A $2 + \epsilon$ approximation algorithm for the k-MST problem

Sanjeev Arora
Princeton University

George Karakostas
Princeton University

Abstract

For any $\epsilon > 0$ we give a $(2+\epsilon)$-approximation algorithm for the problem of finding a minimum tree spanning any k vertices in a graph (k-MST), improving a 3-approximation algorithm by Garg [5]. As in [5] the algorithm extends to a $(2+\epsilon)$-approximation algorithm for the minimum tour that visits any k vertices, provided the edge costs satisfy the triangle inequality.

1 Introduction

Given an undirected graph $G = (V, E)$ with non-negative edge costs and an integer k, the k-MST problem is that of finding a tree of minimum cost that spans exactly k vertices of G; we call such a tree a k-tree. The problem is known to be **NP**-complete. A sequence of results reduced the approximation factor for this problem from an initial $O(\sqrt{k})$ [10] to $O(\log^2 k)$ [2] and $O(\log k)$ [9] to constant factor [4]. The best constant approximation factor of 3 was achieved by Garg [5]. For the special case of finding the k-MST of points on the Euclidean plane, a PTAS is known to exist ([1] and independently, [8]).

A problem similar to the k-MST is k-TSP: find a tour of minimum cost that visits exactly k vertices; we call such a tour a k-tour. Since the traveling salesman problem (TSP) reduces to k-TSP, we expect to have a bounded approximation guarantee algorithm for the k-TSP only under the assumption that the edge costs satisfy the triangle inequality. Garg's 3-approximation algorithm for the k-MST could be extended to a 3-approximation algorithm for the k-TSP.

We present, for any $\epsilon > 0$, an $n^{O(1/\epsilon)}$ time algorithm that computes a $2 + \epsilon$ approximation to the k-MST problem. We use a simple modification to Garg's 3-approximation algorithm, which uses (as does the earlier [4]) an LP relaxation and the primal-dual framework. We note that Garg had exhibited a "tight" instance for his approach, thus suggesting that 3 is the best approximation ratio achievable using his approach.

To explain our improvement, we first recall how Garg obtained an approximation ratio of 3. He uses the obvious LP relaxation for the problem. The integrality gap for this relaxation (i.e., the worst-case ratio of the integer optimum and the fractional optimum) is unbounded. However, he uses an additional lowerbound for the optimum cost (in addition to the LP value), the diameter D of the optimum k-tree. (The diameter can be easily "guessed" by the algorithm by trying all possible $\binom{n}{2}$ internode distances.) Another source of improvement in Garg's paper (as compared to the earlier [4]) is a modification of the 'traditional' primal-dual schema (as described in [7]) to incorporate a final phase of 'potential' reduction that allows him to prove tighter upper and lower bounds. Roughly speaking, the factor 3 ultimately derives from two sources: a (familiar) factor 2 due to the traditional primal-dual analysis, and a factor 1 from the diameter lowerbound.

Our improvement derives from changing the contribution of the diameter lowerbound from OPT to $\epsilon \cdot OPT$ (OPT is the length of a minimum k-tree). We achieve this by allowing the algorithm to "guess" not just 2 vertices in the optimum k-tree (which are needed to guess the diameter), but as many as $O(\frac{1}{\epsilon})$ vertices. This set of vertices have the property that every vertex in the optimum has distance at most $\epsilon \cdot OPT$ to this set. Note that the $n^{O(1/\epsilon)}$ running time is due to the necessity of having to try all $\binom{n}{O(1/\epsilon)}$ choices for this set of vertices.

Finally, we note that the improved $2+\epsilon$ approximation factor has immediate implications for the problem of finding the *minimum latency* tour in a metric space. The *minimum latency problem* is the following: given a graph $G = (V, E)$ with edge costs that satisfy the triangle inequality and a starting vertex, find a tour that visits each vertex and minimizes the sum of the latencies of the vertices, where the latency of a vertex is defined as its distance in the tour from the starting vertex. The first constant factor approximation algorithm for this problem was given in [3], while [6] give the currently best factor of $3.55 \times \alpha$, where α is the approximation factor for the minimum k-tour. Therefore our result improves the approximation factor for the minimum latency problem from 10.77 to 7.11, albeit with a higher running time.

