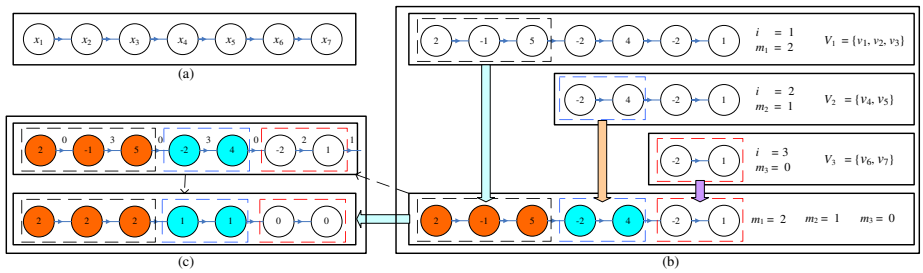

Figure 4: Illustration of Algorithm 1 for solving (22) with the tree being a sequential list. Plot (a) depicts the sequential list. Plot (b) illustrates the calling of Atda three times to the input $\mathbf{v} \in \mathbb{R}^7$: $v_1 = 2, v_2 = -1, v_3 = 5, v_4 = -2, v_5 = 4, v_6 = -2, v_7 = 1$. The last row of plot (c) depicts the solution $\tilde{\mathbf{x}} \in \mathbb{R}^7$: $\tilde{x}_1 = \tilde{x}_2 = \tilde{x}_3 = 2$, $\tilde{x}_4 = \tilde{x}_5 = 1$, and $\tilde{x}_6 = \tilde{x}_7 = 0$. The values on the edges of the first row of plot (c) correspond to the dual variables, from which we can also obtain the optimal solution $\tilde{\mathbf{x}}$ (refer to the proof of Theorem 3 for details).

Figure 4 illustrates how Algorithm 1 solves (22) via $\tilde{\mathbf{x}} = \text{Atda}(T, 0)$, where $\mathbf{v} \in \mathbb{R}^7$ is the input, and $\tilde{\mathbf{x}} \in \mathbb{R}^7$ is the optimal solution. Figure 4 (a) shows the target tree $T^t = (V^t, E^t)$ with $V^t = \{x_1, x_2, \ldots, x_7)$ and $E^t = \{(x_{i-1}, x_i), i = 2, 3, \ldots, 7\}$. From the target tree T^t, we can construct the input tree $T = (V, E)$ using $\mathbf{v}$. As shown in Figure 4 (b), we call Atda three times in order to yield the optimal solution depicted in the last row of Figure 4 (c). Firstly, we identify the maximal root-tree $T_1 = (V_1, E_1)$ with $V_1 = \{v_1, v_2, v_3\}$ and $E_1 = \{(v_1, v_2), (v_2, v_3)\}$. The value of T_1 is $m_1 = m(T_1) = m_{\max}(T) = 2$. Therefore, we set $\tilde{x}_1 = \tilde{x}_2 = \tilde{x}_3 = m_1 = 2$. We remove T_1 from T, and apply the above process recursively to obtain the optimal solution $\tilde{\mathbf{x}}$ given in the last row of Figure 4 (c). For the convenience of understanding the proof of Theorem 3, we also give the value of the dual variables (refer to the proof of Theorem 3 for details) in the first row of Figure 4 (c), from which we can also compute the optimal solution $\tilde{\mathbf{x}}$. The following theorem shows that, $\tilde{\mathbf{x}} = \text{Atda}(T, 0)$ provides the optimal solution to (22).

Theorem 3. $\tilde{\mathbf{x}} = \text{Atda}(T, 0)$ *provides the unique and optimal solution to* (22).

Proof: We prove the theorem using the KKT conditions [2], i.e., verifying that the solution $\tilde{\mathbf{x}} = \text{Atda}(T, 0)$ satisfies the corresponding KKT conditions.

