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

Emmett Breen Renee Mirka Zichen Wang David P. Williamson

{ejb284, rem379, zw336, davidpwilliamson}@cornell.edu
Cornell University, Ithaca, NY 14850

Abstract

This paper revisits the 2-approximation algorithm for k-MST presented by Garg [9] in light of a recent paper of Paul et al. [14]. In the k-MST problem, the goal is to return a tree spanning k vertices of minimum total edge cost. Paul et al. [14] extend Garg's primal-dual subroutine to improve the approximation ratios for the budgeted prize-collecting traveling salesman and minimum spanning tree problems. We follow their algorithm and analysis to provide a cleaner version of Garg's result. Additionally, we introduce the novel concept of a kernel which allows an easier visualization of the stages of the algorithm and a clearer understanding of the pruning phase. Other notable updates include presenting a linear programming formulation of the k-MST problem, including pseudocode, replacing the coloring scheme used by Garg with the simpler concept of neutral sets, and providing an explicit potential function.

1 Introduction

Given as input an undirected graph $G = (V, E)$ and non-negative costs c_e for each edge $e \in E$, the goal of the k-MST problem is to find a tree spanning at least k vertices of minimum total cost. In the rooted case of the problem, a special root vertex r is denoted which must be contained in the final tree, whereas no such r exists for the unrooted case. These two cases are equivalent for the k-MST problem: Garg [9] shows that an α-approximation algorithm for one version can be translated into an α-approximation algorithm for the other. An α-approximation algorithm for k-MST is a polynomial-time algorithm which returns a solution at most α times the cost of the optimal tree.

Approximation algorithms have been considered for k-MST since Ravi et al. introduced the problem [15]. They proved the problem is NP-hard and presented a $3\sqrt{k}$-approximation algorithm. Since then, one line of work by numerous authors [4, 6, 8, 3] developed a $(2+\epsilon)$-approximation algorithm running in time polynomial in ϵ^{-1} [2]. Currently, the best known approximation is a 2-approximation algorithm by Garg using primal-dual techniques [9]. A separate line of work considers a special case of k-MST where the input is given by n points in the plane with edge costs determined by the Euclidean metric [15, 10, 7, 5, 13, 1, 12].

This paper focuses on Garg's result. His 2-approximation algorithm is not only the best known for k-MST, but Garg also shows that his algorithm leads to a 2-approximation algorithm for the k-TSP problem (where one seeks a min-cost tour on at least k vertices instead of a tree and edge costs satisfy the triangle inequality) and 3-approximation algorithm for the budgeted version of k-MST (where a budget B is given and one seeks to maximize the number of vertices in a tree of cost at most B) due to an observation by Johnson et al. [11]. More recently, Paul et al. have extended Garg's technique to improve the approximation algorithms for the budgeted prize-collecting traveling salesman and minimum spanning tree problems [14]. We revisit Garg's algorithm and analysis in light of the paper of Paul et al. In particular, we seek to provide a cleaner version of Garg's algorithm and analysis by adapting the results of Paul et al. We use Paul et al.'s explicit potential function and use of *neutral* sets to replace the coloring scheme used by Garg. Both of these changes improve the clarity and comprehensibility of the initial analysis. Additionally, and distinct from both previous papers, we introduce the notion of a *kernel* in this primal-dual algorithm; its use in part enables a clearer visualization of the mechanics of the algorithm. We believe this perspective yields a more accessible version of Garg's result.

The structure of the paper is as follows. Section 2 introduces a linear programming formulation of the k-MST problem. Section 3 describes the primal-dual subroutine used by Garg, and Section 4 provides an overview of the

entire algorithm along with a lower bound on the cost of the optimal tree. In Sections 5 and 6, we describe the details of finding a specific tree through parameter setting and modifying the results of the primal-dual subroutine. Finally, Section 7 proves the 2-approximation.

2 Linear Programming Formulation

In this section, we provide a linear programming (LP) relaxation of the k-MST problem. The constraints in the dual LP will determine when an *event* happens in the primal-dual subroutine.

For each $S \subseteq V$, let variable $z_S \in \{0,1\}$ denote whether S constitutes the vertices of the spanning tree. For each edge $e \in E$, let variable $x_e \in \{0,1\}$ denote whether edge e is included in the spanning tree. Then, the following is a linear programming relaxation for the k-MST problem:

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

The first constraint guarantees that the spanning tree is connected. If S is a strict subset of the vertices S_T of a spanning tree T, then there must be at least one edge across $\delta(S)$, the cut of S. Given there are no negative-cost edges, any optimal integral solution to the LP will be a tree and have $x_e \leq 1$ for each $e \in E$, and thus these constraints are omitted. We shall also be careful with our subsequent approximation algorithm to not violate these two constraints. We can now write down the dual of this linear program:

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

To produce a tree, we use a primal-dual subroutine for these formulations that will be described in the next section. However, we first observe (similar to Paul et al.) that for any λ_1 and y satisfying the edge constraints, we can find a λ_2 value satisfying the subset constraints. Particularly, we can let λ_2 be the maximum of 0 and $\max_{S \subset V} \{\lambda_1 |S| - \sum_{S_T \subset S} y_{S_T}\}$ and we have a feasible dual solution. Therefore, a key component of this algorithm and the following analysis is choosing a λ_1 value leading to a primal-dual subroutine solution with a tree of the appropriate size. In what follows, we'll see that too small of a λ_1 value leads to too few selected edges, whereas a value of λ_1 that is too large leads to too many selected edges. More details are provided in Section 5.

3 Primal-Dual Subroutine

We now present the primal-dual subroutine used by Garg. The algorithm assumes a fixed λ_1 and, instead of explicitly finding the minimal λ_2 value, greedily grows a forest with respect to a heuristic function of the dual variables. In our case, the function is the *potential* that we define as below.

