Improved Algorithms for Two Energy-Optimal Routing Problems in Ad-Hoc Wireless Networks

Samuel Baugh*, Gruia Calinescu†, David Rincon-Cruz‡, Kan Qiao§
* University of Chicago, Chicago, USA
†Department of Computer Science, Illinois Institute of Technology, Chicago, USA
‡ Knox College, USA
§ Google Inc., USA

***Abstract*—We study two problems of assigning transmission power to the nodes of ad hoc wireless networks to minimize total power consumption while satisfying certain connectivity constraints. The first problem requires to establish k node-disjoint paths from a given source to a given destination. Our new algorithm works for the most general cost model, and returns an optimal solution in time $O(n^3)$ (where n is the number of nodes), improving the running time over the previously published algorithm by a factor of k.**

The second problem assumes that the power requirement between any two nodes is symmetric, and that there are exactly two possible power levels, one of which is negligible. A source is given and all the nodes in the network must be reachable from the source (unidirectional links allowed for broadcast). We obtain a $(4 + \ln n)$-approximation algorithm, while the best published approximation ratio is $2(1 + \ln(n - 1))$. It was also known that no approximation ratio better than $O(\ln n)$ is possible unless $P = NP$.

I. Introduction

As nodes in ad-hoc wireless networks often possess limited battery life, it is beneficial to implement routing schemes that conserve transmission energy. Multi-path routing techniques yield various advantages for wireless networks, but consume more energy than their single-path counterparts. This renders minimum-energy schemes particularly important in the multi-path case. However, determining an energy-optimal scheme can be computationally expensive, especially when many disjoint paths are required. In this paper, we present an algorithm that alleviates the added expense of finding many minimum-energy disjoint paths by improving upon the time complexity of the previously best-known algorithm by a factor of k (where k is the number of node-disjoint paths desired). We then prove the correctness and asymptotic run-time of this algorithm. We assume throughout the paper that the wireless network is multi-hop and static.

Precisely, we improve upon an algorithm, introduced Srinivas and Modiano [21], to minimize energy costs in a power-constrained wireless ad-hoc network. Ad-hoc networks are those without pre-set infrastructures, so every node assumes the role of both host and router. When one wishes to send a packet from a source node to a destination node where direct transmission is not possible, a routing scheme (a sequence of nodes from the source node to the destination node) is required.

As there are generally multiple possible routes from the source to the destination node, one has the option to choose whether to transmit among one or multiple routes simultaneously. In either case, one may want to choose the path or paths that minimizes a particular cost, such as energy or bandwidth. This paper, as well as [21], concerns itself with multiple minimum-energy *disjoint* paths. The reason for considering multiple disjoint paths is that ad-hoc wireless networks are generally known to be more unreliable than wired networks. For example, physical obstructions, channel fading, and node failure can result in transmission failure and data loss. Sending the packet among multiple transmission paths significantly decreases the chance of failure, as the transmission will succeed as long as one path does not fail. This strategy is best realized when the paths are disjoint, as intersecting paths can be taken down by a single node failure.

The reason for minimizing energy is that energy conservation is particularly important in ad-hoc wireless networks. For example, the nodes of mobile, sensor, and vehicular mobile networks all have limited battery life, and often spend most of their energy in communication [21]. Naturally, multiple path routing schemes require more energy than their single path counterparts, further emphasizing the necessity to minimize energy. By using a minimum-energy multiple path routing scheme, a network can improve its chances of successful transmission while minimizing the negative effect of multiple path schemes on energy transmission and thus on network lifetime.

Denoting the number of paths desired as k, an algorithm to find a k disjoint paths routing scheme of minimum energy was introduced by Srinivas and Modiano [21]. Their primary algorithm is exact (not an approximation algorithm) and runs with a worst-case asymptotic complexity of $O(kn^3)$. This complexity is not terrible for a exact solution algorithm, especially when k is fixed or low. In fact, Srinivas and Modiano [21] additionally propose a specialized algorithms for the case $k = 2$ that reduces the general run-time (but not the asymptotic complexity). However, all things being equal, the chance of transmission failure increases with the size of the network. As such, more paths may be desired as network grows larger to counteract the increase in transmission failure.

This use case in mind, this paper introduces an algorithm that eliminates the algorithm asymptotic dependence on k by reducing the time complexity of the Srinivas and Modiano algorithm to $O(n^3)$. In the next session, we formally set up the problem borrowing the notation of Srinivas and Modiano, as well as describe their algorithm. In Section III, we describe the algorithm's procedure and prove its asymptotic bound. Our algorithm is obtain by a deeper look inside the Min-Cost Flows (as in Subchapter 8.4 of [23]) algorithm employed as black-box by [21], and relies on properties of the flows.

Besides point-to-point communication, broadcast is an important primitive that must be supported, again while minimizing total power consumption. However, this is NP-hard, and we employ approximation algorithms. A source is given and all the nodes in the network must be reachable from the source (unidirectional links allowed for broadcast). We assume that the power requirement between any two nodes is symmetric, and that there are exactly two possible power levels, one of which is negligible. Inspired by the methodology of [4] and an algorithm of [14], we obtain a $O(4 + \ln n)$-approximation algorithm, while the best published approximation ratio is $2(1 + \ln(n - 1))$ (throughout this paper, $\ln$ stands for natural logarithm). It was also known that no approximation ratio better than $O(\ln n)$ is possible unless $P = NP$. In fact, a polynomial-time algorithm with approximation ratio of $(1 - \epsilon) \ln n$ is also very unlikely to exist [12], for any $\epsilon > 0$.

