INTERFERENCE IN MULTI-HOP RADIO NETWORKS

Yanic Heer
Betreuer: Thomas Moscibroda
ETH Zürich

ABSTRACT

Many problems are of the following form: Given a graph find a network satisfying some connectivity constraints while minimizing the degree. We describe in this document such problems.

Date: WS 2005/2006.

1. INTRODUCTION

In this section we describe what is known about different variants of the directed minimum-degree spanning tree problem and similar problems. The two main variants of minimum degree problems are those working on directed and on undirected graphs.

So far research for the undirected case proceeded further. In the minimum-degree spanning tree problem we are given an undirected graph G and have to compute a spanning tree of minimum tree among all spanning trees. The next lemma shows that minimum-degree spanning tree is NP-hard

Lemma 1.1. *Minimum-degree spanning tree is NP-hard*

Proof. We reduce Hamilton path to minimum-degree spanning tree. Given an undirected graph G for which we have to decide whether there is a Hamilton path, we compute a minimum-degree spanning tree in G. It is easy to see that G has a Hamilton path if and only if G has a spanning tree with maximum degree 2. □

Because of the above lemma research focused on developing approximation algorithms for approximating minimum-degree spanning tree. The first approximation algorithm developed by Fürer and Raghavachari in [5] achieved a ratio of $\log(n)$(where n is the number of vertices). It works by building a tree step by step by taking the union of low-degree graphs. In a second paper Fürer and Raghavachari improved upon their first algorithm and presented an additive plus 1 algorithm for the undirected case ([6]). Since minimum-degree spanning tree is NP-hard this is the best ratio we can hope to achieve.

We can generalize minimum-degree spanning tree by allowing some vertices must not be covered. This corresponds to the minimum-degree steiner tree. We are given a set of terminals $T \subset V(G)$ and we have to compute a spanning tree covering all terminals with minimum degree among all such trees. The first approximation algorithm for this problem appeared in [1] , it achieved a ratio of $\log(|T|)$ using network flows. Later Fürer and Raghavachari could generalize their $+1$ approximation algorithm for minimum-degree spanning tree to minimum-degree steiner tree ([4]). Recently, Klein presented a method for approximating the minimum-degree problem for all connectivity constraints which can be modelled by proper functions ([9]). Proper functions, made popular by Goemans and Williamson in ([8]), define for each cut how many edges cross that cut. Note that the minimum-degree spanning and steiner tee problem can be modelled as proper functions. Their algorithm achieves a ratio of $cOpt + \log_c n$ where n is the number of

vertices and Opt is the degree of the degree minimal tree satisfying the connectivity constraints. Consider the following variant of minimum-degree spanning tree: We are given an undirected graph and a degree bound and have to find a spanning tree violating the degree bound not too much and with as little weight as possible. The first algorithm developed in [14] computes a tree of degree at most $O(d\log(n/d)$ and weight at most $O(\log(n))$ times the weight of the lightest tree respecting the degree bound d. Könemann and Ravi ([10]) improve this by describing an algorithm which finds a tree of degree $O(d+\log(n))$ whose cost is at most $O(1)$ times the cost of an optimal tree of degree at most d.

For the directed case much less is known. The best algorithm currently known achieves a ratio of $cOpt + \log(n)$ in pseudo-polynomial time ([11]).

A further variant of minimum-degree spanning tree arises when each edge covers not only one point, but a whole set of points. The goal is to minimize the maximum degree (where the degree of a point is the number of times it is contained in a tree edge). There are several directions in which research proceeded:

- Restricting the points to be in a euclidean space
- Discarding all connectivity constraints

If we restrict the points to be in the 2D-euclidean space and adapt the meaning of an edge we arrive at the model considered in [7] and describe later. Another minimum degree problem is the one introduced in [15]. The points are located on a line and we have to set the ranges the points such that the resulting graph is connected. The resulting graph contains an edge (u, v) if and only if u reaches v and u reaches v. A point reaches a point if its range is greater than the distance to that point. The degree (or interference as called in the paper) is the maximum number of times a point is covered by the edges of a range assignment. The goal is to compute a range assignment t minimizing the interference. The approximation algorithm in [15] achieves a ratio of $\sqrt[4]{n}$ if every node can connect with every other node.

If we discard the connectivity constraints the problem is equivalent to the minimum membership problem. In this problem we are given a collection of sets and we have to find a subset of edges E' covering every point with minimum maximum degree. The problem allows a $(1+\epsilon)log(n)$ approximation which is nearly optimal([12]).

The remaining parts of this document are organized as follows: In section two we introduce some facts about complexity and inapproximability, in section three we present an approximation algorithm for

minimum degree directed spanning tree. In section three we show that it is hard to approximate minimum degree steiner tree better than $(1-\epsilon)\ln(n)$. In section four we describe a $+1$ approximation algorithm for minimum degree spanning tree. In the last section we describe an algorithm for a geometric variant of minimum degree spanning tree.

2. BASICS

In this section we summarize some basics such as definition of approximation algorithms, run times and inapproximability.

Since it seems unlikely that NP=P there are lots of interesting problems which do not have efficient algorithms (where efficient means algorithms with polynomial run time in the input size). Thus the concept of approximation algorithms was developed. A p-approximation algorithm for a minimization problem computes a solution which is at most p times greater. Additionally an approximation algorithm should run in polynomial time. Most approximation algorithms for minimization fits into this scheme. Sometimes we consider additive approximation algorithms. We say an approximation algorithm is a $+p$ algorithm if for every problem instance $A \leq Opt + p$. The runtime should be again polynomial in the input size.

