Revisiting Garg's 2-Approximation Algorithm for the k-MST Problem in Graphs

Emmett Breen, Renee Mirka, **Zichen Wang**, David P. Williamson

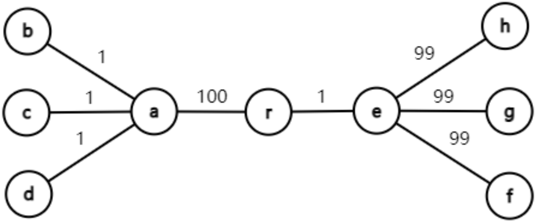
Cornell University

January 2023

Table of Contents

The k-MST Problem

- Given a graph $G = (V, E)$ with edges costs $c_e \geq 0$, want to find a spanning tree of k vertices such that its edge cost is minimized.
- k-MST is known to be NP-hard; Greedy algorithms for the MST problem, such as the Prim's algorithm, are too myopic for k-MST.



Previous Work

- (Ravi et al., 1996) First introduced the k-MST problem and gave a $3\sqrt{k}$ approximation algorithm.
- (Awerbuch et al., 1995) $\log^2 k$ approximation algorithm.
- (Blum et al., 1996) constant approximation algorithm.
- (Garg, 1996) 3 approximation algorithm.
- (Arya and Ramesh, 1998) 2.5 approximation algorithm.
- (Arora and Karakostas, 2006) $(2+\epsilon)$ approximation algorithm.
- (Garg, 2005) 2 approximation using primal-dual techniques.
- (Paul et al., 2020) Detailed analysis of primal-dual techniques on the budget TSP problem.

Garg's Algorithm

- Assign every vertex an initial potential.
- Spend potentials to grow components and cover edge costs.
- If a component runs out of potential, it becomes neutral;
  if the cost of an edge is entirely covered, then merge its two adjacent components.
- Prune the resulting tree by a coloring scheme.
- Find the smallest initial potential that returns at least k vertices.
- Pick exactly k vertices.

Our Work

- Write down explicitly the primal and dual linear programs.
- Give explicit expression of the potential function.
- Supplement intuitive understanding of the algorithm.
- Introduce Paul et al.'s analysis techniques to the k-MST problem.
- Introduce the novel concept of kernels to replace Garg's vertex coloring.

Table of Contents

Primal

$z_S \in \{0,1\}$ denote whether S constitutes the vertices of the spanning tree.
$x_e \in \{0,1\}$ denote whether edge e is included in the spanning tree.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \sum_{e: e \in \delta(S)} x_e \geq \sum_{T: S \subset T} z_T \text{ for } \forall S \subset V \\
& \sum_{S \subseteq V} |S| z_S \geq k \\
& \sum_{S \subseteq V} z_S \leq 1 \\
& z_S,\ x_e \geq 0.
\end{aligned}
$$

Dual

$$
\begin{aligned}
\text{minimize} \quad & \lambda_2 - \lambda_1 k \\
\text{subject to} \quad & \sum_{S: e \in \delta(S)} y_S \leq c_e \text{ for } \forall e \in E \\
& \sum_{T \subset S} y_T + \lambda_2 \geq \lambda_1 |S| \text{ for } \forall S \subset V \\
& \lambda_1, \lambda_2, y_S \geq 0.
\end{aligned}
$$

- Instead of minimizing $\sum c_e x_e$ s.t. $\sum |S| z_s \geq k$, we consider minimizing $\sum c_e x_e - \lambda \sum |S| z_s$.
- λ_1 represents the weight ratio between the two objectives.
- λ_2 is set to the optimal value given λ_1.

Potential & Neutral Set

Definition (Potential)

For any subset $S \subseteq V$, define the **potential** of S to be

$$\pi(S) = \lambda_1|S| - \sum_{T:T\subset S} y_T$$

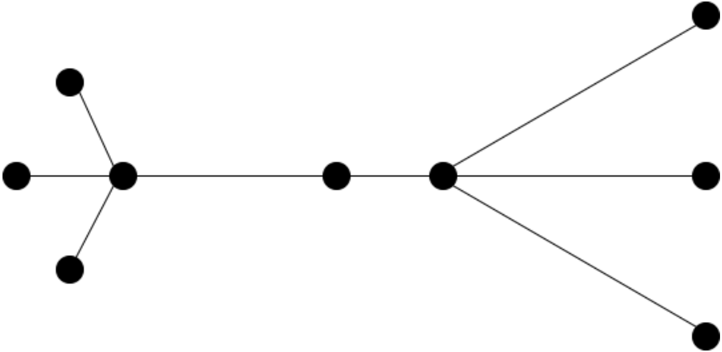
Definition (Neutral Set)

A subset $S \subseteq V$ is **neutral** if $\sum_{T:T\subseteq S} y_T = \lambda_1|S|$.