2 The LP formulation

Like Garg [5], we are going to deal with the rooted version of the k-MST problem. Namely, in addition to G, the weights of the edges and k, we are also given a vertex $r \in V$ to be included in the solution. Notice that the general problem reduces to this problem by simply 'guessing' the root r and solving the rooted version. As mentioned earlier, the algorithm will guess a set of $O(1/\epsilon)$ vertices from the optimum k-tree. We rely on the following simple lemma.

LEMMA 2.1. *For every ϵ the following is true in any tree of total length OPT. There is a set of $O(1/\epsilon)$ edges and vertices such that: (a) Every other tree edge has length less than $\epsilon \cdot OPT$ (b) all other vertices in the tree are at distance at most $O(\epsilon OPT)$ from one of the vertices or edges in the set.*

Proof. First pick all edges that are longer than ϵOPT, together with the vertices incident to them. There are at most $\frac{1}{\epsilon}$ such edges. Now pick the rest of the vertices as follows:

First label all unpicked vertices in the tree as *unmarked*. Repeat the following procedure until all vertices in the tree are either *picked* or *marked*:

- Label as *marked* all *unmarked* vertices that are at a distance at most ϵOPT from some picked vertex.
- If there is no *unmarked* vertex left, then terminate. Otherwise, pick an unmarked vertex whose distance from one of the picked vertices is between ϵOPT and $2\epsilon OPT$ (there is at least one such vertex, since we have already picked the vertices adjacent to edges of length at least ϵOPT). Mark this vertex and go to the previous step.

It is obvious that the vertices picked by the procedure above satisfy the lemma requirements.

Our algorithm will "guess" the sets $W \subseteq V$ and $F \subseteq E$ from the optimum k-tree whose existence is guaranteed by Lemma 2.1. Thus $|W| = |F| = O(\frac{1}{\epsilon})$. By "guessing" we mean that the algorithm tries all possibilities (in $n^{O(1/\epsilon)}$ time) and runs the primal-dual procedure described below for each of them. It is guaranteed to ultimately guess the correct W, F whose existence is guaranteed by the lemma, in which case the primal-dual method below will return a tree of cost $(2+\epsilon)OPT$.

Now we provide details of the primal-dual framework.

The constant factor approximation algorithms of [4], [5] are based on the relaxation of the following integer program for the rooted k-MST .

$$
\begin{array}{rcll}
\text{minimize} & & \sum_{e\in E} x_e c_e & \\
\text{subject to} & & & \\
\sum_{e\in\delta(S)} x_e & \geq & x_v & (\forall v, S : v \in S \subseteq V - r) \\
\sum_{v\in V} x_v & = & k & () \\
x_v & \leq & 1 & (\forall v \in V) \\
x_v & \in & \{0,1\} & (\forall v \in V) \\
x_e & \in & \{0,1\} & (\forall e \in E)
\end{array}
$$

That is, if a vertex v is such that $x_v = 1$, then every set $S \subseteq V - r$ with $v \in S$ must have an outgoing edge $e \in \delta(S)$ such that $x_e = 1$. From now on whenever we refer to a set S it is understood that S does not contain the root r.

As mentioned, our algorithm will "guess" a set $W \subseteq V$ of vertices and a set $F \subseteq E$ of edges and require that the solution should pick them. This means that the corresponding variables are set to 1 in the above LP. Note that if $S \subseteq V - r$ is such that $\delta(S) \cap F \neq \emptyset$ then the equation for S is trivially satisfied and can be omitted.

This suggests a simplification. We collapse every connected component defined by F into a supervertex at the start of the algorithm. Collapsing means repeating the following for all edges in F: if the edge is $\{u, v\}$, then u, v are merged into a single supervertex whose distance to any third vertex x is defined to be $\min\left\{c_{\{u,x\}}, c_{\{v,x\}}\right\}$.

All supervertices created in this way are added to W. Thus we can w.l.o.g. think of F as $\emptyset$. ¿From now on, we use this new weighted graph and use the symbol E for its edge set. Let $w = |W|$.