DEFINITION 3.1. *For any subset $S \subseteq V$, the **potential** of S is*

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

DEFINITION 3.2. *A subset $S \subseteq V$ is* ***neutral*** *if* $\sum_{T:T\subseteq S} y_T = \lambda_1|S|$*, or equivalently, if* $y_S = \pi(S)$.

Intuitively, each vertex has a budget of λ_1, and we spend the budget amassed in connecting vertices to cover the costs of the edges. In this way, we are able to consider the vertices and edges of a set S as a single variable $\pi(S)$ that measures how much budget we have left to spend on future edges. The objective is to connect k vertices with the cheapest possible edge cost; this corresponds well to maximizing the potential.

Algorithm 1 Primal-Dual Subroutine PD(λ_1)

```
y_S ← 0
PD ← ∅
C ← {{v} : v ∈ V}
while C ≠ ∅ do
  raise all y_S corresponding to active components uniformly until either
  if y_S = π(S) then
    C ← C − S
  else if Σ_{S:e∈δ(S)} y_S = c_e for some e between sets S_1, S_2 then
    PD ← PD ∪{e}
    C ← C − S_1 − S_2
    C ← C ∪ {S_1 ∪ S_2}
  end if
end while
return PD
```

The primal-dual subroutine is given in Algorithm 1. Described in words, initially $y_S = 0$ for all S, and all sets consisting of a single vertex are active. At any stage of the algorithm with active components, we uniformly increase y_S corresponding to all active components until either a set event or an edge event happens. Here, a *set event* is a set becoming neutral, while an *edge event* is the constraint corresponding to an edge becoming *tight* (that is, the dual constraint is met with equality). If a set becomes neutral, then we mark this set inactive and remove it from the set of active components. If the dual constraint for an edge between sets S_1 and S_2 reaches equality, then the edge is tight. We add this edge to the set of selected edges, mark S_1 and S_2 inactive if not already, and mark $S_1 \cup S_2$ active. We'll sometimes say that we have *merged* the two sets.

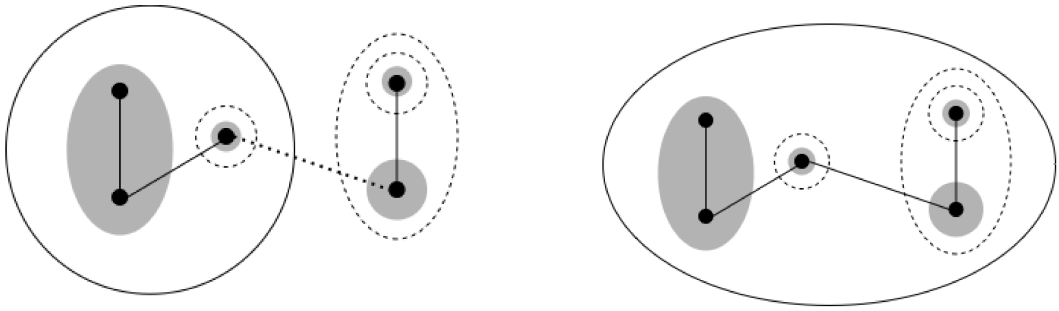
We make a few observations about the structure of the result of the primal-dual subroutine. First, the collection $\mathcal{S}$ of all sets that are ever active during the subroutine is *laminar*; that is, for any pair of sets $A, B \in \mathcal{S}$, either $A \subseteq B$, $B \subseteq A$, or $A \cap B = \emptyset$. Initially, all sets consisting of a single vertex are active. By design, the only way a new, larger set becomes active is through merging two previously active sets when an edge event occurs. This maintains the laminar property. Secondly, the subroutine returns a forest. Each time an edge event occurs, two trees are connected into a larger tree. Furthermore, no cycles can exist. If an edge (u, v) was added that created a cycle, there must have been an active set that contained exactly one of u or v. However, since u and v must already be connected (otherwise this edge would not complete a cycle), any active set that contains u or v must contain them both. If two edges that would complete a cycle go tight at the same time, we choose one of the edges through a tie-breaking procedure described in Sections 5 and 6.

We observe that this procedure always maintains a feasible dual solution. Note that all dual edge constraints are initially satisfied with $y_S = 0$. Furthermore, if an edge becomes tight, the sets corresponding to the endpoints of the edge are marked inactive, so the constraint will never be violated.

Active sets play a crucial role in the subroutine, as these are the sets with the capability for growth. Because of this, we formalize the decomposition of currently-active or once-active sets into neutral subsets and subsets of always-active vertices through the notion of a *kernel*, one of the key points of distinction between our work and Garg's initial presentation as well as Paul et al.'s work.

DEFINITION 3.3. *For an active set S corresponding to a tree T in the set of tight edges of PD, the* ***kernel*** *of S, denoted by $K(S)$, is the smallest cardinality subset of S such that*

1. *if $v \in S$ has always been part of an active set, then $v \in K(S)$,*



(a) An active and inactive set with their kernels before an edge event occurring at the dashed edge.

(b) The new active set and its kernel after the merge.

Figure 1: These figures illustrate how the sets and kernels change when an active set merges with an inactive set. Sets surrounded by a bold (resp. dashed) line are active (resp. inactive). Sets with a grey background are the kernel of the smallest active or inactive set they are contained in.

2. *$K(S)$ is connected in T, and*
3. *for every once-active set $I \subset S$ either $I \subset K(S)$ or $I \cap K(S) = \emptyset$.*

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

Since initially $y_S = 0$ for all subsets S and all sets consisting of a single vertex are active, every vertex is the kernel of itself at the start of the primal-dual subroutine. By the definition of the kernel, we know that the kernel of a once-active (but now inactive) set is the kernel of the set when it went neutral. It remains to understand how the kernel changes as active sets grow throughout the primal-dual subroutine. We illustrate this growth through two possible cases:

1. an active set S merges with an inactive set I or
2. an active set S merges with another active set A.

