An Improved Algorithm for the Min-max k Tree Cover Problem

Aaron Weinmann
s6aawein@uni-bonn.de

November 22, 2022
0. Short Introduction

1. 4-approximation Algorithm

2. 3-approximation Algorithm

2a. Proof Sketch

(3. Related Problem: BTC)
Part 0: Introduction and Preliminaries
Nurse Station Location Problem

- We can relate this to the Nurse Station location problem:
- How to distribute the nurses in a hospital, s.t. the longest morning visit route is minimized
- TSP with $k$ salesmen
- MSTs for constant factor approximation?
Min-Max k-Tree Cover Problem (MMkTC)

**Tree Cover**

Let $G = (V, E)$ be an undirected graph, $w : E \rightarrow \mathbb{Z}^+$ a weight function. A set $T_1, T_2, \ldots, T_k$ of subtrees of $G$ is called a *tree cover of $G$* if every vertex of $V$ appears in at least one $T_i$.

**Min-Max k-tree Cover Problem (MMkTC)**

Given a weighted graph $G$ and a positive integer $k$, find a tree cover of size $k$, s.t. the weight of the largest tree in the cover is *minimized*.

- NP-complete (Even et al.’03)
- 4-approximation algorithm exists (Even et al.’03)
- 3-approximation algorithm exists (Khani et al.’14)
Greedy 4-approximation for MMkTC

- Guess $\lambda$ as value for optimal solution with binary search
- Delete edges $> \lambda$
- Find MSTs on connected components
- Split each MST into trees with weight in range $[2\lambda, 4\lambda]$
4-approximation example:

An arbitrary weighted undirected graph:
4-approximation example:

Guess a $\lambda$ and delete edges $> \lambda$:
4-approximation example:

Delete edges $> \lambda$: 

![Graph](image_url)
4-approximation example:

Find MSTs on connected components:
4-approximation example:

Split MSTs into trees with weight in $[2\lambda, 4\lambda]$
4-approximation example:

Split MSTs into trees with weight in $[2\lambda, 4\lambda]$
4-approximation example:

Split MSTs into trees with weight in \([2\lambda, 4\lambda]\)
4-approximation example:

Split MSTs into trees with weight in \([2\lambda, 4\lambda]\)

combine light subtrees into medium tree
Introducing:

An improved 3-approximation algorithm for the MM$k$TC Problem
Idea:

**INPUT**
- $G = (V, E)$: weighted, undirected graph
- $\lambda$: guess for the value of the optimal solution $OPT$

**OUTPUT**
- If $\lambda \geq OPT$: $S = \{S_1, \ldots, S_k\}$: k-tree cover, where $w(S_i) \leq 3\lambda \ \forall S_i$
- If $\lambda < OPT$: adjust $\lambda$
General Strategy:

- Delete all edges $e$ with $w(e) > \frac{\lambda}{2}$
- Classify CCs into light ($W_T(C) \leq \lambda$) and heavy ($W_T(C) > \lambda$)
- Decide for each light component $C$:
  a) Take $C$ into $S$
  b) Connect $C$ to another light component and add to $S$
  c) Attach $C$ to a heavy component
- Split heavy components in not too many small trees
Problem: How do we decide this?

⇒ Build an auxiliary graph $H$ after deleting edges with weight $> \frac{\lambda}{2}$:

Graph $H(G, a, b)$

- **V:**
  - $l$ regular (light) nodes $v_1, \ldots, v_l$
  - $a$ dummy (null) nodes
  - $b$ dummy (heavy) nodes

- **E:**
  - $e_{v_i, v_j}$ with weight 0 iff $\frac{\lambda}{2} < d(C_i, C_j) \leq \lambda$ in $G$.
  - Between all null nodes and all regular nodes with weight 0.
  - If $A(C_i)$ finite, connect $v_i$ to all heavy nodes with weight $A(C_i)$.