Primal-Dual Algorithm

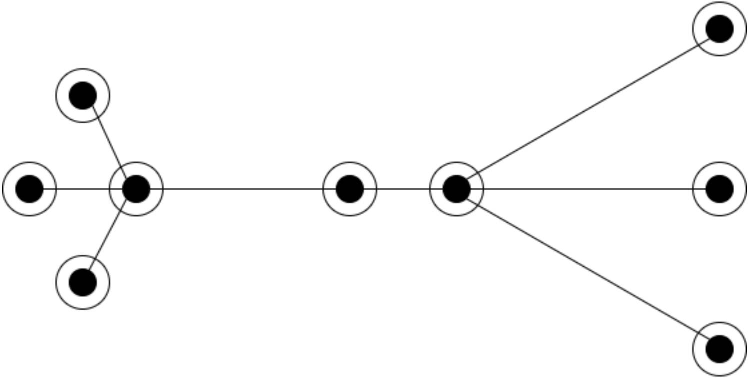
set all vertices to be active components.
while not all sets neutral **do**
  raise all active y_S uniformly until either
  if a set runs out of potential **then**
    mark it neutral.
  else if an edge becomes tight **then**
    combine the two adjacent components of the edge into a new active component.
  end if
end while

Example



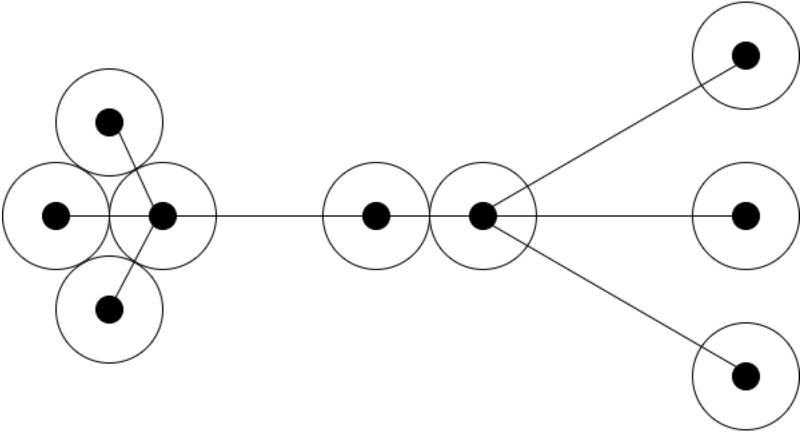
Edge costs represented by edge length.

Example (Cont.)



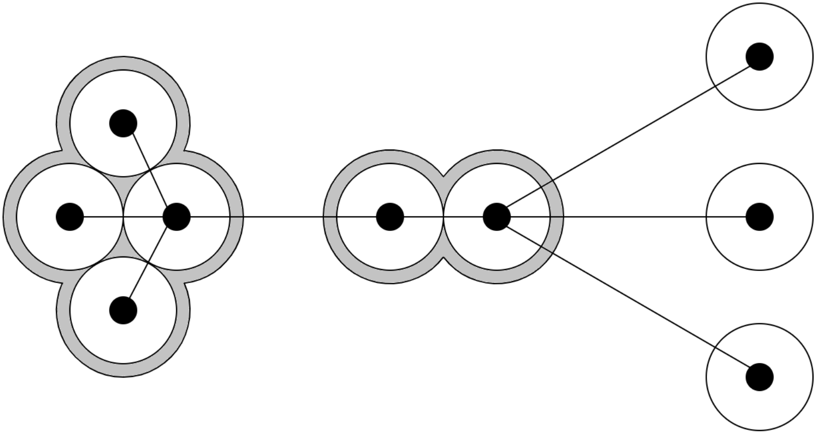
At start, all vertices are themselves active.

Example (Cont.)



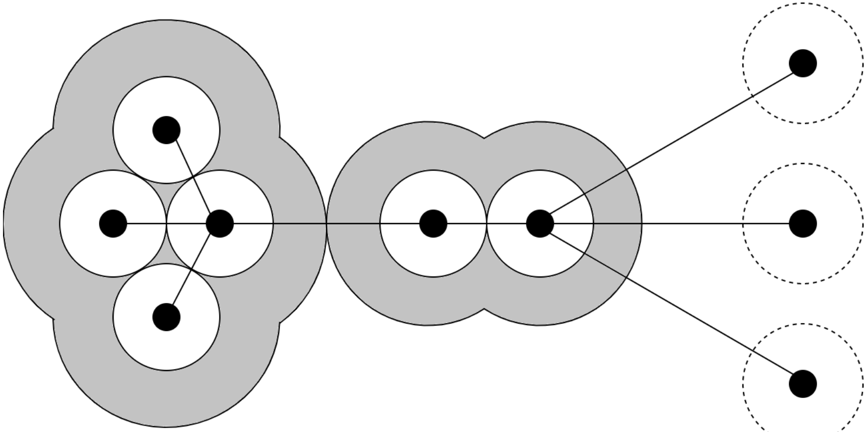
First edge events.