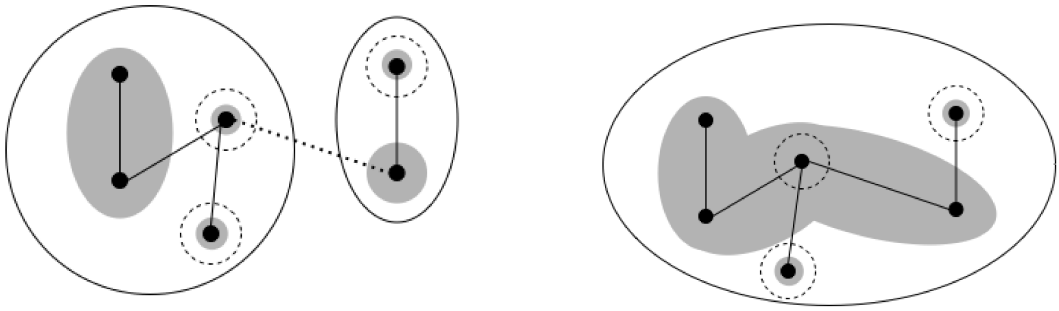
In the first case, the kernel of $S \cup I$ is simply the kernel of S. Since I was inactive at the time of the merge, every vertex in I has been part of an inactive set. Therefore adding any vertices of I to $K(S)$ would simply increase the cardinality of the kernel. Since we want the minimal cardinality set maintaining connectivity (and $K(S)$ is already connnected since it was previously a kernel), we do not add any of the vertices of I. See Figure 1.

In the second case, the kernel of $S \cup A$ will contain both the kernel of S and the kernel of A since both of these contain vertices which have always been contained in active sets. However, we must be careful since the edge event merging S and A may not occur on an edge connecting the kernel of S to the kernel of A and the resulting set should be connected. In this case, we must add the fewest possible number of once-active subsets of S and A by including only the ones on the path from $K(S)$ to $K(A)$. This maintains that $K(S \cup A)$ is connected in the new tree. See Figure 2.

The kernel is used in the next step of the algorithm after the primal-dual subroutine – the *pruning* phase. In this phase, we want to remove sets of vertices that do not help us achieve our end goal. Particularly, we want to remove neutral sets without disconnecting any tree in the forest returned by the primal-dual subroutine. We do so by pruning each tree T, with vertex set S_T, in the forest returned by PD to return exactly the edge-set determined by the kernel of S_T. In particular, a pruned tree T is given by $E(K(S_T)) \cap T$ where $E(S) = \{(u, v) \in E : u, v \in S\}$. The following lemma shows that this has our desired effect; the proof is delayed until Section 7. For sets $S' \subset S \subset V$, let $\delta_{K(S)}(S') = \{(u, v) \in E(S) : u \in S', v \in K(S) - S'\}$.

LEMMA 3.1. *The kernel $K(S)$ of a set S contains no neutral subset $N \subset K(S)$ such that $|\delta_{K(S)}(N)| = 1$.*

Clearly the size of each tree may decrease during the pruning phase, so measures must be taken to ensure we have a tree of suitable size at the end of the primal-dual subroutine. Recalling that we can find a feasible λ_2



(a) Two active sets with their kernels before an edge event occurring at the dashed edge.

(b) The new active set and its kernel after the merge.

Figure 2: These figures illustrate how the sets and kernels change when an two active sets merge. Sets surrounded by a bold (resp. dashed) line are active (resp. inactive). Sets with a grey background are the kernel of the smallest active or inactive set they are contained in.

value for any λ_1 and feasible y_S solution, we carefully select λ_1 guaranteeing at least one kernel in our primal-dual output forest has at least k vertices. After pruning, we execute a picking routine to obtain a tree with exactly k vertices such that the 2-approximation holds. The details of this selection of λ_1 and the picking routine implemented after the primal-dual subroutine and pruning occur are described in the following sections.

4 Algorithm Overview

In this section, we provide an overview of how the forest from the primal-dual subroutine is used to construct a feasible k-MST. Assume we have run the primal-dual subroutine with some value of λ_1 to find a feasible dual solution $(y, \lambda_1, \lambda_2)$, though we may not know λ_2 exactly. We also have a forest F from which we need to select a tree spanning k vertices. In order to do so, we need to guarantee that there exists a tree in F containing at least k vertices after it is pruned. If the fixed λ_1 is too small, this may not be the case as a small λ_1 allows sets to become neutral earlier which limits the potential for growth. On the other hand, if λ_1 is too large, the primal-dual subroutine degrades to greedily selecting the cheapest edges, which is sub-optimal. In order to balance these two scenarios, we search for an appropriate λ_1 value. Specifically, we identify a λ_1 value such that $PD(\lambda_1^-)$ returns a forest where every pruned tree is too small and $PD(\lambda_1^+)$ contains at least one pruned tree large enough. Here $x^- = x - \epsilon$ and $x^+ = x + \epsilon$, where ϵ is arbitrarily small. Once we have this threshold value of λ_1, we prune our selected tree to return a kernel with at least k vertices and bounded cost on the corresponding tree. Finally, we select a sub-tree of our pruned tree containing exactly k vertices.

Before describing how we choose a λ_1 value, we derive a lower bound of the cost of the optimal k-MST in terms of the potential function that works for any choice of λ_1. In the following sections, we will see how to set λ_1 to give an upper bound. Together, this will give us a 2-approximation.

