Covering Graphs using Trees and Stars

Amitabh Sinha

GSIA, Carnegie Mellon University

(Work done while visiting MPII Saarbrucken, Germany)

Covering Graphs using Trees and Stars

Amitabh Sinha

Joint work with G. Even

GSIA, Carnegie Mellon University

(Work done while visiting MPII Saarbrucken, Germany)

Covering Graphs using Trees and Stars

Amitabh Sinha

Joint work with G. Even   N. Garg

GSIA, Carnegie Mellon University

(Work done while visiting MPII Saarbrucken, Germany)

Covering Graphs using Trees and Stars

Amitabh Sinha

Joint work with G. Even   N. Garg   J. Könemann

GSIA, Carnegie Mellon University

(Work done while visiting MPII Saarbrucken, Germany)

Covering Graphs using Trees and Stars

Amitabh Sinha

Joint work with G. Even N. Garg J. Könemann R. Ravi

GSIA, Carnegie Mellon University

(Work done while visiting MPII Saarbrucken, Germany)

Motivation: Nurse station location

- Hospital;
  k nurses (each with her own station);
  n patients in various beds.

Motivation: Nurse station location

- Hospital;
  k nurses (each with her own station);
  n patients in various beds.

- At 8 am, each nurse begins her “morning round” of patients under her care.

Motivation: Nurse station location

- Hospital;
  k nurses (each with her own station);
  n patients in various beds.

- At 8 am, each nurse begins her “morning round” of patients under her care.

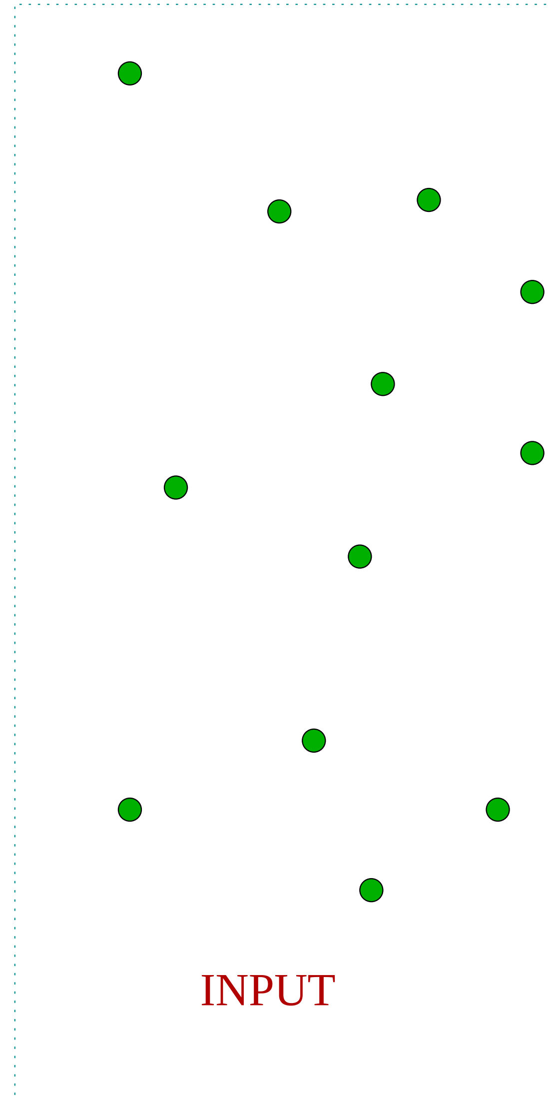
- Morning round ends when *all* nurses have returned to their bases.

Motivation: Nurse station location

- Hospital;
  k nurses (each with her own station);
  n patients in various beds.
- At 8 am, each nurse begins her "morning round" of patients under her care.
- Morning round ends when *all* nurses have returned to their bases.
- Objective: Assign patients to nurses so that morning rounds end ASAP.

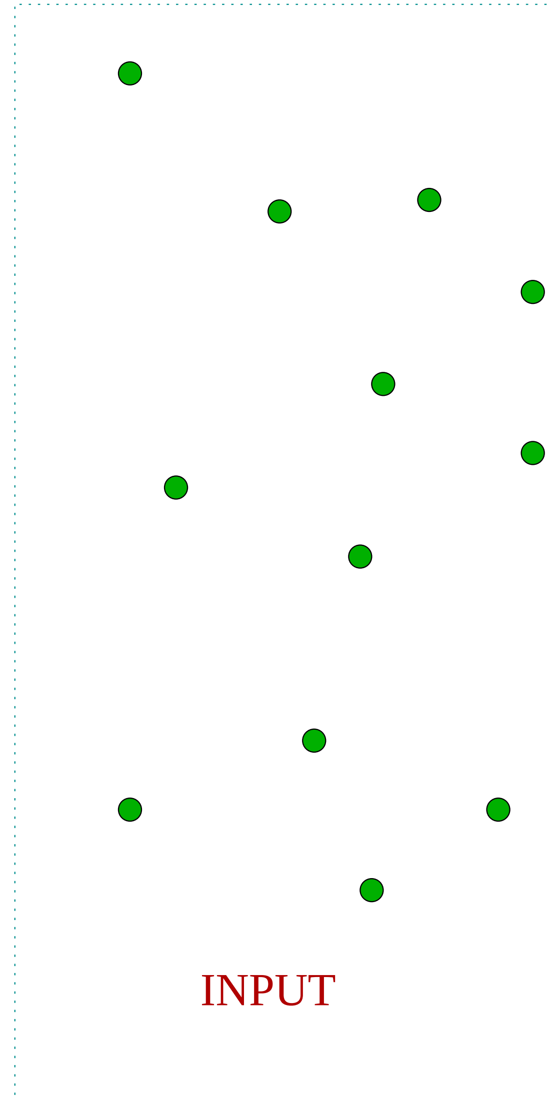
Problem definition

- Input: Graph $G = (V, E)$, edge weights w, integer k.



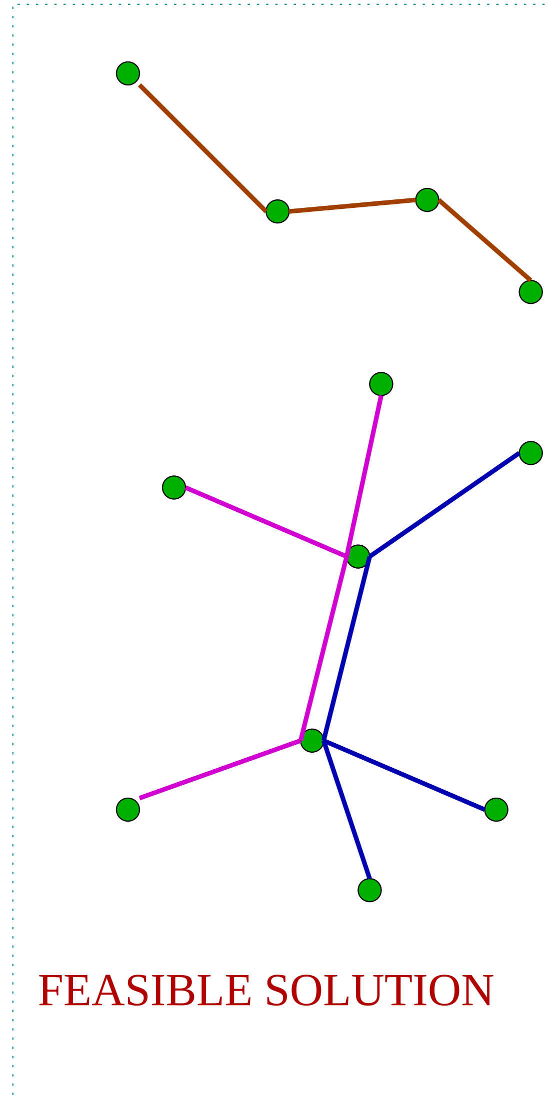
Problem definition

- Input: Graph $G = (V, E)$, edge weights w, integer k.
- k-Tree cover: Set of trees $\{T_1, T_2, \ldots, T_k\}$ such that $\cup_{i=1}^{k} V(T_i) = V$.