Now we are left with the following fractional relaxation for the k-MST problem in which the guessed set of vertices W has to included in the solution.

$$
\begin{array}{rcll}
\text{minimize} & & \sum_{e\in E} x_e c_e & \\
\text{subject to} & & & \\
\sum_{e\in\delta(S)} x_e & \geq & x_v & (\forall v, S : v \in V \setminus W, v \in S) \\
\sum_{e\in\delta(S)} x_e & \geq & 1 & (\forall v, S : v \in S \cap W) \\
\sum_{v\in V\setminus W} x_v & = & k - w & \\
x_v & \leq & 1 & (\forall v \in V \setminus W) \\
x_v & \geq & 0 & (\forall v \in V \setminus W) \\
x_e & \geq & 0 & (\forall e \in E)
\end{array}
$$

The dual of this linear program has a variable $y_{v,S}$ for all v, S such that $v \in S \subseteq V - r$ (notice that v can be any vertex, even a vertex in W), a variable p_v for vertex $v \in V \setminus W$ and a variable p:

$$
\max\ (k-w)p - \sum_{v\in V\setminus W} p_v + \sum_{v\in W} \sum_{S: v\in S} y_{v,S}
$$
subject to

$$\begin{array}{rcll}
\sum_{S:v\in S} y_{v,S} + p_v & \geq & p & (\forall v \in V \setminus W) \\
\sum_{S:e\in\delta(S)} \sum_{v\in S} y_{v,S} & \leq & c_e & (\forall e \in E) \\
p_v & \geq & 0 & (\forall v \in V \setminus W) \\
y_{v,S} & \geq & 0 & (\forall v, S : v \in S) \\
p & \text{free} & &
\end{array}$$

For notational ease we introduce some new (dummy) variables: α_v for each vertex v and z_S for each subset S. These are shorthands for the following quantities:

$$\alpha_v = \sum_{S:v\in S} y_{v,s} \tag{2.1}$$

$$z_S = \sum_{v\in S} y_{v,S} \tag{2.2}$$

The following lemma is a simple modification of a lemma in [5].

LEMMA 2.2. *In the optimum solution to the dual, p has a value between the $(k-w)^{th}$ and the $(k-w+1)^{th}$ smallest values of α_v. The optimum value of the dual program is the sum of the $k-w$ smallest α_v's and the α_v's of the vertices in W.*

Proof. Notice that if $\alpha_v \geq p$ then $p_v = 0$ else $p_v = p - \alpha_v$. If p is greater than the $(k-w+1)^{\text{th}}$ α_v then we can decrease p and all positive p_v's by some small ϵ, thus increasing the objective function by at least ϵ. A similar argument holds for the case p is strictly less than the $(k-w)^{\text{th}}$ smallest α_v. So only the vertices with the $(k-w)$ smallest values of α_v have positive p_v's and for these vertices $\alpha_v = p - p_v$. Thus the optimum value is the sum of the $k-w$ smallest α_v's plus the α_v's of the vertices in W.

We call any function $\pi : V \to \mathcal{R}]^+$ with $\pi(r) = 0$ a *potential assignment.*

DEFINITION 1. *A potential assignment π is* feasible *if there exists an assignment of non-negative values to $y_{v,S}$ for all $v, S : v \in S$ such that for any vertex $v \in V$, $\pi(v) \leq \sum_{S:v\in S} y_{v,S}$ and for any edge $e \in E$, $\sum_{S:e\in\delta(S)} \sum_{v\in S} y_{v,S} \leq c_e$ and*

Note that the values α_v that follow from any solution of the dual program define a feasible potential assignment.