The Lagrangian of (22) can be written as

$$L(\mathbf{x}, \mathbf{y}) = \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|_2^2 + \sum_{i=2}^{p}(x_i - x_{i-1})y_{i-1} - y_p x_p,$$

where $\mathbf{y} \in \mathbb{R}^p$, $y_{i-1}, i = 2, 3, \ldots p$ is the dual variable for the constraint $x_i \geq x_{i-1}$ and y_p is the dual variable for the constraint $x_p \geq 0$.

The objective function of (22) is strictly convex, so that it admits a unique solution. In addition, the constraints in (22) satisfy the Slater's condition. Therefore, it suffices to verify

that the primal optimal $\tilde{\mathbf{x}}$ and the dual optimal $\tilde{\mathbf{y}}$ satisfy the following KKT conditions:

$$\begin{aligned}
\tilde{x}_1 \geq \tilde{x}_2 \geq \tilde{x}_3 \geq \ldots \geq \tilde{x}_p &\geq 0 && (23)\\
\forall i = 2, \ldots, p, (\tilde{x}_i - \tilde{x}_{i-1})\tilde{y}_{i-1} &= 0 && (24)\\
\tilde{y}_p \tilde{x}_p &= 0 && (25)\\
\tilde{x}_1 - v_1 - \tilde{y}_1 &= 0 && (26)\\
\forall i = 2, \ldots, p, \tilde{x}_i - v_i + \tilde{y}_{i-1} - \tilde{y}_i &= 0 && (27)\\
\forall i = 1, 2, \ldots, p, \tilde{y}_i &\geq 0. && (28)
\end{aligned}$$

For the length of the sequence $\{m_i, i = 1, 2, \ldots, k\}$, it is clear $1 \leq k \leq p$. We set the optimal dual variable $\tilde{\mathbf{y}}$ as follows:

$$\tilde{y}_i = \tilde{x}_i - v_i + \tilde{y}_{i-1}, i = 1, 2, \ldots, p, \tag{29}$$

where $\tilde{y}_0 = 0$. In the following, we show that $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ satisfy the KKT conditions.

According to Algorithm 1, we have

$$m_1 > m_2 > \ldots > m_k \geq 0.$$

In addition, the indices of the nodes in V_i are smaller than those of the nodes in V_{i+1}. It follows from $\tilde{x}_j = m_i, \forall v_j \in V_i$ that (23) holds.

If $k = 1$, (24) naturally holds, as $\tilde{x}_j = m_1, \forall v_j \in V_1 = V$, where V contains all the nodes of the input tree. Next, we consider the case when $k \geq 2$. To prove (24), it suffices to establish the relationship:

$$\tilde{y}_{j_i} = 0, \forall i = 1, 2, \ldots, k-1,$$

where j_i denotes the largest index of the nodes in the set V_i. Since $\tilde{x}_j = m_i = m(T_i), \forall v_j \in V_i, i = 1, 2, \ldots, k-1$, we have $\sum_{j:v_j \in V_i} \tilde{x}_j = \sum_{j:v_j \in V_i} v_j = |V_i| m_i, \forall i = 1, 2, \ldots, k-1$. Incorporating (29), we have

$$\tilde{y}_{j_i} = \sum_{l=1}^{i} \left(\sum_{v_j \in V_l} \tilde{x}_j - \sum_{v_j \in V_l} v_j \right) + \tilde{y}_0 = \tilde{y}_0 = 0, \forall i = 1, 2, \ldots, k-1. \tag{30}$$

When $\tilde{x}_p = 0$, (25) naturally holds. For the case $\tilde{x}_p > 0$, we can verify $\tilde{y}_p = 0$ following the similar argument for deriving (30).

As indicated by (29), (26) and (27) naturally hold.