In many cases we allow slightly worse runtimes instead of polynomial time runtime we often consider so-called quasi polynomial runtime $O(n^{\log(n)})$ where n denotes the size if the input. This runtime is justified by the an assumption which says the the class problems solved by algorithms with quasi polynomial run time is not equal to NP.

An important concept is inapproximability or hardness of approximation. It is used to show what approximation ratio can be achieved at best. In the simplest case one shows that it is NP-hard to compute a p-approximation for a given problem. More modern approaches introduced gap producing reductions.

Definition 2.1. ([2]) Let P_1 and P_2 be two minimization problems. A gap-preserving reduction from P_1 to P_2 with parameters (c, p) and (c', p') is a polynomial-time algorithm f. For each instance I of P_1 algorithm f produces an instance $I' = f(I)$ of P_2. The optima of I and I', say $OPT(I)$ and $OPT(I')$ respectively, satisfy the following property

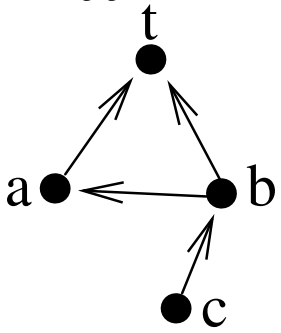
$OPT(I) \leq c \Longrightarrow OPT(I') \leq c'$
$OPT(I) > \frac{c}{p} \Longrightarrow OPT(I') > \frac{c'}{p'}$

If P_1 cannot be approximated better than p then the above reduction implies that P_2 cannot be approximated better than p'.

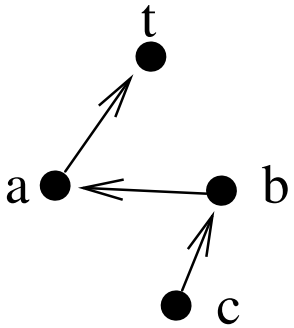
3. DIRECTED MINIMUM DEGREE SPANNING TREE

In this section we describe an approximation algorithm for the directed minimum degree spanning tree problem (DMDST).

This algorithm is described in [11]. The directed minimum spanning tree problem is to compute a directed spanning tree with minimum maximum in-degree with a fixed root t. For example consider the following graph G:



A directed minimum spanning in G is the following tree:



First some notation. Let V be the vertex set of the input graph. Let E be the edge set of the input graph. r denotes the root vertex. The degree of a point u in a spanning tree $T = (V, E')$ is denoted by $deg_T(u) = |(k, u) : (k, u) \in E'|$. $deg(T) = \max_{u \in V} deg_T(u)$ is the maximum degree of a point in T. A directed spanning tree (or branching) is a tuple $(T, r), T = (V, E(T)), r \in V$ such that each point has a directed path to r. Given a directed graph G the directed minimum spanning tree problem is to compute a directed spanning tree (T, r) with minimum $deg(T)$. It can be shown that in a minimum degree spanning tree T each point, except the root t, has a unique outgoing edge $p(u)$. This means for each $u \in V - \{t\}$ there is exactly one edge $(u, k) \in E(T)$.

Lemma 3.1. *$p(u)$ is unique in each spanning tree T and each vertex u.*

Proof. (Adapted from [16])

Consider a vertex v. We iteratively add edges from $E(T)$ to grow a tree with unique $p(u)$. Let S_i be the set of vertices reached when i edges have been added; initialize $S_0 = \{v\}$. There is always an edge in T entering S_i. We add one such edges to the new tree and add its head to S_{i+1}. This repeats until we have reached all vertices. Since a vertex is never reached twice $p(u)$ is unique for each vertex. □

We say w is the ancestor of u if it can be reached by a directed path starting at u. We say w is a descendent of u if u is the ancestor of w. Two vertices u, v are unrelated if either of them is the ancestor of the second vertex. Otherwise we say u and v are unrelated. For two unrelated vertices u and v the least common ancestor w is the vertex with minimal distance in hop-metric from u which is both the ancestor of u and v. Let C_v be the set of vertices contained in the subtree rooted at v.

Minimum degree problems have an objective function which is non-local. This means we can insert edges to an existing solution or change some subtrees without changing the value of the objective function. This makes lower bounding of the optimum difficult for this kind of problems. Most algorithms for minimum degree problems use the concept of witness sets W and blocking sets B. The basic idea is to choose W and B such that satisfying the connectivity constraints for a vertex in W implies there is a unique edge entering a vertex in B. Then the average degree of a vertex in B is a lower bound for the minimum degree. Let T^* be an directed minimum-degree spanning tree.

First we show how to lower bound the minimum degree *Opt* for a given directed graph G.

Lemma 3.2. *([11]) Let $G = (V, E)$ be a directed graph and $r \in V$. Suppose there are subset of vertices $W \subset V$ and $B \subset V$ that satisfy the following properties:*

1. Any path from a vertex $v \in W$ to r must have an incoming edge into a vertex in B,

2. For any two vertices $v, w \in W$, any path from v to r can intersect a path from w to r only after it passes through a vertex in B. In other words, G has no branching wherein the path from v to the least common ancestor of v and w does not contain a vertex of B.

Then the degree of a directed minimum degree spanning tree rooted at r of G satisfies, $Opt \geq \lceil |W| / |B| \rceil$.

