FPTAS’s for Trimming Weighted Trees *

Mingyu Xiao¹, Takuro Fukunaga², and Hiroshi Nagamochi²

¹ School of Computer Science and Engineering, University of Electronic Science and Technology of China, China, myxiao@gmail.com
² Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Japan, {takuro,nag}@amp.i.kyoto-u.ac.jp

Abstract. Given a tree with nonnegative edge cost and nonnegative vertex weight, and a number \( k \geq 0 \), we consider the following four cut problems: cutting vertices of weight at most or at least \( k \) from the tree by deleting some edges such that the remaining part of the graph is still a tree and the total cost of the edges being deleted is minimized or maximized. The MinMstCut problem (cut vertices of weight at most \( k \) and minimize the total cost of the edges being deleted) can be solved in linear time and space and the other three problems are NP-hard. In this paper, we design an \( O(nl/\varepsilon) \)-time \( O(l^2/\varepsilon + n) \)-space algorithm for MaxMstCut, and \( O(nl(1/\varepsilon + \log n)) \)-time \( O(l^2/\varepsilon + n) \)-space algorithms for the other two problems, MinLstCut and MaxLstCut, where \( n \) is the number of vertices in the tree, \( l \) the number of leaves, and \( \varepsilon > 0 \) the prescribed error bound.

Key words. Graph Cut, FPTAS, Tree, Tree Knapsack

1 Introduction

Given an undirected tree \( G = (V,E) \) with a nonnegative vertex weight \( w(v) \), \( v \in V \) and a nonnegative edge cost \( c(e) \), \( e \in E \), and a nonnegative number \( k \geq 0 \), the maximum cutting vertices of weight at most \( k \) problem (MaxMstCut) is to find a nonempty and proper subset \( X \) of \( V \) that maximizes the total cost of edges with exact one endpoint in \( X \) under the constraint that (i) the subgraph \( G[V-X] \) induced by \( V-X \) is connected and (ii) the total weight of vertices in \( X \) is at most \( k \). Analogously, we also define MaxLstCut, MinMstCut and MinLstCut, by replacing “maximizes” with “minimizes” or replacing “at most” with “at least” in the definition. If \( G \) is a rooted tree, then we impose the third constraint that the root is contained in the subgraph \( G[V-X] \) in finding a solution \( X \). These problems are also called the cutting \( k \) vertices (or vertices of weight \( k \)) problem in some references. In this paper, we mainly study algorithms for the above four cut problems in rooted and unrooted trees. In our algorithms, we may compute a solution \( X \subset V \) by identifying the corresponding set \( C = E(X,V-X) \) of edges joining \( X \) and \( V-X \), called cut set.

* A preliminary version of this paper was presented in the 4th international frontiers of algorithmics workshop (FAW 2010) and appeared in [15].
Graph cut problems have great applications and have been extensively studied in the literature. The \textit{minimum cut problem} is regarded as a basic problem among this kind of problems. Many extensions of the classical minimum cut problem, including the \textit{multi-way cut} problem [3, 13], the \textit{balanced cut} problem [5] and the cutting \(k\) vertices problem [4], are also important and fundamental problems. The multi-way cut problem extends the minimum cut problem in a way of partitioning the graph into more than two components. The balanced cut and cutting \(k\) vertices problems are to find an optimal cut under some constraints on the size of the two parts of the cut. These two problems have similar requirements. However, the balanced cut problem is to deal with the case of almost balanced sizes of the two sides of the cut, while the cutting \(k\) vertices problem is more to consider cuts with one small side. Although the minimum cut problem can be solved in polynomial time by flow and other techniques, most extended cut problems are NP-hard. Few “positive” results on the cutting \(k\) vertices problem are known. Considering the hardness of the problems, we study the four cut problems \textsc{MaxMstCut}, \textsc{MaxLstCut}, \textsc{MinMstCut} and \textsc{MinLstCut} in the graphs restricted to trees. These problems in trees are still important to be investigated, since they naturally generate the famous \textit{knapsack problem} from several ways.

In fact, the cut problems in rooted trees are related to some special cases of the \textit{precedence constraint knapsack problem} (PCKP) [7, 9]. In PCKP, a directed acyclic graph is given, each vertex in the graph is regarded as an item to be selected into the knapsack, and if a vertex is selected into the knapsack then all its \textit{ancestors} (vertices having a direct path to the vertex) must also be selected into the knapsack. \textsc{MaxMstCut} (resp., \textsc{MinLstCut}) can model the precedence constraint knapsack problem in rooted trees with all edges directed toward the root (resp., the leaves), which is called the \textit{in-tree} (resp., \textit{out-tree}) \textit{knapsack problems} [2, 7].

As a prior work on \textsc{MinLstCut} in rooted trees, there exists a \((4/3 + \epsilon)\)-approximation algorithm for the generalized partially totally unimodular cover problem with any \(\epsilon > 0\), presented by Kömemann, Parekh and Segev [10]. Consider the matrix, whose rows are corresponding to the vertices and columns corresponding to the edges, such that its \((i, j)\)-th component is 1 if the \(i\)-th vertex is a descendant of the \(j\)-th edge, and 0 otherwise. If we order the vertices by DFS, then 1’s appear consecutively in each column. Such a matrix is known to be totally unimodular. Although we do not give a formulation of the partial totally unimodular cover problem, this observation indicates that \textsc{MinLstCut} is contained by their problem. Another related work is the algorithms for the tree knapsack problems. Cho and Shaw presented a depth-first dynamic programming algorithm for the out-tree knapsack problem in [2] that can be modified to get an \(O(n(W - k))\)-time pseudo polynomial-time algorithm for \textsc{MinLstCut} in rooted trees, where \(W\) is the total weight of all vertices in the tree. This pseudo polynomial-time algorithm can also be used to derive approximation schemes, but the running time depends on \(W\). Johnson and Niemi [7] designed several algorithms for two different tree knapsack problems. Their \(O(n^3/\epsilon)\)-time FPTAS (fully polynomial-time approximation scheme) for the in-tree knapsack problem...
FPTAS’s for Trimming Weighted Trees 3

can solve MaxMstCut in rooted trees in the same time. There are few nontrivial
results on the problems in unrooted trees. In this paper, except showing that
MinMstCut in trees can be solved in linear time and space easily, we present:
(i) $O(nl/\varepsilon)$-time $O(l^2/\varepsilon + n)$-space FPTAS’s for MaxMstCut in rooted and
unrooted trees;
(ii) $O(nl(1/\varepsilon + \log n))$-time $O(l^2/\varepsilon + n)$-space FPTAS’s for MinLstCut and
MaxLstCut in rooted and unrooted trees.

The rest of the paper is organized as follows. Section 2 gives our notation
system. Section 3 considers the hardness of the problems. Section 4 presents
pseudo polynomial-time algorithms that will be used to design FPTAS’s. Sec-
section 5 presents FPTAS for MaxMstCut in rooted trees and Section 6 presents
FPTAS’s for MinLstCut and MaxLstCut in rooted trees. Section 7 designs FP-
TAS’s for MaxMstCut, MinLstCut and MaxLstCut in unrooted trees. Finally
Section 8 makes some concluding remarks.

2 Preliminaries

Let $\mathbb{R}^+$ denote the set of positive reals, and let $\mathbb{R}^* = \mathbb{R}^+ \cup \{0\}$. We may denote
by $V(G)$ and $E(G)$ the sets of vertices and edges of an undirected graph $G$, respectively. For a graph $G = (V, E)$ and a subset $X \subseteq V$, let $G[X]$ denote the
subgraph of $G$ induced by the vertices in $X$, and let $E(X, V - X)$ denote the set
of edges with one endpoint in $X$ and the other in $V - X$. For a vertex weight
$w : V \to \mathbb{R}^*$ and an edge cost $c : E \to \mathbb{R}^*$, let $w(X)$ denote $\sum_{v \in X} w(v)$ and
c($X$) denote $\sum_{e \in E(X, V - X)} c(e)$, respectively.

Given an unrooted tree $G$ and a vertex $r$ in it, we use $G_r$ to denote the
rooted tree obtained from $G$ by choosing $r$ as the root. In a rooted tree, let $D(v)$
(resp., $A(v)$) denote the set of descendants (resp., ancestors) of a vertex $v$, where
$v \in D(v)$ and $v \in A(v)$. Denote the edge sets in the induced graphs $G[D(v)]$ and
$G[A(v)]$ by $E_D(v)$ and $E_A(v)$, respectively. Let $e_p(v)$ denote the edge joining $v$
and its parent. For each edge $e = (u, v)$, where $u$ is the parent of $v$, define $D(e)$
and $E_D(e)$ to be $D(v)$ and $E_D(v)$, respectively.

A set $X$ is called root-connecting if $G[V - X]$ is connected and contains the
root. We use $n$ and $l$ to denote the number of vertices and the number of leaves
(i.e., vertices of degree 1) in the graph respectively. To distinguish edge weight
with vertex weight, we say edge cost instead of edge weight.

A special case of our cut problems is that the weight of vertices in the graph
can only be 0 or 1. We will call the problems under this constraint the terminal
case. We may call vertices $v$ with $w(v) = 1$ terminals, and use $T$ to denote the
set of terminals in the graph and let $t = |T|$.