The condition (28) can be rewritten as:

$$\tilde{y}_j \geq 0, j : v_j \in V_i, i = 1, 2, \ldots k. \tag{31}$$

Next, we prove the result for $i = 1$ and the underlying methodology can be extended to prove the remaining inequalities of (31). According to (29), we have

$$\tilde{y}_j = \sum_{l=1}^{j} \tilde{x}_l - \sum_{l=1}^{j} v_l + \tilde{y}_0 = j(m_1 - \frac{\sum_{l=1}^{j} v_l}{j}) \geq 0, \forall v_j \in V_1,$$

where the last inequality holds as m_1 is the largest maximal value of T. □

A.3 Computational Cost of Anae for Finding the Maximal Root-Tree

We study the computational cost of Anae (a naive algorithm with enumeration) for finding the maximal root-tree. Before conducting analysis, we note that, it is challenging, if not impossible, to analyze the time complexity for Algorithm 1 for an arbitrary tree. Therefore, in the sequel, we focus on analyzing the time complexity for three special trees: 1) a sequential list, 2) a binary tree, and 3) a tree with depth 1. In addition, we assume that the value of the root-tree can be evaluated in $O(1)$[3], and thus make use of the number of root-trees for measuring the complexity.

[3]For the sequential list and the tree with depth 1, the $O(1)$ evaluation of the mean can be easily obtained. For the binary tree, we can also achieve this goal by enumerating the root-trees in a proper order.

A Sequential List When the tree is a sequential list depicted in Figure 1 (b), the worst-case time complexity of Algorithm 1 is $O(p^2)$. It is easy to check that, the worst case happens when the input $\mathbf{u} \in \mathbb{R}^p$ satisfies $u_i > u_{i-1} > 0$, so that, after each call of Algorithm 1, we output one node only. In this case, we need to enumerate $\sum_{i=1}^{p} i = p(p+1)/2$ root-trees.

A Tree with Depth 1 When the tree is of depth 1 depicted in Figure 1 (c), the worst case time complexity is $O(2^p)$. Specifically, for the input tree, it has 2^{p-1} root-trees; and after removing the first root-tree, the resulting trees are of depth 0, so that we need to enumerate at most $p-1$ additional root-trees. Therefore, Anae has exponential complexity.

A Full Binary Tree When the tree is a complete binary tree depicted in Figure 1 (a), the worst-case happens when the input $\mathbf{u}$ follows a max-heap structure and the entries of $\mathbf{u}$ are positive and different. When the complete binary tree has depth d, it has $p = 2^{d+1} - 1$ nodes. For the input tree T, it has $|R(T)| = 1 - d + \sum_{i=1}^{d} 2^{(2^i)}$ root-trees, which satisfies $2^{2^d} \leq |R(T)| < 2^{2^d+1}$. After removing the root node, the resulting two trees have $2(1 - d + \sum_{i=1}^{d-1} 2^{(2^i)})$ root-trees, which satisfies $2(1 - d + \sum_{i=1}^{d-1} 2^{(2^i)}) < 2^{2^d+1}$. Therefore, $\sum_{j=0}^{d} 2^j (1-d+\sum_{i=1}^{d-i} 2^{(2^i)})$, the total number of root-trees is in the interval $(2^{2^d}, (d+1)2^{2^d+1})$, or equivalently $(2^{(p+1)/2}, \log_2(p+1)2^{(p+3)/2})$. Therefore, Anae has exponential complexity.

A.4 Proof of Lemma 1

Lemma 1 is an extension of the following lemma, and can be proven in a similar way.

Lemma 2. *For a nonempty tree $T = (V, E)$ rooted at v_{i_0}, denote its maximal root-tree as $T_{\max} = (V_{\max}, E_{\max})$. Let the root v_{i_0} have n child nodes, denoted by $v_{i_1}, \ldots, v_{i_n}$. If $n \geq 1$, we denote the subtree of T rooted at v_{i_j} as $T^j = (V^j, E^j), j = 1, 2, \ldots, n$, the maximal root-tree of T^j as $T^j_{\max} = (V^j_{\max}, E^j_{\max})$, and $\tilde{m} = \max_{j=1,2,\ldots,n} m(T^j_{\max})$. Then, we have:*