LEMMA 2.3. *Let π be a feasible potential assignment and T be any tree that is rooted at r. Then T has cost at least $\sum_{v\in T} \pi(v)$.*

Proof. Let $(y_{v,S})$ be the nonnegative vector that witnesses the feasibility of π. Then we have

$$\begin{array}{rcl}
\sum_{e\in T} c_e & \geq & \sum_{e\in T} \sum_{S:e\in\delta(S)} \sum_{v\in S} y_{v,S} \\
& \geq & \sum_{e\in T} \sum_{S:e\in\delta(S)} \sum_{v\in T\cap S} y_{v,S} \\
& = & \sum_{v\in T} \sum_{S\ni v} \sum_{e\in T\cap\delta(S)} y_{v,S} \\
& \geq & \sum_{v\in T} \sum_{S\ni v} y_{v,S} \\
& \geq & \sum_{v\in T} \pi(v)
\end{array}$$

where the first inequality is due to feasibility, the second follows after we drop some terms, the third after some rearrangement and the penultimate line follows by the observation that any set S that intersects the tree must have a tree edge leaving it.

3 The algorithm

As mentioned before, the algorithm will guess a set of vertices and edges as given by Lemma 2.1 and the value of OPT. Collapsing the edges into supervertices, the algorithm will be left with a set W of vertices that have to be included in any solution.

As in Garg's algorithm, we will use the procedure TREEGROW below for progressively larger values of C (from 1 to OPT) until the returned tree has at least k nodes. Then the tree will be pruned to contain exactly k nodes.

3.1 Procedure TREEGROW

This procedure is given a number $C \in \Re^+$, a root r and a set W of vertices. It generates a tree T_C rooted at r that contains W and possibly some other nodes. (The number of nodes is determined by C, as will become clear.) The procedure also generates a feasible potential assignment π_C.

Initially, we assign each vertex $v \in V \setminus W$ a credit C, which it will use to increase the value of variables $y_{v,S}$ for $S \ni v$. Vertices in W have infinite credit (or credit at least $OPT + 1$). At each step the algorithm maintains a forest, whose components are either *active* or *inactive*. The component connected to the root is always inactive. Any other component is active iff it contains a vertex with positive credit. Initially the forest consists of singletons and all components except root r are active. As the dual variables rise in value, edges go tight (i.e., the inequality for this edge in the LP relaxation becomes tight). Whenever an edge goes tight, it is added to the forest and the components attached at its endpoints are merged. (We will later also describe a delete phase in which edges may be deleted.)

At each step, the algorithm picks, for every active component S in the current forest, one vertex, say v, whose credit is positive. This selects a dual variable $y_{v,S}$ for S. Having selected a dual variable for each component, the algorithm raises all of them simultaneously and at the same rate, and decreases the credit of the picked vertices at the same rate. (Thus the credit of the vertices pays for the dual increases.) The step ends when either (a) the credit of some vertex drops to zero or (b) some edge becomes tight. If (a) happens, the algorithm just finds another vertex in that component whose credit is positive; if no such vertex can be found, then the component becomes *inactive.*

Now a few observations. First, the dual increase rule can be also viewed as increasing z_S at a uniform rate for all active components. (See (2.2) for definition of z_S.) This is because the algorithm increases $y_{v,S}$ for only one vertex $v \in S$ at any step. Note also that all vertices in W are going to be included in the final solution, because their credit stays positive throughout and so they stay in active components, until their component attaches to the root component.

Delete Phase. At the end, we delete all inactive components that are not on a path from an active component to the root (i.e. inactive components that are leaf subtrees of the root component). This process is done in a bottom-up fashion: first we prune all the leaf inactive components (except the root component, if it is a leaf), then we are left with a new (pruned) tree, we again prune the inactive leaves and so on, until we are left with a tree where all leaves are active components with the possible exeption of the root component.

Constructing a feasible potential assignment. Now we construct a potential assignment π_C whose feasibility is witnessed by the dual values $y_{v,S}$ constructed by the algorithm. For each vertex v whose final credit is 0 (note: $v \notin W$), $\pi_C(v)$ is C. Note that any vertex whose final credit is positive must be in the root component, and before it joined the root component it was always in an active component. For such vertices we do the following procedure. We start decreasing the initial credit of all such vertices uniformly and till we notice that one of them, say u, would now finish the algorithm with zero final credit. Then $\pi_C(u)$ is set to the (new) initial credit of u. We continue in this manner until we ensure that all vertices finish the algorithm with zero final credit.

Finally, the tree T_C output by the procedure is the (pruned) component containing the root.

By construction, if the algorithm is started with initial credits given by π_C, then we end up with a collection of inactive components —one of them containing the root —which all have zero final credit. Thus we have the following lemma.