Example (Cont.)



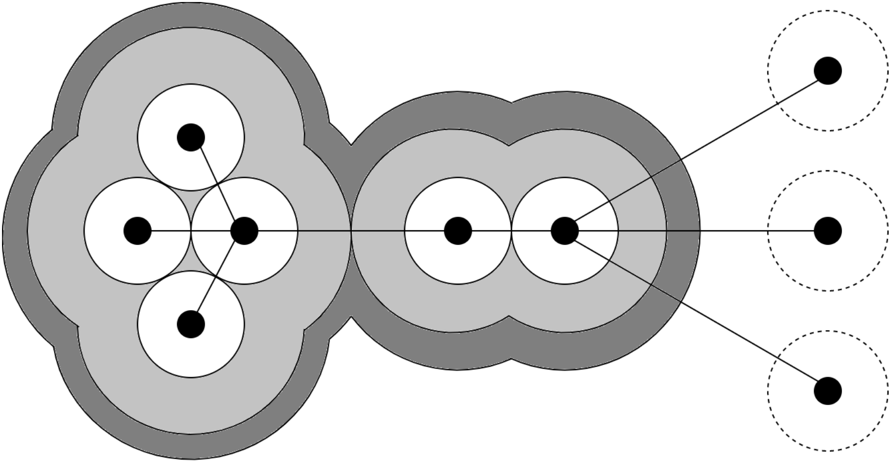
The merged component, denoted in grey, continues to grow.

Example (Cont.)



Dashed circles denote neutral sets.

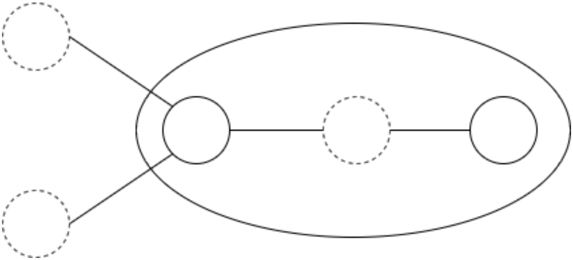
Example (Cont.)



The final result.

Insights

- Connect to neutral sets in the hope that they connect to other active sets.
- Prune fruitless neutral sets so we don't waste our potential.
- Equivalently, keep track of the useful parts that fuel the component's growth. This leads us to the idea of the kernel of tree T, denoted as $K(T)$.



Algorithm Overview

while binary search on λ_1 **do**
  T = primal_dual(λ_1)
  if $K(T) \geq k$ **then**
    lower λ_1
  else
    higher λ_1
  end if
end while
return pick k vertices from $K(T)$

Table of Contents

Kernels

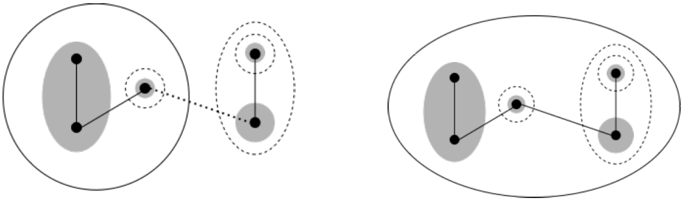
Definition (Kernel)

For any subset $S \subseteq V$, define the **kernel** of S, denoted $K(S)$, to be the smallest connected subset of S such that

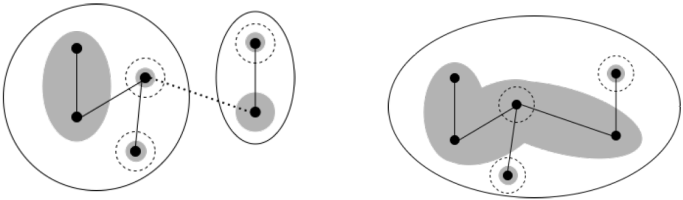
- if a vertex $v \in S$ has always been part of an active component, then $v \in K(S)$.
- if a subset I is once-active, then $I \subseteq K(S)$ or $I \cap K(S) = \emptyset$.

The kernel of a once-active set S is the kernel at the moment S becomes neutral.

Kernels (Cont.)



An active set merges with an inactive set. Kernels denoted in grey.



An active set merges with an active set.

Table of Contents

Open Questions

- How does the primal-dual algorithm perform empirically?
- On what special graphs can the algorithm achieve better approximation?

Thank you!