1. *If $n = 0$, then $T_{\max} = T$.*
2. *If $n \geq 1$, $v_{i_0} \leq 0$, and $\tilde{m} = 0$, then $T_{\max} = T$.*
3. *If $n \geq 1$, $v_{i_0} > 0$, and $v_{i_0} > \tilde{m}$, then $V_{\max} = \{v_{i_0}\}$ and $E_{\max} = \emptyset$.*
4. *If $n \geq 1$, $v_{i_0} > 0$, and $v_{i_0} \leq \tilde{m}$, then $V^j_{\max} \subseteq V_{\max}$, $E^j_{\max} \subseteq E_{\max}$ and $(v_{i_0}, v_{i_j}) \in E_{\max}$, $\forall j : m(T^j_{\max}) = \tilde{m}$.*
5. *If $n \geq 1$, $v_{i_0} \leq 0$, and $\tilde{m} > 0$, then $V^j_{\max} \subseteq V_{\max}$, $E^j_{\max} \subseteq E_{\max}$ and $(v_{i_0}, v_{i_j}) \in E_{\max}$, $\forall j : m(T^j_{\max}) = \tilde{m}$.*

Proof: There are two cases with $T_{\max}$, the maximal root-tree of T: 1) $T_{\max}$ has only one node v_{i_0}, which, together with (4) lead to

$$m(T_{\max}) = \max(v_{i_0}, 0), \tag{32}$$

and 2) the nodes of $T_{\max}$ include the root v_{i_0} and the nodes of $\tilde{T}^j = (\tilde{V}^j, \tilde{E}^j)$ for some $j = j_1, j_2, \ldots, j_q$, $1 \leq j_l \leq n, l = 1, 2, \ldots, q$, where $\tilde{T}^j$ is a given root-tree of T^j. According to Definition 3, we have

$$m(T_{\max}) = \max\left(\frac{v_{i_0} + \sum_{l=1}^{q} |\tilde{V}^{j_l}| m(\tilde{T}^{j_l})}{1 + \sum_{l=1}^{q} |\tilde{V}^{j_l}|}, 0\right) \leq \max\left(\frac{v_{i_0} + \sum_{l=1}^{q} |\tilde{V}^{j_l}| \tilde{m}}{1 + \sum_{l=1}^{q} |\tilde{V}^{j_l}|}, 0\right). \tag{33}$$

If $n = 0$, the tree T only has one node, so that it has a unique root-tree, which is T itself.

If $n \geq 1$, $v_{i_0} \leq 0$, and $\tilde{m} = 0$, it follows from (32) and (33) that $m(T_{\max}) = 0$.

It follows from (4), (5), and (6) that, if $m_{\max}(T) = m(T_{\max}) = 0$, then $T_{\max} = T$.

If $n \geq 1$, $v_{i_0} > 0$, and $v_{i_0} > \tilde{m}$, we obtain from (32) that $m(T_{\max}) = v_{i_0}$ when $T_{\max}$ has only one node v_{i_0}, and obtain from (33) that $m(T_{\max}) < v_{i_0}$ if $T_{\max}$ contains nodes besides

the root v_{i_0}. Therefore, we have that, if $n \geq 1$, $v_{i_0} > 0$, $v_{i_0} > \tilde{m}$, then $V_{\max} = \{v_{i_0}\}$ and $E_{\max} = \emptyset$.

If $n \geq 1$, $v_{i_0} > 0$, and $v_{i_0} \leq \tilde{m}$, we obtain from (32) that $m(T_{\max}) = v_{i_0}$ when $T_{\max}$ has only one node v_{i_0}, and obtain from (33) that $m(T_{\max}) \leq \tilde{m}$ if $T_{\max}$ contains nodes besides the root v_{i_0}. Therefore, we have $m(T_{\max}) \leq \tilde{m}$. According to (4) and (5), $T^j_{\max}, \forall j : m(T^j_{\max}) = \tilde{m}$ should be included into the maximal root-tree $T_{\max}$.

If $n \geq 1$, $v_{i_0} \leq 0$, and $\tilde{m} > 0$, we can verify that $m(T_{\max}) < \tilde{m}$, and thus it follows from (4) and (5) that, $T^j_{\max}, \forall j : m(T^j_{\max}) = \tilde{m}$ should be included into the maximal root-tree $T_{\max}$.