Let $\mathcal{S}$ be the laminar set of all once-active sets plus the set of all vertices. Denote T^* the optimal k-MST, S_{T^*} its set of vertices, and S_1 the set with the minimal potential in $\mathcal{S}$ such that $S_{T^*} \subset S_1$. Since $V \in \mathcal{S}$, such an S_1 always exists. In general, for a tree, we will refer to the set of edges by T and the corresponding set of vertices by S_T.

LEMMA 4.1. *For any* $S \subseteq V$, $\sum_{U:U\subseteq S} y_U \leq \lambda_1 |S|$.

Proof. If S was once active, then the inequality holds by the design of the primal-dual subroutine. We prove this by induction. Suppose S_1 and S_2 merged to form S, and

$$\sum_{U:U\subseteq S_1} y_U \leq \lambda_1 |S_1|,$$

$$\sum_{U:U\subseteq S_2} y_U \leq \lambda_1 |S_2|.$$

Initially $y_S = 0$, so

$$\begin{aligned}\sum_{U:U\subseteq S} y_U &= \sum_{U:U\subseteq S_1} y_U + \sum_{U:U\subseteq S_2} y_U + y_S \\ &\leq \lambda_1|S_1| + \lambda_1|S_2| \\ &= \lambda_1|S|.\end{aligned}$$

We increase y_S either until S merges with another active set or until $\sum_{U:U\subseteq S} y_U = \lambda_1|S|$ and S gets marked neutral. In either case, we no longer increase y_S, so the claim continues to hold. For an arbitrary set S, we can partition it into maximal disjoint laminar subsets $S_1, S_2, ..., S_c \in \mathcal{S}$. Therefore,

$$\sum_{U:U\subseteq S} y_U = \sum_{i=1}^{c} \sum_{U:U\subseteq S_i} y_U \leq \sum_{i=1}^{c} \lambda_1|S_i| = \lambda_1|S|.$$

□

THEOREM 4.1. *The minimal spanning tree has cost at least $\lambda_1 \cdot k - \pi(S_1)$.*

Proof. By the potential definition and Lemma 1,

$$\begin{aligned}\lambda_1|S_1| &= \sum_{U:U\subset S_1} y_U + \pi(S_1) \\ &= \sum_{U:U\subseteq S_1 - S_{T^*}} y_U + \sum_{\substack{U:U\subset S_1 \\ U\cap S_{T^*}\neq\emptyset}} y_U + \pi(S_1) \\ &\leq \lambda_1|S_1 - S_{T^*}| + \sum_{\substack{U:U\subset S_1 \\ U\cap S_{T^*}\neq\emptyset}} y_U + \pi(S_1), \text{ so} \\ \lambda_1|S_{T^*}| &\leq \sum_{\substack{U:U\subset S_1 \\ U\cap S_{T^*}\neq\emptyset}} y_U + \pi(S_1) \\ &\leq \sum_{e\in S_{T^*}} \sum_{U:e\in\delta(U)} y_U + \pi(S_1) \\ &\leq \sum_{e\in S_{T^*}} c_e + \pi(S_1).\end{aligned}$$

The first inequality follows from Lemma 1. The third inequality says that for every set that intersects S_{T^*}, an edge in its cut must lie in T^*. Since we only allow edges with non-negative costs, the spanning tree is minimal when it covers exactly k vertices. By rearranging, we obtain the claim. □

5 Setting λ_1

We now describe how to set λ_1 to enable us to pick k vertices from a pruned tree in our primal-dual subroutine forest. Recall $x^- = x - \epsilon$ and $x^+ = x + \epsilon$, where ϵ is arbitrary small. We will later use *infinitesimal* to refer to variables that approximate their originals as $\epsilon \to 0$. We adapt two lemmas from Paul et al. to the k-MST situation. The first tells us that we can find our desired threshold value λ_1.

LEMMA 5.1. *In polynomial time, we can find a threshold value λ_1 such that all pruned trees of $PD(\lambda_1^-)$ have less than k vertices and there exists at least one pruned tree in $PD(\lambda_1^+)$ with at least k vertices.*

As the details of this proof closely mirror those of Paul et al., we omit them here and refer the reader to Lemma 2 [14]. The key idea is that the time for each set and edge event to occur can be represented as a linear function in λ_1; see Figure 3. Then by observing the threshold λ_1 must occur at an intersection of these lines, we

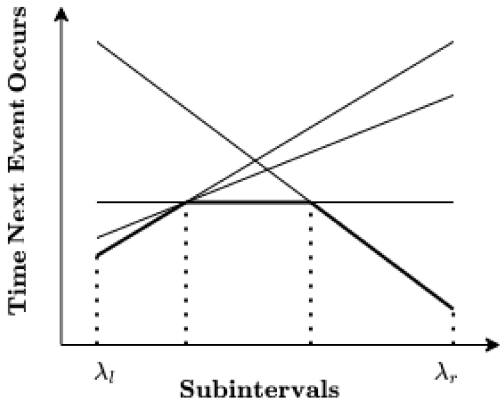


Figure 3: Finding Threshold λ_1. Each line represents the time for an event to occur next in terms of λ_1. The bold line shows the next event, where each segment is from a given subinterval.

can consider smaller and smaller intervals of λ determined by the intersections until we find our desired value. In particular, we recurse onto a smaller subinterval where the next event to occur is consistent throughout the subinterval and consider the possible events that can follow. Eventually, there must be a time (intersection) where the difference in next events translates to a difference in sizes of the resulting kernels; this is where our threshold λ_1 occurs. The next lemma tells us two important properties regarding the primal-dual subroutine for values around our threshold λ_1.