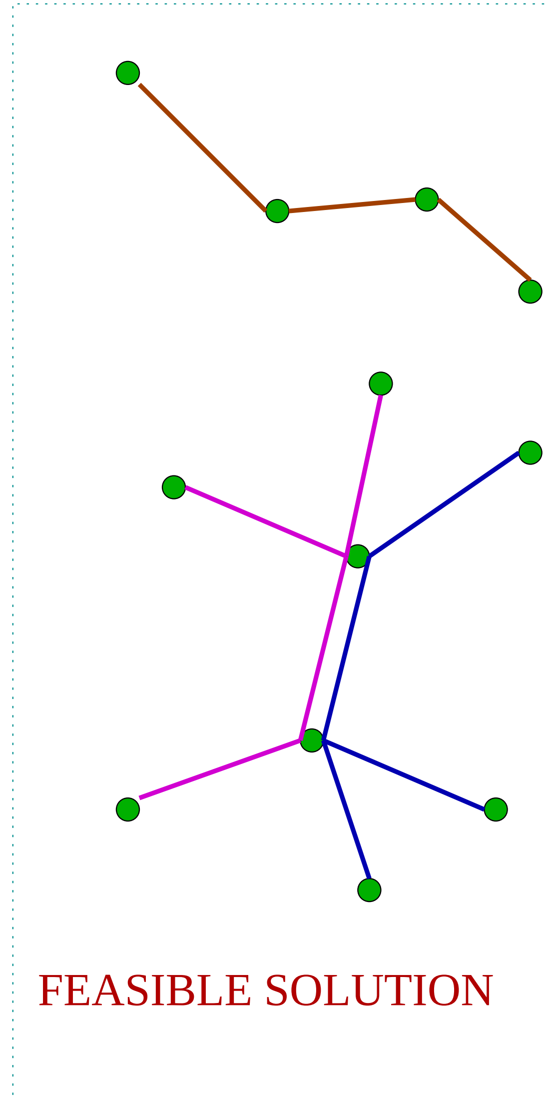
Problem definition

- Input: Graph $G = (V, E)$, edge weights w, integer k.
- k-Tree cover: Set of trees $\{T_1, T_2, \ldots, T_k\}$ such that $\cup_{i=1}^{k} V(T_i) = V$.



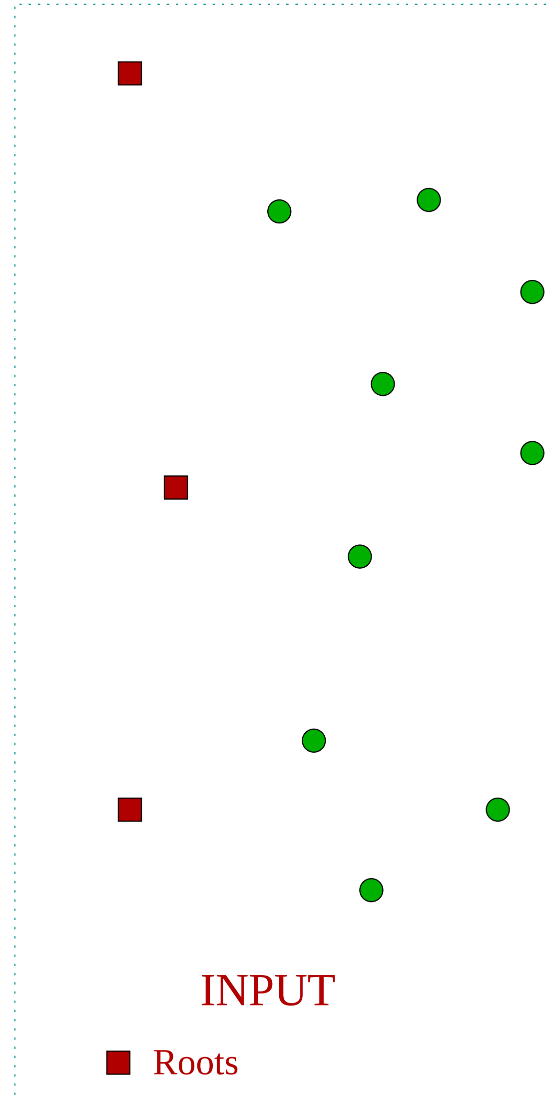
Problem definition

- Input: Graph $G = (V, E)$, edge weights w, integer k.
- k-Tree cover: Set of trees $\{T_1, T_2, \ldots, T_k\}$ such that $\cup_{i=1}^{k} V(T_i) = V$.
- Objective: Minimize $\max_i w(T_i)$.



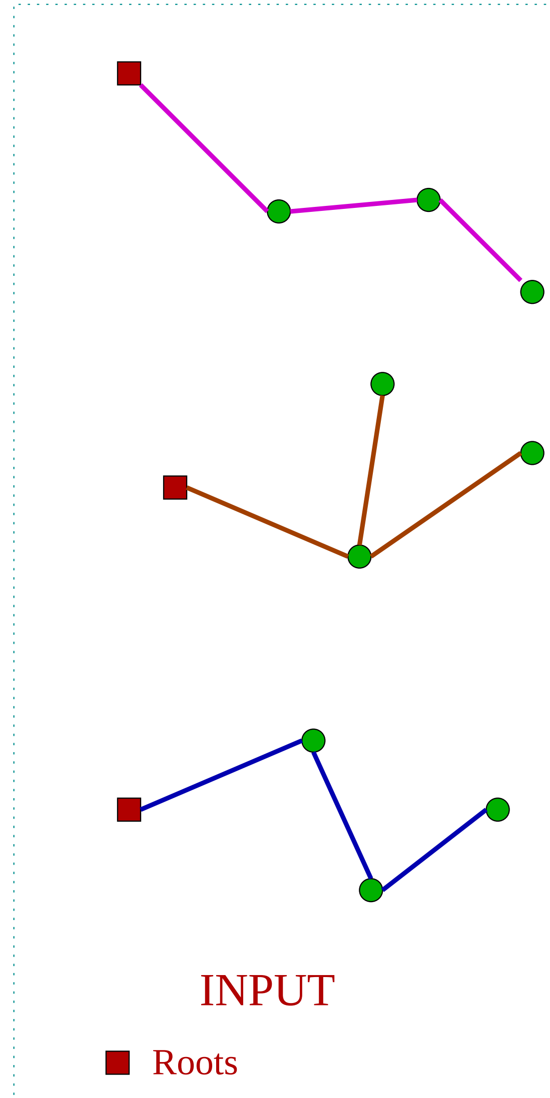
Problem definition

- Input: Graph $G = (V, E)$, edge weights w, integer k.
- k-Tree cover: Set of trees $\{T_1, T_2, \ldots, T_k\}$ such that $\cup_{i=1}^{k} V(T_i) = V$.
- Objective: Minimize $\max_i w(T_i)$.
- Rooted version: Given *roots* $R \subset V$, find a k-Tree cover with each tree using a distinct root in R.



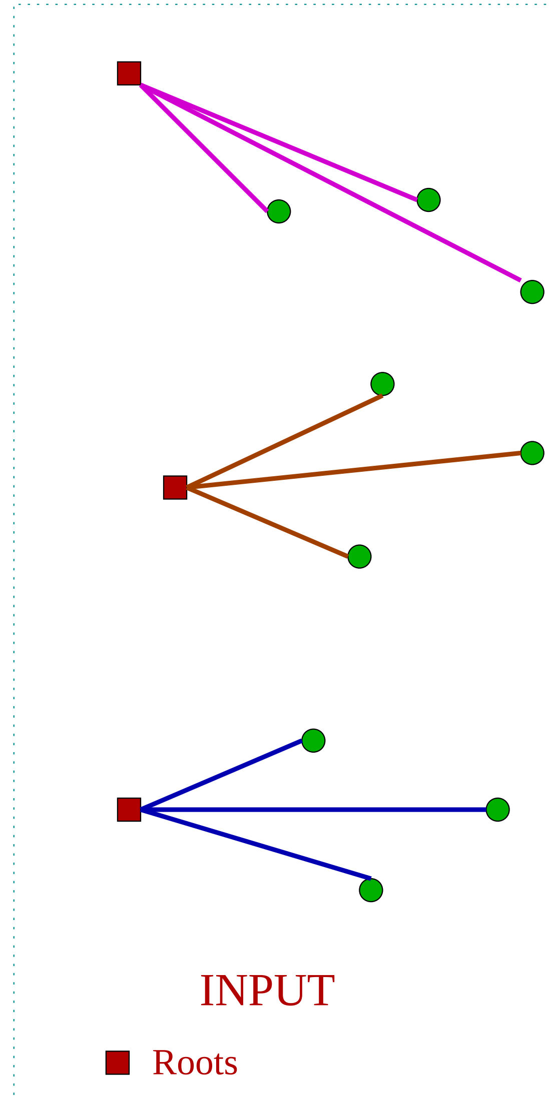
Problem definition

- Input: Graph $G = (V, E)$, edge weights w, integer k.
- k-Tree cover: Set of trees $\{T_1, T_2, \ldots, T_k\}$ such that $\cup_{i=1}^{k} V(T_i) = V$.
- Objective: Minimize $\max_i w(T_i)$.
- Rooted version: Given *roots* $R \subset V$, find a k-Tree cover with each tree using a distinct root in R.