3 Computational Complexity

3.1 In General Graphs

First, we study the hardness of our cut problems in general graphs. We consider
the terminal case of the problems in nonnegative edge-cost graphs without the
constraint (i) on the connectivity of $G[V - X]$, where we can assume that given graphs are complete by introducing a new edge with cost 0 to each pair of non-adjacent vertices. Thus we treat our problems in complete graphs. For terminal cases, it may also be interesting to study the exact version of the problems: cutting exact $k$ terminals away from the graph by minimizing (maximizing) the cost of the cut. We use MinExtCut and MaxExtCut to denote these two problems. They will also be considered later.

Since the maximum cut problem is a well-known NP-hard problem, we know that terminal MaxMstCut, MaxLstCut and MaxExtCut are NP-hard. We will pay more attention to the minimum version of the problems. In both terminal MinMstCut and MinLstCut, the target $k$ should be less than $|T|/2$, otherwise it is no meaning to distinguish them.

It is easy to see that when $k$ is a constant, terminal MinMstCut, MinLstCut and MinExtCut are polynomial-time solvable as we can exhaustively search for all $k$ (or at most $k$)-terminal groups (For MinLstCut, we will find a minimum cut in each graph obtained by contracting a $k$-terminal group into a single terminal). When $k$ is part of input, it is also easy to show the NP-hardness of terminal MinExtCut by reducing from the balanced cut problem [1]. Marx [11] also studied the hardness of terminal MinExtCut from the Parameterized Complexity view. He proved that the problem is W[1]-hard, even when both $k$ and size of the corresponding cut set are taken as parameters, which shows the corresponding parameterized problem unlikely has any parameterized algorithm.

Terminal MinExtCut remains NP-hard even if $k \leq |T|/2$, and this implies that terminal MinLstCut is NP-hard, because a terminal MinExtCut instance with $k \leq |T|/2$ can be polynomially reduced to a terminal MinLstCut instance by adding heavy edges between every two terminals so that any optimal solution of the terminal MinLstCut instance cuts exactly $k$ vertices away from the root.

On the other hand, (terminal) MinMstCut is polynomially solvable by using an algorithm due to Nagamochi [12] for computing families of extreme sets (see Appendix A).

3.2 In Weighted Trees

Since the cut problems in general graphs are hard, we pay more attention to the scenarios where the graph is a tree. For the terminal case of the problems in trees, they can be solved in polynomial time by using dynamic programming (see Section 4). When vertex weight is nonnegative real number, MinLstCut and MaxMstCut in stars (trees having at most one vertex of degree $> 1$) are equal to the classical knapsack problem (each leaf in the star is corresponding to an item in the knapsack problem). For the hardness of MaxLstCut, we reduce another well-known NP-hard problem, the subset sum problem, to it. The subset sum problem is a special case of the knapsack problem, in which the profit of each item is equivalent to the weight of the item. Given a subset sum instance $(w(1), w(2), \ldots, w(n), h)$ with $n$ items, where $w(i)$ is the weight of item $i$ and $h$ is the capacity of the knapsack, we construct a tree $T$ with $2n + 1$ vertices in this way: Each item $i$ in the subset sum problem is corresponding to two vertices $a_i$,
and \( b_i \) in the tree. The weight of \( b_i \) is 0 and the weight of \( a_i \) is \( w(i) \). There is also an auxiliary vertex \( v \) with weight 0 in \( T \). There is an edge between \( a_i \) and \( b_i \) with cost \( w(i) \) and an edge between \( a_i \) and \( v \) with cost 0. The target \( k \) in the tree is set to be \( W - h \), where \( W \) is the total weight of items. Note that the cut set corresponding to an optimal solution to MaxLstCut in \( T \) will contain exact one edge in \( \{a_i, b_i, a_i, v\} \) for each \( i \). Then if item \( i \) is selected into the knapsack, we include edge \( a_i, b_i \) into the cut set. Otherwise, we include edge \( a_i, v \) into the cut set. For each optimal solution to MaxLstCut we can also construct a solution to the knapsack problem. Therefore, we can prove the correctness of this reduction.

MinMstCut is polynomial-time solvable in general graphs even without the constraint on the vertex weight. We further show that the problem can be solved in linear time when the graph is restricted to a tree. Note that the edge cost and vertex weight are nonnegative. The cut set corresponding to an optimal solution only contains one edge. We only need to identify a feasible edge with minimum cost. An edge \( e \) in a rooted tree is a feasible edge if \( w(D(e)) \leq k \), and an edge \( e \) in an unrooted tree is a feasible edge if at least one of the two components of \( G - e \) is of total vertex weight \( \leq k \). We can find all feasible edges in a DFS (For an unrooted tree, we arbitrarily select a vertex as the root and consider if \( \min \{w(D(v)), W - w(D(v))\} \leq k \) for each vertex \( v \)). In the DFS, we only need to remember the feasible edge with minimum cost among all visited edges. Therefore, we can solve the problem in linear time and space. We will not discuss MinMstCut in the rest of the paper.

4 Pseudo Polynomial-time Algorithms

In this section, we give dynamic programming algorithms for our problems in rooted trees, which are pseudo polynomial-time algorithms and will be used to derive FPTAS’s. One technique we used to design the dynamic programming algorithms is called the “left-right” method for trees in [7]. To get the pseudo polynomial-time algorithms, we assume that all edge cost are integers in this section. In fact, our algorithm will find an optimal solution \( X \) only when \( c(X) \leq q \) holds for a prescribed threshold \( q \).

We order all the leaves in the tree according to a DFS (first visited has small index). Let \( L = \{z_1, z_2, \ldots, z_l\} \), \( l = |L| \), be the set of leaves, where leaves \( z_1, z_2, \ldots, z_l \) appear from left to right in this order (see Fig. 1 for an illustration). To construct a root-connecting set \( X \), if a vertex \( v \) does not belong to \( X \) (i.e., \( v \) is not allowed to be cut away from the rooted tree), then all the edges in the path from \( v \) to the root are not allowed to be deleted.

Let \( V' \) be the set of non-root vertices in the tree. For \( 0 \leq i \leq l \) and \( 0 \leq j \leq q \), define \( \mathcal{X}(i, j) \) to be the set of all root-connecting sets \( X \subseteq V' \) that satisfy two constraints: \( L \cap X \subseteq \{z_1, z_2, \ldots, z_i\} \) (only the first \( i \) leaves are allowed to be cut away); and \( c(X) = j \). We set

\[
OPT(i, j) = \begin{cases} 
\min \{w(X) \mid X \in \mathcal{X}(i, j)\} & \text{for MaxMstCut} \\
\max \{w(X) \mid X \in \mathcal{X}(i, j)\} & \text{for MaxLstCut and MinLstCut}.
\end{cases}
\]
Let \( X(i, j) \) store a set \( X \in \mathcal{X}(i, j) \) attaining \( w(X) = OPT(i, j) \), where we let \( X(i, j) = \perp \) and \( OPT(i, j) = \infty \) if \( X(i, j) = \emptyset \). Clearly for the maximum integer \( c_0 \) such that \( OPT(l, c_0) \leq k \), \( X(l, c_0) \) is an optimal solution to MaxMstCut.

Similarly for the maximum (resp., minimum) integer \( c_0 \) such that \( OPT(l, c_0) \geq k \), \( X(l, c_0) \) is an optimal solution to MaxLstCut (resp., MinLstCut).

We maintain a table of size \((l + 1) \times (q + 1)\) to store \( OPT(i, j) \) and \( X(i, j) \) for \( 0 \leq i \leq l \) and \( 0 \leq j \leq q \). To get a recursive formula among these, we consider a partition of the edge set \( E \) into \( l \) subsets \( P_i = E_A(z_i) - E_A(z_{i+1}) \), \( i = 1, 2, \ldots, l \), where we regard \( E_A(z_{l+1}) \) as \( \emptyset \). Note that \( P_i \) is the set of edges \( e \) such that deleting \( e \) cuts \( z_i \) away without cutting \( z_{i+1} \) away (see Fig. 1 for an illustration of \( P_i \)). Then, for any edge \( e \in P_i \), deleting \( e \) cuts away leaves \( z_0, z_{i+1}, \ldots, z_i \) with consecutive indices. Let \( \alpha(e) \) be the smallest index \( a \) of the leaves in \( V(e) \). Then all values \( OPT(i, j) \) and \( X(i, j) \) for MaxMstCut can be computed by the recursive formula in Fig. 2.

Algorithm DP for MaxLstCut and MinLstCut can be obtained by replacing the if condition in Step 2 with \( OPT(\alpha(e) - 1, j - c(e)) + w(e) > OPT(i, j) \) and modifying the choice of \( c_0 \) in Step 3.

**Lemma 1.** For a rooted tree \( G \) with nonnegative integer edge cost and nonnegative vertex weight, Algorithm DP\((G, k, q)\) can be implemented to run in \( O(qn) \) time and \( O(ql + n) \) space.