II. Set-Up and Previous Work

For the purpose of energy conservation, each node can (possibly dynamically) adjust its transmitting power, based on the distance to the receiving node and the background noise. In the most common power-attenuation model [18], the signal power falls as $\frac{1}{l^\kappa}$ where l is the distance from the transmitter antenna and κ is a real *constant* dependent on the wireless environment, typically between 2 and 5. Assume that all receivers have the same power threshold for signal detection, which is typically normalized to one. With this assumption, the transmission power required to support a link between two nodes separated by a distance l is l^κ. A crucial issue is how to find a route with minimum total energy consumption for a given communication session. This problem is referred to as *Minimum-Energy Routing* in [20, 25].

In the most general cost model, the input is a simple directed graph $G = (V, E)$, and a cost function (sometimes called "power requirement" or "power (transmission) threshold") $c : E \to \mathbb{R}^+$. A *power assignment* is a function $p : V \to \mathbb{R}^+$. A *unidirectional link* from node u to node v is established under the power assignment p if $uv \in E$ and $p(u) \geq c_{u,v}$. Let $H(p)$ denote the set of all unidirectional links established between pairs of nodes in V under the power assignment p.

Min-Power k-Unicast also has as input a source $s \in V$ and a destination/sink $t \in V$, and the connectivity requirement is that the digraph $(V, H(p))$ contains k node-disjoint paths from s to t. When $k = 1$, the problem can easily be reduced to a shortest path computation; as an aside, this can be done in a more sophisticated way also if symmetric links are required as shown in [1].

Finding minimum-cost paths on a weighted graph has a well known tradition in theoretical computer science through the classic algorithms of Dijkstra and Bellman-Ford. Building on this work, Suurballe [22] presented an algorithm to find multiple paths that minimizes the paths' overall cost. A direct application of Suurballe's algorithm does not work for our model, however, because when broadcasting information to a node at a certain transmission range, all nodes within a lesser or equal transmission range also receive the signal. As such, our algorithm does not wish to find a set of paths P that minimizes $\sum_{(i,j) \in P} c_{i,j}$ (the total cost of the paths) but rather $\sum_{\text{node } i} p(i)$ where $p(i) = \max\{c_{i,j} : (i,j) \in P\}$ (the aggregate energy spent by the nodes). In other words, our algorithm wishes to account for the energy savings of transmitting simultaneously to more than one node.

Since we are looking for disjoint paths (specifically, node-disjoint paths) the only node that can take advantage of this saving is the source node. That is, each node that is not the source is a member of only one path, and as such, transmits to only one other node. The source node, on the other hand, must transmit to k nodes, and as such can take advantage of other energy saving. Srinivas and Modiano account for this observation by fixing the source's transmission power and incrementing it over the set of nodes that can be reached. With each resulting graph, they run Suurballe's algorithm to determine the minimum energy k disjoint paths for that transmission power. They then choose the minimum-energy scheme out of the set of those computed. This algorithm runs Suurballe's algorithm an order of $O(n)$ times (the number of nodes that can be possibly reached by the source), and since Suurballe's algorithm itself runs with a complexity of $O(kn^2)$ their resulting complexity is $O(kn^3)$.

A closer look reveals that the Srinivas and Modiano algorithm also has running time of $k \cdot deg(s)(m + n \log n)$, where $m = |E|$ and $deg(s)$ is the (out)degree of the source s in the input graph G. Then our algorithm has running time of $deg(s)(m + n \log n)$.

Min-Power Broadcast also has as input a source $s \in V$, and the connectivity requirement is that the digraph $(V, H(p))$ contains a path from the source s to every other vertex. The same problem was also studied in the bidirected input model (sometimes called "undirected" or "symmetric" in the literature), where the edge set of the input E is bidirected, (that is, $uv \in E$ if and only if $vu \in E$, and if weighted, the two arcs have the same cost). In some wireless settings, it is reasonable to assume an *Euclidean* input model, where $c_{u,v}$ is proportional to the Euclidean distance from the position of u to the position of v, raised to a power κ, where κ is fixed constant between 2 and 5.

A survey of Power Assignment problems is given by Santi [19]; as there we only consider centralized algorithms (there is a vast literature on distributed algorithms). The general (directed) input model is appropriate in certain scenarios (i.e, it can take into account the residual battery of the nodes [4]), while the bidirected input model is more general than the Euclidean input model and is also appropriate in some scenarios (i.e, when obstacles make communication between two nodes more power-consuming, or even impossible). From an approximation algorithm standpoint, it appears to be easier to tackle the two-dimensional Euclidean input model than the three-dimensional input model. The three-dimensional input model is easier than the bidirected input model, and the general input model appears to be harder than the bidirected input model. Even in the two-dimensional Euclidean input model, Min-Power Broadcast was proven NP-hard [9]. Only in the one-dimensional Euclidean input model, Min-Power Broadcast has polynomial-time algorithms.