Proof. Let T^* be an optimal branching rooted at r for the DMDST problem. Since it is a branching, it contains a path from any vertex to the root. By Condition 1 of the lemma, a path from a vertex $v \in W$ to

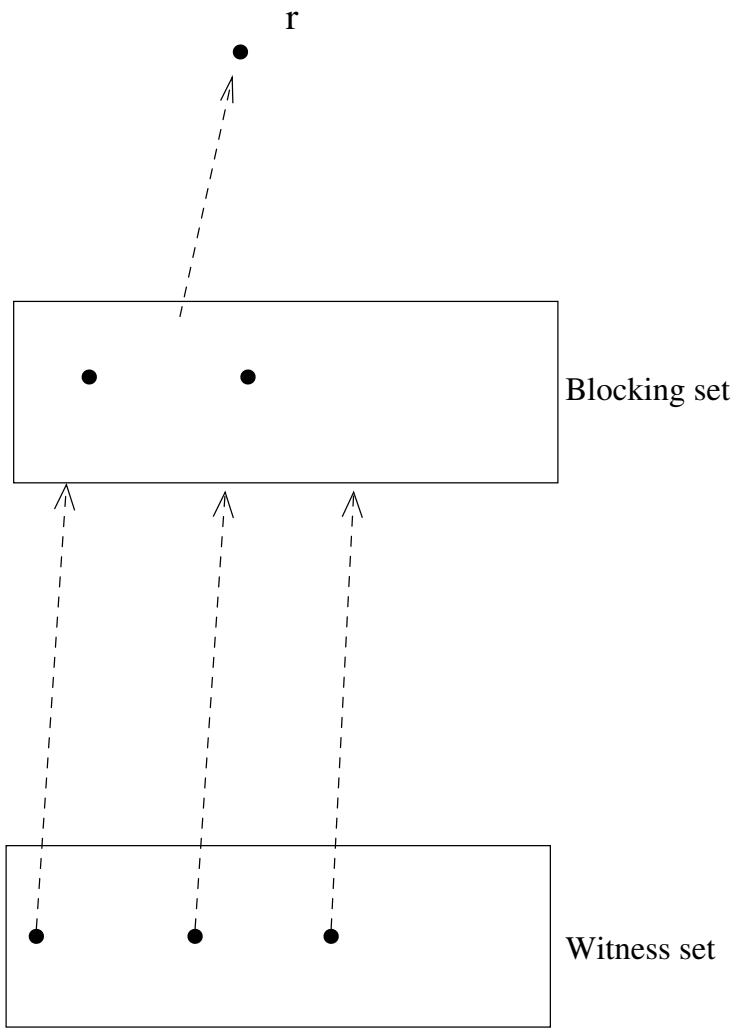


FIGURE 3.1. How to lower bound the optimal degree

r contains at least one edge into a vertex in B. Let $f_v \neq v$ be the closest ancestor of v such that $f_v \in B$. Let P_v be the path from v to f_v. By condition 2 of the lemma, the paths $\{P_v : v \in W - \{r\}\}$ are all internally disjoint. Therefore we have identified $|W|$ paths in T^*, and each of these paths has an incoming edge to some vertex in B. Therefore the average degree of a vertex in B is at least $|W| / |B|$, implying that there is at least one vertex in T^* whose degree is $\lceil |W| / |B| \rceil$ or more. □

The idea of the algorithm is to apply local improvement steps. We start with an arbitrary tree T in G. Let d be the current maximum degree. Then we check for each edge entering a high-degree vertex whether an alternative path to the root is possible, i.e one without

going through a vertex with degree $d-1$ or more. If there is such a path we connect v via this path with remaining tree.

(1) Delete (v,p) from G.
(2) Let d be the degree of p. For each vertex $u \in V$ whose in-degree in T is greater than $d-1$, delete from G edges going into u that are not in T.
(3) Run Breadth-first search from v, and test if the root r is reachable from v.
(4) If there is no path from v to r, return False after restoring all edges of G.
(5) Otherwise, BFS finds a path P from v to r. Let w be the first vertex on the path with the property that $(w,x) \in P$ and $w \in C_v$ and $x \notin C_v$.
(6) For each edge (a,b) in the subpath of P from v to x, replace the edge from $(a,p(a))$ in T by (a,b).
(7) Restore all edges of G and return True.

The improvement procedure is applied to all vertices with degree within $\log_c n$ within the current degree of T. When no improvement is possible the algorithm terminates.

(1) Find a branching T of G rooted at r. Let its degree be k. Fix some constant $c > 1$.
(2) For each edge $(v,p) \in T$, run Improvement(T,v,p) if the degree of p in T is more than $k - \log_c(n)$. If the degree of T has changed, reset k to be its new degree.
(3) Repeat the above step until Improvement(T,v,p) returns false for every edge $(v,p) \in T$ for which it is called.
(4) Return T.

The runtime of the algorithm is derived via the potential function method. The potential of a vertex of degree d is defined to be n^d. Summing over all vertices we obtain for the total potential at most $nn^k = n^{k+1}$ where k is the current degree. Next we observe the effect of an improvement step on the potential. The vertex to which we apply the improvement has degree at least $d \geq k - \log_c n$. After the improvement it has degree $d-1$ and the degree of all other vertices may increase to $d-1$. Therefore the reduction in potential is at least n^{d-2}. Since $k > k - \log_c n$ the new potential is a fraction of $n^{-\log_c n - 3}$ of the old potential. After at most $n^{\log_c n + 3}$ steps the potential is one which implies termination of the algorithm. Each step can be implemented in time $O(n^3)$.