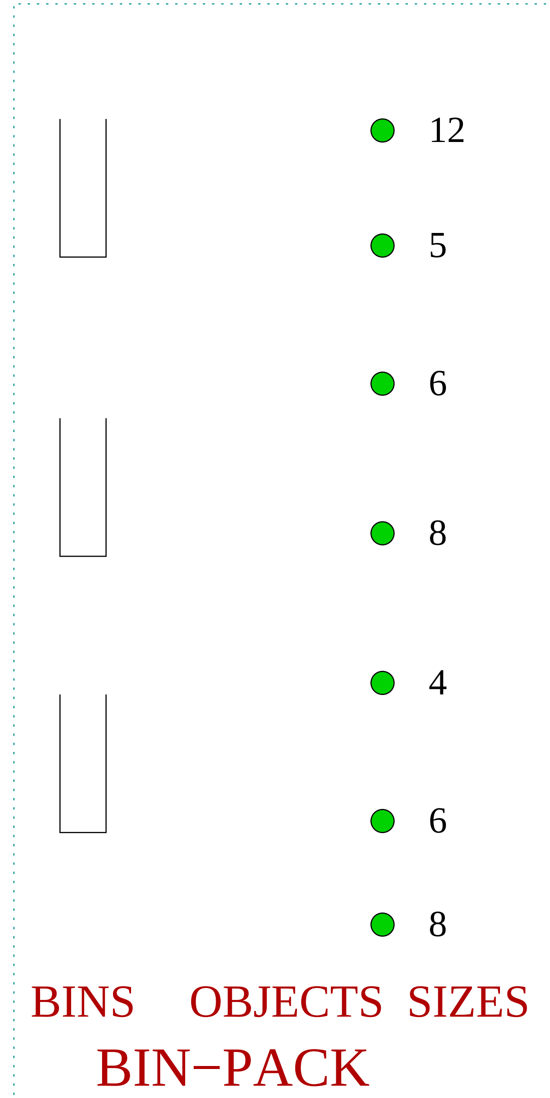
Problem definition

- **Input**: Graph $G = (V, E)$, edge weights w, integer k.
- **k-Tree cover**: Set of trees $\{T_1, T_2, \ldots, T_k\}$ such that $\cup_{i=1}^{k} V(T_i) = V$.
- **Objective**: Minimize $\max_i w(T_i)$.
- **Rooted version**: Given *roots* $R \subset V$, find a k-Tree cover with each tree using a distinct root in R.
- **Star cover**: Cover with stars, same objective; may be rooted or unrooted.



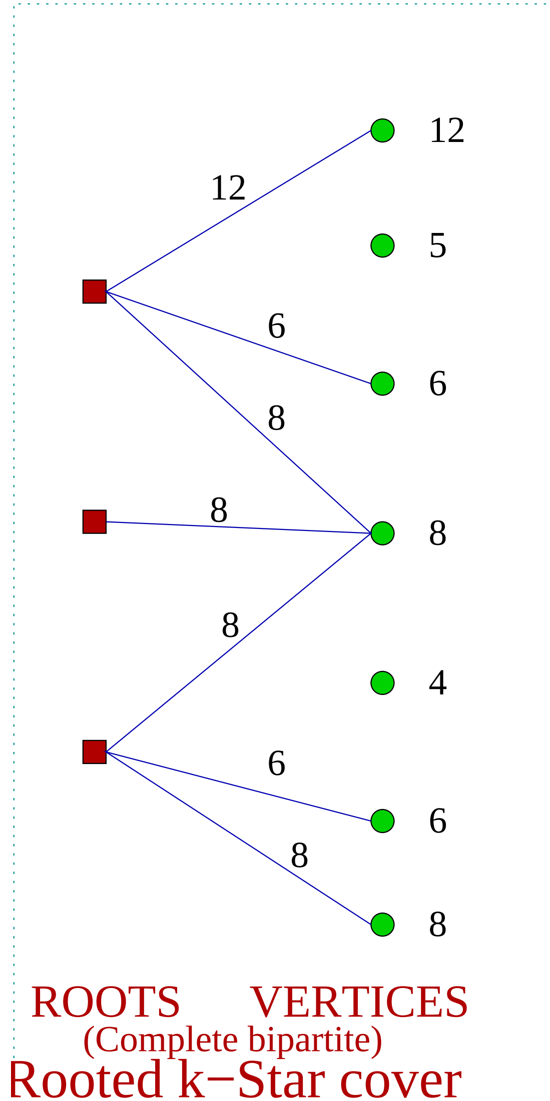
Hardness (of rooted k-star cover)

- Reduction from BIN-PACK:
  Given elements U with sizes s_u, k bins of size B. Can we pack elements in k bins?



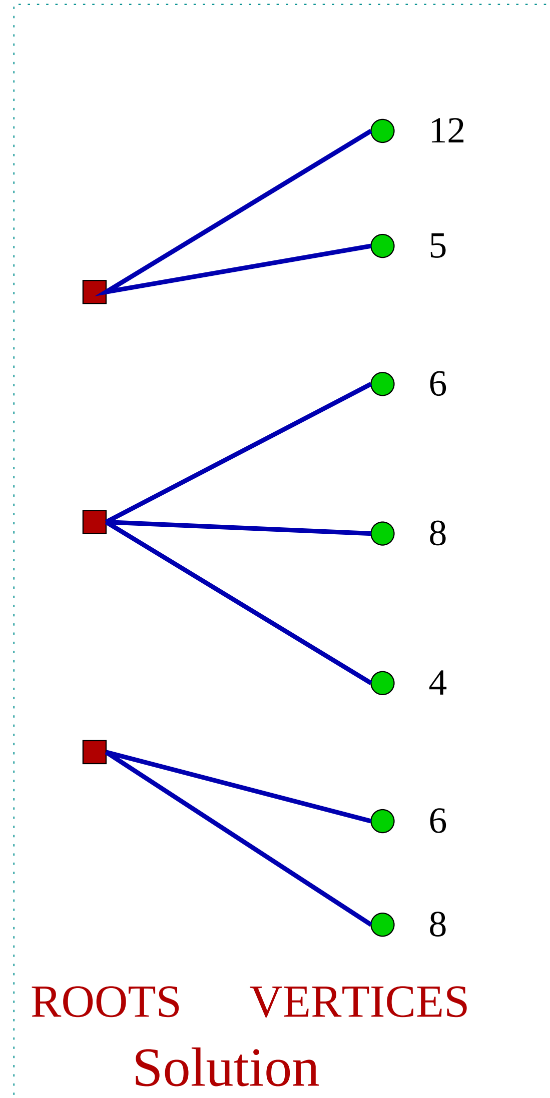
Hardness (of rooted k-star cover)

- Reduction from BIN-PACK: Given elements U with sizes s_u, k bins of size B. Can we pack elements in k bins?
- Convert to Rooted k-star cover: Complete bipartite graph between elements and bins, edge weights = element sizes, bins = roots.