Each iteration for an edge \( e \in P_i \) in the fourth line of Step 2 can be executed in \( O(1) \) time by representing \( X = X(i, j) \) as the edge set \( E(X, V - X) \) and updating \( E(X, V - X) \) by adding the edge \( e \). Since \( \{P_i \mid i = 1, 2, \ldots, l\} \) is a partition of the edge set \( E \), each edge is considered only once for each fixed \( j \) in the algorithm. Since \( j \) runs from 1 to \( q \), the running time of the algorithm is \( O(qn) \). The space of the algorithm is \( O(ql + n) \).

Since DP\((G, k, q)\) only computes \( OPT(i, j) \) for \( j \) from 0 to \( q \), when the cost of an optimal solution is greater than \( q \), DP\((G, k, q)\) cannot find an optimal solution.
Algorithm DP($G, k, q$)
Input: A tree $G = (V, E)$ rooted at a vertex $r$ with nonnegative integer edge cost $c$ and nonnegative vertex weight $w$, a number $k \geq 0$, and an integer $p \geq 0$.
Output: A maximum cost root-connecting set $X$ such that $w(X) \leq k$ and $c(X) \leq p$, if any, or $\bot$ otherwise.

1. For $j = 1$ to $q$, do $OPT(0, j) \leftarrow \infty$ and $X(0, j) \leftarrow \bot$.
   For $i = 0$ to $l$, do $OPT(i, 0) \leftarrow 0$ and $X(i, 0) \leftarrow \emptyset$.
2. For $i = 1$ to $l$, do
   For $j = 1$ to $q$, do
     $OPT(i, j) \leftarrow OPT(i - 1, j)$ and $X(i, j) \leftarrow X(i - 1, j)$.
     For each $e \in P_i$ with $c(e) \leq j$, do
       If $OPT(\alpha(e) - 1, j - c(e)) + w(e) < OPT(i, j)$, then
         $OPT(i, j) \leftarrow OPT(\alpha(e) - 1, j - c(e)) + w(D(e))$ and
         $X(i, j) \leftarrow X(\alpha(e) - 1, j - c(e)) \cup D(e)$.
   3. Let $c_0$ be the maximum integer such that $OPT(l, c_0) \leq k$.
   4. Return $X(l, c_0)$.

Fig. 2. Algorithm DP($G, k, q$) for MaxMstCut

Note that the recursion in Fig. 2 can be rewritten to do over vertex weight instead of edge cost by maintaining a table of size $(l + 1) \times (k + 1)$ to store $OPT'(i, j)$ (and also $X'(i, j)$) for $0 \leq i \leq l$ and $0 \leq j \leq k$, where $OPT'(i, j) = \max\{c(X) \mid X \in X'(i, j)\}$ and $X'(i, j)$ is the set of all root-connecting sets $X \subseteq V'$ that satisfy two constraints: $L \cap X \subseteq \{z_1, z_2, \ldots, z_l\}$ and $w(X) = j$. Then we can get running time bound $O(kn)$ time and space bound $O(kl + n)$ for the modified algorithm (We also need to assume that all vertex weight are integers for this case). This also implies that the terminal case of our cut problems can be solved in $O(kn)$ time and $O(kl + n)$ space.

5 FPTAS for MaxMstCut in Rooted Trees

First, we consider MaxMstCut in rooted trees and then extend our results to the problem in unrooted trees in Section 7.

5.1 Preprocesses

We show some conditions on edges that can be ruled out from a given instance of MaxMstCut. An edge $e$ is called an overload edge if $w(D(e)) > k$, i.e., the total weight of vertices cut away from the tree by deleting it exceeds $k$. Clearly, we can simply contract all overload edges to form the new rooted tree. A non-leaf edge $e$ is called a dominated edge if $c(e)$ is not larger than the total of $c(e')$ of the child-edges $e'$ of $e$ (i.e., the edges in $E_D(e)$ adjacent to $e$). Thus, for any root-connecting set $X$ with a dominated edge $e = (u, v) \in E(X, V - X)$, where $v \in X$, $X' = X - \{v\}$ satisfies $c(X') \geq c(X)$ and $w(X') \leq w(X)$ by nonnegativeness
of $c$ and $w$, indicating that there is an optimal solution $X^*$ to MaxMstCut such that $e \notin E(X, V - X)$. Hence we can also safely contract all dominated edges. This can be done in a bottom-up manner from leaves toward the root in linear time. Observe that, in a rooted tree with no dominated edges, for each non-root vertex $v$, $D(v)$ has the maximum cut cost $c(D(v))$ over all root-connecting sets $X \subseteq D(v)$.

5.2 A $\frac{1}{3}$-Approximation Algorithm

To design our FPTAS for MaxMstCut, we first give an $O(n \log l)$-time $\frac{1}{3}$-approximation algorithm. The idea of the algorithm comes from a well-known greedy approximation algorithm for the knapsack problem, which is to greedily select items of the most dense. For each non-root vertex $v$ with $D(v) > 0$, the density of $v$ is defined to be $\rho(v) = c(e)/w(D(v))$, where $c = c_p(v)$. The simple approximation algorithm is presented in Fig. 3, where through out the computation, we use the density $\rho(v)$ of vertices $v$ in the graph obtained in Step 1, and never recompute new densities even after fixing some vertices as part of the current root-connecting set $Y$.

**Algorithm APPRO($G, k$)**

**Input:** A rooted tree $G = (V, E)$ with nonnegative edge cost $c : E \to \mathbb{R}^*$ and nonnegative vertex weight $w : V \to \mathbb{R}^*$ and a number $k > 0$.

**Output:** A $\frac{1}{3}$-approximation solution to MaxMstCut in $G$.

1. Contract all overload and dominated edges. Let $V'$ be the set of non-root vertices in the resulting tree. Compute the density $\rho$.
2. $Y \leftarrow$ the union of sets $D(v)$ with $w(D(v)) = 0$ and $v \in V'$.
3. While $V' - Y \neq \emptyset$, do
   - Select $v \in V' - Y$ such that $\rho(v)$ is maximized (possibly $Y \cap D(v) \neq \emptyset$).
   - If $w(Y \cup D(v)) \leq k$, then $Y \leftarrow Y \cup D(v)$.
   - Else return one of $Y$ and $D(v)$ that attains max{$c(Y), c(D(v))$}, and halt.
4. Return $Y$.

**Fig. 3. Algorithm APPRO($G, k$)**

Lemma 2. Algorithm APPRO($G, k$) is an $O(n \log l)$-time $\frac{1}{3}$-approximation algorithm for MaxMstCut in rooted trees with nonnegative edge cost and nonnegative vertex weight.

Proof. Since we have contracted all dominated edges, we know that there is an optimal root-connecting set $X^*$ contains all sets $D(v)$ with $w(D(v)) = 0$ in the tree. Then we can simply put them into the solution set directly. Next, if the algorithm stops at Step 4, it will return an optimal solution $Y = V'$, since $c(V')$ attains the maximum cut cost $c(X)$ over all root-connecting sets $X$. 
Otherwise the algorithm will return \( Y \) or \( D(v') \) in Step 3, where \( v' \) is the last vertex being selected in the algorithm. Note that \( D(v') \) is excluded from \( Y \) in the algorithm. Thus \( Y \) is a feasible solution when the algorithm halts. \( D(v') \) is also a feasible solution, because all overload edges are contracted in Step 1. Assume that \( X^* \) is an optimal solution. We partition \( Y \) and \( X^* \) into disjoint subsets: \( Y = D(v_1) \cup D(v_2) \cup \cdots \cup D(v_y) \) and \( X^* = D(u_1) \cup D(u_2) \cup \cdots \cup D(u_x) \), where \( v_i \) is not a descendent of any vertex in \( Y - \{v_1\} \) and \( u_i \) is not a descendent of any vertex in \( X^* - \{u_i\} \). We further assume that \( \{v_1, \ldots, v_y\} \cap \{u_1, \ldots, u_x\} = \emptyset \).

Let \( X_1 = D(u_1) \cup D(u_2) \cup \cdots \cup D(u_h) \) and \( X_2 = X^* - X_1 = D(u_{h+1}) \cup \cdots \cup D(u_x) \). Obviously, \( X_1 \subseteq Y \) and \( X_1 \) is still a feasible solution. By the nonnegativeness of \( c \), \( c(Y) \geq c(X_1) \). Since at the time when the algorithm selects \( v' \), no vertex in \( \{u_{h+1}, \ldots, u_x\} \) has been selected by the greedy strategy, we know that \( \rho(v_i) \geq \rho(v') \geq \rho(u_j) \) holds for any \( i = 1, \ldots, y \) and \( j = h + 1, \ldots, x \). Since \( w(Y \cup D(v')) > k \geq w(X_2) \), this implies that \( c(Y \cup D(v')) > c(X_2) \). Hence we have \( c(X^*) = c(X_1) + c(X_2) \leq c(Y) + c(Y \cup D(v')) \leq 2c(Y) + c(D(v')) \). Therefore, one of \( c(Y) \) and \( c(D(v')) \) must have cost at least \( \frac{1}{2} c(X^*) \).