LEMMA 5.2. *Throughout the two subroutines $PD(\lambda_1^-)$ and $PD(\lambda_1^+)$, the following two properties hold:*

- *All active components are the same except for during infinitesimal time.*
- *For all $S \subseteq V$, the difference between y_S^+ and y_S^- is infinitesimal. Here y_S^+ and y_S^- are the dual variables when running $PD(\lambda_1^+)$ and $PD(\lambda_1^-)$, respectively.*

Here, again we present an overview of the proof of Lemma 3 [14] with a few minor changes. The main observation is the claim could only fail if two different events occur at the same time in $PD(\lambda)$, and the events cause lasting disparity in $PD(\lambda_1^-)$ and $PD(\lambda_1^+)$. Furthermore, there are four possible ways that two different events could occur at the same time:

1. different sets go neutral in $PD(\lambda_1^-)$ and $PD(\lambda_1^+)$,
2. an edge goes tight in $PD(\lambda_1^-)$ while a set goes neutral in $PD(\lambda_1^+)$,
3. a set goes neutral in $PD(\lambda_1^-)$ while an edge goes tight in $PD(\lambda_1^+)$, or
4. different edges go tight in $PD(\lambda_1^-)$ and $PD(\lambda_1^+)$.

In the first case, the times for the two sets to go neutral must differ by an infinitesimally small amount and thus one will go neutral immediately after the other. The second case cannot occur for this problem since the time for a set to go neutral has a positive slope in λ_1 while the time for an edge to go tight has a negative slope in λ_1.

For the third case, if the tight edge has an endpoint in an active set different than the set going neutral, the edge will still go tight immediately after the set event. If the edge going tight in $PD(\lambda_1^+)$ merges the set going neutral in $PD(\lambda_1^-)$ and another neutral set, the merged set must have infinitesimally small potential and will go neutral immediately after the edge event. This maintains the active sets. Furthermore, if an active set merges with the set going neutral in the future in $PD(\lambda_1^-)$, the edge will go tight immediately after, still maintaining the active sets.

Finally, for the fourth case, if the edges are between different components one will go tight immediately after the other (similar to the first case). Meanwhile, if they are between the same components, only one will go tight in each subroutine but the resulting active components will be the same.

The main role of Lemma 4 is insight into the potential differences between $PD(\lambda_1^-)$ and $PD(\lambda_1^+)$. In particular, there may be subsets marked neutral in $PD(\lambda_1^-)$ but with infinitesimally small potential in $PD(\lambda_1^+)$ or there may be different edges that went tight between the same components. If, every time two events tied in $PD(\lambda)$, we broke the tie by selecting what $PD(\lambda_1^+)$ would do, we will end up with at least one kernel having at least k vertices. On the other hand, we can consider breaking ties in favor of $PD(\lambda_1^-)$ one at a time. By doing so, we will find the smallest i such that if the first i ties are broken according to $PD(\lambda_1^-)$ and the rest by $PD(\lambda_1^+)$, we return a forest with all kernels containing less than k vertices. By Lemma 3, the dual variables only change by an infinitesimally small amount, and the only differences occur during the pruning phase.

In finding this value of i, we have also either identified a neutral subset X such that if X remained active our forest would contain a kernel of appropriate size, or found two edges e and f between the same components such that adding f instead of e returns a forest containing a kernel of appropriate size. These two cases will play a role in picking our final set of vertices in the next section.

6 Constructing a Tree

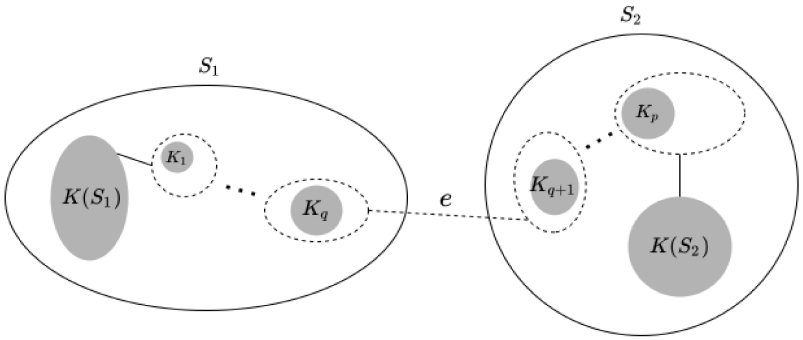
Let λ_1 be our found threshold value and $(y, \lambda_1, \lambda_2)$ our feasible dual solution acquired through tie-breaking in the manner described at the end of Section 5. Currently, all kernels of our primal-dual subroutine output forest have fewer than k vertices, but Section 5 tells us that either we have identified a neutral subset X such that if X remained active our forest would contain a kernel with at least k vertices (Case I) or found two edges e and f between the same components such that adding f instead of e returns a forest containing a kernel with at least k vertices (Case II). The final construction of our tree depends on which of these cases are present and requires us to pick k vertices. Specifically, $pick(X, w, k)$ returns a sub-tree of X with k vertices and contains the vertex w. The idea is to inspect the last two subsets that merged to form the set. Suppose that X_1 and X_2 merged to form X, with edge (u, v) connecting them, and that X_1 contains w. If X_1 contains at least k vertices, then we invoke $pick(X_1, w, k)$. If X_1 has less than k vertices, then we pick all vertices in X_1 and continue to $pick(X_2, v, k - |X_1|)$. We repeat the process recursively until we have picked exactly k vertices.