LEMMA 3.1. *The potential assignment π_C is feasible.*

Proof. The proof follows from two observations. First, the potential assignment (i.e., initial credit values) are being used to exactly pay for the increase in the dual values, so $\pi_C(v) = \sum_{S \ni v} y_{v,S}$. Second, the edges are added to the forest as soon as they become tight, so edges never go overtight.

Let $k_C = |T_C|$. The next lemma follows from the construction of π_C.

LEMMA 3.2. *$W \subseteq T_C$ and T_C contains the $k_C - |W|$ vertices in $V \setminus W$ with the smallest potentials. All vertices outside T_C have potential C.*

Proof. The vertices that did not get included in T_C were among the ones who expended all their credit C on growing their dual values before the end of the algorithm. Hence their initial credit was not changed at the end, so their potential is C. Vertices in $T_C \setminus W$ have potential at most C.

3.1.1 The lower & upper bounds

Now we try to relate the cost of T_C with OPT. Since π_C is feasible, Lemma 2.3 implies that OPT is lowerbounded by the sum of potentials of nodes in the optimum tree. Hence we have the following Corollary to the previous two lemmas.

COROLLARY 1. *If W is contained in the optimum k-tree, then OPT is at least the sum of the potentials of vertices in W and the sum of the $k - w$ smallest potentials in $V \setminus W$.*

For an upper bound we give the standard proof of upper bounds in primal-dual schemes (cf. [7], [4]):

LEMMA 3.3.

$$\sum_{e \in T_C} c_e \leq 2 \sum_{v \in T_C} \pi_C(v).$$

Proof. By the way the potentials were defined, we have

$$\sum_{v \in T_C} \pi_C(v) = \sum_{S \subseteq T_C} \sum_{v \in S} y_{v,S} = \sum_{S \subseteq T_C} z_S$$

We wish to upperbound

$$\begin{aligned} \textstyle\sum_{e \in T_C} c_e &= \textstyle\sum_{e \in T_C} \sum_{S: e \in \delta(S)} z_S \\ &= \textstyle\sum_{S \subseteq T_C} y_S |T_C \cap \delta(S)| \end{aligned}$$

So the Lemma will be proved if we could show

$$\sum_{S \subseteq T_C} z_S |T_C \cap \delta(S)| \leq 2 \times \sum_{S \subseteq T_C} z_S \tag{3.3}$$

We prove (3.3) by induction for every step of the algorithm. Initially this is true since all the z_S's are zero.

At any stage of the algorithm, let H be a tree whose vertices are the components formed till this stage, and whose edges are the edges of T_C that are missing from these components (i.e, will connect them at the end). Let N_a, N_i denote the sets of active and inactive components at this stage respectively. Recall that the algorithm raises all current z_S's at the same rate. When each z_S for the active components rises for ϵ, the algorithm increases the LHS of (3.3) by $\epsilon \sum_{v \in N_a} \deg_v$ and the RHS by $2\epsilon|N_a|$. It suffices to show that the average degree of the components in N_a is at most 2.

Owing to the **Delete** phase, the leaves in H cannot be inactive components, except possibly the component containing r. Thus all components $S \in N_i$ except possibly one have degree at least 2, and so the sum of the degrees of components in N_i is at least $2|N_i| - 1$. The sum of the degrees of components in tree H is $2|H| - 1$, so the sum of the degrees of components in N_a is at most $2|H| - 2|N_i| = 2|N_a|$. Hence the average degree of components in N_a is at most 2.

3.2 Algorithm

Now we can prove our main result

THEOREM 3.1. *For any constant $\epsilon_0 > 0$, there is a polynomial time algorithm that approximates the optimal solution to the k-MST problem within a factor $2 + \epsilon_0$.*

Proof. The algorithm is a simple modification of Garg's algorithm. Here we sketch its phases:

Guessing. Let $\epsilon = \frac{\epsilon_0}{2}$ be any constant. First we make a number of guesses. Namely, we guess

1. The root $r \in V$ together with $O(\frac{1}{\epsilon})$ vertices (that are going to be included in the final solution).