As for the running time, we only need to show that Step 3 can be done in \( O(n \log l) \) time. First of all, we compute the densities of all vertices in the tree in a DFS. If a vertex \( v \) has an ancestor \( u \) such that \( \rho(v) < \rho(u) \), we say that \( v \) is a weak vertex and \( e_p(v) \) a weak edge. Note that the algorithm will never select a weak vertex in Step 3. We can reduce the tree by contracting all weak edges. Detecting and contracting all weak edges can be done in a DFS. After reducing the tree, the tree has the monotonicity property: on any path from the root to a leaf, the vertex densities give a nonincreasing sequence. Then we can sort all the vertices according to their densities in \( O(n \log l) \) time as we do in Mergesort. After sorting the vertices, we only need to select vertices according to the sorting list. Then we can do Step 3 in \( O(n \log l) \) time.

**Note.** This greedy algorithm can get approximation ratio \( \frac{1}{2} \) for the knapsack problem. Note that when the graph is a star, in the above proof we will have that \( c(X_2) \leq c(Y \cup \{v'\} - X_1) \) (since \( w(Y \cup \{v'\} - X_1) > k - w(X_1) \geq w(X_2) \) and \( \rho(v_i) \geq \rho(v') \geq \rho(u_j) \) for any \( i = 1, \ldots, y \) and \( j = h + 1, \ldots, x \). Then \( c(X^*) = c(X_1) + c(X_2) \leq c(Y) + c(\{v'\}) \). At least one of \( c(Y) \) and \( c(\{v'\}) \) must have cost at least \( \frac{1}{2} c(X^*) \). Ibarra and Kim [6] also introduced an \( O(n^2) \)-time \( \frac{1}{2} \)-approximation algorithm for MaxMstCut in rooted trees. In their algorithm, they will update the densities of vertices when each time selecting a vertex of maximum density and deleting it from the tree. We will use our \( \frac{1}{2} \)-approximation algorithm instead of their \( \frac{1}{2} \)-approximation algorithm, because we cannot improve the running time bound of our FPTAS by improving the approximation ratio from \( \frac{1}{2} \) to \( \frac{1}{2} \), but will worsen the running time bound if the running time of the constant approximation algorithm changes from \( O(n \log l) \) to \( O(n^2) \).

---

\(^3\) The correctness of the algorithm (Theorem 2 in [6]) was proved by assuming an important property (the fourth line of the proof of Theorem 2), the complete proof of which was omitted.
5.3 The FPTAS

Next, we design the FPTAS for MaxMstCut. Let \( R = \text{APPRO}(G, k) \). Algorithm \( \text{DP}(G, k, 3c(R)) \) will find an optimal solution for our problem (assume that all edge costs are integers). If the cost of each edge is a small number, i.e., bounded by a polynomial of \( n \), then \( \text{DP}(G, k, 3c(R)) \) runs in polynomial time. To obtain an FPTAS, we scale the cost of each edge down to be bounded by a polynomial of \( n \) and use Algorithm \( \text{DP} \) on the new instance to get a solution. By scaling with respect to the desired error \( \epsilon \), we can get a solution with cost within \((1 - \epsilon) c(X^*) \) and \( c(X^*) \) in polynomial time with respect to both \( n \) and \( 1/\epsilon \), where \( X^* \) is an optimal solution to the original instance \((G, k)\). The FPTAS for MaxMstCut is described in Fig. 4.

### Algorithm Fptas1\((G, k, \epsilon)\)

**Input:** A rooted tree \( G = (V, E) \) with nonnegative edge cost \( c : E \to \mathbb{R}^* \) and nonnegative vertex weight \( w : V \to \mathbb{R}^* \), and two numbers \( k \geq 0 \) and \( 1 > \epsilon > 0 \).

**Output:** A \((1 - \epsilon)\)-approximation solution to MaxMstCut.

1. \( R \leftarrow \text{APPRO}(G, k) \) and \( g \leftarrow c(R) \).
2. If \( g = 0 \), return \( R \) as an optimal solution and halt.
3. \( Q \leftarrow \epsilon g / l \) and replace \( c(e) \) with \( c'(e) = \lfloor c(e)/Q \rfloor \) to get \( G' \).
4. Return \( \text{DP}(G', k, [3g/Q]) \).

**Fig. 4. Algorithm Fptas1\((G, k, \epsilon)\)**

**Theorem 1.** For MaxMstCut in a rooted tree with nonnegative edge cost and nonnegative vertex weight, algorithm \( \text{Fptas1}(G, k, \epsilon) \) runs in \( O(n/\epsilon) \) time and \( O(l^2/\epsilon + n) \) space, and returns a root-connecting set \( Y \) such that

\[
\forall \ne c(Y) \geq (1 - \epsilon) c(X^*),
\]

where \( X^* \) is an optimal solution to MaxMstCut.

**Proof.** Since \( \text{APPRO}(G, k) \) runs in \( O(n \log l) \) time and \( O(n) \) space, and \( \text{DP}(G', k, [3g/Q]) \) runs in \( O(3gn/Q) = O(n/\epsilon) \) time and \( O(3g/l + n) = O(l^2/\epsilon + n) \) space, we get the running time and space bounds.

Next, we prove (1). It is clear that for any edge \( e \), \( 0 \leq c(e) - Qc'(e) < Q \).

Since the cardinality of any cut is at most the number \( l \) of leaves in \( G \), any root-connecting set \( X \neq \emptyset \) satisfies \(|E(V - X, X)| \leq l \) and

\[
0 \leq c(X) - Qc'(X) < lQ.
\]

Since \( c'(X^*) \leq \lfloor c(X^*)/Q \rfloor \leq [3g/Q] \) holds, \( \text{DP}(G', k, [3g/Q]) \) finds an optimal solution \( Y \) for the scaled instance, which satisfies \( c'(X^*) \leq c'(Y) \leq [3g/Q] \).

Since \( c'(Y) \geq c'(X^*) \geq g \), we have

\[
c(Y) \geq Qc'(Y) \geq Qc'(X^*) \geq c(X^*) - lQ = c(X^*) - \epsilon g \geq (1 - \epsilon)c(X^*).
\]
FPTAS’s for MinLstCut and MaxLstCut in Rooted Trees

We cannot directly extend the FPTAS for MaxMstCut in Section 5 to an FPTAS for MinLstCut or MaxLstCut in rooted trees, because we have no similar constant factor algorithm as Appro\((G, k)\) for MinLstCut or MaxLstCut. To get a constant factor algorithm, we will use a “rounding” technique [14]. This technique is used as a basic trick for deriving FPTAS’s for many problems. We will also use a new technique, called “binary search”, to get the final FPTAS. In this section, we design the FPTAS for MinLstCut, which can also be modified to solve MaxLstCut.

6.1 The Preliminary Algorithm

First we present an algorithm \textsc{Scaling} for MinLstCut. Whether a given MinLstCut instance \((G, k)\) has an optimal solution \(X^*\) with \(c(X^*) = 0\) or not can be tested in linear time by checking if \(w(V') \geq k\) for the set \(V'\) of non-root vertices in the tree obtained by contracting all edges with positive costs. We here assume that \(c(X^*) \geq c_{\min}\) for the minimum \(c_{\min}\) of positive edge costs. Given an instance such that \(p \leq c(X^*)\) for a positive number \(p\) and a prescribed number \(\varepsilon > 0\), algorithm \textsc{Scaling} will run in \(O(\ln/\varepsilon)\) time and \(O\left(\frac{l^2}{\varepsilon} + n\right)\) space and return a feasible solution with value \(c(Y)\) satisfying \((1 + \varepsilon)c(X^*) \geq c(Y)\), if \(c(X^*) \leq 2p\), or \(Y = \perp\) if \(c(X^*) > 2p\).

\begin{algorithm}
\textbf{Algorithm} \textsc{Scaling}(G, k, \varepsilon, p)
\begin{itemize}
    \item \textbf{Input:} A rooted tree \(G = (V, E)\) with nonnegative edge cost \(c : E \rightarrow \mathbb{R}^*\) and nonnegative vertex weight \(w : V \rightarrow \mathbb{R}^*\), and three numbers \(k \geq 0, \varepsilon > 0\) and a number \(p \geq c_{\min}\).
    \item \textbf{Output:} A feasible solution to MinLstCut, if any, or \(\perp\) otherwise.
\end{itemize}
\begin{enumerate}
    \item \(Q \leftarrow \varepsilon p / l\) and replace \(c(e)\) with \(c'(e) = \lfloor c(e) / Q \rfloor\) to get \(G'\).
    \item Return DP\((G', k, \lfloor 2p / Q \rfloor)\).
\end{enumerate}
\end{algorithm}