Algorithm 2 Pick Routine $pick(X, w, k)$

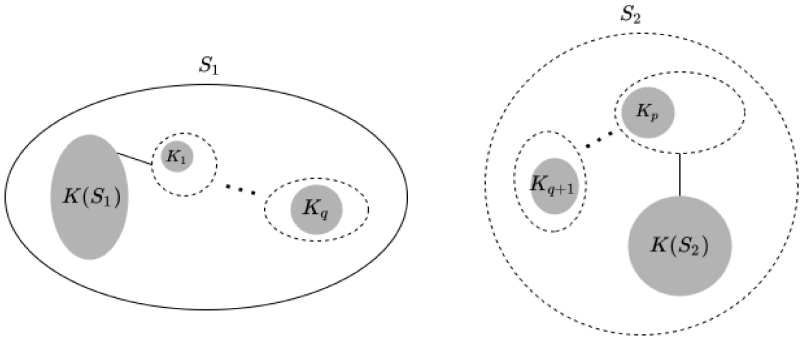
```
let X_1 and X_2 be the two subsets that merged on edge e = (u, v) to form X
suppose with lost of generality that w ∈ X_1
if |X_1| > k then
   call pick(X_1, w, k)
else if |X_1| < k then
   pick all vertices in X_1
   call pick(X_2, v, k − |X_1|)
else
   pick all vertices in X_1
end if
```

In Case I, a set goes neutral the same time an edge goes tight; see Figure 4. Let $K(S_1)$ and $K(S_2)$ be the two kernels of the merging two subsets S_1 and S_2. If we break the tie by choosing the set event, then we would end up with both $K(S_1)$ and $K(S_2)$ having less than k vertices. On the other hand, if we break the tie by choosing the edge event, then the new kernel $K(S_1 \cup S_2) = K(S_1) \cup N_1 \cup \cdots \cup N_p \cup K(S_2)$ would have at least k vertices. Here N_i denotes the neutral sets on the path from $K(S_1)$ to $K(S_2)$. Now we show how to pick exactly k vertices from this new kernel using the pick routine. Starting from $K(S_1)$, we select neutral sets $N_1, N_2, \ldots, N_{q-1}$ until adding another neutral set N_q will cause $K(S_1) \cup N_1 \cup \cdots \cup N_q$ to have at least k vertices. Suppose edge $e = (u, v)$ links N_{q-1} to N_q, then $pick(N_q, v, r)$ will pick the remaining vertices needed, where $r = k - |K(S_1)| - \sum_{i=1}^{q-1} |N_i|$ is the number of additional vertices we need to pick.

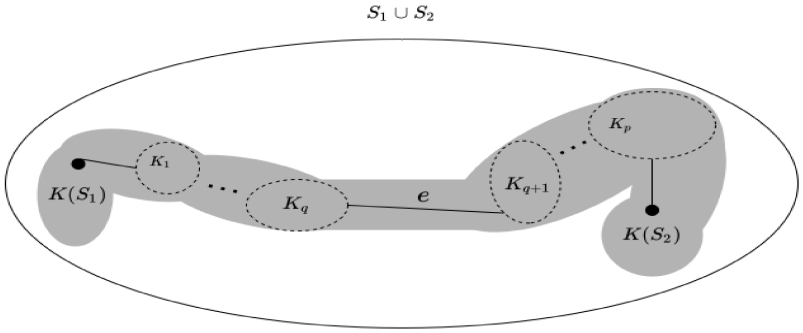
In Case II, two edges go tight simultaneously; see Figure 5. Again, let $K(S_1)$ and $K(S_2)$ be the two kernels of the merging two subsets S_1, S_2. Denote e, f the two edges both between S_1 and S_2. If we choose edge e, then



(a) Sets S_1 and S_2 and their kernels before the edge event causing e to go tight and the set event causing S_2 to go neutral tie in $PD(\lambda_1)$.

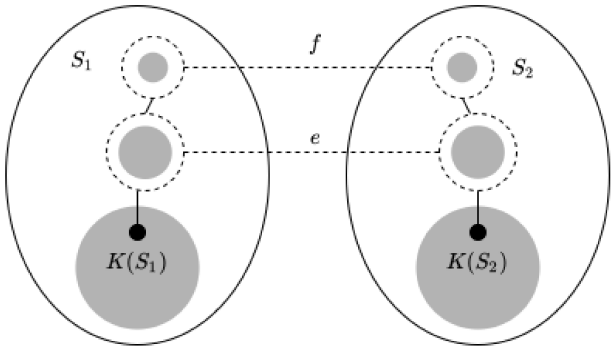


(b) Sets S_1 and S_2 and their kernels after the tie is broken by allowing S_2 to go neutral.

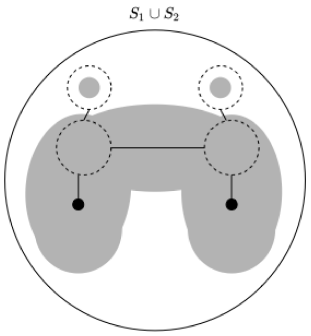


(c) Set $S_1 \cup S_2$ and its kernel after the tie is broken by allowing e to go tight.

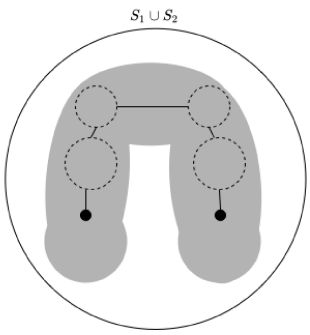
Figure 4: Case I. S_2 goes neutral the same time e goes tight.



(a) Sets S_1 and S_2 and their kernels before the edge events causing e and f to go tight tie in $PD(\lambda_1)$.



(b) Set $S_1 \cup S_2$ and its kernel after the tie is broken so e goes tight.



(c) Set $S_1 \cup S_2$ and its kernel after the tie is broken so f goes tight.

Figure 5: Case II. Edges e and f between S_1 and S_2 go tight at the same time.

the new kernel $K(S_1 \cup S_2) = K(S_1) \cup N_1 \cup ... \cup N_p \cup K(S_2)$ would have less than k vertices, where $N_1, \ldots, N_p$ are the neutral sets between $K(S_1)$ and $K(S_2)$ using edge e. On the other hand, if we choose edge f, then the new kernel $K(S_1 \cup S_2) = K(S_1) \cup N'_1 \cup ... \cup N'_q \cup K(S_2)$ would have at least k vertices, where $N'_1, \ldots, N'_q$ are the neutral sets between $K(S_1)$ and $K(S_2)$ using edge f. The difference clearly results from the neutral sets in between $K(S_1)$ and $K(S_2)$. Once again, we pick $K(S_1), N'_1, N'_2, \ldots, N'_{t-1}$ where picking N'_t would give at least k vertices and invoke $pick(N'_t, v, r)$ to finish the rest (N'_t, v, r defined similar to Case I).