In this paper, we tackle Broadcast in the bidirected input model with the further restriction that c only takes two values; that is $c : E \to \{0, 1\}$. This corresponds to the case when wireless nodes can only adjust their power on two levels, one of which is negligible. This problem is still NP-hard, and a standard reduction from Set Cover shows that no approximation ratio better than $O(\ln n)$ is possible unless $P = NP$ (using [12]). This reduction was known in 2000 and appears in several papers [3, 4, 8, 9, 15, 16]. We obtain a $4 + \ln n$-approximation algorithm, while the best previously published approximation algorithms [4, 7, 17] have a ratio of $2 \ln n$ (our own paper [6] also obtains this ratio, with faster algorithms). Actually, [4] suggests the existence of a $1.35 \ln n$ approximation algorithm (in the most general cost model) based on the ideas of [14]. A $1.5 \ln(n)$ approximation algorithm (also in the most general cost model) was claimed by [13]; however this paper has errors. These may be fixable. Note that the MIN-POWER BROADCAST WITH POWER LEVELS $\{0, 1\}$ problem is to minimize the number of nodes with assigned power 1. We assume that the input graph is connected (this can easily be checked).

III. MIN-POWER k-UNICAST

A. Description

The algorithm we present here uses the technique of incrementing the source's transmission and computing minimum-cost flows as featured in the algorithm of Srinivas and Modiano, however at each iteration our algorithm uses the results of the previous iteration along with properties of network flows to compute the next k disjoint paths in less time. As such, instead of running Suurballe's full algorithm at each iteration (which runs in $O(kn^2)$ time) our algorithm can run Dijkstra's algorithm (which runs in $O(n^2)$ time) at each iteration to improve the overall run time by a factor of k while preserving the optimality of the results. The runtime bounds of $O(kn^3)$ and $O(n^3)$ are attributed to the cost of each algorithm's sub-routine multiplied by the upper bound for the number of neighbors of the start vertex (which is n). We will start describing this algorithm with formal definitions used in in the proof of correctness of Algorithm 1 (our algorithm). We also include Algorithm 2, the Srinivas and Modiano algorithm, for comparison. Algorithm 2 looks much simpler as most computation is hidden in the (previously known) SuurballePaths procedure.

B. Formal Definitions

Let $G = (V, E)$ be a directed graph with non-negative weight function $c : E \to \mathbb{R}$. Let k be a positive integer. Let $s, t \in V$ be the *source node* and *sink node* respectively. Let k be the number of paths desired. Denote the neighbors of s as $\{s_1, \ldots, s_{deg(s)}\}$ where $i < j \Rightarrow c_{s,s_i} \leq c_{s,s_j}$ (thus assuming these nodes are sorted accordingly).

Define $G^i = (V, E^i)$, where $E^i = E \setminus \{(s, s_j) | j > i\}$. In G^i, all the capacities on arcs are 1. All the arcs of G^i have costs given by c except for the arcs leaving s, which have cost 0. Let G'_f be the residual graph of G' with respect to flow f. As notation, $c(P)$ is the sum of the costs of arcs in path P and $c(f)$ is the sum of the costs of the arcs e on which $f(e) = 1$.

C. Node Disjoint Transformation

In order to ensure that all paths are node disjoint, for every node u, we add another node to the graph, u', and the arc uu' of cost 0. We then remove all outgoing arcs from u, and direct them from u'. Thus, only one path can use u as a vertex. This is a standard well-known transformation. Moreover, remove s and t' (and all their incident arcs) from the graph, and have s' as the source of the flow and t as the sink. We still call G^i the transformed graph. An example of this transformation appears in Figure 1.

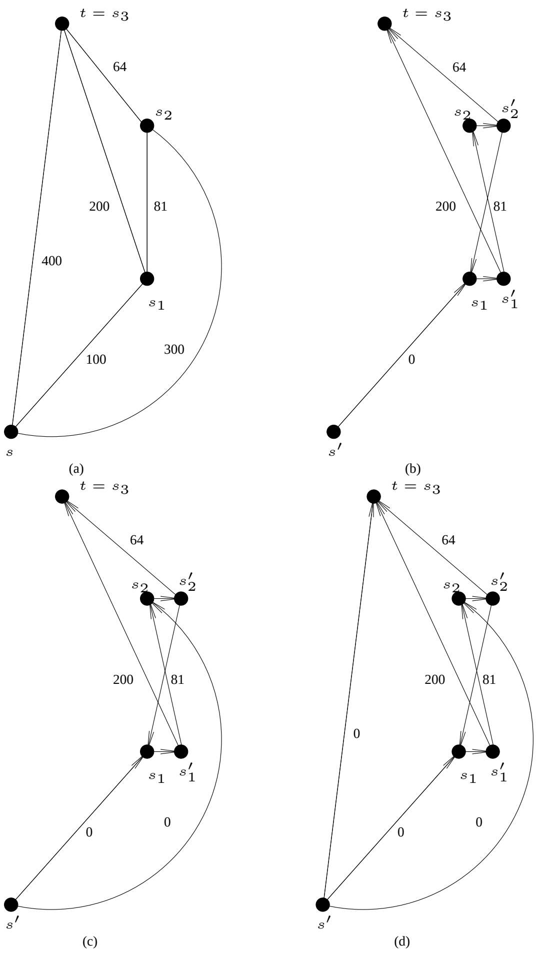

Fig. 1: (a) The input G is here bidirected, and we represent antiparallel arcs by undirected segments. Here $deg(s) = 3$. (b) G^1. (c) G^2. (d) G^3.