By the definition of DP\((G', k, \lfloor 2p / Q \rfloor)\) for MinLstCut, it will return a feasible solution \(Y\) such that \(c'(Y) \leq \lfloor 2p / Q \rfloor\) if any, or \(\perp\) otherwise. Note that if it returns \(\perp\) then we know \(c(X^*) > 2p\) since \(c(X^*) \leq 2p\) implies \(c'(X^*) \leq \lfloor 2p / Q \rfloor\). Assume \(p \leq c(X^*) \leq 2p\) and let \(Y = \text{Scaling}(G, k, \varepsilon, p)\). Analogously with the analysis of Theorem 1, we can prove that directly
\[
    c(Y) \leq Q \cdot c'(Y) \leq Q \cdot c'(X^*)
    \leq c(X^*) + lQ = c(X^*) + \varepsilon p.
\]
Hence if \(c(X^*) \geq p\) then \(c(Y) \leq (1 + \varepsilon)c(X^*)\) holds. Therefore, \textsc{Scaling}(G, k, \varepsilon, p) works as claimed.
To get an approximation scheme, we need to compute a value \( p^* \) such that
\[ p^* \leq c(X^*) \leq 2p^* \] and call \( \text{SCALING}(G, k, \varepsilon, p^*) \). To detect such \( p^* \), we repeatedly check intervals \([c_{\min}, 2c_{\min}], [2c_{\min}, 4c_{\min}], \ldots\) as in the loop in Fig. 6 to compute \( p^* \).

\begin{enumerate}
  \item Let \( p \leftarrow c_{\min} \);
  \item If \( \text{SCALING}(G, k, \varepsilon = 1, p) \neq \perp \), return \( p \) as \( p^* \) and halt;
  \item Otherwise, let \( p \leftarrow 2p \) and go to Step 2.
\end{enumerate}

**Fig. 6. Computing \( p^* \)**

It is clear that the process in Fig. 6 will return \( p^* \) by calling \( \text{SCALING}(G, k, \varepsilon = 1, p) \) for at most \( \lceil \log(c(X^*)/c_{\min}) \rceil \) times. Therefore, we can compute \( p^* \) in \( O(nl\log(c(X^*)/c_{\min})) \) time and \( O(l^2 + n) \) space.

**Theorem 2.** There is an \( O(nl(1/\varepsilon + \log(c(X^*)/c_{\min}))) \)-time and \( O(l^2/\varepsilon + n) \)-space algorithm that computes a solution with error \( \varepsilon \) for MinLstCut in a rooted tree with nonnegative edge cost and nonnegative vertex weight.

Since the running time of the approximation algorithm depending on \( c(X^*) \), it is not a real polynomial-time algorithm. Next, we further improve the running time bound from \( O(nl(1/\varepsilon + \log(c(X^*)/c_{\min}))) \) to \( O(nl(1/\varepsilon + \log n)) \).

### 6.2 Binary Search and Further Improvement

In Fig. 6, we set \( p \) as 1 in the beginning to compute \( p^* \), and the algorithm may run \( \lceil \log(c(X^*)/c_{\min}) \rceil \) loops. If we get a value \( p_1 \) such that \( p_1 \leq c(X^*) \leq np_1 \) and set \( p \) as \( p_1 \) in the first step, then we can compute \( p^* \) in \( O(nl\log n) \) time. We will use a technique called “binary search” to compute such a value \( p_1 \) such that \( p_1 \leq c(X^*) \leq np_1 \) or a value \( p'_1 \) such that \( p'_1 \leq c(X^*) \leq 2p'_1 \) in \( O(nl\log n) \) time.

We sort the positive numbers in \( \{c(e) > 0 \mid e \in E\} \) into an increasing sequence \( c_1 \leq c_2 \leq \cdots \leq c_h \) in \( O(nl\log n) \) time, where \( c_1 = c_{\min} \leq c(X^*) \). We first detect an index \( j \) such that \( c_j \leq c(X^*) \leq 2c_{j+1} \) by applying algorithm \( \text{SCALING} \) in \( O(l^2) \) times. For a given number \( p > 0 \), let \( Y \leftarrow \text{DP}(G, c', w, k, [2p/Q]) \) for \( Q = p/l \) and \( c' = \lfloor c/Q \rfloor \). As observed, if \( Y = \perp \), then \( c(X^*) > 2p \), else \( Y \) is a feasible solution such that \( c(Y) \leq c(X^*) + p \), from which we know either \( c(X^*) \geq p \) (when \( c(Y) \geq 2p \)) or \( c(X^*) \leq c(Y) < p \) (when \( c(Y) < 2p \)). Overall, the algorithm \( \text{DP} \) tells us either \( p \leq c(X^*) \) or \( c(X^*) < 2p \). Staring with the initial interval \( I = [1, 2, \ldots, h] \), we repeat the following until the current interval contains only one index: Choose the middle index \( i \) in the current interval \( I = [a, a + 1, \ldots, b] \) and test whether \( c_i \leq c(X^*) \) or \( c(X^*) < 2c_i \). Then reset \( I \) to be interval \([i + 1, i + 2, \ldots, b]\) for the former case, and set \( I \) to be \([a, a + 1, \ldots, i - 1]\) for the latter. The procedure finds the above index \( j \) in \( O(nl\log n) \) time and \( O(l^2 + n) \) space.
For such an index $j$, we can find a $(1 + \varepsilon)$-approximation by taking a better one from the following two solutions $Y_1$ and $Y_2$. (i) $c_{j+1} \leq c(X^*) \leq 2c_{j+1}$: Compute $Y_1 = \text{Scaling}(G, k, \varepsilon, c_{j+1})$ in $O(nl/\varepsilon)$ time and $O(l^2 + n)$ space; (ii) $c_j \leq c(X^*) < c_{j+1}$: No edge with cost greater than $c_j$ is used in $E(X^*, V - X^*)$. Contract all those edges to obtain a new tree, wherein the total cost of all edges is not greater than $jc_j$, and hence $c(X^*) \leq jc_j$. By setting $p$ to be $c_j$ in the algorithm in Fig. 6, we can find $p^*$ with $p^* \leq c(X^*) \leq 2p^*$ by at most $\log(c(X^*)/c_j) \leq \log n$ iterations. Let $Y_2 = \text{Scaling}(G, k, \varepsilon, p^*)$. Although we do not know which of (i) and (ii) occurs, $Y_1$ or $Y_2$ gives a $(1 + \varepsilon)$-approximation, and the required complexity is $O(nl(1/\varepsilon + \log n))$ time and $O(l^2/\varepsilon + n)$ space.

The FPTAS for MinLstCut in this section can be directly modified to a similar FPTAS for MaxLstCut, which preserves the approximation ratio. We here only mention that a similar binary search can be obtained for MaxLstCut to reduce the time complexity from $O(nl(1/\varepsilon + \log(c(X^*)/c_{\min})))$ to $O(nl(1/\varepsilon + \log n))$. For this, it suffices to show that, for a given number $p > 0$, we can get $p \leq c(X^*)$ or $c(X^*) < 2p$ in $O(nl)$ time and $O(l^2/\varepsilon + n)$ space. Compute $Y^* \leftarrow \text{DP}(G', k, [p/Q])$ for $Q = p/l$ and $c' = [c/Q]$ in $G'$. As observed, if $Y = \perp$, then $c(X^*) > p$, else $Y$ is a feasible solution such that $c(Y) \geq c(X^*) - p$, from which either $c(X^*) < 2p$ (when $c(Y) < p$) or $c(X^*) \geq c(Y) \geq p$ (when $c(Y) \geq p$) holds. Hence we know $p \leq c(X^*)$ or $c(X^*) < 2p$.

**Theorem 3.** There is an $O(nl(1/\varepsilon + \log n))$-time and $O(l^2/\varepsilon + n)$-space algorithm that computes a solution with error $\varepsilon$ for MinLstCut and MaxLstCut in a rooted tree with nonnegative edge cost and nonnegative vertex weight.

## 7 FPTAS’s for Unrooted Trees

If the tree is an unrooted tree, we still can solve our problems by choosing each vertex as a root and solving the $n$ problems in the rooted trees. But the running time bound will increase by a factor of $\Theta(n)$. There is a simple divide-and-conquer method to improve the result. By choosing a vertex as $r$ in the tree, we can either solve the problem in the tree rooted at $r$ or solve several (constrained) problems in subtrees obtained by separating at $r$ (See Fig. 7 for the illustration). Note that in each subtree we still keep a copy of $r$, called $r_i$, with the weight being the sum of the weight of all the vertices separated away from this subtree (including vertex $r$ itself). In each subtree we should find a solution under the constraint that the new vertex $r_i$ is cut away from the tree. To reduce the running time effectively, each we can select a balanced separator as the considered vertex $r$ (Each subtree obtained after deleting $r$ has at most half of number of vertices). With this method, we can solve the problems in unrooted trees by increasing running time bound by only a factor of $\Theta(\log n)$. In this section, we present an algorithm that iteratively chooses a leaf-balanced separator of the tree and solves the problem in a divide-and-conquer way. With some nontrivial analysis, we show that this method can solve our problems in unrooted trees without increasing the running time. First of all, we give the precise definitions of constrained problems...
and leaf-balanced separator. We design linear-time algorithms for our problems in paths (unrooted trees with only two leaves), which will be used as subalgorithms in our FPTAS’s.

7.1 Constrained Problems and Leaf-Balanced Separator

Given a set of inputs of MaxMstCut (MinLstCut, or MaxLstCut) additionally with a prescribed set $L'$ of leaves, the constrained MaxMstCut (MinLstCut, or MaxLstCut) requires to find a solution to cut away all leaves in $L'$ (possibly with other leaves) from the given tree, where $L'$ may be empty. We design an algorithm $\text{DP}'(G, L', k, q)$ for the constrained MaxMstCut (MinLstCut, or MaxLstCut). Thus for $L' = \emptyset$, $\text{DP}'(G, \emptyset, k, q)$ is exactly $\text{DP}(G, k, q)$.