This completes the proof. □

A.5 Illustration of Abuam

We briefly explain Algorithm 2 as follows. The input tree T is built with the target tree T^t and the input $\mathbf{v}$. In line 1, we generate T_0, which is the root-tree of T that only contains the root. According to the definition of the maximal root-tree in Definition 4, we have that, T_0 is a root-tree of $T_{\max}$, the maximal root-tree of T. Lines 2-4 correspond to case 1 of Lemma 2 (see A.4), and lines 5-20 repeatedly make use of Lemma 1 to compute the maximal root-tree. Lines 7-9 correspond to case 1 of Lemma 1, and the maximal root-tree of T is T itself. In lines 10-12, T^j is a subtree of T rooted at v_{i_j}, with v_{i_j} being a child node of a given node in V_0. We find $T^j_{\max}$, the maximal root-tree of T^j, and compute $\tilde{m}$, the maximal value among $m(T^j_{\max}), j = 1, 2, \ldots, n$. As we need to recursively call Abuam in line 11, the maximal root-trees are generated in a bottom-up manner, i.e., the maximal root-tree of the subtree rooted at the parent node is obtained after the maximal root-trees of the subtrees rooted at the child nodes are all processed. This explains why the algorithm is performed in a bottom-up manner. Lines 13-14 correspond to case 2 of Lemma 1, lines 15-16 correspond to case 3 of Lemma 1, and lines 17-19 correspond to cases 4 & 5 of Lemma 1.

Next, we illustrate how Algorithm 2 finds the maximal root-tree for the two examples used in Figure 2 and Figure 4. For the convenience of discussion, we denote T_i as the maximal root-tree of the subtree rooted at v_i.

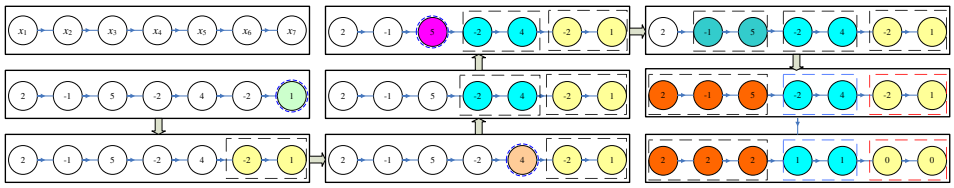

Figure 5: Application of Algorithm 2 for finding the maximal root-tree for a sequential list. The target tree is depicted in the top left box, the solution to (3) is shown in the bottom right box, and the internal boxes depict the maximal root-trees for the subtrees associated with the nodes in a bottom-up manner.

A Sequential List In Figure 5, we illustrate how Algorithm 2 finds the maximal root-tree for the same sequential list used in Figure 4. According to lines 2-3 (see also case 1 of Lemma 2), we have $T_7 = (\{v_7\}, \emptyset)$. It follows from lines 17-18 that (see also case 5 of Lemma 2), to get T_6, we need to merge v_6 with T_7, as $v_6 = -2 < m(T_7) = 1$, and then we have $T_6 = (\{v_6, v_7\}, \{(v_6, v_7)\})$ according to lines 7-8. According to lines 15-16 (see also case 3 of Lemma 2), we have $T_5 = (\{v_5\}, \emptyset)$, as $v_5 = 4 > m(T_6) = 0$. Following a similar analysis, we have $T_4 = (\{v_4, v_5\}, \{(v_4, v_5)\})$, $T_3 = (\{v_3\}, \emptyset)$, and $T_2 = (\{v_2, v_3\}, \{(v_2, v_3)\})$. For the maximal root-tree of the input tree T, its maximal tree $T_1 = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3)\})$ as $v_1 = 3 = m(T_2)$ (see also case 4 of Lemma 2). As shown in the bottom right box, the solution to (3) is identical to the one shown in Figure 4.