D. Non-Negative Arc Cost Transformation

Let $G' = (V', E')$ be a digraph with costs c' on the arcs, and without negative-cost cycles. Following Johnsons' algorithm (see Chapter 25 of [11]), define a modified cost function a' such that

$$\forall e \in E',\ a'(e) = c'(e) + d'(u) - d'(v), \tag{1}$$

where u is the tail of e, v is the head of e, and d' is the distance (in G') vector from some node $\bar{s}$ in V' that can reach all the other nodes of G' (it is known that this distance vector exists if G' does not have a negative-cost cycle). It is known that $a'(e) \geq 0$ for all $e \in E'$ and we include the argument for completeness. Let P be the path $P : \bar{s} \rightsquigarrow u$ such that $c(P) = d'(u)$, then consider $P + e$, where u is the tail of e and v is the head of e. Then $c'(P + e) = c'(e) + d'(u)$;

however, $d'(v)$ is the cost of the least cost path from $\bar{s}$ to v in G' and therefore $d'(v) \leq c'(e) + d'(u)$.

It is also known (Chapter 25 of [11]), that this is a valid transformation, specifically that the least cost path with respect to a' and c' are equivalent (see the next lemma). As such, we can use a' as a substitute for finding the least cost paths in G'. The advantage of doing this is that non-negative arc weights are necessary for the correctness of Dijkstra's Algorithm, which runs with a lower time bound than the negative cycle detecting Bellman-Ford Algorithm (with a bound of $O(n^2)$ versus a bound of $O(n^3)$).

Lemma 1 (Reformulation of Lemma 25.1 of [10])**.** *Let $G' = (V', E')$ be a digraph with costs c' on the arcs, and let $d'(v)$ be a vector (of length $|V'|$) satisfying that, for all $e \in E'$ with tail u and head v, $c'(e) + d'(u) - d'(v) \geq 0$. Let $a'(e) = c'(e) + d'(u) - d'(v)$. We can conclude that shortest paths in G' exists, and, moreover, P is a shortest path w.r.t a' if and only if P is a shortest path w.r.t c'.*

E. Auxiliary arcs

Given a directed spanning subgraph H of G, the *power* of a vertex u in H is given by $p_H(u) = \max_{uv \in E(H)} c_{u,v}$. The *power* of H is given by $p(H) = \sum_{u \in V} p_H(u)$. It is easy to check that $p(H)$ is the minimum total power that results in establishing all the arcs of H (and possibly more arcs). The algorithms below returns a set of arcs OPT; the power assignment, assuming all arcs of OPT are original (defined below), is given by $p(v) = p_{OPT}(v)$.

Let $M = n \cdot \max_{u,v \in V} c_{u,v}$. For technical reasons we add to each G^i the following *auxiliary* arcs:

1) For all u, the arc $s'u$ with capacity $2n+1$ and cost M, and
2) For all s_i, the arc $s_i s'$ with capacity $2n+1$ and cost M

We call $\hat{G}^i$ the obtained multi-di-graphs. The auxiliary arcs have such high cost that, if any feasible solution exists at all in G, any optimal solution in $\hat{G}^{deg(s)}$ has flow 0 on all the auxiliary arcs). All $\hat{G}^i$ have the same vertex set, which we call $\hat{V}$ The total number of auxiliary arcs added is $n - 1 + deg(s)$ and thus the size of $\hat{G}^i$ is at most twice the size of G^i. We call an arc of $\hat{G}^i$ that is not auxiliary an *original* arc.

F. The algorithm

We make the convention that djikstraPath(G', u, a) returns the shortest path distance vector from u to all the vertices of G', assuming a is the weight on the arcs of G' (this works correctly and in the promised runtime only if a is non-negative), and that djikstraPath(G', u, v, a) in addition to these shortest-path distances, also returns the shortest path from u to v. We make sure to only call djikstraPath when u can reach all the nodes in the graph. The procedure positiveCostTransformation(G', d') returns, for every arc of G', a new cost obtained by Equation (1).