The algorithm $\text{DP}'(G, L', k, q)$ can be obtained by modifying $\text{DP}(G, k, q)$ as follows. In the fifth line of Step 2 in Fig. 2, $\text{DP}(G, k, q)$ initializes $OPT(i, j)$ and $X(i, j)$ by

$$OPT(i, j) \leftarrow OPT(i - 1, j), \ X(i, j) \leftarrow X(i - 1, j).$$

In the modified algorithm $\text{DP}'$, the same initialization is used if leaf $z_i \notin L'$. On the other hand, if $z_i \in L'$, we initialize them by

$$OPT(i, j) \leftarrow \infty, \ X(i, j) \leftarrow \bot$$

for MaxMstCut and initialize them by

$$OPT(i, j) \leftarrow -1, \ X(i, j) \leftarrow \bot$$

for MinLstCut and MaxLstCut.

By this modification, for MaxMstCut, all cuts that keep $z_i \in L'$ in the tree are assigned an infinite cost. Thus the algorithm computes a minimum cost solution of the constrained MaxMstCut if a feasible solution exists. For MinLstCut and MaxLstCut, the modified algorithm $\text{DP}'$ also works as claimed. Note that the running time bound of $\text{DP}'$ does not change. We can also design FPTAS’s for our constrained problems with the same running time bound.

![Fig. 7. Separation of a tree at vertex $r$](image-url)
use $\text{Fptas}^1(G, L', k, \varepsilon)$ (resp., $\text{Fptas}^2(G, L', k, \varepsilon)$) to denote the corresponding FPTAS in Section 5 for MaxMstCut (resp., FPTAS in Section 6 for MinLstCut and MaxLstCut) in a rooted tree $G$ with the constraint that all leaves in $L'$ should be cut away in finding a solution.

Next, we introduce the concept of leaf centroid, which will be used to separate the tree into several small trees.

**Lemma 3.** For any tree, there is a vertex such that each of the subtrees obtained by removing it contains at most half of number of leaves in the original tree, and this vertex can be found in linear time.

We will call the vertex described in Lemma 3 a leaf centroid. Leaf centroid is an extension of centroid introduced in [8]. Lemma 3 can be proved easily.

Before we present the algorithm for unrooted trees, we consider a special case that the graphs are restricted to paths (unrooted trees with only two leaves), the algorithms for which will be used as subalgorithms in our FPTASs for general unrooted trees.

### 7.2 Linear Algorithms for the Problems In Paths

In this subsection, we mainly design some linear-time algorithms for (constrained) MinLstCut, MaxLstCut, MinMstCut and MaxMstCut in a path with general edge cost and nonnegative vertex weight. Note that our algorithms allow negative edge cost. MinLstCut and MaxLstCut (MinMstCut and MaxMstCut) are equivalent since we can reset the edge cost to get each other. Then we only need to consider two problems, MinLstCut and MinMstCut. In fact, we have designed a linear-time algorithm for MinMstCut in trees with nonnegative edge cost and nonnegative vertex weight in Section 3.2. However, that algorithm does not allow negative edge cost.

Given a path $P = a_1a_2 \cdots a_n$ with $n$ vertices, where the vertices are arranged from the left end $a_1$ to the right end $a_n$. For each vertex $a_i$ in the path, we let $V_1(a_i) = \{a_1, a_2, \ldots, a_i\}$ and $V_2(a_i) = \{a_i, a_{i+1}, \ldots, a_n\}$. We can compute $w(V_1(a_i))$ and $w(V_2(a_i))$ for all $i$’s in linear time.

First, we present a linear-time algorithm for MinLstCut in a path in Fig. 8. In the algorithm, $S(i)$ will store an optimal solution $V' \subseteq V$ under the constraint that the corresponding cut set $E(V')$ contains two edges and the left edge is the $i$-th edge $a_ia_{i+1}$ from left to right, if any, or $S(i) = \bot$ otherwise.

In Step 2, we check the case that the cut set of the optimal solution contains only one edge. In Step 5, we check the case that the cut set of the optimal solution contains two edges, which will be $a_ia_{i+1}$ and $a_{j-1}a_j$. Note that in each iteration in Step 5, $i$ or $j$ will increase by 1. We require that $i < j \leq n$, and then there are less than $2n$ iterations. Each iteration can be done in $O(1)$ time. Therefore the whole algorithm runs in linear time. To prove the correctness of the algorithm, we only need to prove that $S(i)$ works as claimed. If $S(i) = \bot$, it is easy to see that there is no solution the corresponding cut set of which contains $a_ia_{i+1}$ as the left edge. Otherwise, we consider the case of $j^* > 0$. Note each time before
we increase \( j \) by 1 in Step 5, edge \( a_j \) is the minimum edge in the path \( a_1 \cdots a_j \). Since the algorithm will increase \( j \) to the maximum index such that \( w(V_1(a_j) \cup V_2(a_j)) \geq k \), at the time before \( i \) changes to \( i+1 \), \( S(i) \) has already stored an optimal solution under the constraint. Therefore, it works as claimed. Since the algorithm will return the best one among \( \{ S(S(1)), S(S(2)), \ldots, S(S(n-1)) \} \), the algorithm will find an optimal solution to MinLstCut.

**Algorithm** \( \text{Path1}(P, L', k) \)

**Input:** A path \( P = a_1a_2 \cdots a_n \) with an edge cost \( c \) and a nonnegative vertex weight \( w : V \rightarrow \mathbb{R}^+ \), a subset \( L' \subseteq \{ a_1, a_n \} \) of leaves, and a number \( k \geq 0 \).

**Output:** An optimal solution \( S \) to MinLstCut with the constraint that \( L' \subseteq S \), if any, or \( \bot \) otherwise.

1. Let \( S \leftarrow \bot \).
2. For \( i \) from 1 to \( n-1 \), do
   - If \( a_i \not\in L' \) and \( w(V_2(a_{i+1})) \geq k \) and \( c(V_2(a_{i+1})) < c(S) \), then \( S \leftarrow V_2(a_{i+1}) \).
   - If \( a_n \not\in L' \) and \( w(V_1(a_i)) \geq k \) and \( c(V_1(a_i)) < c(S) \), then \( S \leftarrow V_1(a_i) \).
3. For \( i \) from 1 to \( n-1 \), do \( S(i) \leftarrow \bot \).
4. Let \( i \leftarrow 1 \), \( j \leftarrow 3 \) and \( j^* \leftarrow 0 \).
5. While \( (i < n-1) \) do
   - Let \( V' = V_1(a_i) \cup V_2(a_j) \).
     - If \( w(V') \geq k \) then
       - If \( c(V') < c(S(i)) \), then \( S(i) \leftarrow V' \) and \( j^* \leftarrow j \).
       - If \( j < n \), then \( j \leftarrow j+1 \).
     - Else
       - \( i \leftarrow i+1 \).
       - If \( i = j-1 \), then \( j \leftarrow j+1 \) and \( j^* \leftarrow 0 \).
       - If \( j^* \neq 0 \), then \( S(i) \leftarrow V_1(a_i) \cup V_2(a_j) \).
6. Return \( S \) as the best among \( \{ S(S(1)), S(S(2)), \ldots, S(S(n-1)) \} \).

**Fig. 8. Algorithm** \( \text{Path1}(P, L', k) \)

The technique for the algorithm in Fig. 8 can also be used to derive a linear-time algorithm for MinMstCut. By modifying \( \text{Path1} \), we present algorithm \( \text{Path2} \) for MinMstCut in a path in Fig. 9. Differently from \( \text{Path1} \), \( \text{Path2} \) will use \( S(j) \) to store an optimal solution \( V' \subseteq V \) under the constraint that the corresponding cut set \( E(V') \) contains two edges and the right edge is the \((j-1)\)th edge \( a_{j-1} \) from left to right. The analysis for this algorithm is also similar to that for \( \text{Path1} \). In \( \text{Path2} \), Step 2 is to deal with the case that the cut set contains only one edge, and Step 5 considers the case that the cut set contains two edges. In each iteration in Step 5, \( i \) or \( j \) will increase by 1, and we require that \( i < j \leq n \). Then the algorithm will run in linear time. The correctness of the algorithm follows from the following observation. In Step 5, the algorithm will keep edge \( a_i \cdots a_{i+1} \) as a minimum edge in the path \( a_1a_2 \cdots a_{i+1} \) (when \( i^* \neq 0 \),
where \( i \) satisfies that \( w(V_1(a_i) \cup V_2(a_j)) \leq k \). Since for each \( j \), the algorithm will increase \( i \) to the maximum index satisfying the constraint, the algorithm can find an optimal solution for \( S(j) \).

### Algorithm Path2\( (P, L', k) \)

**Input**: A path \( P = a_1a_2 \cdots a_n \) with an edge cost \( c \) and a nonnegative vertex weight \( w : V \to \mathbb{R}^+ \), a subset \( L' \subseteq \{a_1, a_n\} \) of leaves, and a number \( k \geq 0 \).

**Output**: An optimal solution \( S \) to MinMstCut with the constraint that \( L' \subseteq S \), if any, or \( \perp \) otherwise.

1. Let \( S \leftarrow \perp \).
2. For \( i \) from 1 to \( n - 1 \), do
   - If \( (a_i \notin L' \text{ and } w(V_2(a_{i+1})) \leq k \text{ and } c(V_2(a_{i+1})) < c(S)) \), then \( S \leftarrow V_2(a_{i+1}) \).
   - If \( (a_n \notin L' \text{ and } w(V_1(a_i)) \leq k \text{ and } c(V_1(a_i)) < c(S)) \), then \( S \leftarrow V_1(a_i) \).
3. For \( j \) from 3 to \( n \), do \( S(j) \leftarrow \perp \).
4. Let \( i \leftarrow 1 \), \( j \leftarrow 3 \) and \( i^* \leftarrow 0 \).
5. While \( (j \leq n) \) do
   - Let \( V' \leftarrow V_1(a_i) \cup V_2(a_j) \).
   - If \( w(V') \leq k \) then
     - If \( c(V') < c(S(j)) \), then \( S(j) \leftarrow V' \) and \( i^* \leftarrow i \).
     - If \( i \leq j - 2 \), then \( i \leftarrow i + 1 \).
   - Else
     - \( j \leftarrow j + 1 \).
   - If \( i^* \neq 0 \), then \( S(j) \leftarrow V_1(a_{i^*}) \cup V_2(a_j) \).
6. Return \( S \) as the best among \( \{S, S(3), S(4), \ldots, S(n)\} \).

Fig. 9. Algorithm Path2\( (P, L', k) \)

### 7.3 The Algorithms for Unrooted Trees

Now we are ready to design and analyze the algorithm for general unrooted trees. Recall that \( G_r \) is the rooted tree obtained from an unrooted tree \( G \) by choosing \( r \) as the root. For a neighbour \( v \) of \( r \), let \( G_r(v) \) denote the tree obtained by contracting \( V \setminus D(v) \) in \( G_r \) into a single (nonroot) vertex \( r_v \) with weight \( w(r_v) \) being the total weight of vertices in \( V \setminus D(v) \). Separating a vertex \( r \) in an unrooted tree \( G \) means to construct unrooted subtrees \( G_r(v) \) for all neighbours \( v \) of \( r \). In our algorithm, we will select the vertex \( r \) for such separation as the leaf centroid of the tree. Note that \( w_r(v) \) can be computed in a DFS for all pairs of \( r \) and its neighbour \( v \). We will assume that we can get all the subtrees after separating a tree at a vertex \( r \) in linear time. The main steps of our algorithm for (constrained) MaxMstCut in unrooted trees is presented in Fig 10.
Algorithm \textsc{Unroot}($G, L', k, \varepsilon$)

\textbf{Input:} An unrooted tree $G = (V, E)$ with nonnegative edge cost $c : E \to \mathbb{R}^+$ and nonnegative vertex weight $w : V \to \mathbb{R}^+$, a subset $L'$ of leaves in $G$, and two numbers $k \geq 0$ and $1 > \varepsilon > 0$ (Initially, $L' = \emptyset$).

\textbf{Output:} A $(1 - \varepsilon)$-approximation solution $S$ to MaxMstCut with the constraint that $L' \subseteq S$.

1. If $G$ has only two leaves, solve the problem in linear time by using the algorithm presented in Section 7.2 and return the solution.
2. Let $r$ be a leaf centroid of $G$ and $G_r$ the rooted tree obtained by selecting $r$ as the root in $G$.
3. $S \leftarrow \text{Fptas}^1(G_r, L', k, \varepsilon)$. 
4. For each neighbour $v$ of $r$, do
   4.1 Construct the subtree $G_r(v)$.
   4.2 Let $S' \leftarrow \text{Unroot}(G_r(v), L' \cup \{rv\}, k, \varepsilon)$.
   4.3 If $c(S) < c(S')$, let $S \leftarrow S'$.
5. Return $S$.

Fig. 10. Algorithm \textsc{Unroot}($G, k, L', \varepsilon$)

Algorithm \textsc{Unroot} for MinLstCut and MaxLstCut can be obtained by replacing the called algorithm \text{Fptas}^1($G_r, k, L', \varepsilon$) with \text{Fptas}^2($G_r, L', k, \varepsilon$) in Step 3 and modify the if condition in Step 4.3.

The correctness of algorithm \textsc{Unroot} follows from this observation: Let $X$ be an optimal solution, if $r \in V - X$ then we can find a solution within the error bound in the first step, if $r \in X$ then the optimal solution is an optimal solution to the problem in $G_r(v)$ for some neighbour $v$ of $r$ with the constraint $r \in X$, which implies that we can solve the problem by solving the constrained problems in $G_r(v)$. We will solve the problem directly if the tree has only two leaves. Then the algorithm will run in $O(\log l)$ iterations by Lemma 3. Note that when the tree has at least three leaves, no leaf centroid is a leaf of the tree, and hence $L'$ does not contain a possible root, implying that our algorithm always receives a well-defined input.

As for the running time, we have showed that the problems can be exactly solved in linear time when the graphs are restricted to paths. The hard part is the analysis for the general case.

Since the case for MaxMstCut will be covered by the case of MinLstCut and MaxLstCut, in this section we will analyze the algorithm for MinLstCut. We consider the search tree $T$ generated in the algorithm. Each node of $T$ corresponds to an instance of the constrained MinLstCut in unrooted trees. The root of $T$ represents the original instance $G^{(0)} = G$.\footnote{We may use the same notation to denote the instance in a node of $T$ and the corresponding tree in the instance.} The children of a node $v$ in $T$ correspond to the instances called recursively by the fourth step of the algorithm when it solves the instance corresponding to $v$. In each non-leaf node of $T$, the algorithm will call \text{Fptas}^2 once in Step 3. The whole running time
of the algorithm will be the sum of all the running time taken by \( \text{Fptas2'} \) in non-leaf nodes and the running time taken by the linear-time algorithms in leaf nodes. Note that the running time used in each leaf node is also bounded by the running time of \( \text{Fptas2'} \).

In \( \mathcal{T} \), we say that a node is in level \( i \) if the path from the node to the root \( G^{(0)} \) contains \( i \) edges. We fix an ordering of the nodes in the same level of \( \mathcal{T} \) according to a BFS. We will use \( G^{(i)}_j \) to denote the instance corresponding to the \( j \)-th node in the \( i \)-th level of \( \mathcal{T} \). Let \( N^{(i)} \) denote the total number of vertices of all trees in the \( i \)-th level instances. We see that \( N^{(i)} \leq 2n \) for any \( i \), since no edge in the original \( G^{(0)} \) appears more than once over all instances in the same level.

Fig. 11 illustrates a search tree \( \mathcal{T} \). Let \( n^{(i)}_j \) and \( l^{(i)}_j \) denote the numbers of vertices and leaves in the instance \( G^{(i)}_j \). Then a time bound on the basic computation for executing \( \text{Fptas2'} \) to \( G^{(i)}_j \) is given by
\[
 c l^{(i)}_j n^{(i)}_j (\frac{1}{\varepsilon} + \log 2n) \quad \text{for a constant } c > 0.
\]
To analyze the running time, we first get a bound on the time \( T^{(i)}_{j_0} \) of basic computations for solving all problems in the trees corresponding to the children of instance \( G^{(i-1)}_{j_0} \) for some \( j_0 \). Then we bound the time \( T^{(i)} \) of basic computations for solving all instances in the \( i \)-th level of \( \mathcal{T} \). Finally, we get the whole running time of the algorithm by summing up \( T^{(i)} \) for all \( i \)'s.

![Fig. 11. An illustration of \( \mathcal{T} \)](image)

Take an instance \( G^{(i-1)}_{j_0} \) not corresponding to a leaf of \( \mathcal{T} \). Without loss of generality, we assume that \( G^{(i-1)}_{j_0} \) has child instances \( G^{(i)}_j \) (\( j = 1, 2, \ldots, x \)). Then we have
\[
 3 \leq n^{(i)}_j \leq n^{(i-1)}_{j_0} - 1, \quad 3 \leq l^{(i)}_j \leq \left\lceil \frac{1}{2} n^{(i-1)}_{j_0} \right\rceil + 1, \quad \sum_{j=1}^{x} n^{(i)}_j \leq n^{(i-1)}_{j_0} + x - 1.
\]
and $\sum_{j=1}^{\infty} f_{j}^{(i)} \leq \sum_{j=1}^{\infty} f_{j}^{(i-1)} + x$. Under these constraints, we can prove the following relation (see Appendix B for the full proof):

$$\sum_{j=1}^{\infty} f_{j}^{(i)} n_{j}^{(i)} \leq \frac{1}{2} \sum_{j=1}^{\infty} f_{j}^{(i-1)} n_{j}^{(i-1)} + \frac{5}{2} n_{j}^{(i-1)}.$$  \hspace{1cm} (2)