Hardness (of rooted k-star cover)

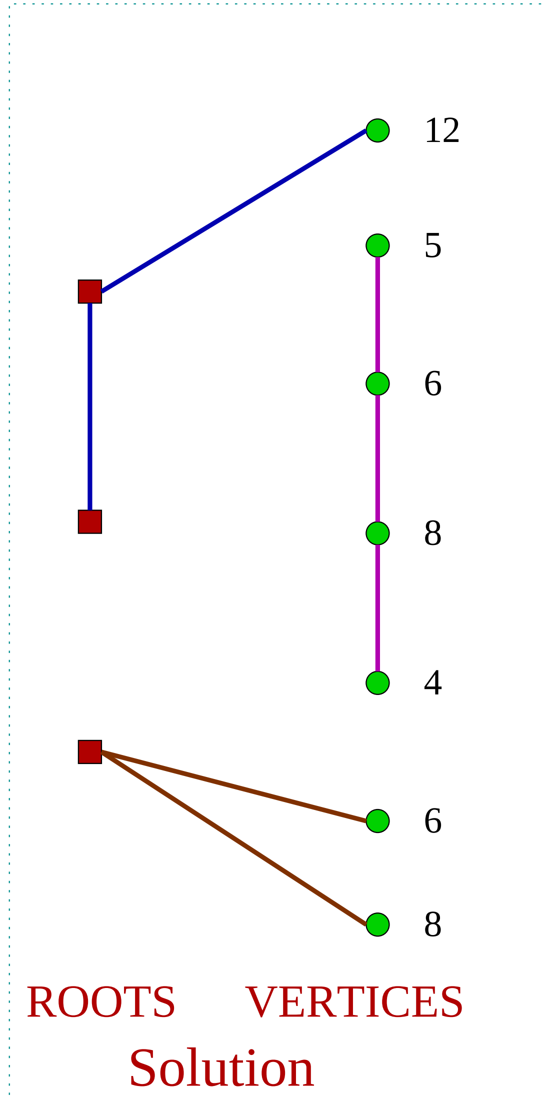
- Reduction from BIN-PACK: Given elements U with sizes s_u, k bins of size B. Can we pack elements in k bins?
- Convert to Rooted k-star cover: Complete bipartite graph between elements and bins, edge weights = element sizes, bins = roots.
- Claim: BIN-PACK is identical to this special case of Rooted k-star cover.



Hardness of others

- Also by reduction from BIN-PACK.

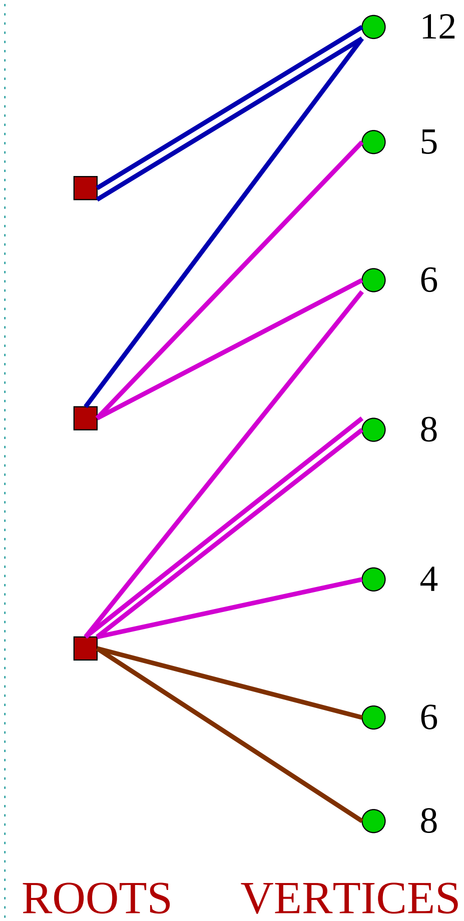
- Key: Poly time algorithm to convert any solution to Rooted k-star cover solution without increasing cost. (In BIN-PACK graph.)



Hardness of others

- Also by reduction from BIN-PACK.

- Key: Poly time algorithm to convert any solution to Rooted k-star cover solution without increasing cost. (In BIN-PACK graph.)

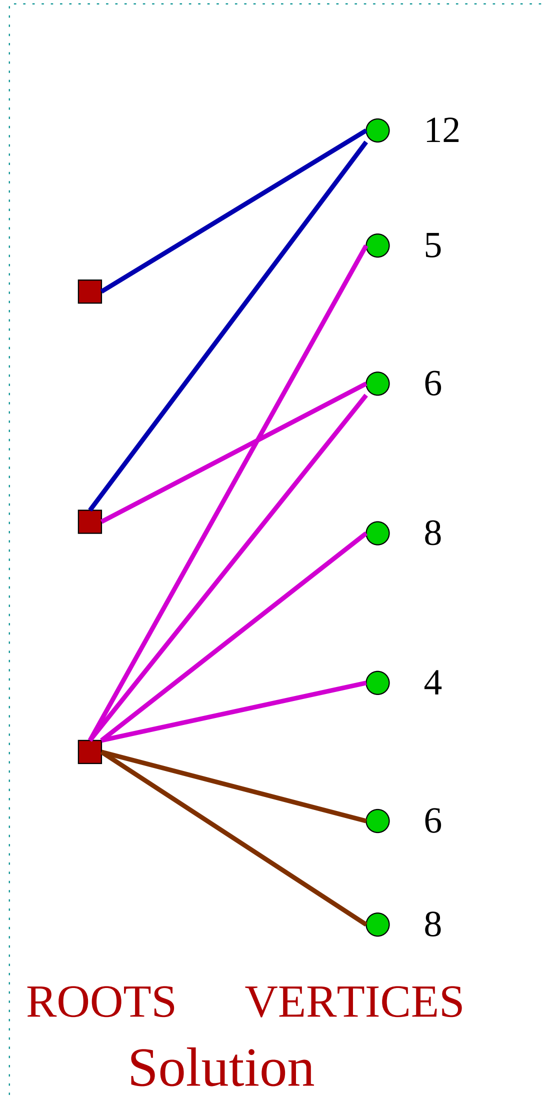


Solution

Hardness of others

- Also by reduction from BIN-PACK.

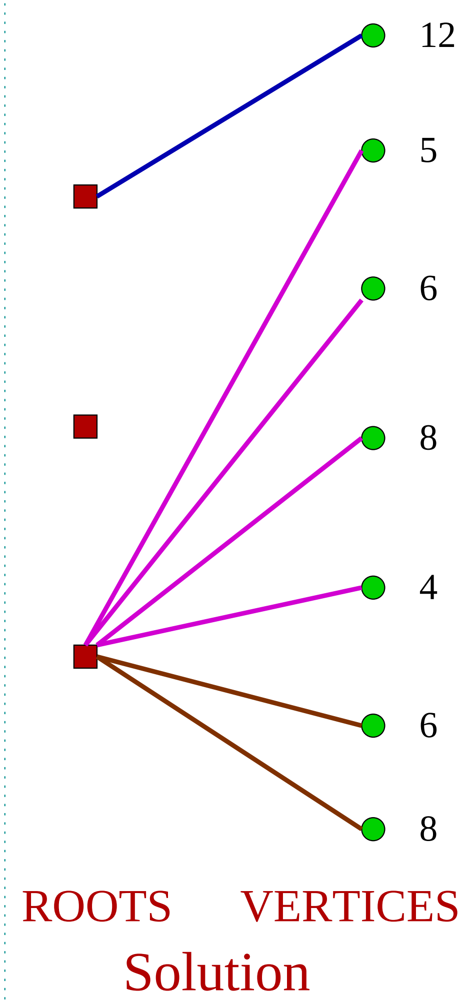
- Key: Poly time algorithm to convert any solution to Rooted k-star cover solution without increasing cost. (In BIN-PACK graph.)



Hardness of others

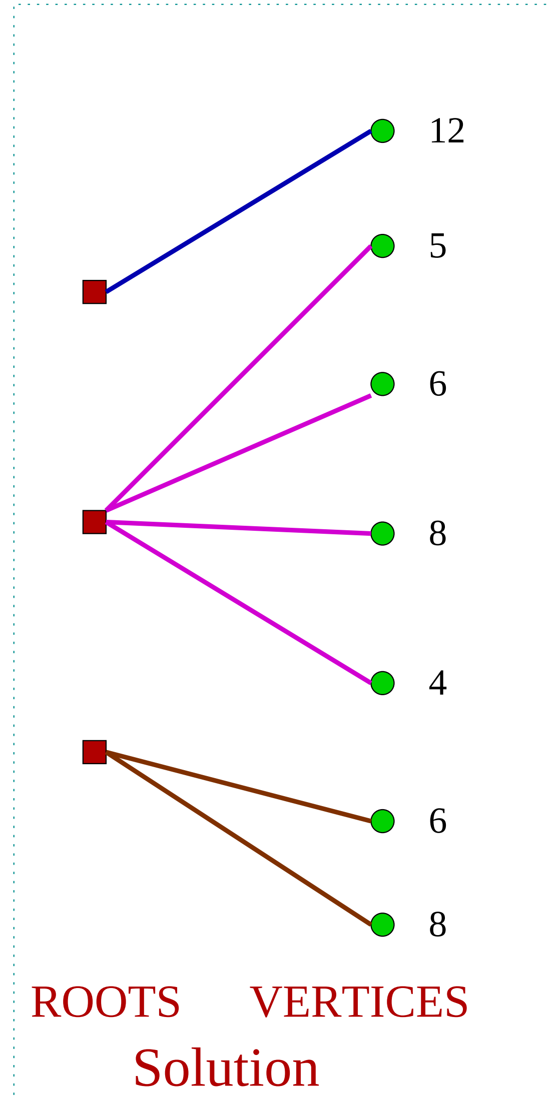
- Also by reduction from BIN-PACK.