G. Proof of Correctness

We will make use of the following classical result (see, for example, Theorem 8.11 of [23]):

Theorem 2. *f is a minimum-cost flow (of a certain value) iff G_f does not contain any negative-cost cycle.*

Algorithm 1 Improved Minimum Energy k-Disjoint Paths

```
1: procedure INPUT IS $G = (V, E)$, SOURCE-SINK PAIR $s, t \in V$, COST FUNCTION $c : E \to \mathbb{R}^+$, AND INTEGER $k > 0$.
2:     G=NodeDisjointTransformation(G)
3:     sortNeighbors(s) such that $c_{s,s_i} \leq c_{s,s_{i+1}}$ for all $i$
4:     $f_0$=flow of value $k$ on auxiliary arc $s't$ and value 0 on all the other arcs;
5:     $i = 1$; $opt = kM$; $d_0$ = array of all values $M$ except $d_0[s'] = 0$
6:     while $i \leq deg(s)$ do
7:         $a_{i-1}$ = positiveCostTransformation($\hat{G}^{i-1}_{f_{i-1}}, d_{i-1}$)
8:         $\hat{G}^i$ =addNeighbor($\hat{G}^{i-1}$) // adds the arc of cost 0 from $s'$ to $s_i$
9:         $(P, d'_i)$ =djikstraPath($\hat{G}^{i-1}_{f_{i-1}}, s_i, s', a_{i-1}$)
10:        For all $v \in \hat{V}$, set $d'_i[v] = d'_i[v] + d_{i-1}[v] - d_{i-1}[s_i]$
11:        if $c(P) < 0$ (costs are in $\hat{G}^{i-1}_{f_{i-1}}$) then
12:            $f_i$=AugmentFlow($\hat{G}^{i-1}_{f_{i-1}}, f_{i-1}, P$) (push one unit of flow on path $P$)
13:            $f_i(s's_i) = 1$
14:        else
15:            $f_i = f_{i-1}$
16:        end if
17:        $a'_i$ =positiveCostTransformation($\hat{G}^i_{f_i}, d'_i$)
18:        $d_i$ =djikstraPath($\hat{G}^i_{f_i}, s', a'_i$)
19:        For all $v \in \hat{V}$, set $d_i[v] = d_i[v] + d'_i[v] - d'_i[s']$
20:        if $c_{s,s_i} + c(f_i) \leq opt$ then
21:            $opt = c_{s,s_i} + c(f_i)$
22:            $OPT = \{uv \in G \mid f_i(u'v) = 1\}$
23:        end if
24:        $i = i + 1$
25:    end while
26:    if $opt < M$ then
27:        return($OPT$)
28:    else
29:        return ("no feasible solution").
30:    end if
31: end procedure
```

Algorithm 2 Srinivas and Modiano Algorithm

```
1: procedure INPUT IS $G = (V, E)$, SOURCE-SINK PAIR $s, t \in V$, COST FUNCTION $c : E \to \mathbb{R}^+$, AND INTEGER $k > 0$.
2:     G=NodeDisjointTransformation(G)
3:     sortNeighbors(s) such that $c_{s,s_i} \leq c_{s,s_{i+1}}$ for all $i$
4:     for $i = k, \ldots, deg(s)$ do
5:         $f_i$=SuurballePaths($G^i, s', t, k$)
6:     end for
7:     $j$ = argmin$_i(c(f_i) + c_{s,s_i})$
8:     return( $OPT = \{uv \in G \mid f_j(u'v) = 1\}$ )
9: end procedure
```

The correctness of the whole algorithm will appear later. For it, we need the main technical contribution of this section.

Theorem 3. *For all $i \geq 0$, f_i, as computed by Algorithm 1, is an integral min-cost flow of value k in $\hat{G}^i$. Moreover, all the a and a' costs used by Algorithm 1 are non-negative (so that indeed, Dijkstra's algorithm can be used to correctly compute distance vectors).*

Proof: The proof is by induction on i, and we also prove that before Line 7, d_{i-1} is the shortest-path vector from s' in $\hat{G}^{i-1}_{f_{i-1}}$, and that s' reaches every other vertex in $\hat{G}^{i-1}_{f_{i-1}}$. Also, we prove that after Line 10, d'_i is the shortest-path vector from s_i in $\hat{G}^{i-1}_{f_{i-1}}$ and that s_i reaches every other vertex in $\hat{G}^{i-1}_{f_{i-1}}$.

The base case is $i = 0$. There are no arcs leaving s' in G^0, and thus any flow of positive value must use auxiliary arcs; all these have cost M, and all the arcs of G^0 have non-negative cost. Thus any flow of value k has a cost at least kM; such a flow is indeed given by having flow k on the auxiliary arc $s't$ and 0 on all the other arcs of $\hat{G}^0$.

We also have that d_0 is the vector of shortest-paths from s' in $\hat{G}^0_{f_0}$, as indeed $\hat{G}^0_{f_0}$ contains all the arcs of $\hat{G}^0$ plus the back arc of the auxiliary arc $s't$ (which has cost $-M$). All the arcs except this back arc have non-negative costs, and all the arcs leaving s' have cost M. Thus one shortest path from s' to any vertex $u \in V \setminus \{s\}$ is the auxiliary arc $s'u$, and one shortest path from s' to any vertex u', for $u \in V \setminus \{s, t\}$, is the auxiliary arc $s'u$ followed by the 0-cost arc uu'. Note also that these arcs allow s' to reach every other vertex in $\hat{G}^0_{f_0}$.