**Weight of Attachment $A(C_i)$**

$A(C_i)$ minimum weight of attaching an MST of $C_i$ to a heavy component.
Example:

Undirected graph:
Example:

Delete all edges $e$ with $w(e) > \frac{\lambda}{2}$:

\[
\begin{array}{c}
\lambda < w(e) \\
\lambda/2 < w(e) \leq \lambda \\
w(e) \leq \lambda/2
\end{array}
\]
Example:

Delete heavy edges:
Example:

Classify CCs:

heavy components

light components
Example:

Build auxiliary graph $H$ on light components:

$H(G, a=0, b=0)$
Example:

Add edges if light CCs were connected with weight $\frac{\lambda}{2} < \cdots \leq \lambda$: 

\[
H(G, a=0, b=0)
\]
Example:

Add *null nodes*:

\[ H(G, a=1, b=0) \]
Example:

Add *heavy* nodes:

$$H(G, a=1, b=1)$$
Example:

Add *heavy* nodes:

H(G, $a=1$, $b=2$)
Example:

Add heavy nodes:

$H(G, a=1, b=3)$
Finally: Pseudocode
Algorithm 1 MMkTC Algorithm

1: Delete all edges with weight more than $\frac{\lambda}{2}$; let $C_1, \ldots, C_{l+h}$ be the set of $l$ light and $h$ heavy components created.

2: for $a : 0 \rightarrow l$, $b : 0 \rightarrow l$ do
   (i) $S \leftarrow \emptyset$
   (ii) Construct graph $H(G, a, b)$
   (iii) Find \textbf{minimum perfect matching} on $H$; or break.
   (iv) \textbf{Attach} some light-components $C_i$ to its \textit{nearest heavy component} based on match with heavy node.
   (v) \textbf{Decompose} heavy components using \textit{Lemma 3} and add the obtained trees to $S$.
   (vi) Add MST of $C_i$ to $S$, based on match with \textit{null} node.
   (vii) Join $C_i$ and $C_j$ with edge and add ST to $S$, based on match with \textit{regular} node.
   (viii) If $|S| \leq k$ then return $S$.

3: end for

4: return failure
Proof Sketch
Proof Sketch:

Theorem:
The algorithm finds a 3-approximation for the MM$k$TC problem

Assume in the following $\lambda \geq OPT$:

I: Bound on the size of the trees
   a) Finding $\lambda \geq OPT$
   b) Every tree in $S$ has weight $\leq 3\lambda$

II: The algorithm returns a set $|S| \leq k$

III: The $S$ covers $V$ completely

IV: Remarks on running time
Proof Sketch:

**Theorem:**

The algorithm finds a 3-approximation for the MM$k$TC problem

Assume in the following $\lambda \geq \text{OPT}$:

I: **Bound on the size of the trees**
   a) **Finding** $\lambda \geq \text{OPT}$
   b) Every tree in $S$ has weight $\leq 3\lambda$

II: The algorithm returns a set $|S| \leq k$

III: The $S$ covers $V$ completely

IV: Remarks on running time
Finding $\lambda$

Assume our algorithm is correct:

- If $\lambda \geq \text{OPT}$ we get a solution $\leq 3\lambda$
- If $\lambda < \text{OPT}$ we get an error

How does this result in a 3-approximation for OPT?
Finding $\lambda$

Assume our algorithm is correct:

- If $\lambda \geq OPT$ we get a solution $\leq 3\lambda$
- If $\lambda < OPT$ we get an error

How does this result in a 3-approximation for $OPT$?

Binary search on the interval $[0, \sum_{e \in E} w(e)]$!
Proof Sketch:

Theorem:
The algorithm finds a 3-approximation for the MM$k$TC problem

Assume in the following $\lambda \geq OPT$:

I: Bound on the size of the trees
   a) Finding $\lambda \geq OPT$
   b) Every Tree in $S$ has weight $\leq 3\lambda$

II: The algorithm returns a set $|S| \leq k$

III: The $S$ covers $V$ completely

IV: Remarks on running time
Decision on light components:

Remember that we use a perfect matching on $H$ for our decisions:

(i) Construct graph $H(G, a, b)$
(ii) Find \textit{minimum perfect matching} on $H$; or break.
(iii) Attach some light-components $C_i$ to its nearest heavy component based on match with heavy node.
(iv) Decompose heavy components using \textit{Lemma 3} and add the obtained trees to $S$.
(v) Add MST of $C_i$ to $S$, based on match with \textit{null} node.
(vi) Join $C_i$ and $C_j$ with edge and add ST to $S$, based on match with \textit{regular} node.
Matching on $H$:

Find minimum perfect matching on $H$:

$H(G, a=2, b=1)$
Matching on H:

Join regular pairs:

H(G, a=2, b=1)
Matching on $H$:

Insert directly into $S$ if matched to $null$ node:

$H(G, a=2, b=1)$
Matching on $H$:

Attach to closest heavy component if matched to heavy node:

$$H(G, a=2, b=1)$$

join CCs and add to $S$

Add to "closest" heavy component

Add to $S$

- regular node
- null node
- heavy node
Every Tree in $S$ has weight $\leq 3\lambda$:

Recall the algorithm:

1. **Decompose** heavy components using Lemma 3 and add the obtained trees to $S$.
2. Add MST of $C_i$ to $S$, based on match with *null* node.
3. Join $C_i$ and $C_j$ with edge and add ST to $S$, based on match with *regular* node.

Trees are added in the following steps:

- **(v)**: Decompose to weight $\leq 3\lambda$ (bound on number of trees later)
- **(vi)**: $W_T(C_i) \leq \lambda$
- **(vii)**: $W_T(C_i) + W_T(C_j) + w(e) \leq \lambda + \lambda + \lambda = 3\lambda$

(by definition of H)

*This is only based on the assumption that we actually find a perfect matching!*
Proof Sketch:

Theorem:
The algorithm finds a 3-approximation for the MMkTC problem

Assume in the following $\lambda \geq \text{OPT}$:

I: Bound on the size of the trees
   a) Finding $\lambda \geq \text{OPT}$
   b) Every tree in $S$ has weight $\leq 3\lambda$

II: The algorithm returns a set $|S| \leq k$

III: The $S$ covers $V$ completely

IV: Remarks on running time
Existence of Perfect Matching

Show the existence of suitable values for $a$ and $b$:

- Construct an $H'$ based on an optimal solution $OPT = \{ T_1, \ldots, T_k \}$:
  - After Step 1, each $T_i$ is in at most 2 components
    - $k_l$ "light" trees (only light comp.)
    - $k_h$ "heavy" trees (only heavy comp.)
    - $k_b$ "bad" trees (one light, one heavy comp.)
  - Calculate maximum matching $M$ on $H'$

$H' = (V', E')$

$V'$: $v'_i$ corresponding to each light component

$E'$: Add edge for every light $T_i$ (even loops)
Construction of $H'$:

Recall Graph from earlier slides:
Construction of $H'$:

Consider Optimal Solution $OPT$:
Construction of $H'$:

Classify:

![Diagram of interconnected circles labeled as light trees.](image-url)
Construction of $H'$:

Classify:

- **Heavy trees**
- **Light trees**
Construction of $H'$:

Classify:

- Heavy trees
- Bad trees
- Light trees
Construction of $H'$:

Look at light components:
Construction of $H'$:

Ignore "bad" components:
Construction of $H'$:

Vertices for each light comp.:
Construction of $H'$:

Edges corresponding to OPT:
Construction of H’:

Edges corresponding to OPT:
Construction of $H'$:

Edges corresponding to OPT:
Construction of $H'$:

Edges corresponding to OPT:
Construction of $H'$:

Find maximum matching $M$
Construction of $H'$:

Recall $H$:
Construction of $H'$:

Heavy nodes (b nodes) for set of isolated nodes $I$:
Construction of $H'$:

Null nodes (a nodes) for set of non-isolated, non-matching $U$
Construction of $H'$:

Build perfect matching in $H$:
Existence of Perfect Matching

- Consider iteration where we have $a = |U|$ null nodes and $b = |I|$ heavy nodes