7 2-Approximation

Now that we have finished describing the mechanics of the algorithm, we can finally present the proof of the 2-approximation. Let T' denote the tree we obtained after pruning with vertex set (kernel) K', $T_0 \subseteq K'$ the tree of k vertices we obtained by the pick routine, and S_{T_0} the set of vertices of T_0. Further, let v be the vertex in S_{T_0} where the pick routine terminated. Before proceeding with the necessary lemmas, we present the proof of Lemma 3.1 from Section 3.

Proof. We prove this by induction on the number of events. In the base case, at the start of the primal-dual subroutine, every vertex is the kernel of itself and no neutral sets have appeared yet.

Suppose that for the first k events the claim holds, and we consider the next event. If the $(k+1)$-st event is a set event, then a set S goes neutral. A set going neutral does not affect its kernel, so as previously, the claim still holds for S; other sets are also unaffected so the claim holds for them.

If the $(k+1)$-st event is an edge event, then an edge e between sets S_1 and S_2 goes tight and the two sets merge to form a new active set S. First of all, this event does not affect the kernel of S_1, S_2, or of any other set besides S, so the claim still holds for all of them. Since at least one of the merging sets is active, we can suppose without loss of generality that S_1 is active. Then, if S_2 is inactive, the kernel of S would be the kernel of S_1. For a neutral set $N \subset K(S)$, $|\delta_{K(S)}(N)| = 1$ for some N would imply that $|\delta_{K(S_1)}(N)| = 1$, which contradicts that the claim holds previously.

If S_2 is active, then the kernel of S would be the kernel of S_1 and S_2, plus some neutral sets on the path between the two kernels. If $N \subset K(S)$, then either N is on the path between $K(S_1)$ and $K(S_2)$ or N is a subset of $K(S_1)$ or $K(S_2)$. But every neutral set N on the path has $|\delta_{K(S)}(N)| = 2$, and by the inductive hypothesis, no neutral subset N of $K(S_1)$ or $K(S_2)$ can have $|\delta_{K(S_i)}(N)| = 1$, $i = 1, 2$.

∎

Again, we utilize a result by Paul et al. (Lemma 4 in [14]).

LEMMA 7.1.

$$\sum_{e \in T_0} \sum_{S : e \in \delta(S)} y_S \leq 2 \sum_{\substack{U : U \cap S_{T_0} \neq \emptyset \\ v \notin U}} y_U.$$

Proof. We will prove the inequality for any arbitrary iteration in the primal-dual algorithm. Consider an iteration in which we let $\mathcal{C}$ be the current set of components C such that $|\delta(C) \cap T_0| \geq 1$. We can partition $\mathcal{C}$ into active components $\mathcal{C}_A$ and inactive components $\mathcal{C}_I$. Let v be the final vertex picked and C_v be the unique set in $\mathcal{C}$ containing v.

We claim that if $C_v \in \mathcal{C}_A$, then $\sum_{C \in \mathcal{C}_A} |\delta(C) \cap T_0| \leq 2|\mathcal{C}_A| - 2$, otherwise if $C_v \in \mathcal{C}_I$, then $\sum_{C \in \mathcal{C}_A} |\delta(C) \cap T_0| \leq 2|\mathcal{C}_A| - 1$. Suppose for now that the claim is true. We now prove the inequality in Lemma 4 by induction on the algorithm. At the start of the algorithm, with $y_S = 0$ for all S, both sides of the inequality are equal to 0. At each iteration, let ϵ be the amount that we raise y_C for each active component $C \in \mathcal{C}_A$. The LHS of the inequality increases by $\sum_{C \in \mathcal{C}_A} |\delta(C) \cap T_0| \epsilon$, while the RHS of the inequality increases by either $2|\mathcal{C}_A|\epsilon$ (if $C_v \in \mathcal{C}_I$) or $2(|\mathcal{C}_A| - 1)\epsilon = (2|\mathcal{C}_A| - 2)\epsilon$ (if $C_v \in \mathcal{C}_A$). Then given the claim, the inequality continues to hold inductively. Thus the lemma statement will hold at the end of the algorithm.

To prove the claim, first suppose that C_v is inactive. By Lemma 1, all other neutral subsets of S_{T_0} have degree at least 2. Since v is the last vertex added, C_v is the only inactive component such that possibly $|\delta(C_v) \cap T_0| = 1$. Thus we have

$$\sum_{C \in \mathcal{C}_I} |\delta(C) \cap T_0| \geq 2|\mathcal{C}_I| - 1.$$

Note that edges of T_0 link components in $\mathcal{C}$ to form a tree, so

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap T_0| + \sum_{C \in \mathcal{C}_I} |\delta(C) \cap T_0| \leq 2|\mathcal{C}_A| + 2|\mathcal{C}_I| - 2.$$

Then the last two inequalities imply

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap T_0| \leq 2|\mathcal{C}_A| - 1,$$

and the claim holds for this case.

Now consider the case where $C_v \in \mathcal{C}_A$. By a similar logic, there is no component $C \in \mathcal{C}_I$ such that $|\delta(C) \cap T_0| = 1$, and therefore

$$\sum_{C \in \mathcal{C}_I} |\delta(C) \cap T_0| \geq 2|\mathcal{C}_I|, \text{ implying}$$

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap T_0| \leq 2|\mathcal{C}_A| - 2,$$

and the claim holds for this case, so the proof of the lemma is complete. ∎

The result of Lemma 7.1 allows us to prove the following upper bound on the cost of our tree.