Now we proceed to the induction step. We know, by the induction hypothesis, that we start iteration i with d_{i-1} the shortest-path vector (from s') in $\hat{G}^{i-1}_{f_{i-1}}$, where s' can reach every other vertex. Then, using the argument that follows Equation (1), we know that a_{i-1} is a non-negative cost function. Thus we can use Djikstra's algorithm in Line 9. As $a_{i-1}(e) = c(e) + d_{i-1}(u) - d_{i-1}(v)$, where u is the tail of e and v is the head of e, we have that the cost of s_iv-path P w.r.t to a_{i-1} is $a_{i-1}(P) = c(P) + d_{i-1}(s_i) - d_{i-1}(v)$; thus $c(P) = a_{i-1}(P) + d_{i-1}(v) - d_{i-1}(s_i)$. This, together with Lemma 1, shows that d'_i is correctly recomputed in Line 10, that is, d'_i is a shortest-path vector from s_i in $\hat{G}^{i-1}_{f_{i-1}}$. Moreover, s_i can reach every vertex in $\hat{G}^{i-1}_{f_{i-1}}$, due to the existence of the forward version of all the auxiliary arcs. Indeed, these forward arcs are never saturated, since each such auxiliary arc has in $\hat{G}^0_{f_0}$ a capacity of at least $2n + 1$, starts with a flow between 0 and k, and further iterations only change the flow on an arc by at most 1.

For Line 11, once a shortest path P is found w.r.t. a_{i-1}, we have that P is also a shortest path w.r.t the costs of the residual network $\hat{G}^{i-1}_{f_{i-1}}$ (based on Lemma 1). We can recompute the cost in $\hat{G}^{i-1}_{f_{i-1}}$ of P in linear time (or we can use $d'[s']$, which is this cost of P as argued above).

We claim that, by Line 16, f_i is a min-cost flow of value k in $\hat{G}^i$. Indeed, we have two cases. If $c(P) \geq 0$ in Line 11, then $\hat{G}^i_{f_{i-1}}$ does not have any negative cost cycles, since any cycle in $\hat{G}^i_{f_{i-1}}$ is either a cycle in $\hat{G}^{i-1}_{f_{i-1}}$ (and by Theorem 2 such cycles have non-negative cost, using the fact that f_{i-1} is a min-cost flow in $\hat{G}^{i-1}_{f_{i-1}}$), or it has the arc $s's_i$, which has cost 0, plus a path from s_i to s' in $\hat{G}^{i-1}_{f_{i-1}}$; this path has non-negative cost since P is a shortest path and $c(P) \geq 0$.

In the second case, $c(P) < 0$. In this case f_i is obtained from f_{i-1} by pushing one unit of flow on P in $\hat{G}^{i-1}_{f_{i-1}}$, and adding one unit of flow on the arc $s's_i$. Let C be an arbitrary cycle in $\hat{G}^i_{f_i}$. We will prove next that $c(C) \geq 0$, from which Theorem 2 allows us to conclude that f_i is indeed a min-cost flow in $\hat{G}^i$.

Indeed, consider the flow f' that has one unit of flow on C and one unit of flow on P; we ignore capacities here, while costs are as in any residual network: forward arcs have the cost in $\hat{G}^i$, and back arcs have the negation of the cost in $\hat{G}^i$. Thus f' ships one unit of flow from s_i to s'. Also, $c(f') = c(C) + c(P)$. If there are arcs of $\hat{G}^i$ such that f' uses both the forward and the back version of the arc (this can happen, as for arcs in P, the antiparallel arc exists in $\hat{G}^i_{f_i}$) modify f' to obtain f'' such that f'' does not use any of these two versions of the same arc. Then we have that f'' also ships one unit of flow from s_i to s', and $c(f'') = c(f') = c(C) + c(P)$. Moreover, f'' does not use any arc antiparallel to an arc of P.

The flow f'' is decomposed (as in Lemma 8.3 of [23]) into one path P'' from s_i to s', and a number of cycles C''_i, with one unit of flow on P'' and one each of these cycles. If f'' uses the back (non-auxiliary) arc s_is' (which exists in $\hat{G}^i_{f_i}$) then we can take P'' to consists only of this arc; this implies that the arc s_is' is not in any of the cycles C''_i (as s_is' is not in P and appears only once on C). Therefore, all the arcs of C''_i are in $\hat{G}^{i-1}_{f_{i-1}}$, as any arc other than the back (non-auxiliary) arc s_is' that is in $\hat{G}^i_{f_i}$ but not $G^{i-1}_{f_{i-1}}$ is the antiparallel arc of an arc of P, and this kind of arcs do not appear in f'', based on how f'' was constructed from f'. By Theorem 2, we have that $c(C''_i) \geq 0$.

Now, P'' is either the back (non-auxiliary) arc s_is', in which case $c(P) < 0 = c(s_is') = c(P'')$, or P'' is a path in $\hat{G}^{i-1}_{f_{i-1}}$ from s_i to s' (we argued earlier that f'' does not use any arc antiparallel to an arc of P, and thus P'' cannot have arcs of $\hat{G}^i_{f_i}$ but not $\hat{G}^{i-1}_{f_{i-1}}$, other than the back (non-auxiliary) s_is'), and we have that $c(P) \leq c(P'')$ by P being a shortest paths.