Next they show how to find a witness a set W and a suitable blocking set B.

Lemma 3.3. *([11])Let T be a branching whose degree d or more. Let S_d be the set of vertices whose degree is d or more. There are at least $(d-1)|S_d|+1$ unrelated vertices such that the parent of each of these vertices is in S_d.*

Proof. The proof is by induction on the cardinality of S_d. If $|S_d| = 1$, then the single vertex in that set has at least d children, and the children of this vertex satisfy the lemma. If $|S_d| > 1$, remove a node $v \in S_d$ and all its descendants from T such that v has no descendents in S_d (except itself). Now the resulting branching has $|S_d| - 1$ nodes of degree d or more, and by the induction hypothesis, has at least $(d-1)(|S_d|-1)+1$ unrelated nodes that are children of S_d. Since all these nodes are unrelated to each other, at most one of these nodes is an ancestor of v. Therefore there are $(d-1)(|S_d|-1)$ nodes left that are not ancestors of v. Now we add the children of v to this set, the set increases by at least d and the number of nodes that we get is $(d-1)(|S_d|-1)+d = (d-1)|S_d|+1$ □

Next we show how to interrelate the branching output by the algorithm with a witness and blocking set.

Lemma 3.4. *([11])Let T be the branching output by our algorithm. Let its degree be k. Then for any $k - \log_c n < d \leq k$.*
$Opt \geq \frac{(d-1)|S_d|+1}{|S_{d-1}|}$

Proof. Let W be the set of vertices as in lemma 3.3 that are children of nodes in S_d, but have no descendents in S_d. We know that $|W| \geq (d-1)|S_d|+1$. Let B be S_{d-1}, the set of all vertices whose degree is at least $d-1$. For each vertex $v \in W$, the algorithm tries to find an improvement that decreases the degree of $p = p(v)$. Since it failed (the condition under which the algorithm stops), any path from v to r that doesn't use (v,p) must go through a vertex x in S_{d-1}. By construction, the internal vertices of the path from v to x is entirely contained in C_v, the descendents of v in T. Since all vertices of W are unrelated to each other, these subtrees are disjoint. Therefore, the set W and B that we have defined satisfy the conditions given in the statement of Lemma 3.2 . Therefore
$Opt \geq \lceil |W| / |B| \rceil \geq \frac{(d-1)|S_d|+1}{|S_{d-1}|}$. □

Theorem 3.5. *([11])The degree of the branching returned by our algorithm is at most $cOpt + \log_c n$, where $c > 1$ is the constant in step 1.*

Proof. Lemma 3.4 establishes a set of lower bounds on Opt for $\log_c n$ different values of d. At least for one of these values of d, $|S_{d-1}| \leq c\,|S_d|$. Using this value of d, we get $k \leq cOpt + \log_c n$. □

This concludes the proof of the approximation ratio.

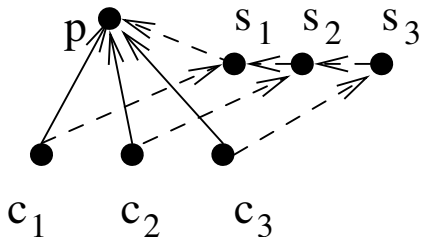
4. HARDNESS OF DIRECTED MINIMUM DEGREE STEINER TREE

The directed minimum degree steiner tree problem is a generalization of the directed minimum degree spanning tree problem. A directed minimum degree steiner tree (DMST) instance consists of a directed graph $G = (V, E)$ and a set $T \subset V$. T are the terminals. We have to compute a tree T such that each terminal is contained in T. T should have minimum maximum in-degree. If $T = V$ then we obtain the directed minimum degree spanning tree problem.

Currently, no non-trivial approximation algorithm for DMST is known. The search for an approximation algorithm is motivated by the fact that there is $+1$ approximation algorithm for the undirected variant. A natural way to start the investigation of DMST is to consider the directed minimum degree steiner tree algorithm. Interestingly, this algorithm while achieving a good ratio for the spanning tree problem totally fails for the DMST problem. In the next subsection we sketch an example in [11] for which the algorithm from the previous section finds a steiner tree with degree $n/2$ while the minimum degree steiner tree has degree 2.

4.1. **Bad example.**

The example graph H has k terminals $c_1, ..., c_k$. For each c_i there is an edge (c_i, p) and an edge (c_i, s_i). s_1 is connected with p and $s_i, i > 1$ is connected with s_{i-1}. The figure shows the example for $k = 3$. Solid edges denote edges in the current tree while dashed edges are not in the tree:



We assume the current steiner tree connects every c_i with p. The degree is then k. The important point is that the degree of p cannot be decreased via improvements. The optimal solution connects each c_i with the corresponding s_i, the resulting maximum degree is 2.

4.2. **log(n) hardness for directed minimum degree steiner tree.**

As a start we show a simple reduction from minimum dominating set to directed minimum degree steiner tree. The well known minimum dominating set problem is defined as follows:

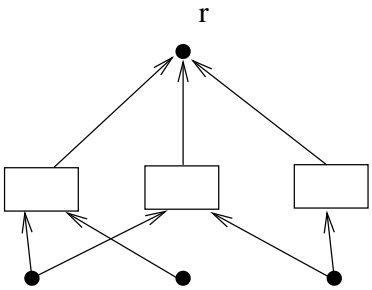
Definition 4.1. Instance:Undirected graph $G = (V, E)$.