- Key: Poly time algorithm to convert any solution to Rooted k-star cover solution without increasing cost. (In BIN-PACK graph.)



Hardness of others

- Also by reduction from BIN-PACK.
- Key: Poly time algorithm to convert any solution to Rooted k-star cover solution without increasing cost. (In BIN-PACK graph.)



Algorithm for Rooted k-tree cover

- Guess-and-check type algorithm.

Algorithm for Rooted k-tree cover

- Guess-and-check type algorithm.
- Guess optimal solution cost B. Let true optimum be B^*.
  - If “fail”, then proof that $B < B^*$.
  - If “success”, then find solution of cost no more than $4B$.

Algorithm for Rooted k-tree cover

- Guess-and-check type algorithm.
- Guess optimal solution cost B. Let true optimum be B^*.
  - If "fail", then proof that $B < B^*$.
  - If "success", then find solution of cost no more than $4B$.
- Binary search yields (weakly) polynomial time 4-approximation algorithm.

Algorithm for Rooted k-tree cover

- Guess-and-check type algorithm.

- Guess optimal solution cost B. Let true optimum be B^*.
  - If "fail", then proof that $B < B^*$.
  - If "success", then find solution of cost no more than $4B$.

- Binary search yields (weakly) polynomial time 4-approximation algorithm.

- Can be made strongly polynomial; approximation ratio worsens to $4 + \epsilon$.

Algorithm: Overview

Given B, set of roots R, and G.

1. Remove all edges with $w_e > B$.

Algorithm: Overview

Given B, set of roots R, and G.

1. Remove all edges with $w_e > B$.
2. Contract R; compute MST M.
   $\{T_i\}_i :=$ forest obtained by expanding R.

Algorithm: Overview

Given B, set of roots R, and G.

1. Remove all edges with $w_e > B$.
2. Contract R; compute MST M.
   $\{T_i\}_i :=$ forest obtained by expanding R.
3. Decompose each T_i into trees $\{S_i^j\}^j + L_i$ s.t.
   $w(S_i^j) \in [B, 2B)$ and $w(L_i) < B$.

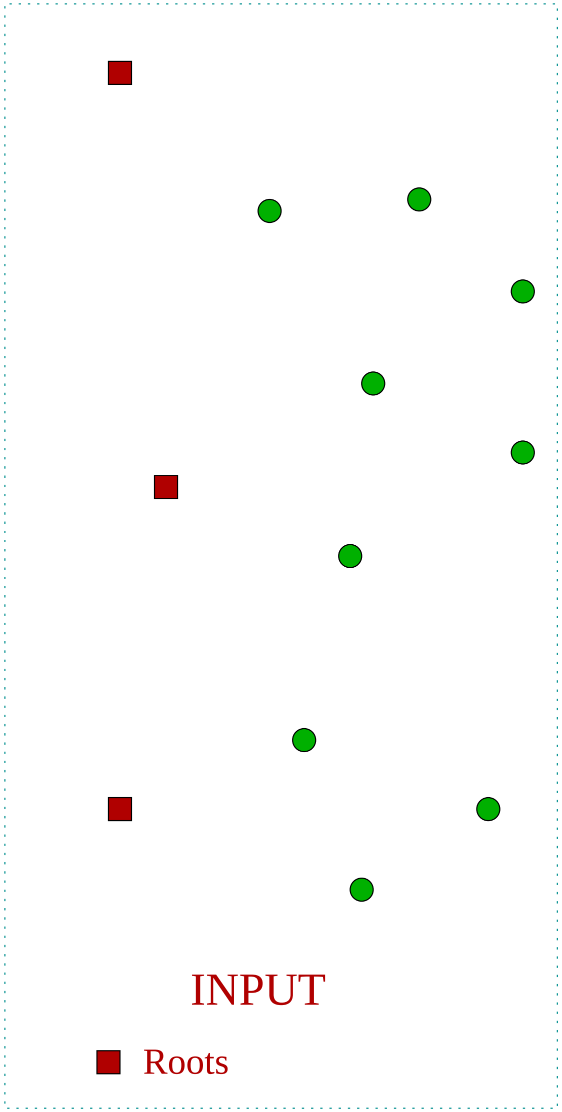
Algorithm: Overview

Given B, set of roots R, and G.

1. Remove all edges with $w_e > B$.
2. Contract R; compute MST M.
   $\{T_i\}_i :=$ forest obtained by expanding R.
3. Decompose each T_i into trees $\{S_i^j\}^j + L_i$ s.t.
   $w(S_i^j) \in [B, 2B)$ and $w(L_i) < B$.
4. Match trees $\{S_i^j\}_i^j$ to roots in R within distance B from it.
   - If possible, return "success".
   - If impossible, return "fail".

Algorithm: Demonstration

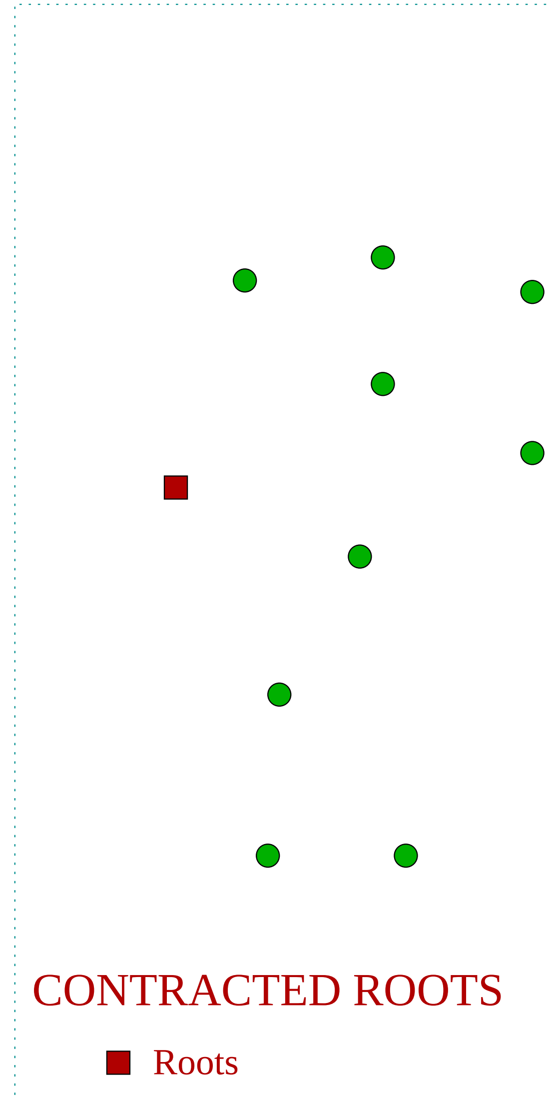
1. Prune.



Algorithm: Demonstration

1. Prune.

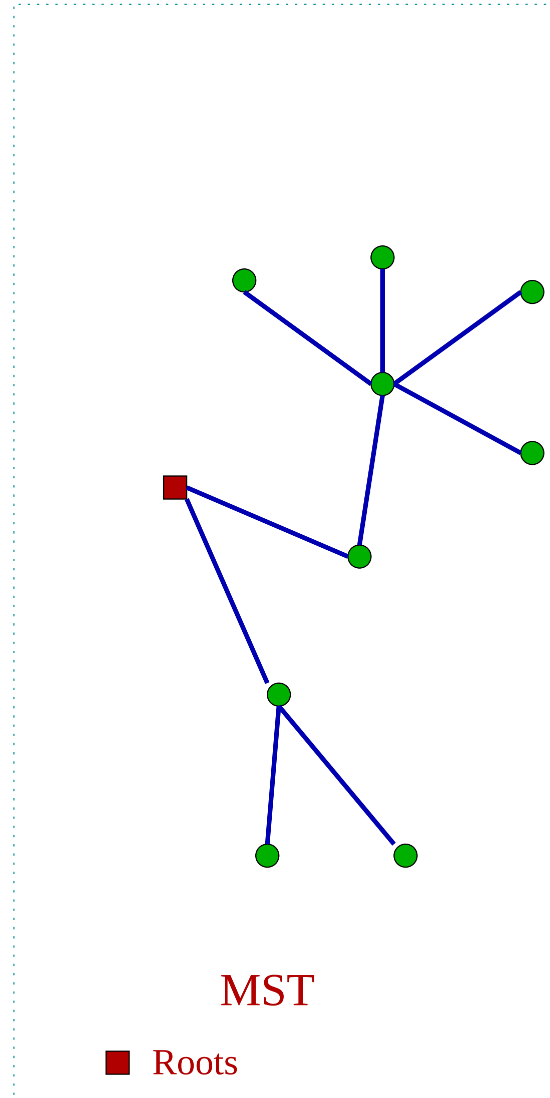
2. Contract R, compute MST.