We have that $c(C) + c(P) = c(P'') + \sum_i c(C''_i)$, and using $c(P) \leq c(P'')$ (from the previous paragraph) and $c(C''_i) \geq 0$ (from one paragraph above), we obtain that $c(C) \geq 0$, as required. Thus indeed, we have that, by Line 16 of the algorithm, f_i is a min-cost flow of value k in $\hat{G}^i$.

Next we prove that a'_i as computed in Line 17 is indeed non-negative; note that this not follow directly from d'_i being a shortest-path vector since d'_i was computed in $\hat{G}^{i-1}_{f_{i-1}}$ and a'_i is on the arcs of $\hat{G}^i_{f_i}$. We do have, however, that s_i reaches every other vertex in $\hat{G}^{i-1}_{f_{i-1}}$, due to the fact s_i can reach s' and s' can reach every other vertex using the forward version of the auxiliary arcs.

We have two cases, depending on the condition in Line 11. First assume $c(P) \geq 0$ and thus $f_i = f_{i-1}$. Then $\hat{G}^i_{f_i}$ has exactly one arc not in $\hat{G}^{i-1}_{f_{i-1}}$: the forward version of the original arc $s's_i$. For arcs e other than the original arc $s's_i$, the fact that $a'_i(e) \geq 0$ follows from the argument following

Equation (1). And for the original arc $s's_i$, we use that this arc has cost 0, and $d'_i(s') \geq 0$ and $d'_{s_i} = 0$.

In the second case, $c(P) < 0$ on Line 11. Recall that one unit of flow is pushed on P and on the original arc $s's_i$. As above, it is enough to prove that $a'_i(e) \geq 0$ for the arcs e that exist in $\hat{G}^i_{f_i}$ and not in $\hat{G}^{i-1}_{f_{i-1}}$. These are the back arc of the original arc $s's_i$, and the opposite arcs for some of the arcs of P. The back arc e'' of the original arc $s's_i$ gets $a'_i(e'') = c(e'')+d'_i(s_i)-d'_i(s') = 0+0-c(P) > 0$. Now let us consider an arbitrary arc e that is the opposite of arc e' on path P. Let u be the tail of e and v be the head of e. As P is a shortest path from s_i in $\hat{G}^{i-1}_{f_{i-1}}$, we have that $d'_i(u) = d'_i(v) + c(e')$. Then $a'_i(e) = c(e) + d'_i(u) - d'_i(v) = -c(e') + d'_i(u) - d'_i(v) = 0$. Thus, also in second case, $a'_i(e) \geq 0$ for all arcs of $\hat{G}^i_{f_i}$.

With a'_i being indeed non-negative, we can use Djikstra's algorithm in Line 18. As $a'_i(e) = c(e) + d'_i(u) - d'_i(v)$, where u is the tail of e and v is the head of e, we have that the cost of $s'v$-path P w.r.t to a'_i is $a'_i(P) = c(P)+d'_i(s')-d'_i(v)$; thus $c(P) = a'_i(P) + d'_i(v) - d'_i(s')$. This, together with Lemma 1, shows that d_i is correctly recomputed in Line 19, that is, d_i is a shortest-path vector from s' in $\hat{G}^i_{f_i}$. Moreover, s' can reach every vertex in $\hat{G}^i_{f_i}$, due to the existence of the forward version of all the auxiliary arcs. Therefore the vector d_{i-1}, in Line 7 of the next iteration, satisfies the induction hypothesis. This completes the induction step. □

Theorem 3 gives indeed an $O(n^3)$ running time, as they are at most n iterations of the **while** loop, and each iteration does linear work (constructing various graphs and distances) and call Djikstra's algorithm exactly two times. The next theorem uses ideas from Srinivas and Modiano [21].

Theorem 4. *The Improved Minimum Energy k-Disjoint Paths Algorithm (detailed in Algorithm 1) returns k disjoint paths of minimum total power.*

Proof: Assume that the optimal solution (power assignment p) has $H = H(p)$ (H is a set of arcs). Make H minimal while ensuring k node-disjoint paths exists from the source s to the sink t in H. Then every node $v \neq s$ which has outgoing arcs in H has exactly one outgoing arc in H. Let j be largest such that the arc $ss_j \in H$. We obtain that $p(H) = c_{s,s_j} + \sum_{v \neq s} c_{v,\bar{v}}$, where we make the convention that $\bar{v}$ is the arc going out of v in H if such an arc exists, and $\bar{v} = v$ and $c_{v,v} = 0$ otherwise.

Note that G^j contains H (after its node-disjoint transformation) and therefore there exists a flow in G^j of value k and cost $\sum_{v \neq s} c_{v,\bar{v}}$ (recall that in G^j the costs of arcs leaving s is 0). Then $\hat{G}^j$ also has a flow of value k and cost at most $\sum_{v \neq s} c_{v,\bar{v}} < M$. From Theorem 3 we get that f_j (the variable-vector of Algorithm 1) is an integral min-cost flow in $\hat{G}^j$, and since $c(f_j) < M$ we obtain that f_j uses no auxiliary arc and therefore f_j is 1 on certain arcs of G^j.