**Lemma**

In the iteration where $a = |U|$ and $b = |I|$ the algorithm computes a minimum perfect matching on $H$ whose cost can be bounded by the sum of all finite $A(C_i)$

**Proof**: Match all light components according to $M$ if possible, leaving only $I$ and $U$. Every $v_i \in I$ can be matched to a null node in $H$, every $v_u \in U$ to a heavy node with weight $A(C_u)$. $\square$
Upper Bound on Number of Trees:

Pseudocode:

(vi) Add MST of $C_i$ to $S$, based on match with null node.
(vii) Join $C_i$ and $C_j$ with edge and add ST to $S$, based on match with regular node.

- Recall: In Steps (vi) and (vii), we add $|U| + |M| \leq k_l$ trees to $S$
- Left to show: In Step (v), we add at most $k_h + k_b$ trees to $S$
Upper Bound for Step (v):

Recall Step (v):

(iv) **Attach** some light-components $C_i$ to its nearest heavy component based on match with heavy node.

(v) **Decompose** heavy components using Lemma 3 and add the obtained trees to $S$.

- similar to 4-approximation algorithm
- need upper bound for spanning tree of heavy components:

$$
\sum_{1 \leq i \leq s} A(C_{l_i}) + \sum_{l+1 \leq i \leq l+h} W_T(C_i) \leq (k_h + k_b) \cdot \frac{3}{2} \lambda - h \frac{\lambda}{2}
$$

⇒ can split heavy CCs in $(k_h + k_b)$ trees with maximum weight $3\lambda$

⇒ In total at most $k_l + k_h + k_b = k$ trees in $S$
Proof Sketch:

Theorem:
The algorithm finds a 3-approximation for the MM$k$TC problem

Assume in the following $\lambda \geq OPT$:

I: Bound on the size of the trees
   a) Finding $\lambda \geq OPT$
   b) Every tree in $S$ has weight $\leq 3\lambda$

II: The algorithm returns a set $|S| \leq k$

III: The $S$ covers $V$ completely

IV: Remarks on running time
$S$ covers $V$ completely

Trivial:
- Every light component is inserted into $S$ or
- gets attached to a heavy component which we split into trees
Proof Sketch:

**Theorem:**
The algorithm finds a 3-approximation for the MM$k$TC problem.

Assume in the following $\lambda \geq OPT$:

I: Bound on the size of the trees
   a) Finding $\lambda \geq OPT$
   b) Every tree in $S$ has weight $\leq 3\lambda$

II: The algorithm returns a set $|S| \leq k$

III: The $S$ covers $V$ completely

IV: Remarks on running time
Remarks on running time

**Algorithm 1 MMkTC Algorithm**

1: Delete all edges with weight more than $\frac{1}{2}$; let $C_1, \ldots, C_{l+h}$ be the set of $l$ light and $h$ heavy components created.

2: for $a : 0 \rightarrow l$, $b : 0 \rightarrow l$ do
   
   (i) $S \leftarrow \emptyset$
   
   (ii) Construct graph $H(G, a, b)$
   
   (iii) Find *minimum perfect matching* on $H$; or break.
   
   (iv) **Attach** some light-components $C_i$ to its nearest heavy component based on match with *heavy* node.
   
   (v) **Decompose** heavy components using *Lemma 3* and add the obtained trees to $S$.
   
   (vi) Add MST of $C_i$ to $S$, based on match with *null* node.
   
   (vii) Join $C_i$ and $C_j$ with edge and add ST to $S$, based on match with *regular* node.
   
   (viii) If $|S| \leq k$ then return $S$.

3: end for

4: return failure
A 2.5-approximation algorithm for BTC
Bounded Tree Cover Problem (BTC)

BTC

For an undirected graph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{Z}$ and a parameter $\lambda$, find a tree cover with minimum number of trees s.t. the weight of every tree in the cover is at most $\lambda$.

Main Differences:
- Delete edges with weight $> \lambda/4$
- No more guessing $\lambda$
Sources:
