

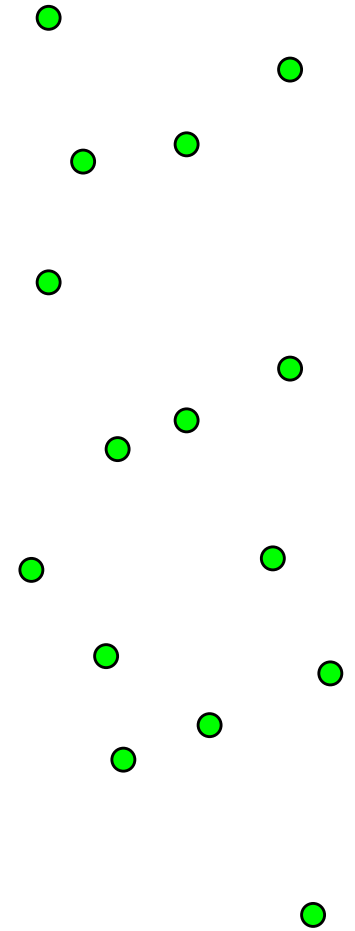
Covering Graphs Using Trees and Stars

Guy Even (Tel-Aviv), Naveen Garg (Delhi),
and

Jochen Könemann, R. Ravi and A. Sinha (Pittsburgh)

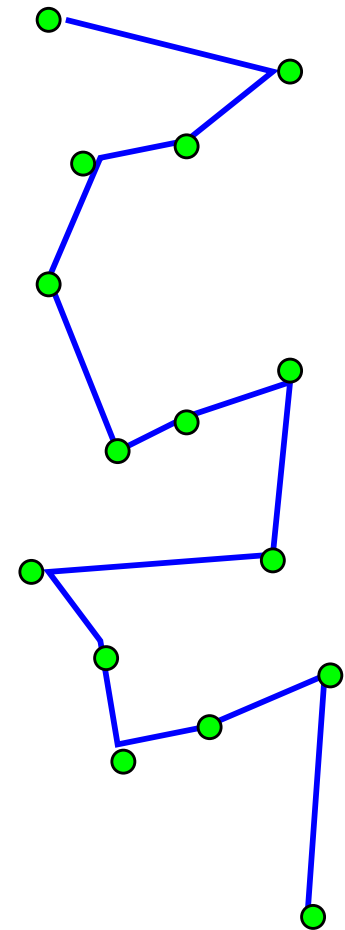
Motivation

- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.



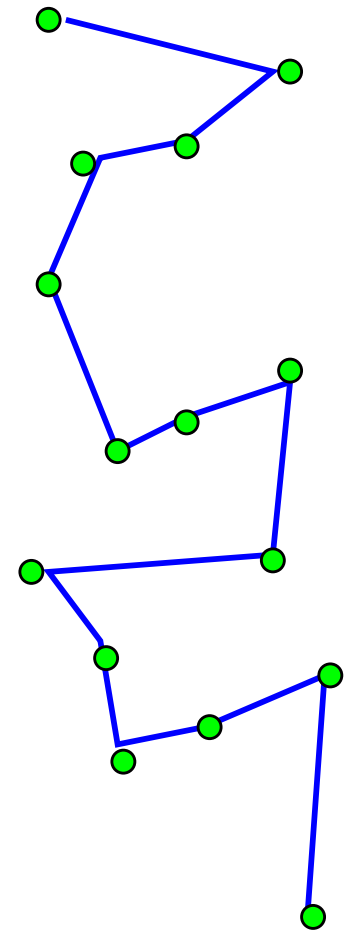
Motivation

- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.
- Suppose regulation dictates: agent may travel at most 100 km.



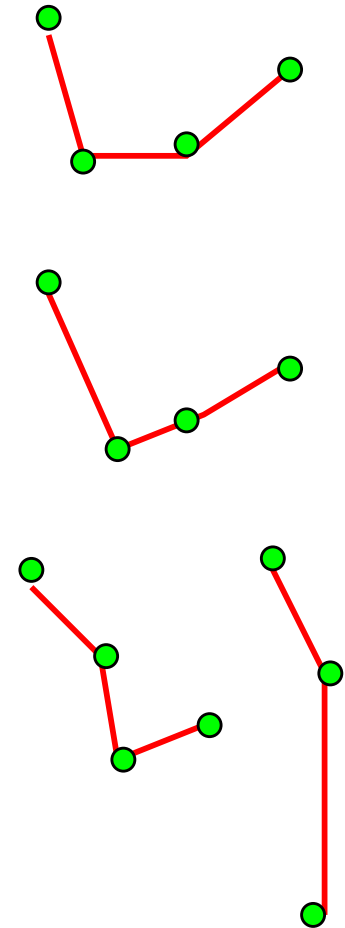
Motivation

- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.
- Suppose regulation dictates: agent may travel at most 100 km.
- \Rightarrow must employ multiple agents.



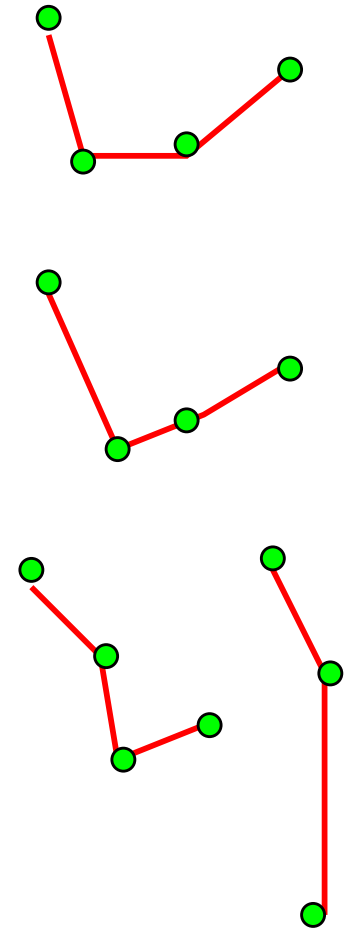
Motivation

- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.
- Suppose regulation dictates: agent may travel at most 100 km.
- \Rightarrow must employ multiple agents.
- \Rightarrow k -traveling salespeople problem.



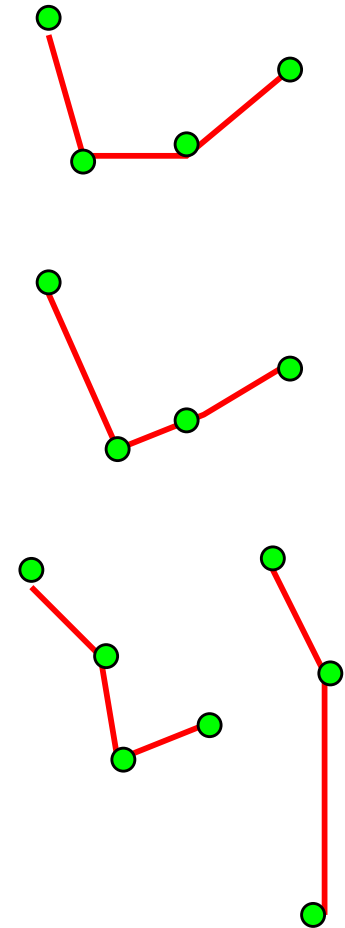
Motivation

- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.
- Suppose regulation dictates: agent may travel at most 100 km.
- \Rightarrow must employ multiple agents.
- \Rightarrow k -traveling salespeople problem.
 - Cover the vertices of a graph with k tours.



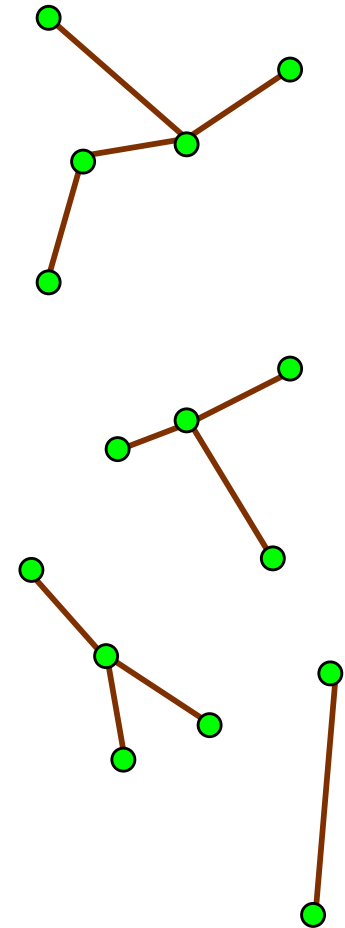
Motivation

- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.
- Suppose regulation dictates: agent may travel at most 100 km.
- \Rightarrow must employ multiple agents.
- \Rightarrow k -traveling salespeople problem.
 - Cover the vertices of a graph with k tours.
 - Balance the load of the agents: minimize the maximum tour.



Motivation

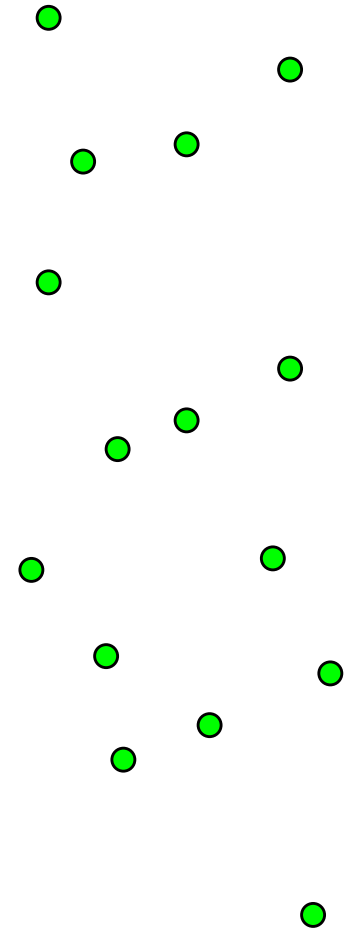
- Consider a TSP instance with a large optimal tour, e.g. $w(\text{tour}^*) > 1000$.
- Suppose regulation dictates: agent may travel at most 100 km.
- \Rightarrow must employ multiple agents.
- \Rightarrow k -traveling salespeople problem.
 - Cover the vertices of a graph with k tours.
 - Balance the load of the agents: minimize the maximum tour.
- MST is a constant ratio approx of a min tour \Rightarrow k -Tree Cover Problem.



k -Tree Cover Problem

k -Tree Cover Problem

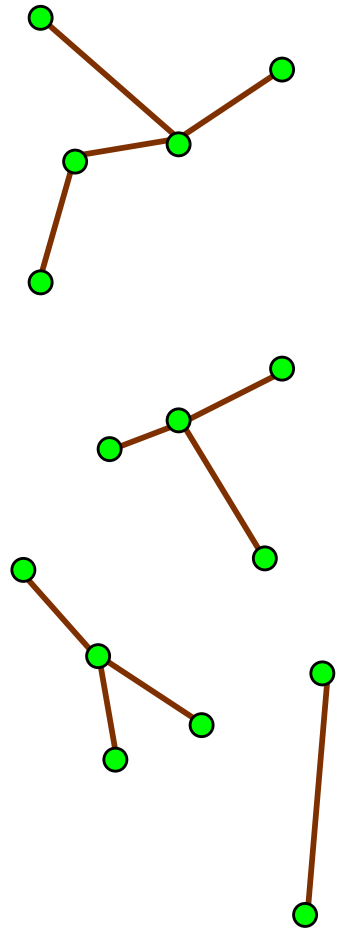
Input: (i) integer k and (ii) $G = (V, E)$ - an undirected graph with positive integral edge weights $w : E \rightarrow \mathbb{N}^+$.



k -Tree Cover Problem

Input: (i) integer k and (ii) $G = (V, E)$ - an undirected graph with positive integral edge weights $w : E \rightarrow \mathbb{N}^+$.

k -tree cover: a set \mathcal{T} of trees $\{T_i\}_i$ such that $V = \bigcup_{i=1}^k V(T_i)$.

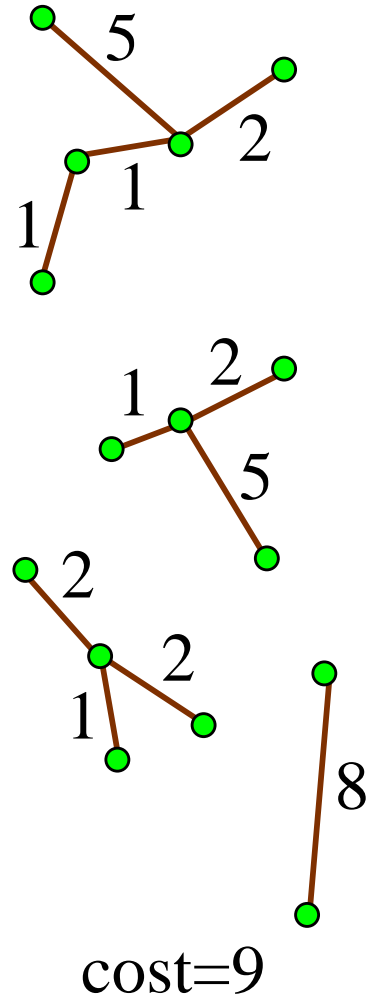


k -Tree Cover Problem

Input: (i) integer k and (ii) $G = (V, E)$ - an undirected graph with positive integral edge weights $w : E \rightarrow \mathbb{N}^+$.

k -tree cover: a set \mathcal{T} of trees $\{T_i\}_i$ such that $V = \bigcup_{i=1}^k V(T_i)$.

cost: $\text{cost}(\mathcal{T}) = \max_{T_i \in \mathcal{T}} w(T_i)$.



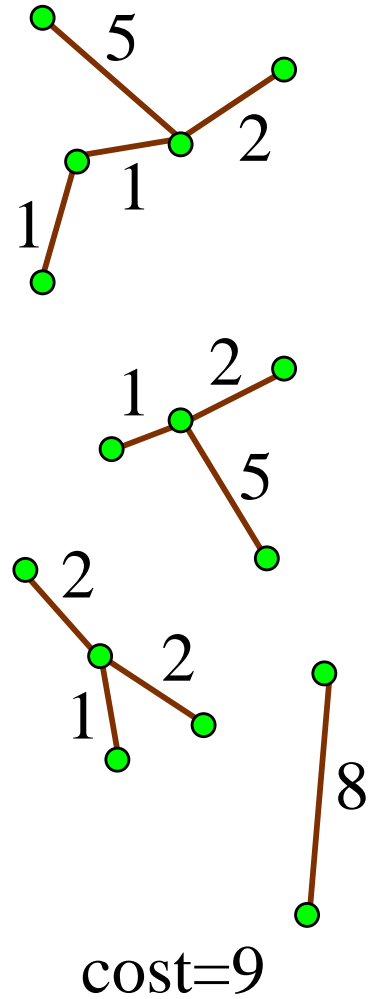
k -Tree Cover Problem

Input: (i) integer k and (ii) $G = (V, E)$ - an undirected graph with positive integral edge weights $w : E \rightarrow \mathbb{N}^+$.

k -tree cover: a set \mathcal{T} of trees $\{T_i\}_i$ such that $V = \bigcup_{i=1}^k V(T_i)$.

cost : $\text{cost}(\mathcal{T}) = \max_{T_i \in \mathcal{T}} w(T_i)$.

goal : find a minimum cost k -tree cover.



k -Tree Cover Problem

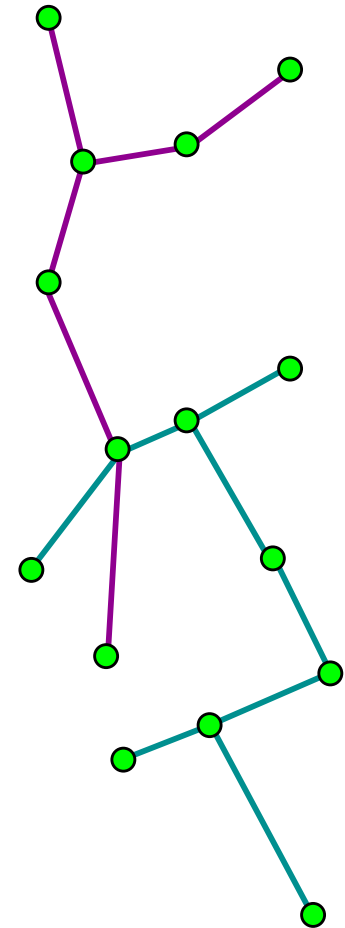
Input: (i) integer k and (ii) $G = (V, E)$ - an undirected graph with positive integral edge weights $w : E \rightarrow \mathbb{N}^+$.

k -tree cover: a set \mathcal{T} of trees $\{T_i\}_i$ such that $V = \bigcup_{i=1}^k V(T_i)$.

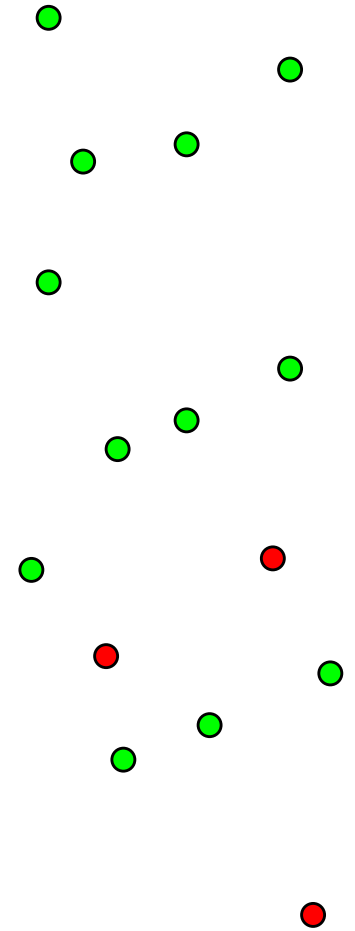
cost : $\text{cost}(\mathcal{T}) = \max_{T_i \in \mathcal{T}} w(T_i)$.

goal : find a minimum cost k -tree cover.

remark : trees may share nodes & edges in a tree cover.



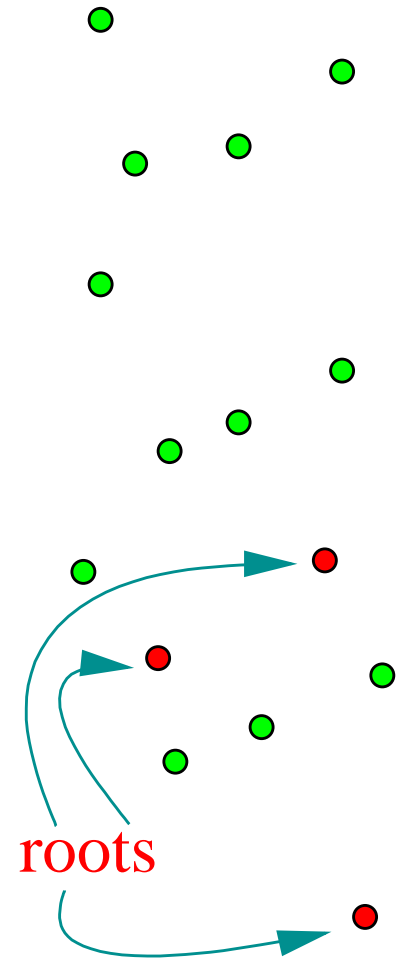
κ -Tree Cover Problem: the rooted case



k -Tree Cover Problem: the rooted case

Roots: Input contains also a set of roots:

$$R = \{r_1, r_2, \dots, r_k\}.$$



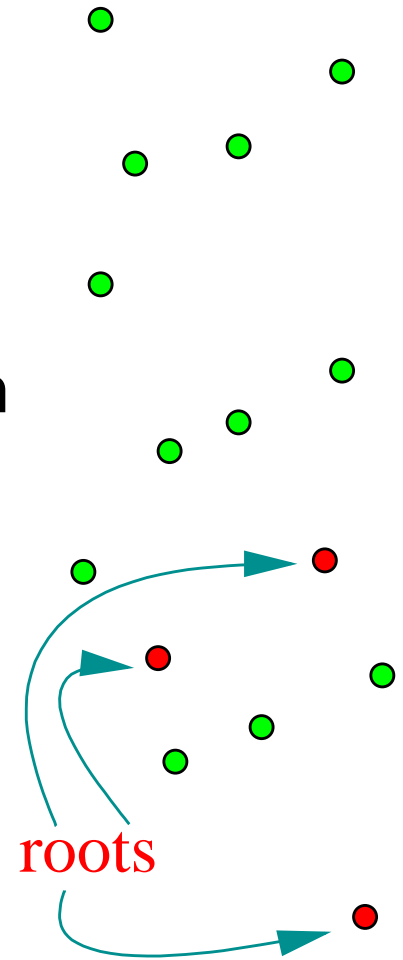
k -Tree Cover Problem: the rooted case

Roots: Input contains also a set of roots:

$$R = \{r_1, r_2, \dots, r_k\}.$$

k -rooted tree cover: a k -tree cover $\{T_i\}_i$ such that

$$\forall i : r_i \in T_i.$$



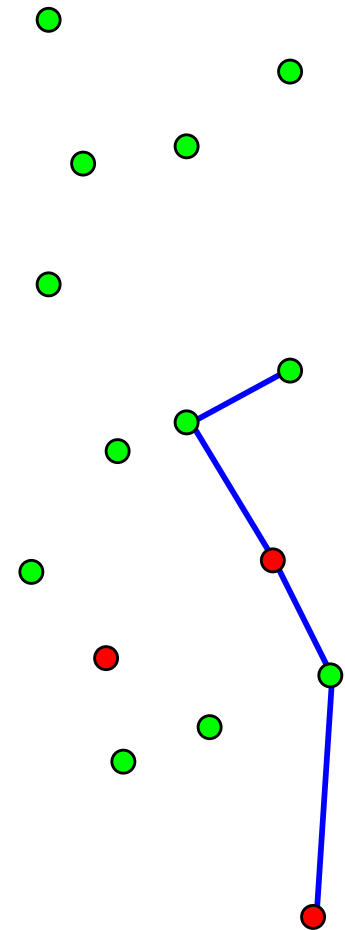
k -Tree Cover Problem: the rooted case

Roots: Input contains also a set of roots:

$$R = \{r_1, r_2, \dots, r_k\}.$$

k -rooted tree cover: a k -tree cover $\{T_i\}_i$ such that

$$\forall i : r_i \in T_i.$$



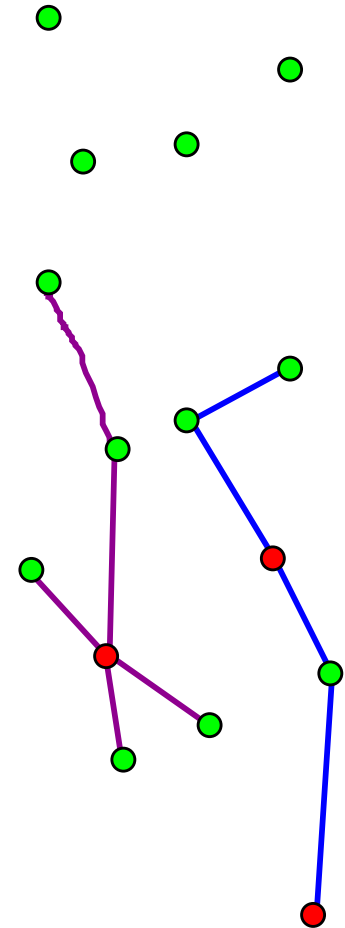
k -Tree Cover Problem: the rooted case

Roots: Input contains also a set of roots:

$$R = \{r_1, r_2, \dots, r_k\}.$$

k -rooted tree cover: a k -tree cover $\{T_i\}_i$ such that

$$\forall i : r_i \in T_i.$$



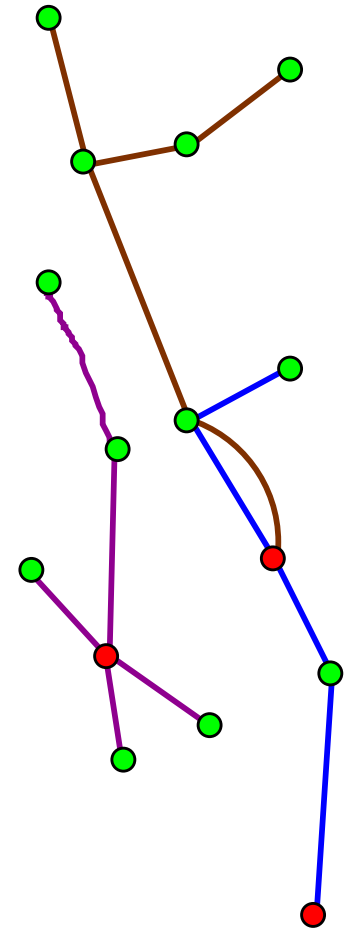
k -Tree Cover Problem: the rooted case

Roots: Input contains also a set of roots:

$$R = \{r_1, r_2, \dots, r_k\}.$$

k -rooted tree cover: a k -tree cover $\{T_i\}_i$ such that

$$\forall i : r_i \in T_i.$$



k -Tree Cover Problem: the rooted case

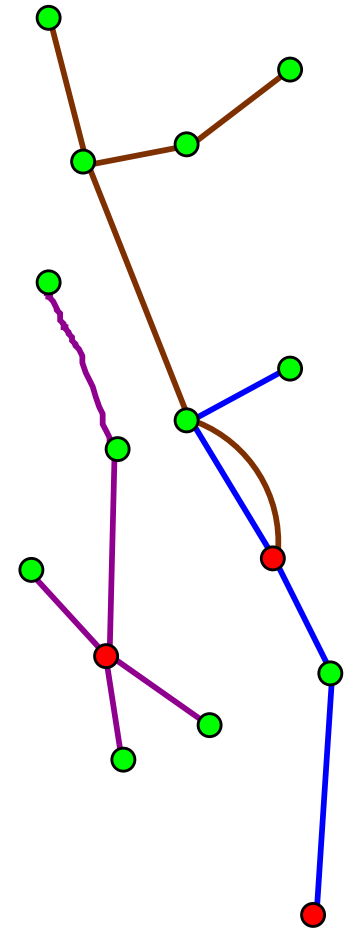
Roots: Input contains also a set of roots:

$$R = \{r_1, r_2, \dots, r_k\}.$$

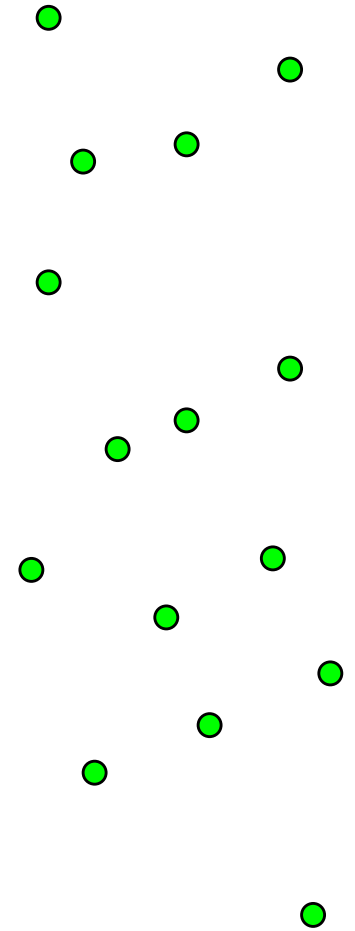
k -rooted tree cover: a k -tree cover $\{T_i\}_i$ such that

$$\forall i : r_i \in T_i.$$

motivation : agents start their tour in different locations.



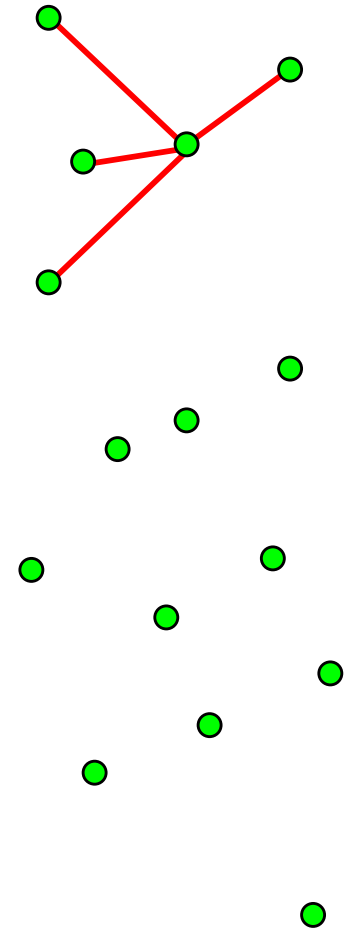
Star Covers



Star Covers

k -Star Cover: a k -tree cover $\{T_i\}_i$ in which

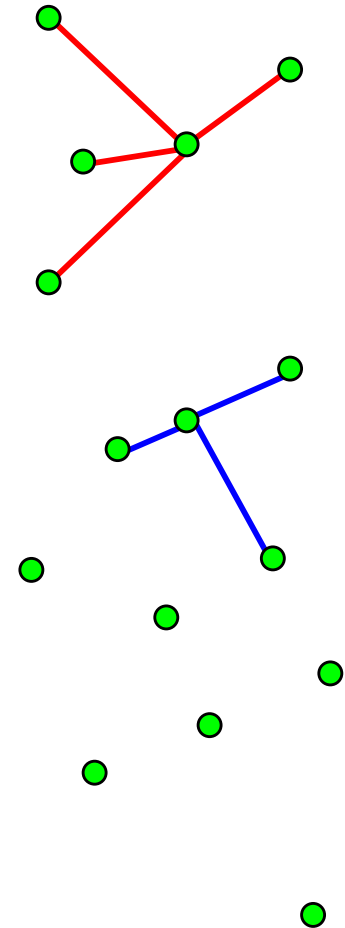
$\forall i : T_i$ is a star



Star Covers

k -Star Cover: a k -tree cover $\{T_i\}_i$ in which

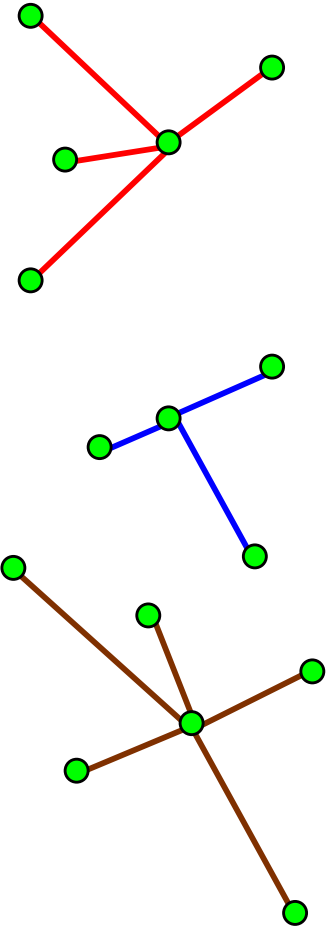
$\forall i : T_i$ is a star



Star Covers

k -Star Cover: a k -tree cover $\{T_i\}_i$ in which

$\forall i : T_i$ is a star

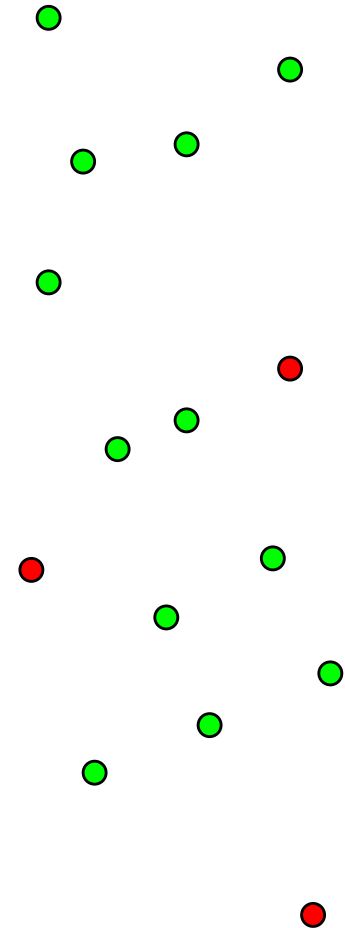


Star Covers

***k*-Star Cover:** a *k*-tree cover $\{T_i\}_i$ in which

$$\forall i : T_i \text{ is a star}$$

rooted/unrooted versions: in rooted version, stars must be rooted at R . Namely, r_i is the root of T_i .

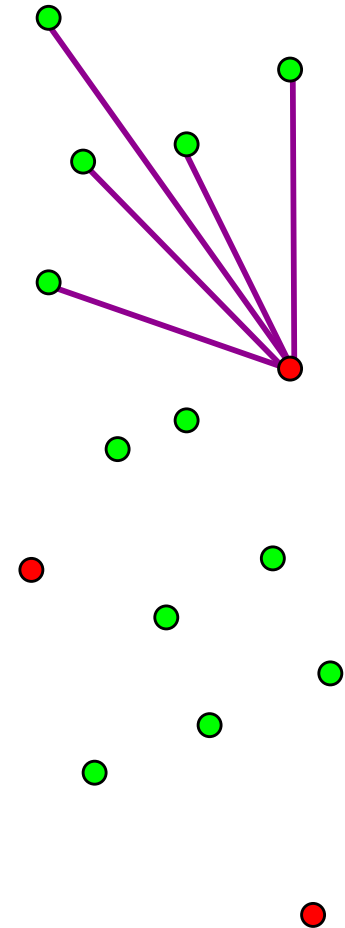


Star Covers

***k*-Star Cover:** a *k*-tree cover $\{T_i\}_i$ in which

$$\forall i : T_i \text{ is a star}$$

rooted/unrooted versions: in rooted version, stars must be rooted at R . Namely, r_i is the root of T_i .

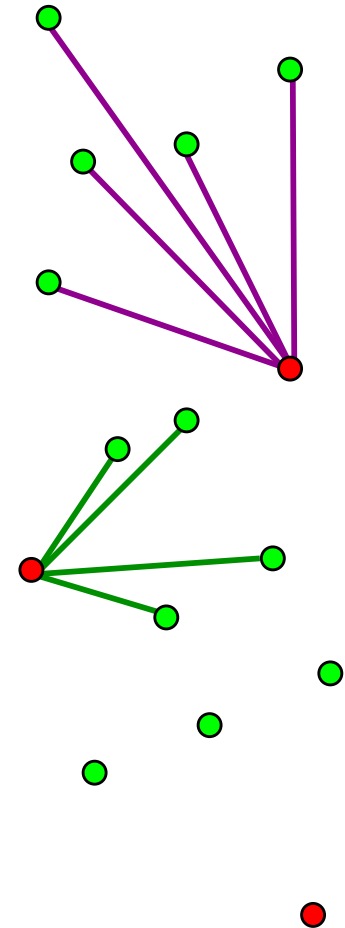


Star Covers

***k*-Star Cover:** a *k*-tree cover $\{T_i\}_i$ in which

$$\forall i : T_i \text{ is a star}$$

rooted/unrooted versions: in rooted version, stars must be rooted at R . Namely, r_i is the root of T_i .

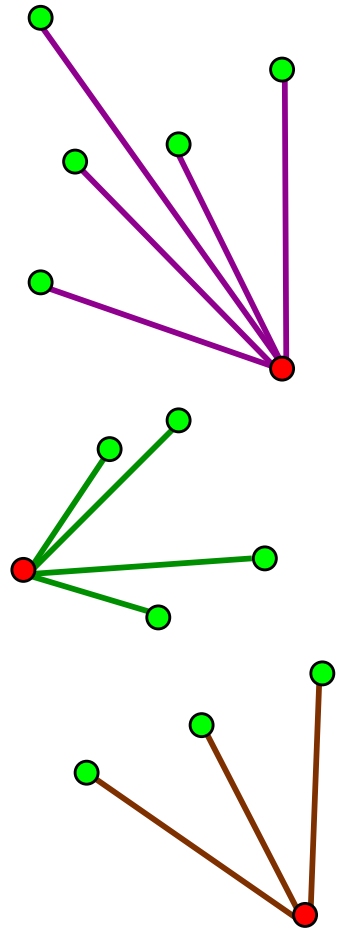


Star Covers

k -Star Cover: a k -tree cover $\{T_i\}_i$ in which

$$\forall i : T_i \text{ is a star}$$

rooted/unrooted versions: in rooted version, stars must be rooted at R . Namely, r_i is the root of T_i .



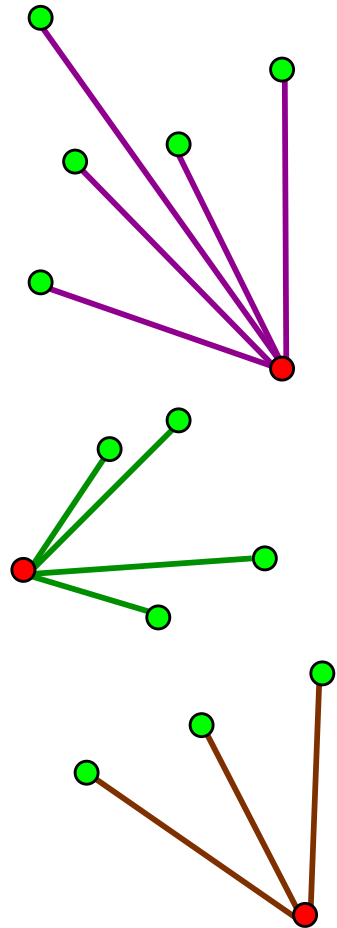
Star Covers

k -Star Cover: a k -tree cover $\{T_i\}_i$ in which

$$\forall i : T_i \text{ is a star}$$

rooted/unrooted versions: in rooted version, stars must be rooted at R . Namely, r_i is the root of T_i .

motivation : agents must return to base after each visit.



Related work

- Arkin, Hassin, & Levin - approx algorithms for covering nodes or subsets of edges by paths, walks, or stars. For example:

Related work

- Arkin, Hassin, & Levin - approx algorithms for covering nodes or subsets of edges by paths, walks, or stars. For example:
 - $O(1)$ -approx for unrooted **path** cover.

Related work

- Arkin, Hassin, & Levin - approx algorithms for covering nodes or subsets of edges by paths, walks, or stars. For example:
 - $O(1)$ -approx for unrooted **path** cover.
 - $O(1)$ -approx for unrooted ***B***-star cover.

Related work

- Arkin, Hassin, & Levin - approx algorithms for covering nodes or subsets of edges by paths, walks, or stars. For example:
 - $O(1)$ -approx for unrooted **path** cover.
 - $O(1)$ -approx for unrooted ***B***-star cover.
 - many other problems...

Related work

- Arkin, Hassin, & Levin - approx algorithms for covering nodes or subsets of edges by paths, walks, or stars. For example:
 - $O(1)$ -approx for unrooted **path** cover.
 - $O(1)$ -approx for unrooted ***B***-star cover.
 - many other problems...
 - Methods do not seem to extend to rooted versions.

Related work

- Arkin, Hassin, & Levin - approx algorithms for covering nodes or subsets of edges by paths, walks, or stars. For example:
 - $O(1)$ -approx for unrooted **path** cover.
 - $O(1)$ -approx for unrooted ***B***-star cover.
 - many other problems...
 - Methods do not seem to extend to rooted versions.
- Chandra Chekuri & Amit Kumar - similar results for star covers.

Related work - cont

- k -Traveling Repairman: Cover with tours, $O(1)$ -approx minimize average latency. [Fakcharoenphol, Harrelson, Rao 2003]

Related work - cont

- **k -Traveling Repairman**: Cover with tours, $O(1)$ -approx minimize average latency. [Fakcharoenphol, Harrelson, Rao 2003]
- **k -Traveling Salesman**: Cover with tours, $O(1)$ -approx minimize total length. [Haimovich, Rinooy Kan, Stougie 1988]

Related work - cont

- **k -Traveling Repairman**: Cover with tours, $O(1)$ -approx minimize average latency. [Fakcharoenphol, Harrelson, Rao 2003]
- **k -Traveling Salesman**: Cover with tours, $O(1)$ -approx minimize total length. [Haimovich, Rinooy Kan, Stougie 1988]
- **Vehicle Routing**: Vast amount of work, e.g. Survey [Toth, Vigo, 2002]

Results

- **Hardness:** All 4 problems are NP-complete. (reduction from Bin-Packing).

Results

- **Hardness:** All 4 problems are NP-complete. (reduction from Bin-Packing).
- **k -tree cover:** 4-approximation algorithm. Strongly polynomial versions are $(4 + \varepsilon)$ -approx.

Results

- **Hardness:** All 4 problems are NP-complete. (reduction from Bin-Packing).
- **k -tree cover:** 4-approximation algorithm. Strongly polynomial versions are $(4 + \varepsilon)$ -approx.
- **k -star cover:**

Results

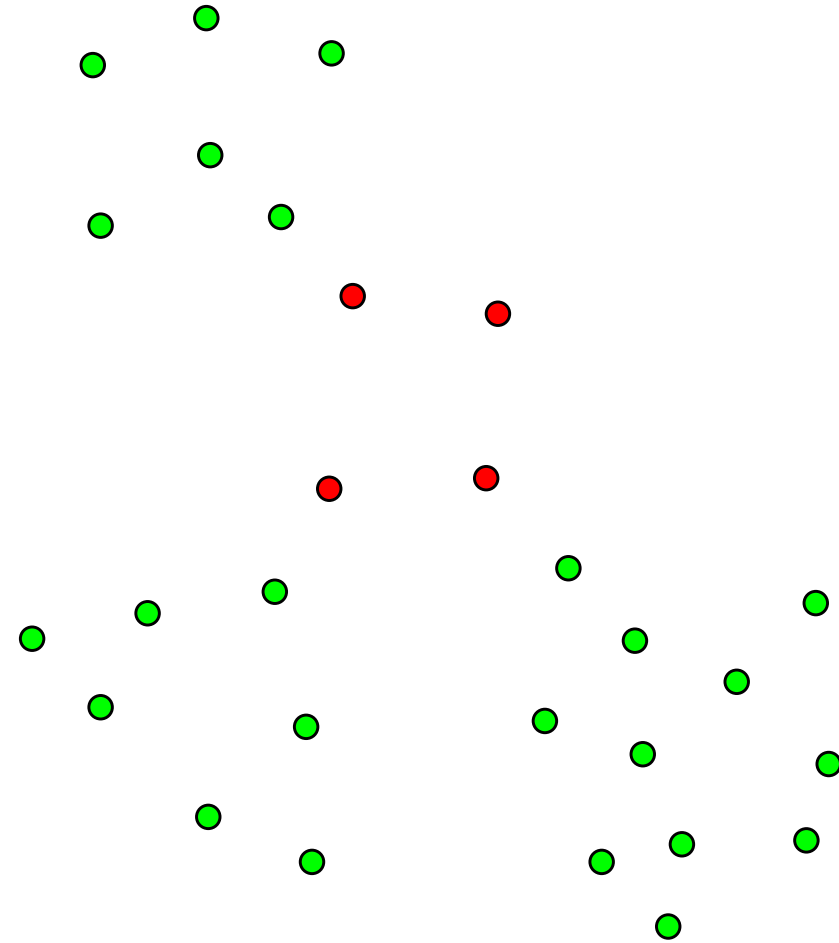
- **Hardness:** All 4 problems are NP-complete. (reduction from Bin-Packing).
- **k -tree cover:** 4-approximation algorithm. Strongly polynomial versions are $(4 + \varepsilon)$ -approx.
- **k -star cover:**
 - Unrooted version: $(4, 4)$ -bi-criteria approximation algorithm (i.e. $4k$ stars of cost $4 \cdot OPT_k$). Extend method of [Shmoys, Tardos, & Aardal, 1997] for capacitated facility location.
Levin - improved to $(3, 3)$ bi-criteria.

Results

- **Hardness:** All 4 problems are NP-complete. (reduction from Bin-Packing).
- **k -tree cover:** 4-approximation algorithm. Strongly polynomial versions are $(4 + \varepsilon)$ -approx.
- **k -star cover:**
 - Unrooted version: $(4, 4)$ -bi-criteria approximation algorithm (i.e. $4k$ stars of cost $4 \cdot OPT_k$). Extend method of [Shmoys, Tardos, & Aardal, 1997] for capacitated facility location.
Levin - improved to $(3, 3)$ bi-criteria.
 - Rooted version: equivalent to min. makespan of k machines and n jobs. 2-approximation of [Shmoys & Tardos, 1993].

approximation algorithm: k -rooted tree cover

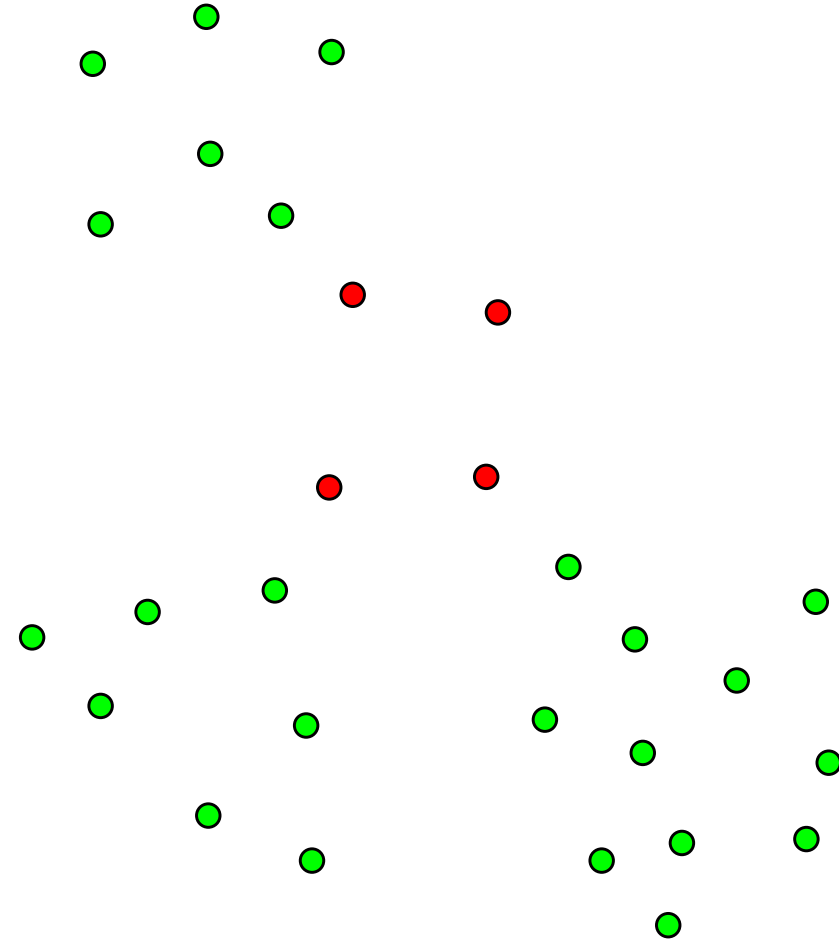
Input: graph, roots, and B - “guess” of opt. cost.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

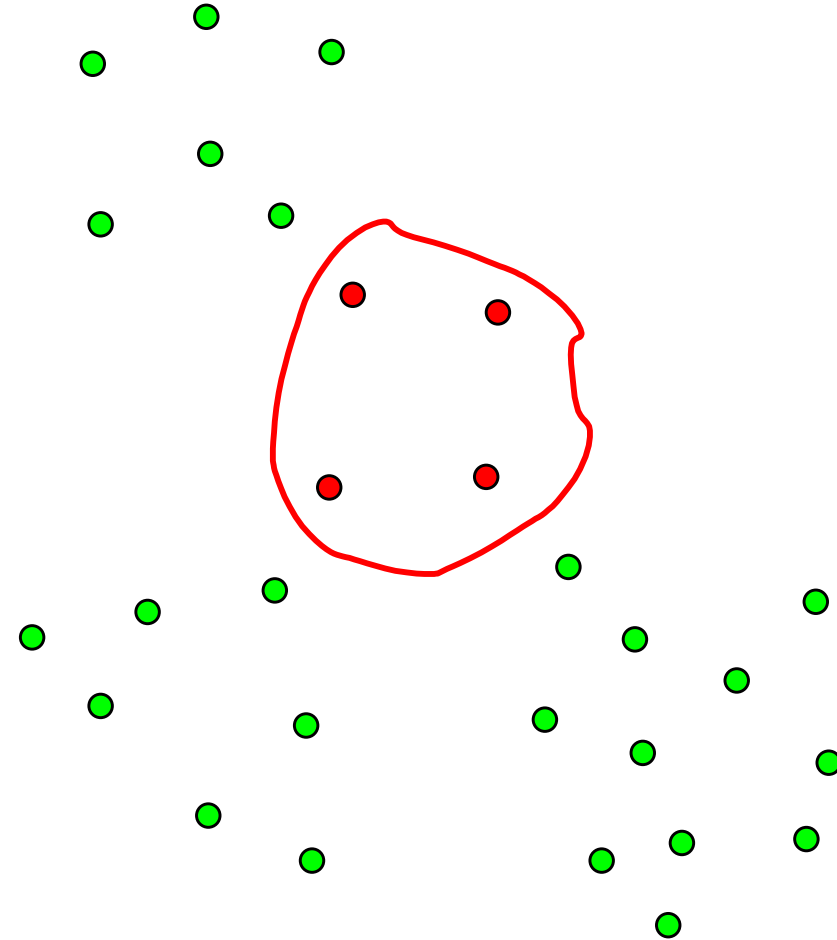
1. contract roots.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

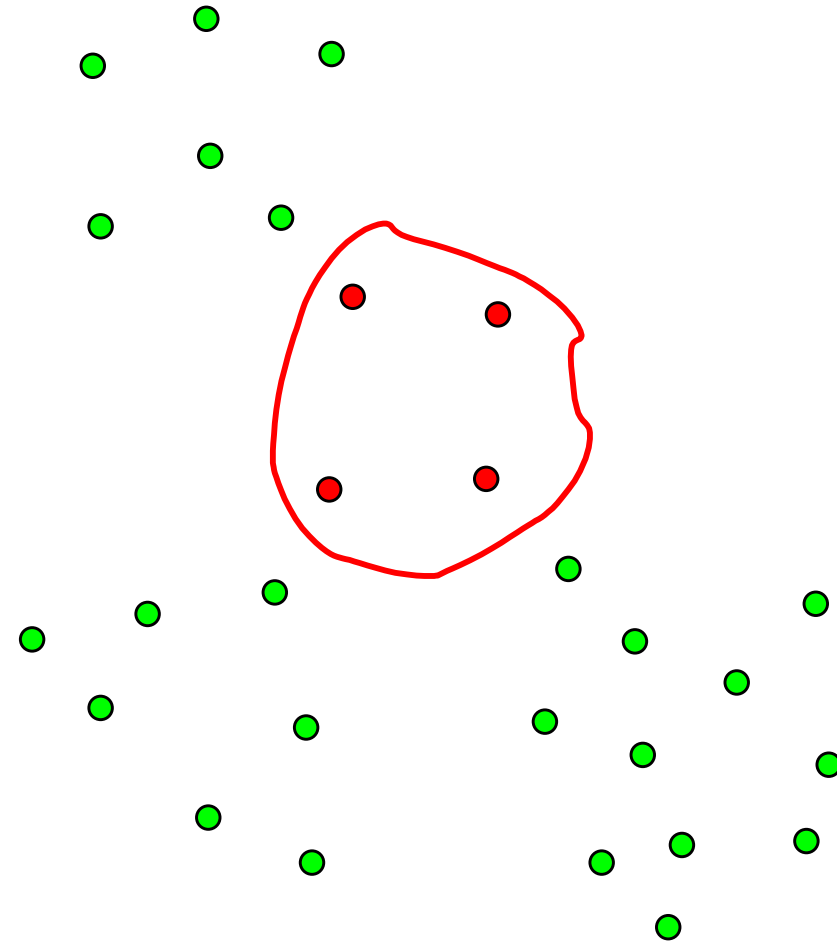
1. contract roots.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

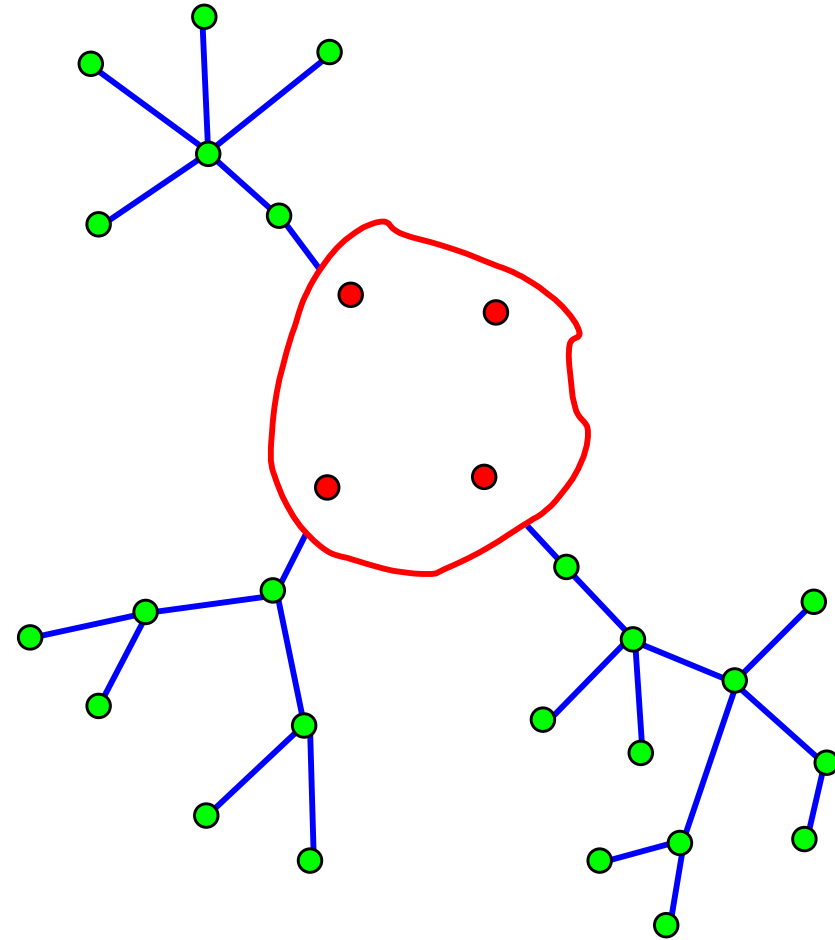
1. contract roots.
2. compute MST.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

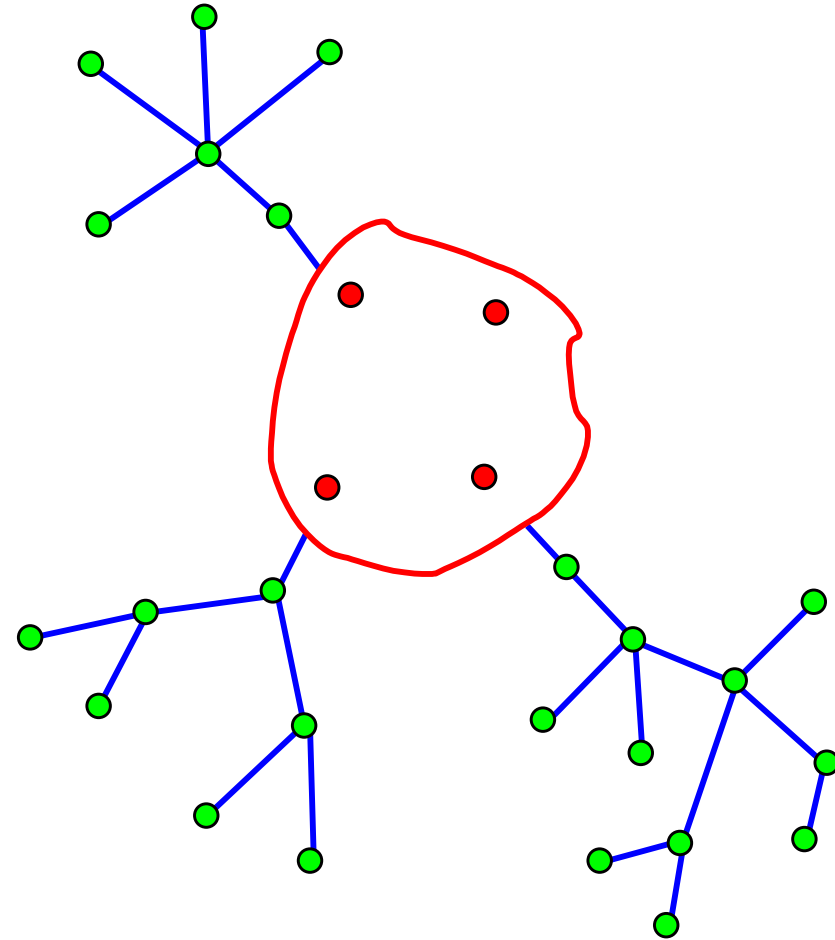
1. contract roots.
2. compute MST.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

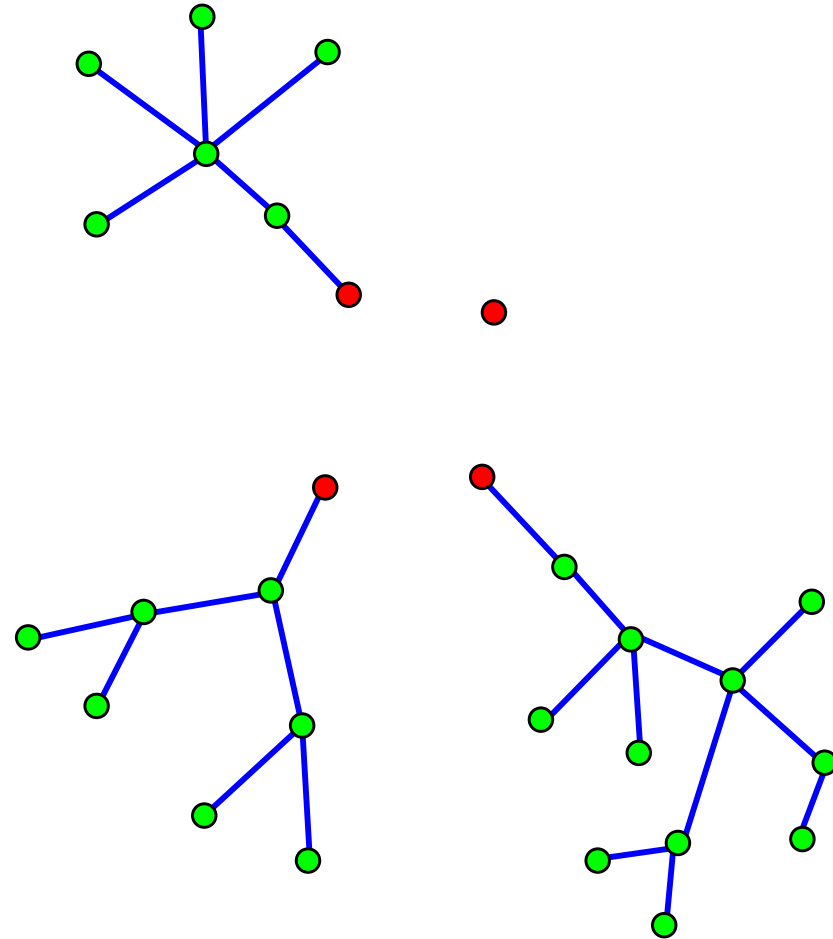
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

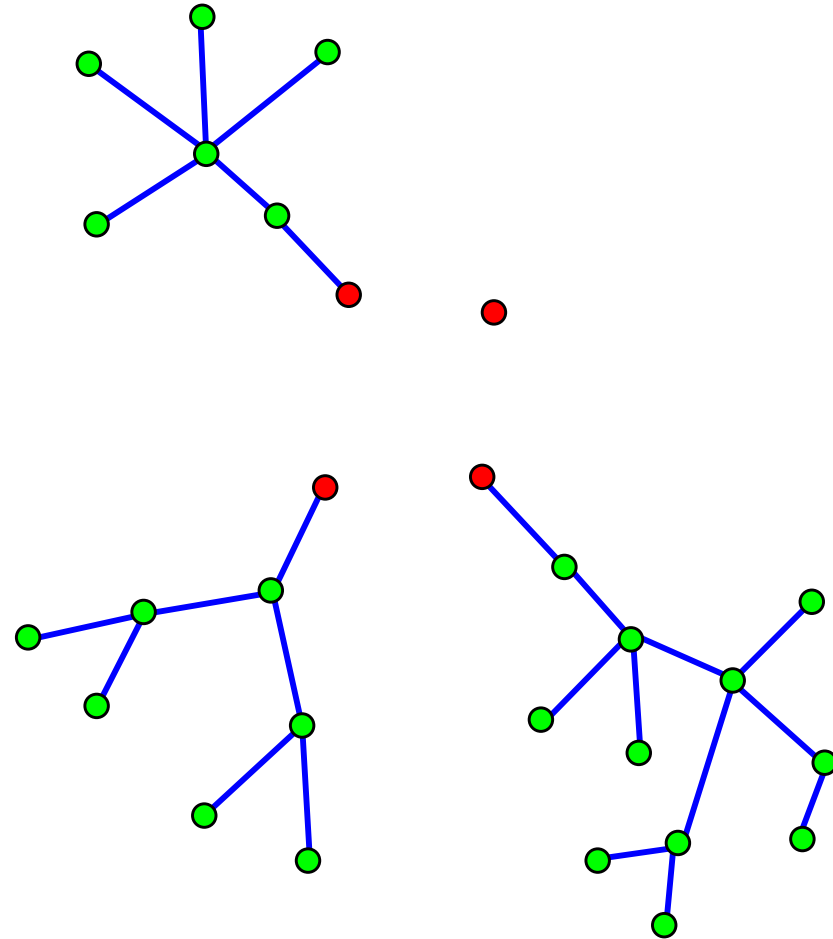
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

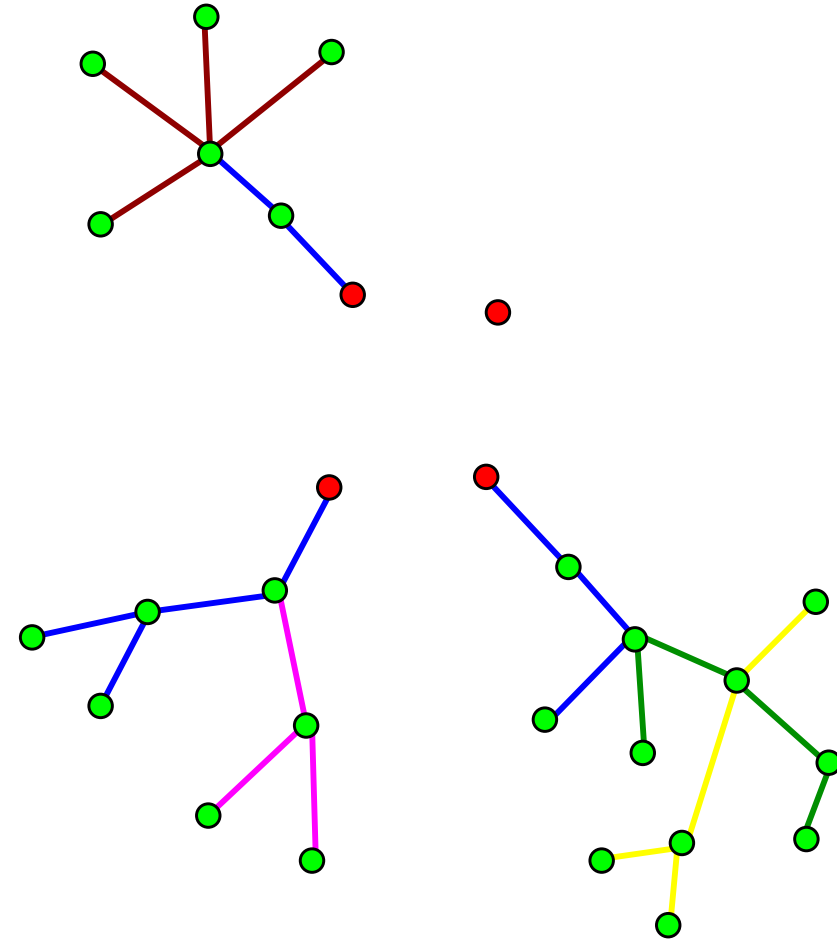
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.
4. edge-decompose trees:
 $w(\text{subtrees}) \in [B, 2B)$,
 $w(\text{leftovers}) < B$.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

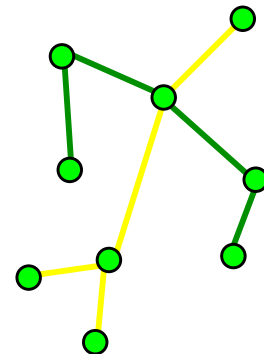
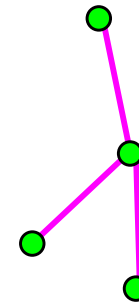
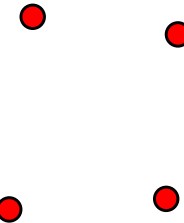
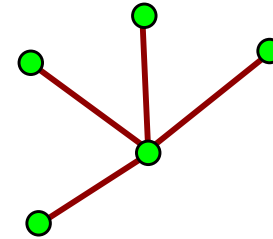
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.
4. edge-decompose trees:
 $w(\text{subtrees}) \in [B, 2B)$,
 $w(\text{leftovers}) < B$.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

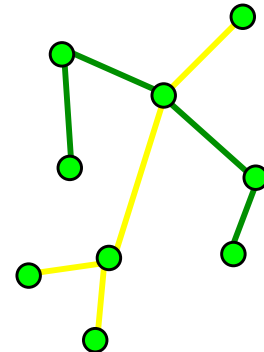
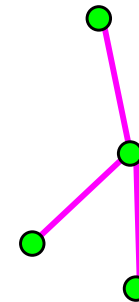
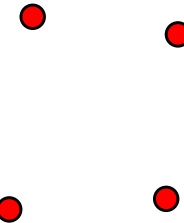
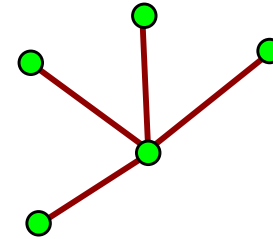
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.
4. edge-decompose trees:
 $w(\text{subtrees}) \in [B, 2B)$,
 $w(\text{leftovers}) < B$.
5. max match subtrees to roots (if $\text{dist} \leq B$).



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

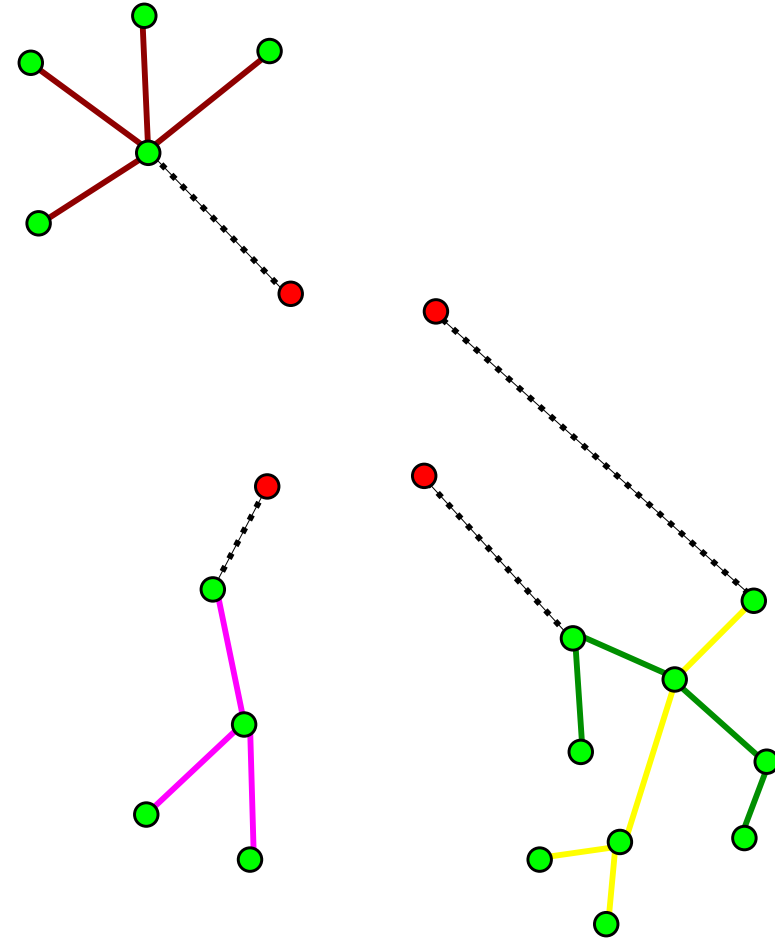
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.
4. edge-decompose trees:
 $w(\text{subtrees}) \in [B, 2B)$,
 $w(\text{leftovers}) < B$.
5. max match subtrees to roots (if $\text{dist} \leq B$).
6. if not all subtrees are matched
 $\Rightarrow B < B^*$.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

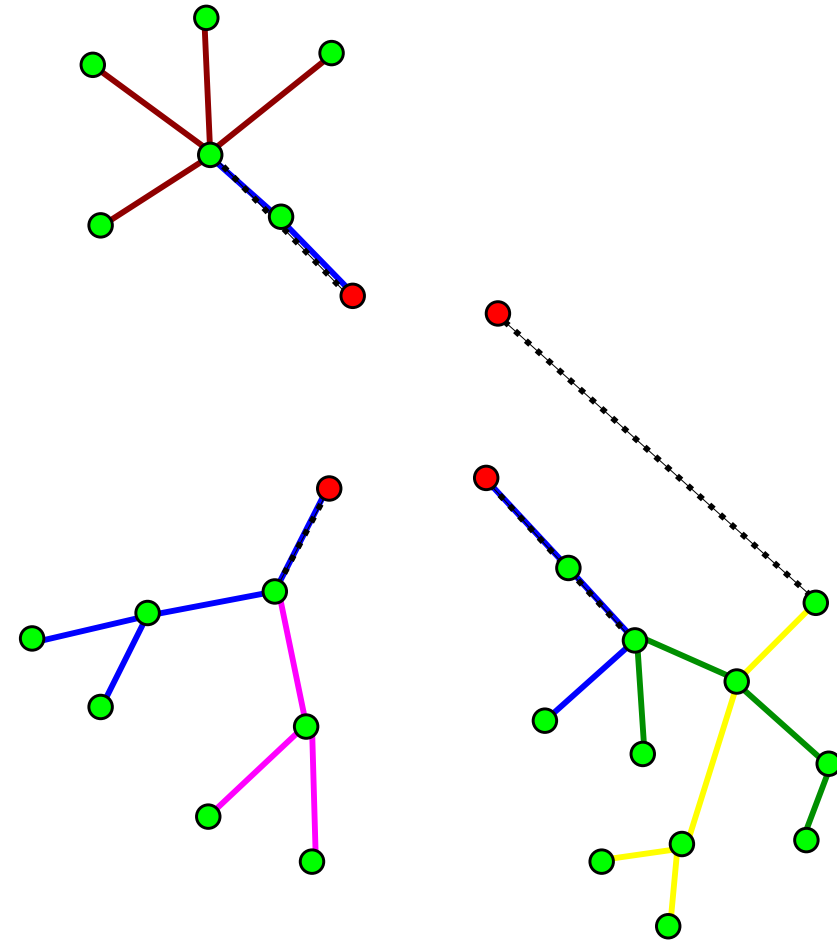
1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.
4. edge-decompose trees:
 $w(\text{subtrees}) \in [B, 2B)$,
 $w(\text{leftovers}) < B$.
5. max match subtrees to roots (if $\text{dist} \leq B$).
6. if not all subtrees are matched
 $\Rightarrow B < B^*$.



approximation algorithm: k -rooted tree cover

Input: graph, roots, and B - “guess” of opt. cost.

1. contract roots.
2. compute MST.
3. un-contract roots: forest of trees rooted at roots.
4. edge-decompose trees:
 $w(\text{subtrees}) \in [B, 2B)$,
 $w(\text{leftovers}) < B$.
5. max match subtrees to roots (if $\text{dist} \leq B$).
6. if not all subtrees are matched
 $\Rightarrow B < B^*$.
7. else return $\forall r_i$: leftover + matched subtree.



4-approx algorithm : k -rooted tree cover

- Claim: success $\Rightarrow \text{cost}(\text{cover}) \leq 4 \cdot B$.

4-approx algorithm : k -rooted tree cover

- Claim: success $\Rightarrow \text{cost}(\text{cover}) \leq 4 \cdot B$.
- Claim: fail $\Rightarrow B < B^*$.

4-approx algorithm : k -rooted tree cover

- Claim: success $\Rightarrow \text{cost}(\text{cover}) \leq 4 \cdot B$.
- Claim: fail $\Rightarrow B < B^*$.
- Binary search on value of $B \Rightarrow$ (weakly) polynomial 4-approx algorithm.

4-approx algorithm : k -rooted tree cover

- Claim: success $\Rightarrow \text{cost}(\text{cover}) \leq 4 \cdot B$.
- Claim: fail $\Rightarrow B < B^*$.
- Binary search on value of $B \Rightarrow$ (weakly) polynomial 4-approx algorithm.
- Scaling \Rightarrow strongly polynomial $(4 + \varepsilon)$ -approx algorithm.

Claim: success \Rightarrow $cost(cover) \leq 4 \cdot B$

Claim: success \Rightarrow $cost(cover) \leq 4 \cdot B$

Each tree in tree cover may consist of:

- a rooted leftover subtree $\Rightarrow cost(leftover) < B$.

Claim: success $\Rightarrow cost(cover) \leq 4 \cdot B$

Each tree in tree cover may consist of:

- a rooted leftover subtree $\Rightarrow cost(leftover) < B$.
- a matched subtree $\Rightarrow cost(subtree) < 2 \cdot B$.

Claim: success $\Rightarrow cost(cover) \leq 4 \cdot B$

Each tree in tree cover may consist of:

- a rooted leftover subtree $\Rightarrow cost(leftover) < B$.
- a matched subtree $\Rightarrow cost(subtree) < 2 \cdot B$.
- matching edge $\Rightarrow cost(edge) \leq B$.

Claim: success $\Rightarrow cost(cover) \leq 4 \cdot B$

Each tree in tree cover may consist of:

- a rooted leftover subtree $\Rightarrow cost(leftover) < B$.
- a matched subtree $\Rightarrow cost(subtree) < 2 \cdot B$.
- matching edge $\Rightarrow cost(edge) \leq B$.

\Rightarrow weight of every tree in solution is $< 4 \cdot B$.

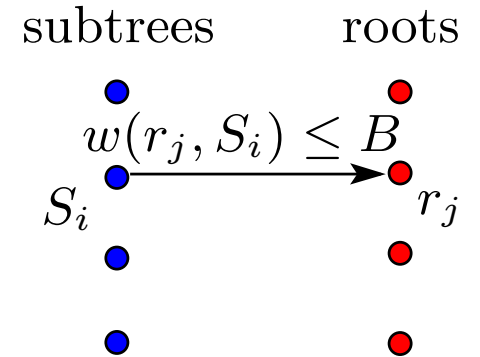
QED

Claim: fail $\Rightarrow B < B^*$

Claim: fail $\Rightarrow B < B^*$

Assume for sake of contradiction:

$B \geq B^*$ and matching failed.



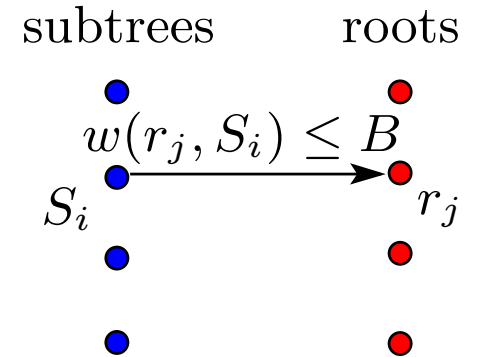
Claim: fail $\Rightarrow B < B^*$

Assume for sake of contradiction:

$B \geq B^*$ and matching failed.

Hall's Theorem: if matching failed, then

$\exists \mathcal{S} \subseteq \text{ subtrees} : |\mathcal{S}| > |N(\mathcal{S})|.$



Claim: fail $\Rightarrow B < B^*$

Assume for sake of contradiction:

$B \geq B^*$ and matching failed.

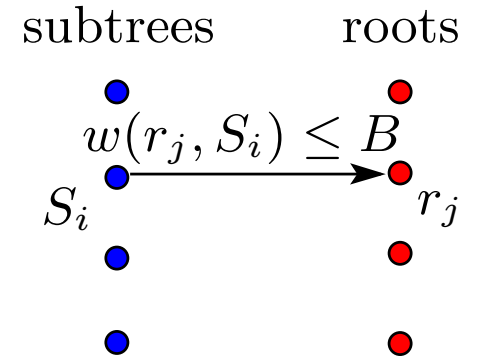
Hall's Theorem: if matching failed, then

$$\exists \mathcal{S} \subseteq \text{subtrees} : |\mathcal{S}| > |N(\mathcal{S})|.$$

Fix OPT:

$$\mathcal{T}^* \triangleq \{T_1^*, \dots, T_k^*\} \text{ where } r_j \in T_j^*.$$

$$\mathcal{T}^*(\mathcal{S}) \triangleq \{T_j^* \mid \exists S_i \in \mathcal{S} : T_j^* \cap S_i \neq \emptyset\}.$$



Claim: fail $\Rightarrow B < B^*$

Assume for sake of contradiction:

$B \geq B^*$ and matching failed.

Hall's Theorem: if matching failed, then

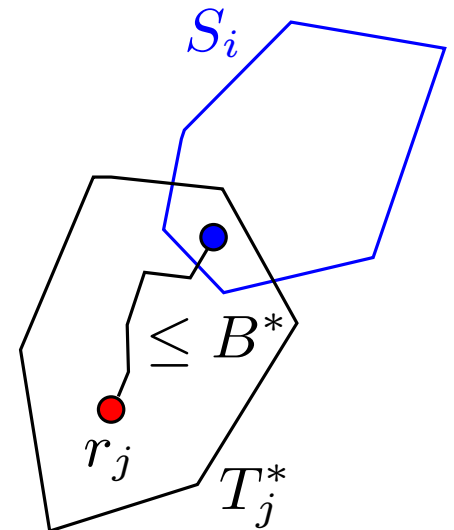
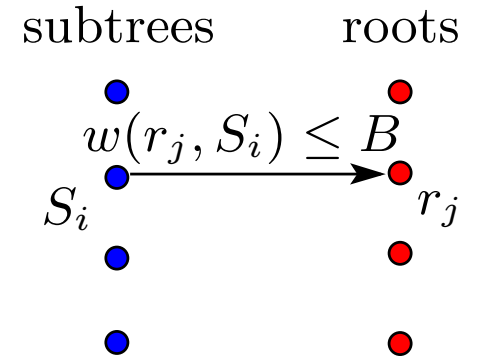
$$\exists \mathcal{S} \subseteq \text{ subtrees} : |\mathcal{S}| > |N(\mathcal{S})|.$$

Fix OPT:

$\mathcal{T}^* \triangleq \{T_1^*, \dots, T_k^*\}$ where $r_j \in T_j^*$.

$\mathcal{T}^*(\mathcal{S}) \triangleq \{T_j^* \mid \exists S_i \in \mathcal{S} : T_j^* \cap S_i \neq \emptyset\}$.

Note: $T_j^* \cap S_i \neq \emptyset \Rightarrow w(r_j, S_i) \leq B^* \leq B$.



Claim: fail $\Rightarrow B < B^*$

Assume for sake of contradiction:

$B \geq B^*$ and matching failed.

Hall's Theorem: if matching failed, then

$$\exists \mathcal{S} \subseteq \text{subtrees} : |\mathcal{S}| > |N(\mathcal{S})|.$$

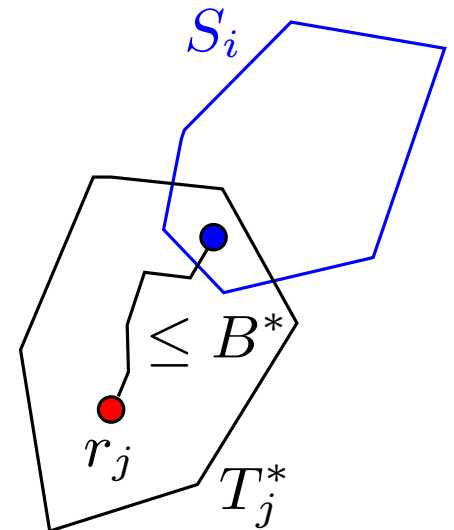
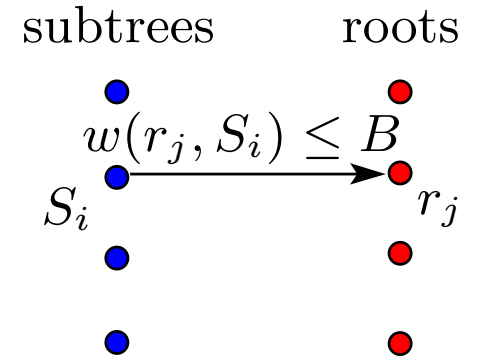
Fix OPT:

$\mathcal{T}^* \triangleq \{T_1^*, \dots, T_k^*\}$ where $r_j \in T_j^*$.

$\mathcal{T}^*(\mathcal{S}) \triangleq \{T_j^* \mid \exists S_i \in \mathcal{S} : T_j^* \cap S_i \neq \emptyset\}$.

Note: $T_j^* \cap S_i \neq \emptyset \Rightarrow w(r_j, S_i) \leq B^* \leq B$.

$$\Rightarrow |\mathcal{T}^*(\mathcal{S})| \leq |N(\mathcal{S})|.$$



Claim: fail $\Rightarrow B < B^*$ - (cont.)

recall:

$$B \geq B^* \quad \text{and} \quad |\mathcal{T}^*(\mathcal{S})| \leq |N(\mathcal{S})| < |\mathcal{S}|.$$

Claim: fail $\Rightarrow B < B^*$ - (cont.)

recall:

$$B \geq B^* \quad \text{and} \quad |\mathcal{T}^*(\mathcal{S})| \leq |N(\mathcal{S})| < |\mathcal{S}|.$$

$$\forall j : w(T_j^*) \leq B^* \Rightarrow w(\mathcal{T}^*(\mathcal{S})) \leq B^* \cdot |\mathcal{T}^*(\mathcal{S})|$$

$$\begin{aligned} \forall i : w(S_i) \in [B, 2B) &\Rightarrow w(\mathcal{S}) \geq B \cdot |\mathcal{S}|. \\ &\Rightarrow w(\mathcal{S}) > w(\mathcal{T}^*(\mathcal{S})). \end{aligned}$$

Claim: fail $\Rightarrow B < B^*$ - (cont.)

recall:

$$B \geq B^* \quad \text{and} \quad |\mathcal{T}^*(\mathcal{S})| \leq |N(\mathcal{S})| < |\mathcal{S}|.$$

$$\forall j : w(T_j^*) \leq B^* \Rightarrow w(\mathcal{T}^*(\mathcal{S})) \leq B^* \cdot |\mathcal{T}^*(\mathcal{S})|$$

$$\begin{aligned} \forall i : w(S_i) \in [B, 2B) &\Rightarrow w(\mathcal{S}) \geq B \cdot |\mathcal{S}|. \\ &\Rightarrow w(\mathcal{S}) > w(\mathcal{T}^*(\mathcal{S})). \end{aligned}$$

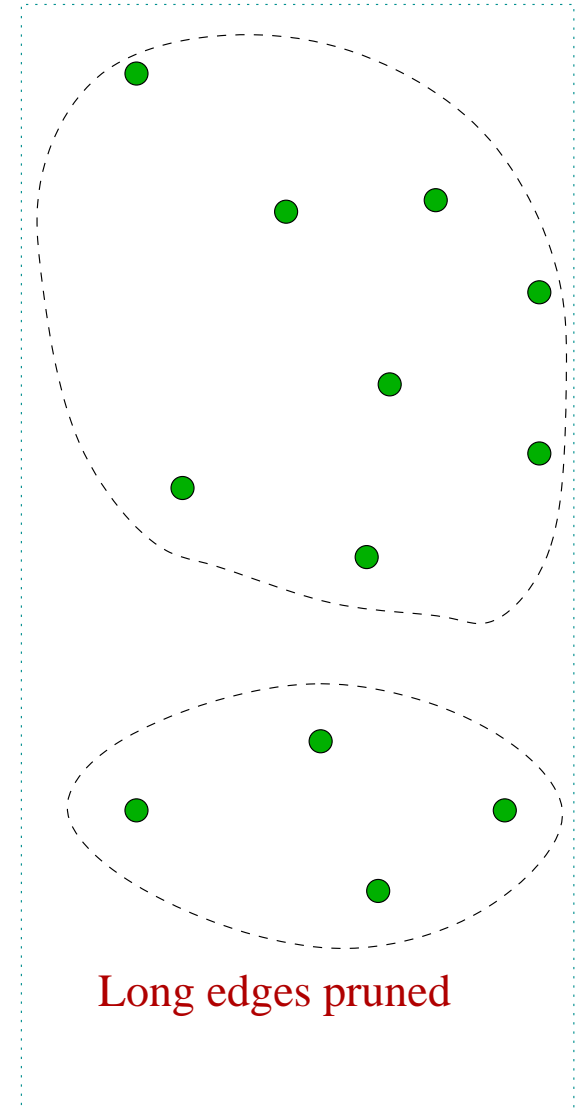
But

$$T' \triangleq MST - \mathcal{S} + \mathcal{T}^*(\mathcal{S})$$

is a spanning tree and $w(T') < w(MST)$, contradiction. QED

Algorithm for Unrooted k -tree cover

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

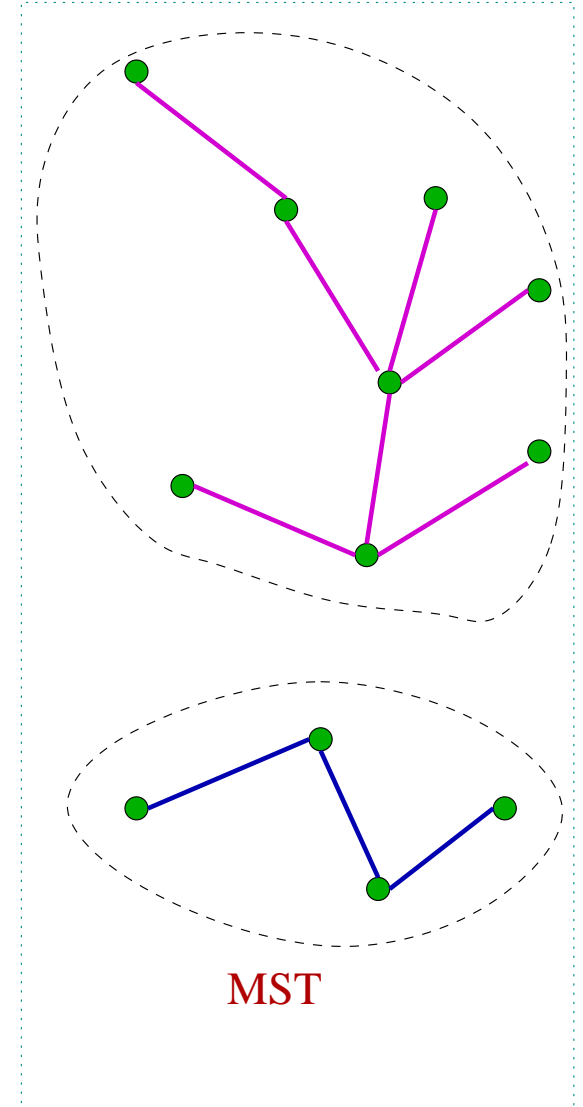


Algorithm for Unrooted k -tree cover

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

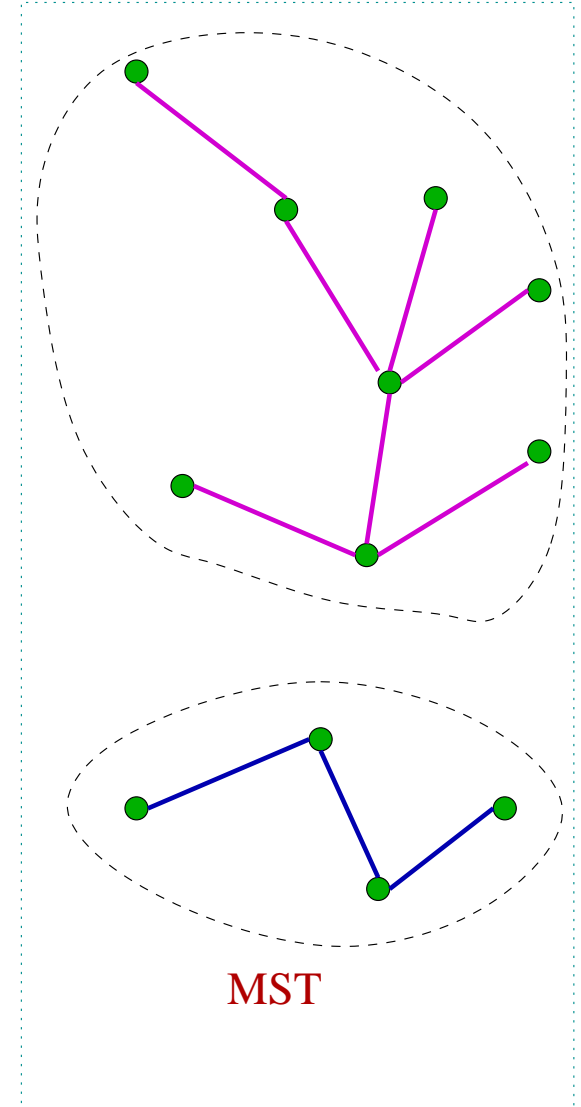
2. $MST_i = \text{MST of } G_i$.

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



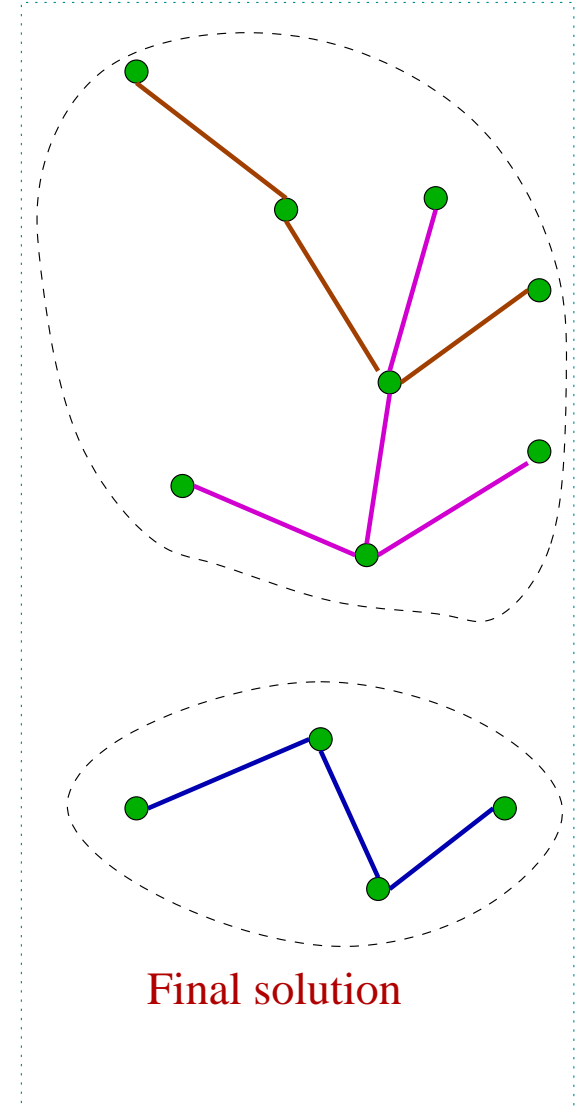
Algorithm for Unrooted k -tree cover

1. Prune edges $w_e > B$.
Let $\{G_i\}_i$ be components.
2. $MST_i = \text{MST of } G_i$.
 $k_i = \lfloor \frac{w(MST_i)}{2B} \rfloor$.
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. $MST_i = \text{MST of } G_i$.
 $k_i = \lfloor \frac{w(MST_i)}{2B} \rfloor$.
3. If $\sum_i (k_i + 1) > k$, return “fail”.
4. Decompose each MST_i into at most $k_i + 1$ trees $S_i^1 + \dots + S_i^{k_i} + L_i$ 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 $B^* \leq B$, then $k_i + 1 \leq k_i^*$ for all i .

Proof: $B^* \leq B \Rightarrow k_i + 1 \leq k_i^*$

Proof: $B^* \leq B \Rightarrow k_i + 1 \leq k_i^*$

Let optimal solution cover G_i with $\{T_1^*, \dots, T_{k_i^*}^*\}$.

Proof: $B^* \leq B \Rightarrow k_i + 1 \leq k_i^*$

Let optimal solution cover G_i with $\{T_1^*, \dots, T_{k_i^*}^*\}$.

Augment it to span G_i by adding $k_i^* - 1$ edges, so:

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

Proof: $B^* \leq B \Rightarrow k_i + 1 \leq k_i^*$

Let optimal solution cover G_i with $\{T_1^*, \dots, T_{k_i^*}^*\}$.

Augment it to span G_i by adding $k_i^* - 1$ edges, so:

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

Since $w(T_i^*) \leq B^* \leq B$,

$$k_i^* \cdot B + (k_i^* - 1)B \geq w(MST_i).$$

Proof: $B^* \leq B \Rightarrow k_i + 1 \leq k_i^*$

Let optimal solution cover G_i with $\{T_1^*, \dots, T_{k_i^*}^*\}$.

Augment it to span G_i by adding $k_i^* - 1$ edges, so:

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

Since $w(T_i^*) \leq B^* \leq B$,

$$k_i^* \cdot B + (k_i^* - 1)B \geq w(MST_i).$$

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

□

Further work

- Improve approximation ratio for (rooted) tree-cover.

Further work

- Improve approximation ratio for (rooted) tree-cover.
- Open: **rooted** path/walk cover.