THEOREM 7.1. *The picked tree has cost at most* $2(\lambda_1 \cdot k - \pi(S_2))$, *where* S_2 *is the maximal potential set that contains the picked tree.*

Proof. By the pick procedure, vertices in $S_2 - S_{T_0}$ are either (i) in a pruned neutral subset N_i' or (ii) in the subset N_p where we started our pick procedure. Thus, we have $S_2 = \bigcup N_i' \cup (N_p - S_{T_0}) \cup S_{T_0}$. Since N_i' are neutral, we have

$$\lambda_1 |\cup N_i'| = \sum_{U : U \subseteq \cup N_i'} y_U.$$

N_p is also neutral, and we can partition its subsets into two types: ones that contain vertices in $N_p - S_{T_0}$ and ones that do not. Then we have

$$\lambda_1 |N_p| = \sum_{\substack{U : U \subset N_p \\ U \cap (N_p - S_{T_0}) \neq \emptyset}} y_U + \sum_{U : U \subseteq N_p \cap S_{T_0}} y_U \leq \sum_{\substack{U : U \subseteq N_p \\ U \cap (N_p - S_{T_0}) \neq \emptyset}} y_U + \lambda_1 |N_p \cap S_{T_0}|$$

by Lemma 1 which implies

$$\lambda_1 |N_p - S_{T_0}| \leq \sum_{\substack{U : U \subseteq N_p \\ U \cap (N_p - S_{T_0}) \neq \emptyset}} y_U.$$

Combining with Lemma 7.1, this gives us

$$\begin{aligned}
\lambda_1 |S_2| &= \sum_{U : U \subset S_2} y_U + \pi(S_2) \\
&\geq \sum_{\substack{U : U \cap S_{T_0} \neq \emptyset \\ v \notin U}} y_U + \sum_{\substack{U : U \subset N_p \\ U \cap (N_p - S_{T_0}) \neq \emptyset}} y_U + \sum_{U : U \subseteq \cup N_i'} y_U + \pi(S_2) \\
&\geq \frac{1}{2} \sum_{e \in T_0} \sum_{S : e \in \delta(S)} y_S + \lambda_1 |N_p - S_0| + \lambda_1 |\bigcup N_i'| + \pi(S_2).
\end{aligned}$$

Rearranging gives $\lambda_1 |S_{T_0}| \geq \frac{1}{2} \sum_{e \in T_0} c_e + \pi(S_2)$. ∎

Combining our lower bound from Theorem 1 with the upper bound in Theorem 2, we achieve the 2-approximation.

THEOREM 7.2. *The tree returned by the picked routine has at most twice the cost of the optimal spanning tree of k vertices, that is*

$$\sum_{e \in T_0} c_e \leq 2 \sum_{e \in T^*} c_e.$$

Proof. Recall from Theorem 1 we have that

$$\sum_{e \in T^*} c_e \geq \lambda_1 \cdot k - \pi(S_1),$$

where S_1 is the set with minimal potential in $\mathcal{S}$ that contains S_{T^*}. Since we include the set of all vertices $V(G)$ in $\mathcal{S}$, the fact that both S_{T^*} and S_{T_0} are subsets of $V(G)$ implies that $\pi(S_2) \geq \pi(V(G))$ and $\pi(S_1) \leq \pi(V(G))$. Thus we have

$$\begin{aligned}
\sum_{e \in T_0} c_e &\leq 2(\lambda_1 \cdot k - \pi(S_2)) && \text{by Theorem 2} \\
&\leq 2(\lambda_1 \cdot k - \pi(S_1)) \\
&\leq 2 \sum_{e \in T^*} c_e.
\end{aligned}$$

∎

References

[1] S. Arora, *Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems*, in Journal of the ACM, 45 (1998), pp. 753–782.

[2] S. Arora and G. Karakostas, *A $2+\epsilon$ approximation algorithm for the k-MST problem*, in Mathematical Programming Series A, 107 (2006), pp. 491–504.

[3] S. Arya and H. Ramesh, *A 2.5-factor approximation algorithm for the k-MST problem*, Information Processing Letters, 65.3 (1998), pp. 117–118.

[4] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, *Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, in Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 1995, pp. 277–283.

[5] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-MST problem in the plane*, in Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 1995, pp. 294–302.

[6] A. Blum, R. Ravi, and S. Vempala, *A constant-factor approximation algorithm for the k MST problem (Extended Abstract)*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, 1996, pp. 442–448.

[7] D. Eppstein, *Faster geometric K-point MST approximation*, in Comput. Geom., 8 (1997), pp. 231–240.

[8] N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, in Proceedings of 37th Conference on Foundations of Computer Science, 1996, pp. 302–309.

[9] N. Garg, *Saving an epsilon: a 2-approximation for the k-MST problem in graphs*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 396–402.

[10] N. Garg and D. S. Hochbaum, *An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane*, in Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 1994, pp. 432–438.

[11] D. S. Johnson, M. Minkoff, and S. Phillips, *The prize collecting Steiner tree problem: theory and practice*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 760–769.

[12] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, in SIAM Journal on Computing, 28 (1999), pp. 1298–1309.

[13] J. S. B. Mitchell, A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation algorithm for the geometric k-MST problem in the plane*, SIAM Journal on Computing, 28.3 (1998), pp. 771–781.

[14] A. Paul, D. Freund, A. Ferber, D. B. Shmoys, and D. P. Williamson, *Budgeted prize-collecting traveling salesman and minimum spanning tree problems*, Mathematics of Operations Research, 45 (2020), pp. 576–590.

[15] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi, *Spanning trees short or small*, in SIAM Journal on Discrete Mathematics, 9 (1996), pp. 178–200.