Solution:A dominating set for S, i.e a subset $V' \subseteq V$ such for every vertex $u \in V$ either $v \in V'$ or a neighbor of u is contained in V'.

Measure:Cardinality of V'.

Next we show how to transform a minimum dominating set instance $G = (V, E)$ to a minimum degree steiner tree (H, r, T) instance. Each vertex v in S is transformed to a terminal T_s in G. Each vertex v is transformed to a non terminal v in G. For each non terminal v we insert a binary tree B_v of height $log(n)$ (see Figure below , the boxes denote the binary trees). Thus B_v contains n vertices where n is the number of vertices in G. Let $B_{v,u}$ be the vertex of B_v associated with $u \in V$. Each T_s has a directed edge $(T_s, B_{v,s})$ to all non terminals u with $s = B$ or B is a neighbor of s. Finally, all non terminals have a directed edge to the root r.

Next we calculate the gap of this reduction.Let Opt_D be the size of the minimum dominating set and Opt the degree of the corresponding directed minimum steiner tree instance:



Lemma 4.2. *$Opt_D \leq a$ implies $Opt \leq a$*

Proof. Given the minimum dominating set we connect all non terminals with r which are contained in the minimum dominating set. Since the minimum set cover is a dominating set, all vertex terminals can be connected to a non terminal respectively with the corresponding tree. The degree of all vertices except the root is at most 3. The root has degree a which is equivalent to the number of nodes connected to the root. □

Lemma 4.3. *$Opt_D \geq \ln(n)a$ implies $Opt \geq \ln(n)a$*

Proof. We show how to extract a minimum dominating set of size less than $\ln(n)a$ given that the degree of the corresponding directed minimum steiner tree instance is less than $\ln(n)a$. Let V' be the collection of non terminals connected with r. We claim that C' is a dominating set. This follows from the observation that each terminal element has a directed path to the root otherwise it would not be a valid directed

steiner tree. Since a terminals T_u can reach the the root only by entering a tree B_v with $v = u$ or $v \in N(u)$(the set of nighbors of u), it follows that V' is a valid dominating set which implies $Opt \geq \ln(n)a$ since otherwise the minimum dominating set would have cardinality at most $\ln(n)a$. □

Combining the two lemmas we obtain a gap of $(1-\epsilon)\ln(n)$ and noteing that minimum dominating set has the same hardness of approximating as minimum set cover([3]) this shows a gap of $(1-\epsilon)\ln(n)$ and proves the following theorem

Theorem 4.4. *Minimum degree steiner cannot be appproximated better than* $(1-\epsilon)\ln(n)$ *unless it holds* $NP \in DTIME(n^{\log(\log(n))})$.

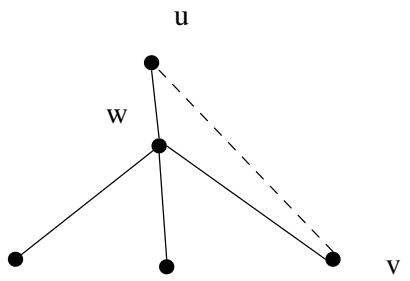


FIGURE 5.1. An improvement

5. +1 ALGORITHM FOR MINIMUM DEGREE SPANNING TREE

The algorithm described in this section is an iterative local improvement algorithm. Minimum degree spanning tree problem is in two respects a peculiar problem. First the function to optimize has a local behaviour, i.e the degree of a non-maximal vertex may increase without changing the value of the objective function. This is in contrast to problems seemingly similar problems like minimum spanning tree where every change of an edge has influence on the sum of the weights. Connected with this peculiarity arises a difficulty, namely the question how to lower bound the optimal degree, as we see later there is a lemma for this which does the job.

As noted earlier the algorithm is based on local improvements. This means we start with an arbitrary spanning tree and apply an improvement until some termination condition is satisfied.

An improvement is an introduction of an edge not in the current tree:

Definition 5.1. Let (u, v) be an edge which is not in T. Let C be the unique cycle generated when $(u, v)$is added to T. Suppose there is a vertex w of degree k in C while the degrees of vertices u and v are at most $k-2$. An improvement to T is the modification of T by adding the edge (u, v) to T and deleting one of the edges incident to w. In such an improvement, we say that w benefits from(u, v).

In the tree shown in figure 5.1 the adding of the edge (u, v) and the deletion of the edge (w, v) is an improvement. By this operation the degree of w decreases by one. If w has maximal degree the number of vertices with maximal degree decreases by one.