Now, we prove a bound for $T_{j_0}^{(i)}$. For all subinstances of $G_{j_0}^{(i-1)}$, the basic computation for executing their FPTAS2 is bounded by $T_{j_0}^{(i)} = \sum_{j=1}^{x} c_{j}^{(i)} n_{j}^{(i)} (1/\varepsilon + \log 2n)$. By (2), we can bound $T_{j_0}^{(i)}$ as follows:

$$T_{j_0}^{(i)} \leq \sum_{j=1}^{x} c_{j}^{(i)} n_{j}^{(i)} (1/\varepsilon + \log 2n) \leq \frac{c}{2} (\sum_{j=1}^{x} f_{j}^{(i-1)} n_{j}^{(i-1)} + 5n_{j}^{(i-1)})(1/\varepsilon + \log 2n).$$  \hspace{1cm} (3)

Next, we prove a bound for $T^{(i)}$, i.e., the running time for solving all instances in the $i$-th level of $T$. Clearly, $T^{(i)} = \sum_{p} T_{p}^{(i)}$. By using these and (3), we can bound $T^{(i)}$ as follows:

$$T^{(i)} \leq \sum_{p} \frac{c}{2} (f_{p}^{(i-1)} n_{p}^{(i-1)} + 5n_{p}^{(i-1)})(1/\varepsilon + \log 2n)$$
$$= \frac{c}{2} \sum_{p} f_{p}^{(i-1)} n_{p}^{(i-1)} (1/\varepsilon + \log 2n) + \sum_{p} \frac{5c}{2} n_{p}^{(i-1)} (1/\varepsilon + \log 2n)$$
$$\leq \frac{c}{4} T^{(i-1)} + \frac{5c}{2} N^{(i-1)} (1/\varepsilon + \log 2n)$$
$$\leq \frac{c}{4} T^{(i-1)} + 5cn (1/\varepsilon + \log 2n).$$

For the maximum level $h = O(\log l)$, we have

$$\frac{1}{2} \sum_{i=0}^{h} T^{(i)} \leq T^{(0)} + \sum_{i=0}^{h} \left( T^{(i)} \frac{1}{2} T^{(i-1)} \right) \leq cnl (1/\varepsilon + \log n) + 5c(h+1)n (1/\varepsilon + \log 2n).$$

Therefore our algorithm for MinLstCut runs in $O(nl(1/\varepsilon + \log n))$ time.

In the same way, we know that MaxMstCut in unrooted trees can also be solved in the same running time as that for the problem in rooted trees. The space used in the algorithms for the problems in unrooted trees also does not increase. Then we get

**Theorem 4.** There is an $O(nl/\varepsilon)$-time and $O(P^2/\varepsilon + n)$-space algorithm that computes a solution with error $\varepsilon$ for MaxMstCut in an unrooted tree with nonnegative edge cost and nonnegative vertex weight.

**Theorem 5.** There is an $O(nl(1/\varepsilon + \log n))$-time and $O(P^2/\varepsilon + n)$-space algorithm that computes a solution with error $\varepsilon$ for MinLstCut and MaxLstCut in an unrooted tree with nonnegative edge cost and nonnegative vertex weight.

### 8 Concluding Remarks

In this paper, we have designed several effective algorithms for some related cut problems in rooted trees by using scaling, rounding, binary search, left-right
DP and some other techniques. We have also extended our algorithms to the corresponding problems in unrooted trees without increasing the running time bounds with nontrivial analysis.

Although the algorithms presented in this paper are effective, there is a gap between the upper bound for the knapsack problem and the upper bounds for our problems. It will be interesting to further reduce the gap. Another question for further study is to consider the problems in more graph classes, such as $k$-trees, directed acyclic graphs and so on. In our problems, we restrict the edge cost and vertex weight to be nonnegative. There are simple reductions among the problems as long as no restriction such as nonnegativeness is imposed on cost/weight/target. We only need to reset the values of the edge cost, vertex weight and target $k$ to get different problems. For the terminal version of the problem without the nonnegativeness constraint on edge cost, we can also solve it by modifying our dynamic programming algorithm in Section 4. However, it is not convenient to discuss approximation algorithm when the optimal value can be negative or positive.

Acknowledgements

Part of the work was done when the first author was visiting Kyoto University in 2009. This work was partially supported by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan, and National Natural Science Foundation of China under the Grant 60903007.

A Algorithm for MinMstCut

**Theorem 6.** An optimal cut to terminal MinMstCut without constraint (i) in a graph with nonnegative edge cost can be obtained in $O(t^3nm\log n)$ time, where $t = |T|$, $n = |V|$ and $m = |E|$.

**Proof.** Let $d(X)$ denote the cut function of $G$, i.e., the sum of the costs of edges between $X$ and $V - X$ in $G$. Let $X^*$ be the set of vertices to be removed by an optimal cut, and assume that $|X^*|$ is minimized over all such sets. Then $w(X^*) \leq k$ and $d(Y) > d(X^*)$ for all nonempty proper subsets $Y \subseteq X^*$. To find such a set $X^*$ efficiently, define a set function $f : 2^X \to \mathbb{R}$ by setting $f(T')$ to be the minimum cost of a cut separating $T' \subseteq T$ and $T - T'$, i.e., $f(T') = \min \{ d(V') \mid V' \subseteq V, V' \cap T = T' \}$. We see that $f$ is a symmetric submodular set function, and we can find the family $\mathcal{X}$ of all extreme subsets $T' \subseteq T$ (i.e., those sets such that $f(T'') > f(T')$ for all nonempty proper subsets $T'' \subset T'$) in $O(|T|^{3}|V||E|\log |V|)$ time by an algorithm due to Nagamochi [12]. The remaining task is to pick up an extreme set $T^*$ with the minimum cost $f(T^*)$ among all extreme set $T' \in \mathcal{X}$ and $w(T') \leq k$. The subset $X^*$ attaining $d(X^*) = f(T^*)$ and $X^* \cap T = T^*$ gives an optimal cut $[X^*, V - X^*]$.  


B Proof of (2)

To prove (2), we first give the following inequality.

**Lemma 4.** Given nonnegative numbers \( x \geq 2, \ a_j, \ b_j \ (j = 1, 2, \ldots, x), \ A, \ B, \ h \) and \( l \) such that \( \sum_{j=1}^x a_j \leq A, \sum_{j=1}^x b_j \leq B, \ a_j \leq h \leq A \) and \( l \leq b_j \ (j = 1, 2, \ldots, x). \) Then for \( q = [A/h], \) it holds

\[
\sum_{j=1}^x a_jb_j \leq h(B - (x - q - 1)l).
\]

**Proof.** Without loss of generality, we assume that \( b_1 \geq b_2 \geq \cdots \geq b_x. \) For \( \Delta_a = qh - \sum_{i=1}^q a_i, \) consider \( a'_j \ (j = 1, \ldots, x) \) such that \( a'_1 = \cdots = a'_q = h, \)
\[ a'_j \leq b_j \ (j = q + 1, \ldots, x), \]
and \( \sum_{j=q+1}^x a'_j = \sum_{j=q+1}^x a_j - \Delta_a. \) Then we have

\[
\sum_{j=1}^x a_jb_j \leq \sum_{j=1}^x a'_jb_j.
\]

Furthermore, consider \( b'_j \ (j = 1, \ldots, x) \) such that where \( \sum_{j=1}^q b'_j = B - (x - q)l \) and \( b'_j = l \ (j = q + 1, \ldots, x) \), and we have

\[
\sum_{j=1}^x a'_jb_j \leq \sum_{j=1}^x a'_jb'_j.
\]

Therefore,
\[
\sum_{j=1}^x a_jb_j \leq h(B - (x - q)l) + \sum_{j=q+1}^x a'_jb'_j
\leq h(B - (x - q)l) + (A -qh)l
\leq h(B - (x - q)l) + hl
= h(B - (x - q - 1)l).
\]

Now we are ready to prove (2).

**Proof.** When \( x = 2, \) it is easy to see that
\[
\sum_{j=1}^2 n_j^{(i)} \leq (\sum_{j=0}^{i-1} \frac{1}{2}) n_j^{(i)} + 2(\frac{1}{2})^{(i-1)} - 2 + \frac{1}{2} \cdot 3
= \frac{1}{2} n_j^{(i)} + \frac{1}{2} n_j^{(i)} + \frac{1}{2} n_j^{(i)} - 4
\leq \frac{1}{2} n_j^{(i)} + \frac{1}{2} n_j^{(i)}.
\]

When \( x \geq 3, \) let \( a_i = n_j^{(i)}, \ b_i = n_j^{(i)}, \ h = [\frac{1}{2}^{(i-1)}] + 1, \ A = n_j^{(i-1)} + x, \ B = n_j^{(i-1)} + x - 1 \) and \( l = 3. \) It is easy to see that when \( x \geq 3, \) it holds \( 2(x - 1) \geq 3q = 3[A/h]. \) By Lemma 4, we get
\[
\sum_{j=1}^x a_jb_j \leq h(B - (x - q - 1)l) = (\frac{1}{2}^{(i-1)} + 2)(n_j^{(i-1)} + x - 1 - 3(x - q - 1))
\leq \frac{1}{2} n_j^{(i-1)} + \frac{1}{2} n_j^{(i-1)}.
\]

This completes the proof.
References