A Tree With Depth 3 In Figure 6, we illustrate how Algorithm 2 finds the maximal root-tree for the same tree with depth 3 employed in Figure 4. In plot (b), we fill the tree with values in $\mathbf{v}$, and compute the maximal root-trees of the subtrees rooted at the leaf

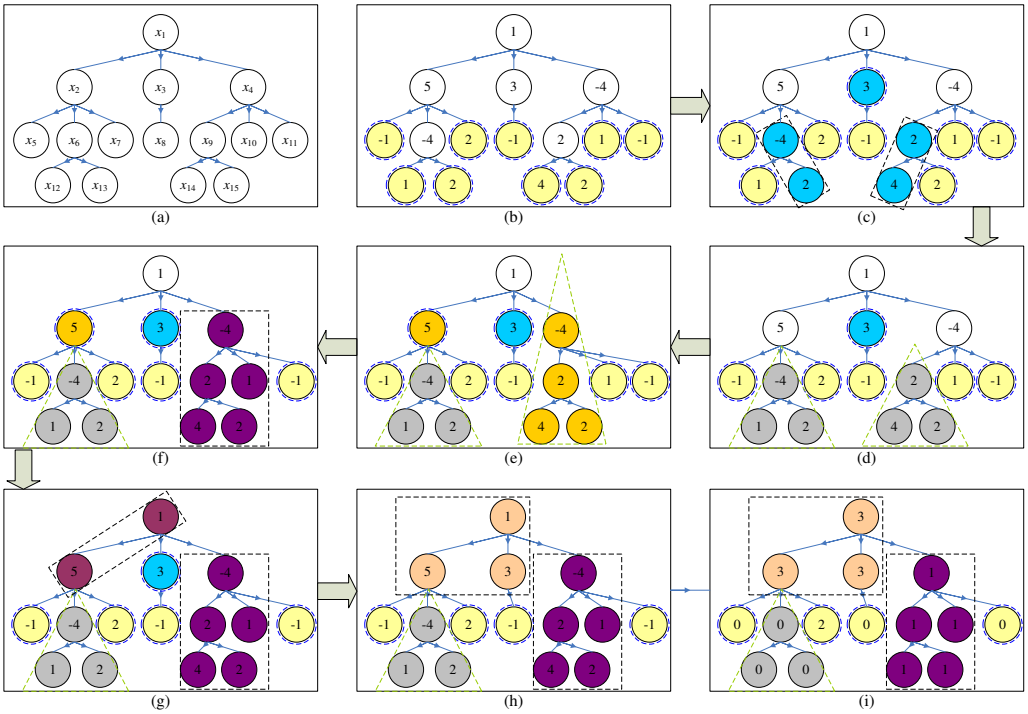

Figure 6: Application of Algorithm 2 for finding the maximal root-tree for a tree with depth 3. Plot (a) depicts the target tree T^t. Plot(b)-plot(h) show the bottom-up procedure for finding the maximal root-trees of all the subtrees of the input tree T. Plot(h) illustrates the super-tree obtained by Abuam, and plot (i) gives the solution to (3), which is identical to the one given in Figure 2 (e).

nodes, according to lines 2-3 (see also case 1 of Lemma 2). In plot (c), we begin the process for finding the maximal root-trees of the subtrees rooted at v_3, v_6, and v_9, respectively, as the maximal root-trees of the subtrees rooted at their child nodes have been identified in plot (b). Plot (d) shows the results for T_6 and T_9. Plots (e-f) depict the process for finding T_4. In plot (g), we begin finding T_1, as T_2, T_3, and T_4 have been computed in plot (e), plot(c), and plot (f), respectively. In plot (h), the top dashed frame corresponds to the maximal root-tree T_1. All the dashed frames in plot (h) constitute a super-tree, with which we can obtain the optimal solution to (3), shown in plot (i). It is clear that, the solution is identical the one shown in Figure 2 (e).

A.6 Time Complexity of Abuam

Let us analyze the time complexity of Algorithm 2 for finding the maximal root-tree of a tree with p nodes. For convenience of analysis, we denote the number of leaf nodes by p_1, the number of internal nodes by p_2, and the number of the super-nodes in the resulting super-tree of Algorithm 2 by $\tilde{p}$. It is clear that $p = p_1 + p_2$, and $1 \leq \tilde{p} \leq p$. We let n_i denote the times line i is called for all the recursive calls of Abuam. We note that, 1) the merge operation defined in Definition 6 can be implemented in $O(1)$ flops (floating point operations), and 2) the most time-consuming steps among lines 5-20 are lines 10-12, with time complexity of $O(n)$.

Since there are p nodes, we have $n_1 = p$. Similarly, as there are p_1 leaf nodes, we have $n_3 = p_1$. Therefore, lines 1-4 cost $O(p)$. Next, we discuss the computational cost for lines 5-20. The key is to estimate the number of calls of lines 10-12. A key observation is $n_8 + n_{14} + n_{16} = p_2$, as there are p_2 internal nodes (for the p_1 leaf nodes, Abuam terminates via line 3). Therefore, we have

$$n_{14} + n_{16} \leq p_2. \tag{34}$$

Table 1: Worst-case time complexity of Anae and Abuam.

	Anae	Abuam
sequential list	$O(p^2)$	$O(p)$
tree with depth 1	exponential	$O(p^2) \Rightarrow O(p \log p)$
binary tree	exponential	at most $O(p^2)$
general tree	exponential	$O(p^2)$

It is clear that, $T_{\max} = T$ line 14 is equivalent to T_0=merge(T_0, $T^j_{\max}$, T), $\forall j = 1, 2, \ldots, n$ and $T_{\max} = T$, as $\tilde{m} = 0$. We have

$$n_{14} + n_{18} \leq p - \tilde{p}, \tag{35}$$

as 1) the number of merges in the recursive calls of Abuam is $p - \tilde{p}$, and 2) line 14 and line 18 consists of at least one merge. It follows from (34) and (35) that

$$n_{14} + n_{16} + n_{18} \leq p - \tilde{p} + p_2, \tag{36}$$

so that lines 10-12 are called at most $p - \tilde{p} + p_2$ times. Due to the structure of the tree, we have $n \leq p_1$, i.e., the number of the nodes that fall outside of T_0 and meanwhile have a node in T_0 as the parent node is less than p_1, the number of leaf nodes. Therefore, we have $n_{11} \leq (p - \tilde{p} + p_2)p_1$.

As $(p - \tilde{p} + p_2)p_1 \leq \frac{(2p-\tilde{p})^2}{2}$, we have that, the overall time complexity for a general tree is at most $O(p^2)$. In addition, when the input tree is of depth 1 (see Figure 1 (c)) and satisfies $0 < v_1 < v_2 < v_3 < \ldots < v_p$ and $\sum_{j=1}^{i-1} v_j \leq v_i, \forall i$, line 11 will be called $\sum_{i=1}^{p-1}(p-i) = \frac{(p-1)p}{2}$ times, so that such $O(p^2)$ complexity can be achieved. However, we note that, for the tree of depth 1, the time complexity can be improved to $O(p \log p)$, by sorting the values of the leaf node in a decreasing order. When the tree is a sequential list, we have $p_1 = 1$, which leads to $n_{11} \leq (p - \tilde{p} + p_2)p_1 = p - \tilde{p} + p - 1 \leq 2p$. Therefore, Abuam has a linear time complexity for the sequential list. The worst-case time complexity of Anae and Abuam is presented in Table 1.

We would like to emphasize that, in the practical application of Abuam, n, the number of the nodes that not only fall outside of T_0 and but also have a node in T_0 as the parent node, is much less than p_1, the number of leaf nodes. In addition, when the solution is sparse, $n_{14} + n_{18}$ can be much less than $p - \tilde{p}$, as line 14 typically merges many nodes at one time. Therefore, we expect that the practical performance is much better than $O(p^2)$ for the general tree, which is verified by our experimental study.

The computational cost of Anae and Abuam is given in Table 1.