To lower bound the optimal degree, we show that if the current tree T has a special structure then the optimal degree is only slightly smaller than the current maximal degree:

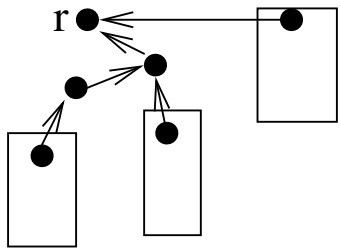


FIGURE 5.2. Illustration for proof of lemma 5.2

Lemma 5.2. *([6]) Let T be a spanning tree of degree k of a graph G. Let Opt be the degree of a minimum degree spanning tree. Let S be the set of vertices of degree k. Let B be arbitrary subset of vertices of degree $k-1$. Let $S \cup B$ be removed from the graph, breaking the tree T into a forest F. Suppose G satisfies the condition that, there are no edges between different trees in F. Then $k \leq Opt + 1$.*

Proof. As there are no edges in G connecting the different subtrees of F, the only way we can make a spanning tree is by connecting these clusters through vertices in S and B. By a simple counting argument, it is easy to show that F contains at least $|S|\,k+|B|\,(k-1)-2(|S|+|B|-1)$ subtrees. Therefore in any spanning tree of G, the average degree of vertices in $S \cup B$ is at least $k-1-(|B|-1)/(|S|+|B|)$. A vertex with maximal degree has at least average degree and hence every spanning tree has at least one vertex of degree at least $k-1$ in $S \cup B$. Therefore $Opt \geq k-1$. □

The algorithm basically just applies improvements to vertices of maximal degree until no improvement is possible. This approach has a severe problem if a new edge (u, v) is introduced which is incident with a $k-1$ vertex, i.e u or v have degree $k-1$, then we may decease the degree of a vertex w to $k-1$ but introduce a new vertex of degree k. So in this case there is no progress possible. The following notation is used:

Definition 5.3. Let T be a spanning tree of degree k. Let $p(u)$ denote the degree of a vertex $u \in T$. Let $(u, v) \notin T$ be an edge in G. Suppose w is a vertex of degree k in the cycle generated by adding (u, v) to T. If $p(u) \geq k-1$, we say that u blocks w from (u, v).

So if we want progress in this case we have to deblock all vertices blocking the improvement, i.e we have to decrease the degrees of those vertices.

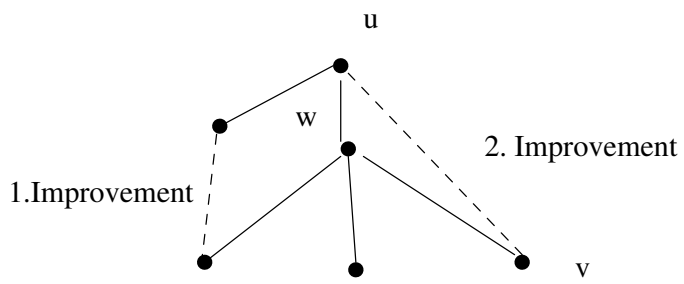


FIGURE 5.3. Improvement sequence

Let S_i be the set of vertices of degree i. The algorithm proceeds in phases. In each phase we remove all vertices in S_{k-1} and S_k. This splits T into trees. The removed vertices are marked as bad, while all other vertices are marked as good. A component consisting of good vertices is called a good component. Now we look at the edges between good components if there are no such edges we stop. In this case lemma 5.2 applies and the current degree is at most $Opt + 1$. Otherwise there is an edge (u, v) connecting two good components, we apply the improvement and merge all components having at least vertex in the cycle generated by the improvement to a single good component. If a bad vertex is contained in the cycle we mark it good. Now we distinguish two cases: If we marked a vertex of degree k good then we apply all improvements. If the bad vertex is of degree $k - 1$ we repeat. The following summarizes the algorithm:

(1) Find a spanning tree T of G
(2) Mark vertices of degree k and $k - 1$ as bad. Remove these vertices from T generating a forest. Mark all other vertices as good. Let F be the set of connected components in the forest.
(3) While there is an edge (u, v) connecting two different components of F and all vertices of degree k are marked bad do
(4) Find the bad vertices in the cycle C generated by T together with (u, v) and mark them as good.
(5) Update F by combining the components along the cycle C and these newly marked vertices into a single component. Note that more than two components of F may be combined into one in this step.
(6) If there is a vertex of degree k marked good find a sequence of improvements which propagate to w a d update T (and if necessary k) and go back to step 2.
(7) Output the final tree T, its degree k and the witness set W consisting of the vertices still marked bad.

In the example above the first improvement marks u as good. Since u is marked as good the improvement (u, v) is possible. Given a vertex w marked of degree k marked good the following procedure finds a sequence of improvements propagating to w:

(1) Apply the improvement (u, v) which marked w as good.
(2) If u or v are of degree $k-1$ apply the improvements which marked them good.
(3) Repeat Step 2 until we insert an improvement which is not incident to a vertex of degree $k-1$.

The above algorithm can be viewed as some kind of lazy evaluation since we mark many vertices as good whose corresponding improvement we never apply.

Next we analyse the algorithm. First we show that whenever we mark a bad vertex of degree $k-1$ as good the above procedure indeed finds a sequence of improvements propagating to it. We say a vertex is marked good at iteration i if its is marked good in this iteration. A vertex whose degree in T is less than $k-1$ is initially good. Let F_i be the subgraph of T containing all vertices marked good at iteration i. Note that $F_i \subset F_{i+1}$ since a good vertex is never marked bad.

Lemma 5.4. *([6]) Suppose that a vertex w is marked good in iteration i, when edge (u, v) is added. in step 4. Then w can be made nonblocking by applying improvements to the components of F_i containing u and v.*

Proof. The proof proceeds by induction on i. If w was marked good at iteration 0, it has degree less than $k-1$ in T, and is therefore nonblocking by definition. Otherwise w belongs to the cycle C found in step 4 at the i iteration. The cycle C is the simple cycle in $T \cup \{(u, v)\}$ where (u, v) is an edge between two good components X_u and X_v of F_{i-1}. Since u was marked good at an iteration $j \leq i-1$, by the induction hypothesis u can be made nonblocking by applying improvements to the component of F_j containing u, a component itself contained in X_u. Similarly, v can be made nonblocking by applying the improvements to X_v. Since X_u and X_v are disjoint, there is no interference between them. A final improvement involving the edge (u, v) suffices to reduce the degree of w, rendering it nonblocking. □

The next lemma relates k with optimal degree after termination.