Algorithm: Demonstration

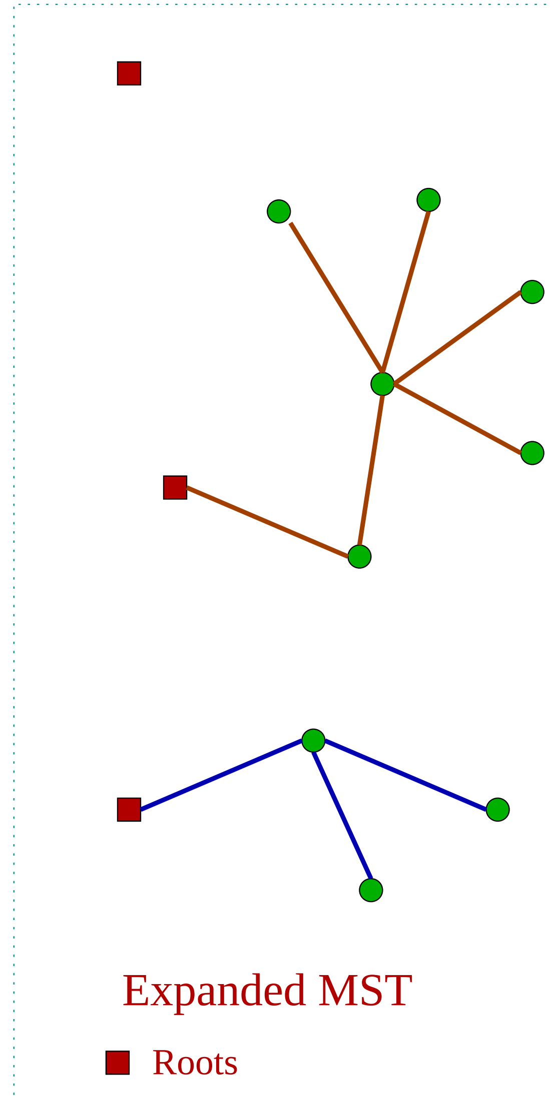
1. Prune.

2. Contract R, compute MST.



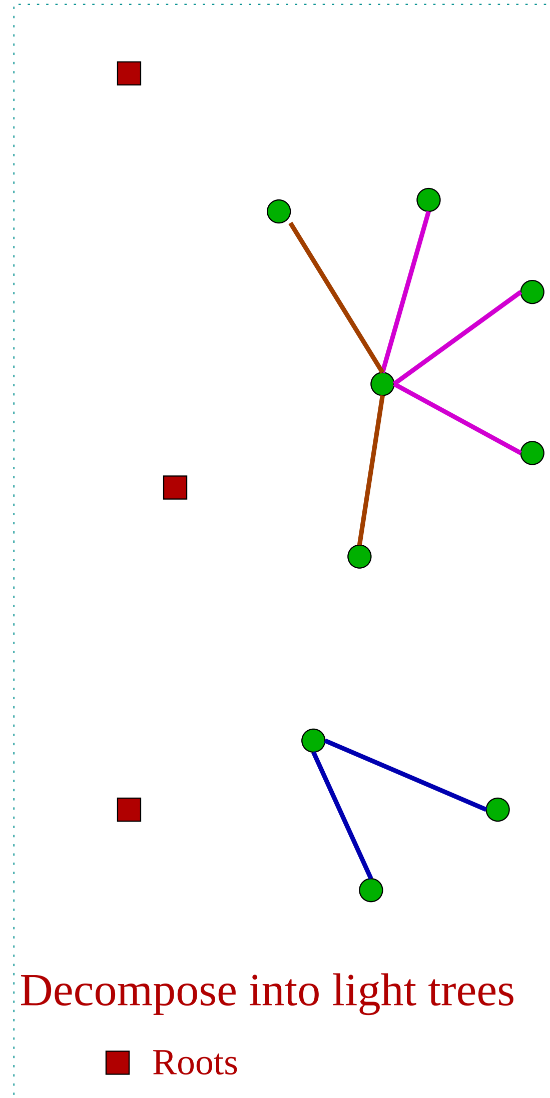
Algorithm: Demonstration

1. Prune.
2. Contract R, compute MST.



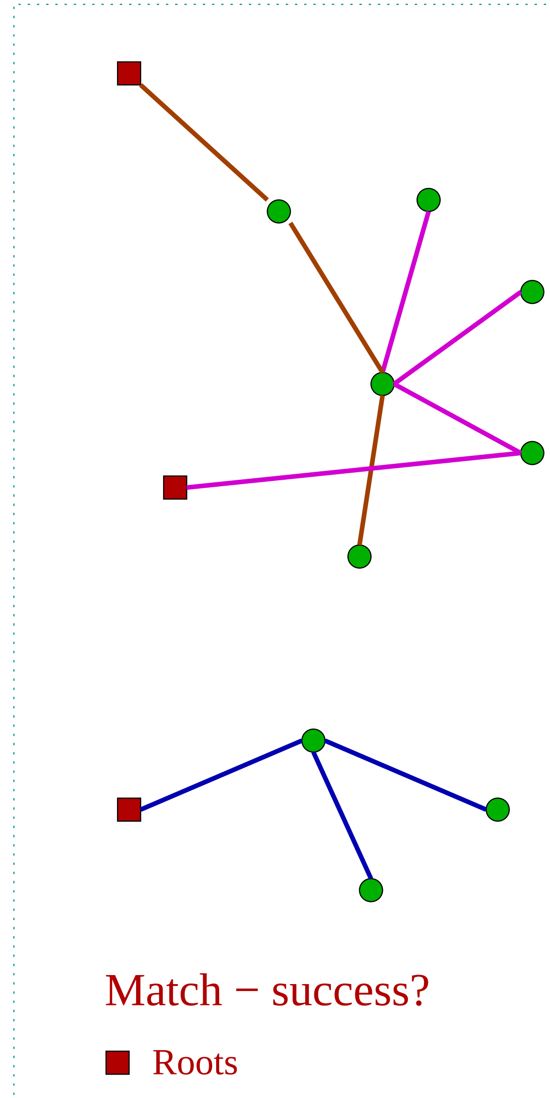
Algorithm: Demonstration

1. Prune.
2. Contract R, compute MST.
3. Decompose.



Algorithm: Demonstration

1. Prune.
2. Contract R, compute MST.
3. Decompose.
4. Match.



Algorithm: Success

Claim: On success, each tree has cost no more than $4B$.

Algorithm: Success

Claim: On success, each tree has cost no more than $4B$.

Proof: Each tree in our solution has 3 components:

Decomposed tree S_i^j, cost $\leq$ $2B$.

Algorithm: Success

Claim: On success, each tree has cost no more than $4B$.

Proof: Each tree in our solution has 3 components:

Decomposed tree S_i^j, cost $\leq$ $2B$.

Edge to root, cost $\leq$ B.

Algorithm: Success

Claim: On success, each tree has cost no more than $4B$.

Proof: Each tree in our solution has 3 components:

Decomposed tree S_i^j, cost $\leq$ $2B$.

Edge to root, cost $\leq$ B.

Leftover tree L_i, cost $\leq$ B.

$\square$

Algorithm: Failure

Lemma: On failure (matching does not exist), $B < B^*$.

Algorithm: Failure

Lemma: On failure (matching does not exist), $B < B^*$.

Alternatively: If $B \geq B^*$, matching exists.

Algorithm: Failure

Lemma: On failure (matching does not exist), $B < B^*$.

Alternatively: If $B \geq B^*$, matching exists.

Proof: Hall's Theorem: We show $|N(S)| \geq |S|$ for all $S \subseteq \{S_i^j\}_i^j$.

Algorithm: Failure

Lemma: On failure (matching does not exist), $B < B^*$.

Alternatively: If $B \geq B^*$, matching exists.

Proof: Hall's Theorem: We show $|N(S)| \geq |S|$ for all $S \subseteq \{S_i^j\}_i^j$.

Consider optimal solution $T^* = \{T_1^*, \ldots, T_k^*\}$. Let $T^*(S) = T^* \cap S$. Hence $|N(S)| \geq |T^*(S)|$.

Algorithm: Failure

Lemma: On failure (matching does not exist), $B < B^*$.

Alternatively: If $B \geq B^*$, matching exists.

Proof: Hall's Theorem: We show $|N(S)| \geq |S|$ for all $S \subseteq \{S_i^j\}_i^j$.

Consider optimal solution $T^* = \{T_1^*, \ldots, T_k^*\}$. Let $T^*(S) = T^* \cap S$. Hence $|N(S)| \geq |T^*(S)|$.

Deleting all edges in S and adding all edges in $T^*(S)$ also yields a spanning tree of G, and since our tree was MST, $w(T^*(S)) \geq w(S)$.

Algorithm: Failure

Lemma: On failure (matching does not exist), $B < B^*$.

Alternatively: If $B \geq B^*$, matching exists.

Proof: Hall's Theorem: We show $|N(S)| \geq |S|$ for all $S \subseteq \{S_i^j\}_i^j$.

Consider optimal solution $T^* = \{T_1^*, \ldots, T_k^*\}$. Let $T^*(S) = T^* \cap S$. Hence $|N(S)| \geq |T^*(S)|$.

Deleting all edges in S and adding all edges in $T^*(S)$ also yields a spanning tree of G, and since our tree was MST, $w(T^*(S)) \geq w(S)$.

$B^*|N(S)| \geq B^*|T^*(S)| \geq w(T^*(S)) \geq w(S) \geq B|S|$. $\square$

Strongly polynomial algorithm

Fix $\epsilon > 0$.

- Sort edges $w_1 \leq w_2 \leq \ldots \leq w_m$.

Strongly polynomial algorithm

Fix $\epsilon > 0$.

- Sort edges $w_1 \leq w_2 \leq \ldots \leq w_m$.
- If algorithm says $w_m = B < B^*$, then contract all edges of weight at most $\frac{\epsilon w_m}{n^2}$. Now binary search in range $[\frac{\epsilon w_m}{n^2}, n w_m]$, which is polynomial.

Strongly polynomial algorithm

Fix $\epsilon > 0$.

- Sort edges $w_1 \leq w_2 \leq \ldots \leq w_m$.
- If algorithm says $w_m = B < B^*$, then contract all edges of weight at most $\frac{\epsilon w_m}{n^2}$. Now binary search in range $[\frac{\epsilon w_m}{n^2}, n w_m]$, which is polynomial.
- Otherwise, find i such that $B^* \in (w_i, 4w_{i+1}]$.

Strongly polynomial algorithm

Fix $\epsilon > 0$.

- Sort edges $w_1 \leq w_2 \leq \ldots \leq w_m$.
- If algorithm says $w_m = B < B^*$, then contract all edges of weight at most $\frac{\epsilon w_m}{n^2}$. Now binary search in range $[\frac{\epsilon w_m}{n^2}, n w_m]$, which is polynomial.
- Otherwise, find i such that $B^* \in (w_i, 4w_{i+1}]$.
- If $\frac{w_{i+1}}{w_i} \leq \frac{n^2}{\epsilon}$, binary search in above range is polynomial.

Strongly polynomial algorithm

Fix $\epsilon > 0$.

- Sort edges $w_1 \leq w_2 \leq \ldots \leq w_m$.
- If algorithm says $w_m = B < B^*$, then contract all edges of weight at most $\frac{\epsilon w_m}{n^2}$. Now binary search in range $[\frac{\epsilon w_m}{n^2}, n w_m]$, which is polynomial.
- Otherwise, find i such that $B^* \in (w_i, 4w_{i+1}]$.
- If $\frac{w_{i+1}}{w_i} \leq \frac{n^2}{\epsilon}$, binary search in above range is polynomial.
- If not, set $w' = n^2 w_i / \epsilon$. If $B^* \in [w_i, w']$, then polynomial.

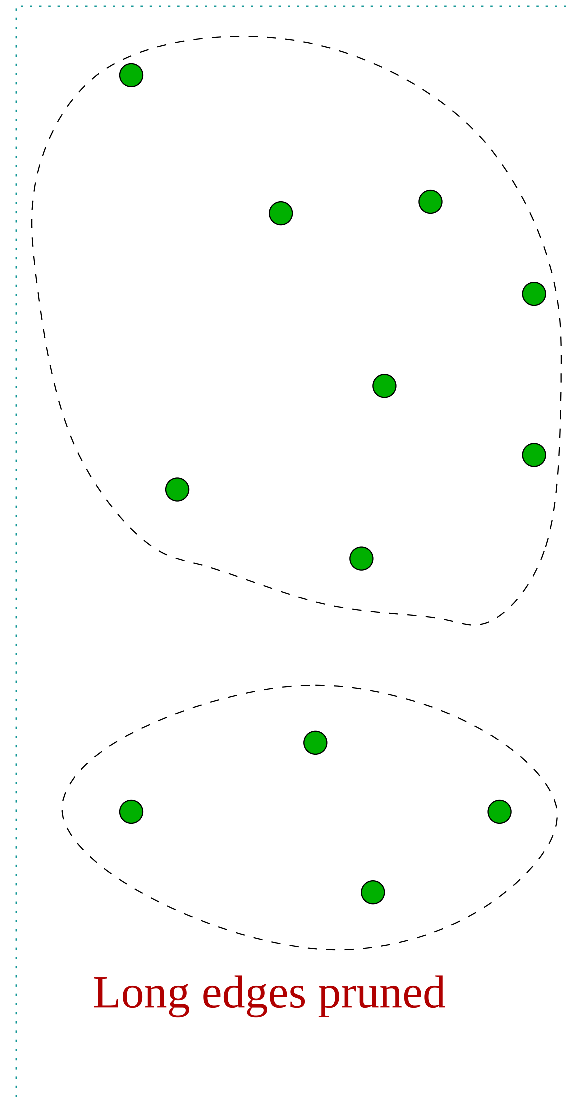
Strongly polynomial algorithm

Fix $\epsilon > 0$.

- Sort edges $w_1 \leq w_2 \leq \ldots \leq w_m$.
- If algorithm says $w_m = B < B^*$, then contract all edges of weight at most $\frac{\epsilon w_m}{n^2}$. Now binary search in range $[\frac{\epsilon w_m}{n^2}, n w_m]$, which is polynomial.
- Otherwise, find i such that $B^* \in (w_i, 4w_{i+1}]$.
- If $\frac{w_{i+1}}{w_i} \leq \frac{n^2}{\epsilon}$, binary search in above range is polynomial.
- If not, set $w' = n^2 w_i / \epsilon$. If $B^* \in [w_i, w']$, then polynomial.
- If not, then contract all edges of weight at most w_i. Now binary search in $[w_{i+1}, 4w_{i+1}]$ is polynomial.

Algorithm for Unrooted k-tree cover

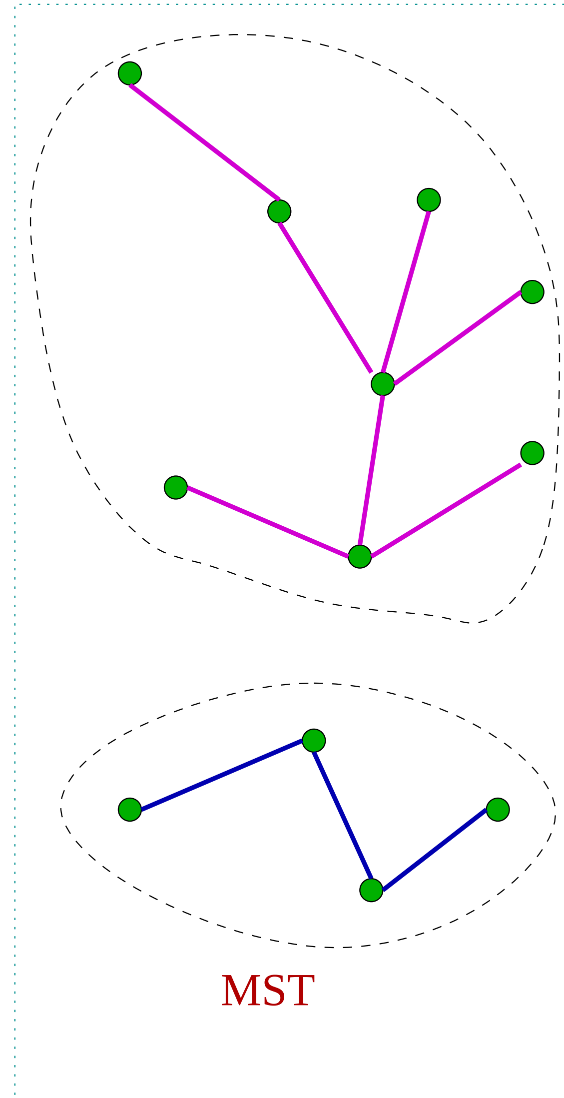
1. Prune edges $w_e > B$.
   Let $\{G_i\}_i$ be components.



Algorithm for Unrooted k-tree cover

1. Prune edges $w_e > B$.
   Let $\{G_i\}_i$ be components.

2. MST_i = MST of G_i.
   $k_i = \lfloor \frac{w(MST_i)}{2B} \rfloor$.

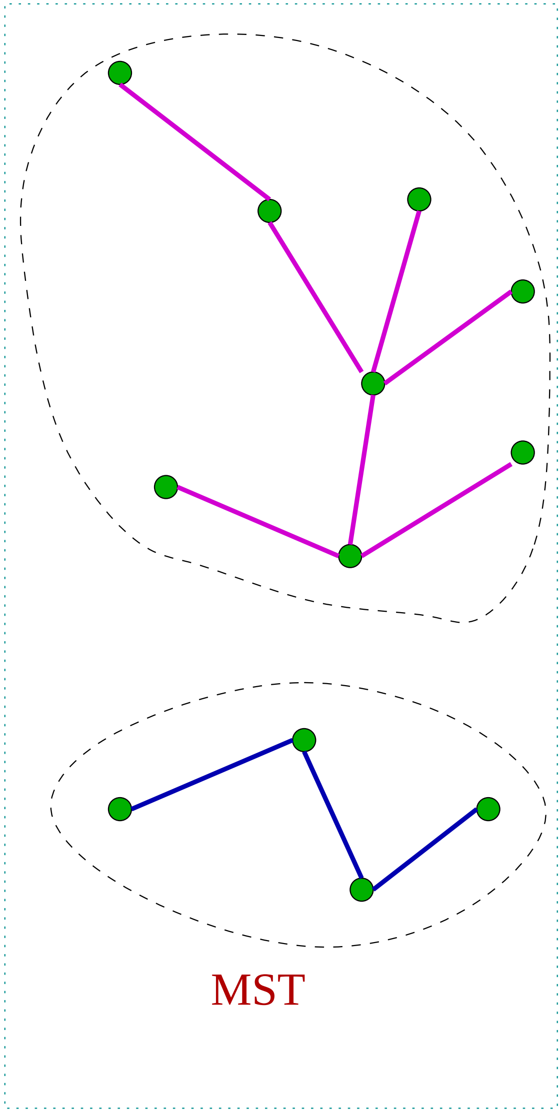


Algorithm for Unrooted k-tree cover

1. Prune edges $w_e > B$.
   Let $\{G_i\}_i$ be components.

2. MST_i = MST of G_i.
   $k_i = \lfloor \frac{w(MST_i)}{2B} \rfloor$.

3. If $\sum_i (k_i + 1) > k$, return "fail".



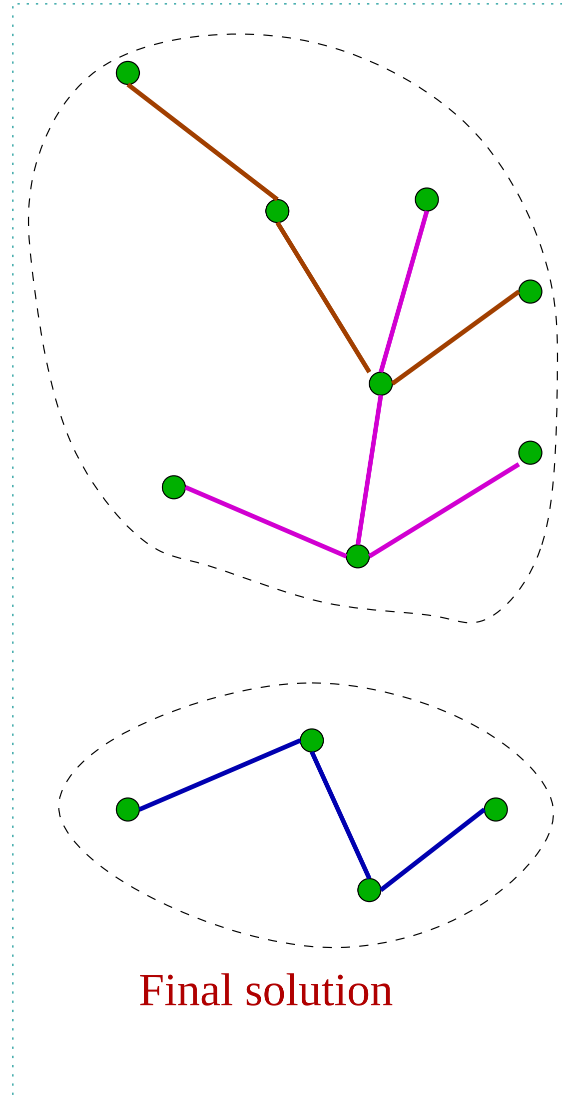
Algorithm for Unrooted k-tree cover

1. Prune edges $w_e > B$.
   Let $\{G_i\}_i$ be components.

2. MST_i = MST of G_i.
   $k_i = \lfloor \frac{w(MST_i)}{2B} \rfloor$.

3. If $\sum_i (k_i + 1) > k$, return "fail".

4. Decompose each MST_i into at most $k_i + 1$ trees $S_i^1 + \ldots + S_i^{k_i} + L_i$ such that $w(S_i^j) \in [2B, 4B)$ and $w(L_i) < 2B$. Return "success".



Analysis

Claim: On success, each tree has weight no more than $4B$.

Analysis

Claim: On success, each tree has weight no more than $4B$.

Claim: On failure, $B < B^*$.

Analysis

Claim: On success, each tree has weight no more than $4B$.

Claim: On failure, $B < B^*$.

Alternatively, if $B \geq B^*$, then $k_i + 1 \leq k_i^*$ for all i.

Analysis

Claim: On success, each tree has weight no more than $4B$.

Claim: On failure, $B < B^*$.

Alternatively, if $B \geq B^*$, then $k_i + 1 \leq k_i^*$ for all i.

Proof: Let optimal solution cover G_i with $\{T_1^*, \ldots, T_{k_i^*}^*\}$. We can make it span G_i by adding at most $k_i^* + 1$ edges, so:

$$\sum_{j=1}^{k_i^*} w(T_i^*) + (k_i^* - 1)B \geq w(MST_i)$$

.

Analysis

Claim: On success, each tree has weight no more than $4B$.

Claim: On failure, $B < B^*$.

Alternatively, if $B \geq B^*$, then $k_i + 1 \leq k_i^*$ for all i.

Proof: Let optimal solution cover G_i with $\{T_1^*, \ldots, T_{k_i^*}^*\}$. We can make it span G_i by adding at most $k_i^* + 1$ edges, so:

$$\sum_{j=1}^{k_i^*} w(T_i^*) + (k_i^* - 1)B \geq w(MST_i)$$

.

Therefore $k_i^* \geq \frac{w(MST_i)}{2B} + \frac{1}{2} > k_i$. □

Extensions and conclusion

- Rooted k-star cover: Reduces to Generalized Assignment problem, yields a 2-approximation.

Extensions and conclusion

- Rooted k-star cover: Reduces to Generalized Assignment problem, yields a 2-approximation.
- Unrooted k-star cover: LP rounding gives bicriteria approximation: Covers with $2k$ stars, each costing no more than twice the optimum.

Extensions and conclusion

- Rooted k-star cover: Reduces to Generalized Assignment problem, yields a 2-approximation.
- Unrooted k-star cover: LP rounding gives bicriteria approximation: Covers with $2k$ stars, each costing no more than twice the optimum.
- Tree cover algorithms also yield constant factor approximations for tour cover, the original nursing station location problem.

Extensions and conclusion

- Rooted k-star cover: Reduces to Generalized Assignment problem, yields a 2-approximation.

- Unrooted k-star cover: LP rounding gives bicriteria approximation: Covers with $2k$ stars, each costing no more than twice the optimum.

- Tree cover algorithms also yield constant factor approximations for tour cover, the original nursing station location problem.

- Questions?

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.