2. The value of the optimal solution OPT (OPT will denote both the optimal solution and its value). In order to keep the number of possible values of OPT within a polynomial of the instance size, we rescale the instance given to us as follows: we guess the furthest vertex in the optimal solution ($O(n)$ choices) and let D denote its distance from the root. D is the diameter of the optimal solution, so $D \leq OPT$. Next we zero out all edge lengths that are less than $\frac{\delta D}{k}$, for some sufficiently small δ. This can affect the cost of the optimal k-tree by at most $k \times \frac{\delta D}{k} \leq \delta OPT$, and the ratio of maximum to minimum internode distance is at most $\frac{k}{\delta} = O(n)$ since δ is a constant. By rescaling we get an instance of internode distances between 1 and $O(n)$. Thus OPT lies between 1 and $O(kn)$.

3. A number m ($0 < m \leq O(\frac{1}{\epsilon})$)

4. A set F of m edges between picked vertices of length greater than ϵOPT, *that do not form a cycle.*

By guessing we mean exhaustive enumeration of all possibilities. Namely, there are $n^{O(\frac{1}{\epsilon})}$ choices for the first step, $O(n)$ choices for the second, $O(\frac{1}{\epsilon})$ for the third and $2^{O(1/\epsilon^2)}$ choices for the fourth step. Notice that some of the possibilities may be inconsistent. For example, for some m in step 3 it may be impossible to have m edges of length greater than ϵOPT. Such choices are discarded right away. Keep the vertices that are either guessed, or adjacent to a guessed edge or in distance less than ϵOPT from a guessed vertex or edge. Discard the rest of the vertices.

Now we try to come up with a tree. As mentioned earlier, we collapse each component defined by F into a single supervertex, thus wlog $F = \emptyset$ and then W consists of the guessed vertices and the supervertices.

We call procedure TREEGROW for increasing values of initial credit C, ranging from 1 to OPT.

Suppose that some value C_0 is a threshold in the following sense: if we give initial credit $C_0 - \delta$ (where δ is infinitesimaly small), the tree $\hat{T}_{C_0-\delta}$ has $k_1 < k$ vertices, but if we give initial credit C_0, the tree $\hat{T}_{C_0}$ has $k_2 \geq k$ vertices. In order to construct a k-tree, we use Garg's procedure (cf. Section 5 in [5]) to pick $s \geq k_2 - k_1$ vertices from $\hat{T}_{C_0}$ and append them to $\hat{T}_{C_0-\delta}$. These extra vertices occur *contiguously* in $\hat{T}_{C_0}$ and can be connected to $\hat{T}_{C_0-\delta}$ by just an edge (cf. Lemma 5.2 in [5]). Since our guessing phase ensured that every vertex in $V \setminus T_{C_0-\delta}$ has distance at most $2\epsilon \cdot OPT$ to a vertex in W (and hence to a vertex in $T_{C_0-\delta}$), this additional edge would incur cost $\leq 2\epsilon \cdot OPT$ This gives us a tree T with k nodes.

Note that the vertices in $V \setminus \hat{T}_{C_0-\delta}$ have the same potential $\pi_{C_0-\delta}(v) = (C_0 - \delta)$ (see our Lemma 3.1). Furthermore, we can pick the extra vertices in such a way that the proof of Lemma 3.3 works (with slight modifications) also for T (see Lemma 5.3 in [5]). Accounting for the extra edge we need in order to connect the new vertices to $\hat{T}_{C_0-\delta}$ by an extra factor of $2\epsilon OPT$, we get

LEMMA 3.4.

$$\sum_{e \in T} c_e \leq 2 \times \sum_{v \in T} \pi_C(v) + 2\epsilon OPT \leq (2 + 2\epsilon) \times OPT.$$

where the second inequality follows from the fact that the tree T we constructed contains the vertices of W plus the $k - |W|$ vertices with the smallest potentials.

Since $\epsilon = \frac{\epsilon_0}{2}$ the theorem follows.

From the above proof it follows that the running time of the original algorithm has increased by at most a factor $O(2^{1/\epsilon_0{}^2} n^{O(1/\epsilon_0)}) = O(n^{O(1/\epsilon_0)})$, since ϵ_0 is a constant.