Lemma 5.5. *When the algorithm stops, $k \leq Opt + 1$.*

Proof. Let S be S_k and B be the set of vertices of degree $k-1$ still marked bad. Note that the algorithm only stops when there are no edges between good components. Hence the tree T along with these

sets S and B satisfies the conditions of Lemma 5.2 and we get the desired results. □

Theorem 5.6. *([6]) The above algorithm is a $+1$ approximation algorithm for the minimum degree spanning tree problem.*

Proof. The sum of degrees of the vertices of a tree is exactly $2n - 2$. Hence the number of vertices of degree k in a tree on n vertices is $O(n/k)$. Since the size of S decreases by one in each phase. There are $O(n/k)$ when the maximal degree is k. Summing up the harmonic series corresponding to different values of k, we conclude that there are $O(n \log(n))$ phases. In each phase we try to find improvements which propagate to a vertex of S_k. Lemma 5.4 makes sure than whenever a vertex of degree k is marked as good, its degree can be decreased by one. Lemma 5.5 shows that after termination the degree is at most $Opt+1$. Each phase of the algorithm can be implemented in polynomial time. □

6. MINIMUM INTEREFERENCE SINK TREE

In this section we describe an algorithm for minimum interference sink tree in [7]. First we introduce some notation for minimum interference sink tree. 2D-max-interference operates on a set of points V in the 2D euclidean space. Therefore there is a distance function $d(u, v)$ defined for each pair of points u and v. Each $u \in V$ has a directed edge (u, v) if and only if $d(u, v) \leq 1$. The edge set consists of all such directed edges. The output of 2D-max-interference is a special tree:

Definition 6.1. Given a set of nodes V and a sink s, a sink tree is a tree spanning V with all arcs pointing towards s.

Given a set of edges the interference of a point is the number of times the point is covered by the edges:

Definition 6.2. The interference value of a single point v is defined as $I(v) = |u \mid u \neq v \wedge v \in D(u, r_u)|$ where $D(u, r_u)$ stands for the transmission circle with point u in its center and radius r_u.

The interference of an edge set is the maximum interference over all points:

Definition 6.3. The interference of a graph $G(V, E)$ is defined as $I(G) = max_{v \in V} I(v)$.

2D-maximum-interference is defined as follows:

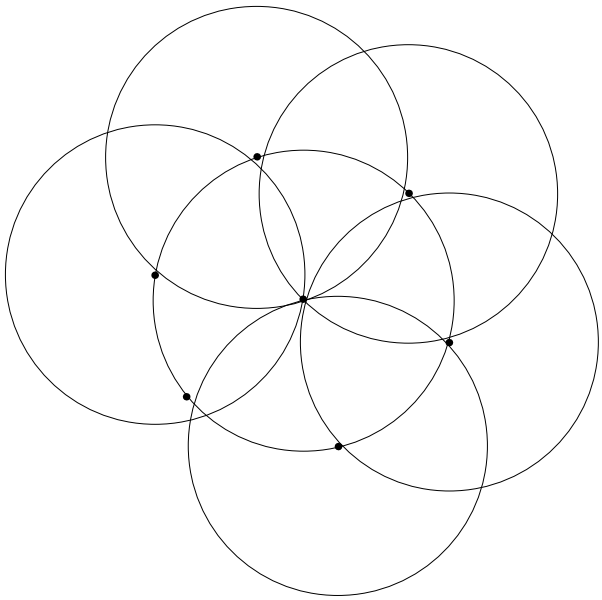


FIGURE 6.1. Number of non-intersecting circles covering a point

Definition 6.4. The minimum interference sink tree problem is defined as the problem of finding a sink tree for a given point set with minimal interference.

Before we describe an algorithm, we compare minimum interference sink tree problem with directed minimum degree spanning tree. The main difference is that an edge in MIST may cover multiple points. In MDST every edge contributes to the in-degree of at least one point.

The inherent geometry of the problem makes it possible to achieve good approximation ratios. The following lemma based on a geometric observation is used in the proof of the approximation ratio:

Lemma 6.5. *([13])Let C be a circle of radius r and let S be a set of circles of radius r such that every circle in S intersects C and no two circles in S intersect each other. Then, $|S| \leq 5$.*

Proof. Suppose $|S| \geq 6$. Let $s_i, 1 \leq i \leq 6$, denote the centers of any six circles in S. Let c denote the center of C. Denote the ray cs_i by $r_i (1 \leq i \leq 6)$. Since there are six ray emanating from c, there must at least one pair of rays r_j and r_k such that the angle between them is at most 60 degrees. Now, it can be verified that the distance between s_j and s_k is at most $2r$, which implies that circles centered at s_j and s_k intersect, contradicting our assumption. Thus $|S| \leq 5$. □

Next we present the algorithm called nearest component connector. As the name says it is an iterative algorithm connecting in every step the nearest components. Initially, every point forms a component. Each component is a MIST, i.e every component has a special node to which every other point in the same component has a directed path to. This point is called local sink. In every round every local sink connects to the nearest point not in the same component. Because of bounded transmission range of a local sink it may happen that we choose a new local sink in the component and connect the old local sink to the new one. The listing belows show the complete algorithm:

(1) $G := (V, E := \emptyset)$
(2) $lsinks := V$
(3) while $|lsinks| \geq 1$ do
(4) for all $s \in lsinks$ do
(5) $E' := \emptyset$
(6) $C :=$component containing s
(7) if s cannot reach any node outside C then
(8) $s' :=$nearest node to s(hop metric) capable of reaching a node outside C
(9) $movesink(G, s, s')$
(10) $s := s'$
(11) end if
(12) $E' := E \cup \{e\}$, where e is the arc from s to its nearest neighbor(Euclidean distance) outside C
(13) end for
(14) if $G' := (V, E \cup E')$ contains cycle then
(15) remove one of the arcs in each cycle from E'
(16) end if
(17) $G := G'$
(18) $lsinks :=$sinks in G
(19) end while
(20) $s :=$only remaining sink in $lsinks$
(21) if $s \neq s_g$ then
(22) $movesink(G, s, s_g)$
(23) end if

The algorithm to shift the local sink within a component goes as follows:

(1) $sp :=$shortest path from s_1 to s_2 according to the hop metric
(2) remove all arcs originating at nodes on sp (including s_2) from E
(3) add arcs on sp to E

Next we prove that the interference of the MIST constructed is at most $12\log(n)$. We show that there are at most $\log(n)$ rounds and the interference for every point in each round is at most 12.

Lemma 6.6. *(Adapted from*[7]*) The number of rounds is at most* $\log(n)$

Proof. We observe that in every round every local sinks establishes an arc to a point outside its component . This arc is deleted if and only if the local sink of the target component establishes an arc to the source component. In this case we have a cycle and one arc is removed. This implies that at least half of the local sinks established survive the deletion process and every such arc reduces the number of local sinks(and components) by half. It follows that after at most $\log(n)$ steps only one component remains which is our termination condition. □

The proof of constant increase of interference per node is heavily based on the geometric lemma presented earlier.

Lemma 6.7. *(Adapted from*[7]*) In each round the interference of every node is increased by at most twelve*

Proof. We note that in every round there are two possible sources of interference increase:

1 The shifting of a local sink within a component

2 The merging of components

By the first operation only points in the same component are affected otherwise the move of the local sink would not be done and each point in that component is covered at most 6 times by arcs in the shortest path otherwise we could introduce a shortcut. For the second operation the key point is that only arcs to nearest neighbors of the local sinks are established. It follows that the set of such arcs covering a point u have no pairwise intersection. Otherwise a local sink could connect to the local sink nearer than u, the lemma 6.5 guarantees that then at most 6 arcs cover any given point. □

It remains to prove that nearest component connector algorithm runs in polynomial time. To see this note that each round can be implemented in polynomial time since there is a only a logarithmic number of rounds: Finding the nearest neighbor of a local sinks can be done by sorting the points according to their distance to that local sink. Moving the sinks can be done in polynomial time by shortest path computations.

References

[1] A. Agrawal, P. Klein, and R. Ravi. How tough is the minimum-degree steiner tree? a new approximate min-max equality. Technical report, TR CS-91-49, Brown University, 1991.

[2] S. Arora and C. Lund. Hardness of approximations. *Approximation algorithms for NP-hard problems table of contents*, pages 399–446, 1996.

[3] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[4] M. Fuerer and B. Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *J. Algorithms*, 17(3):409–423, 1994.

[5] M. Furer and B. Raghavachari. An nc approximation algorithm for the minimum degree spanning tree problem. *Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing*, pages 274–281, 1990.

[6] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 317–324, 1992.

[7] M. Fussen, R. Wattenhofer, and A. Zollinger. Interference arises at the receiver. *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, 1, 2005.

[8] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 307–316, 1992.

[9] P.N. Klein, R. Krishnan, B. Raghavachari, and R. Ravi. Approximation algorithms for finding low-degree subgraphs. *Networks*, 44(3):203–215, 2004.

[10] J. Könemann and R. Ravi. A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 537–546, 1999.

[11] R. Krishnan and B. Raghavachari. The directed minimum-degree spanning tree problem. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 21, 2001.

[12] F. Kuhn, P. von Rickenbach, R. Wattenhofer, E. Welzl, and A. Zollinger. Interference in cellular networks: The minimum membership set cover problem. *Proc. of the 11 thInternational Computing and Combinatorics Conference (COCOON), Kunming, Yunnan, China, August*, 2005.

[13] MV Marathe, H. Breu, HB Hunt III, SS Ravi, and DJ Rosenkrantz. Simple heuristics for unit disk graphs. *Arxiv preprint math.CO/9409226*, 1994.

[14] R. Ravi, MV Marathe, SS Ravi, DJ Rosenkrantz, and H. Hunt. Many birds with one stone: Multi-objective approximation algorithms. *the 25 th Annual ACM Symposium on the Theory of Computing*, pages 438–447, 1993.

[15] P. von Rickenbach, S. Schmid, R. Wattenhofer, and A. Zollinger. A robust interference model for wireless ad-hoc networks. *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 239a–239a, 2005.

[16] D.B. West. *Introduction to graph theory*. Prentice Hall Upper Saddle River, NJ, 1996.