4 The metric k-TSP problem

The metric k-TSP problem is defined as follows: given a graph $G = (V, E)$ with non-negative edge costs that satisfy the triangle inequality, we wish to find a cycle of minimum cost that visits exactly k vertices; we call such a cycle a k-tour. Again wlog we will consider the rooted version of this problem.

For the case of k-MST we used the fact that the final potential assignment π_C was *feasible*, namely any tree T rooted at r and including the preselected vertices in W, had cost at least as much as the sum of the potentials of its vertices. This means that the sum of the $k - |W|$ smallest potentials plus the potentials of the vertices in W were a lower bound for the cost of the edges in the optimal k-tree. Moreover, any feasible assignment has another property: for any tour P, the cost of P is at least twice the sum of the potentials of the vertices in P. We can see that by noticing that for a cycle, every component we grow pays for *two* edges of the cycle (since a component that is entered by the cycle must be left by it as well). Hence, if T denotes the k-tree constructed by the algorithm, OPT_{tour} the minimum k-tour and OPT_{tree} the minimum k-tree, we have:

$$2 \sum_{v \in T} \pi_C(v) \leq OPT_{\text{tour}}$$

since T contains the vertices with the $k - |W|$ smallest potentials together with the vertices of W. Also, from the analysis of the k-MST algorithm we get:

$$OPT_{\text{tree}} \leq 2 \sum_{v \in T} \pi_C(v) + \epsilon OPT_{\text{tour}} \leq (1 + \epsilon) OPT_{\text{tour}}$$

Now we construct a tour out of T, by duplicating its edges and shortcircuiting an eulerian walk on it, incurring an additional approximation factor of 2. Hence, the tour constructed has cost at most $(2+\epsilon') \times OPT_{\text{tour}}$ for any $\epsilon' > 0$.

5 Conclusions

Though the LP relaxation for k-MST has an unbounded integrality gap, Garg was able to achieve an approximation ratio of 3 by allowing his algorithm to do limited "guessing" (namely, guess the root of the optimum solution as well as the diameter). We were able to improve the approximation ratio by allowing the algorithm to do some more guessing. It would be interesting to explore this idea of limited guessing in the context of other problems with nasty integrality gaps.

We also note that there is a good reason why we have been unable to improve the approximation ratio beyond 2: the LP relaxation used here is off by a factor 2 even when $k = n$, i.e., for the minimum spanning tree problem. For the MST problem, the tight relaxation (due to Edmonds) bidirects all edges to get a digraph and finds a min-cost branching. It may be possible to use an analogous relaxation for the k-MST (in conjunction with our idea of guessing $O(1/\epsilon)$ vertices in the optimum) to obtain a PTAS.

References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. In *Journal of the ACM*, 45(5):753–782, 1998.

[2] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum weight k-trees and prize-collecting salesmen. In *Proceedings,27th ACM Symp. on Theory of Computing*, pp 277–283, 1995.

[3] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan and M. Sudan. The minimum latency problem. In *Proceedings,26th ACM Symp. on Theory Of Computing*, pp 163–171, 1994.

[4] A. Blum, R. Ravi, and S. Vempala. A constant factor approximation for the k-MST problem. In *Proceedings,28th ACM Symp. on Theory of Computing*, pp 442–448, 1996.

[5] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings, 37th IEEE Symp. on Foundations of Comp. Science*, pp 302–309, 1996.

[6] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proceedings,ACM-SIAM Symp. on Discrete Algorithms*, pp 152–158, 1996.

[7] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. In *SIAM J. Comput.*, 24:296–317, 1995.

[8] J.S.B. Mitchell. Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems. SIAM Journal on Computing:**28**, 1999.

[9] S. Rajagopalan and V.V. Vazirani. Logarithmic approximation of minimum weight k-trees. Unpublished manuscript, 1995.

[10] R. Ravi, R. Sundaram, M. Marathe, D. Rosenkrantz, and S. Ravi. Spanning trees short and small. In *Proceedings, 5th ACM-SIAM Symp. on Discrete Algorithms*, pp 546–555, 1994.