Then one possibility for OPT (the variable in Algorithm 1) are the arcs e with $f_j(e) = 1$, including some arcs out of s (but none of cost more than c_{s,s_j}, as arcs of higher costs are not in G^j). This gives $p(OPT) \leq c(f_j) + c_{s,s_j} \leq p(H)$. Also, OPT contains k node-disjoint $s - t$ paths, as the capacity of every non-auxiliary arc uu' (for $u \neq s, t$) is 1 and therefore no two paths can use the same vertex u. In other words, an

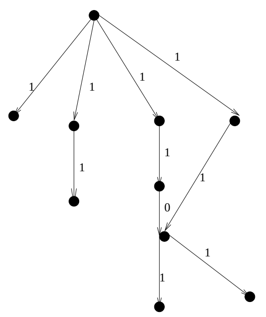

Fig. 2: A spider with four legs, weight equal to $1 + 0 + 1 + 2 + 2 = 6$, and power equal to 5.

optimal power assignment is produced.

Now assume that no feasible solution exists. Then, for all j ($0 \leq j \leq \deg(s)$), G^j does not have a feasible flow with value k, and thus Then $\hat{G}^j$ must use, in any flow with value k, at least one unit of flow on an (auxiliary) arc of cost M, and thus all $c(f_j) \geq M$. The algorithm correctly returns "no feasible solution". □

IV. Min-Power Broadcast with Power Levels $\{0, 1\}$

Phase 1 of the algorithm below resembles the algorithm of [4], and some definitions are common. The overall structure of the algorithm is from [14]. The first phase of our algorithm works in iterations. It starts iteration i with a directed graph H_{i-1} seen as a set of arcs with vertex set V (initially, H_0 contains all the arcs of cost 0). The strongly connected components of H_{i-1} which do not contain the source s and have no incoming arc are called *unhit components*. The algorithm computes a weighted structure called a *spider* (details below) attempting to achieve a reduction in the number of unhit component. The algorithm then adds the spider (seen as a set of arcs) to H_{i-1} to obtain H_i.

Definition 1. *A* spider *is a directed graph consisting of one vertex called* head *and a set of directed paths (called* legs*), each of them from the head to a (vertex called)* foot *of the spider. The* weight *of the spider S, denoted by $w(S)$, is the maximum cost of the arcs leaving the head plus the sum of costs of the legs, where the cost of a leg is the sum of the costs of its arcs without the arc leaving the head.*

See Figure 2 for an illustration of a spider and its weight. The weight of the spider S can be higher than $p(S)$ (here we assume S is a set of arcs), as the legs of the spider can share vertices, and for those vertices the sum (as opposed to the maximum) of the costs of outgoing arcs contributes to $w(S)$. From every unhit component of H_i we arbitrarily pick a vertex and we call it a *representative*. Let $r(S)$ be the number of representatives among the feet of the spider S. Call a spider *invalid* if $r(S) = 1$ and the only representative among the feet of the spider is in the same component of H_i as the head; if not invalid, a spider is *valid*.

Define $u(H)$ as the number of unhit components of digraph H (with vertex set V), and a_i as $u(H_i)$ (therefore $a_0 \leq n$).

Finding the valid spider S_i that minimizes $w(S)/r(S)$ with respect to H_{i-1} (in Phase 1) is achieved by the following

Algorithm 3 3-phase algorithm

```
Initialize $H_0 = \{uv | c_{u,v} = 0\}$.
Phase 1: In each iteration $i$, greedily pick the valid spider $S_i$ that minimizes $w(S)/r(S)$ with respect to $H_{i-1}$, until $a_i \le \frac{\sum_{j=1}^{i} w(S_j)}{\ln n} + 1$.
Phase 2: Assume after iteration $t$, let the current $|a_t|$ representatives and the source be the terminals of a (undirected) Steiner tree instance on the graph $G$. Run an edge-weighted Steiner tree algorithm to obtain set of edges $A$, and then for every edge of $A$ keep one arc in one direction, such that the source $s$ can reach all the $|a_t|$ representatives using the arcs chosen
Phase 3: Let $H$ be the union of the arcs of the spiders $S_1 \dots S_t$ and the arcs chosen in Phase 2. For every $v$, assign and output $p(v) = p_H(v)$.
```

method (a simplification of the method of described in [4]). We try all possible heads h, and both possible discrete power for the head (0 and 1). Define the *children* of the head to be the vertices within its power value - where the head is also considered a child. For each representative r_i, compute the shortest path P_i from a child of h to r_i. Let R be the list of representatives sorted such that such that the lengths of the paths P_i are in nondecreasing order. Then the best valid spider with head h and the given power value can be obtained by trying all integers j ($1 \le j \le |R|$), and taking the paths P_i leading to the first j representatives of R, but ignoring the case $j = 1$ if the corresponding spider is invalid. (A faster methods was described in [5].) We omit for lack of space the argument that this procedure indeed produces a valid spider

Lemma 5. *For every $v \in V$, H (as obtained in Phase 3 of Algorithm 3) contains a path from the source s to v.*

We omit the straightforward proof. Let OPT be an optimal solution, seen as a set of arcs that contain a path from the source s to every vertex of G, and let $opt = p(OPT)$. The following lemma is implicit in the proof of Lemma 2 of [4].

Lemma 6. *Given any graph H_i and set of representatives obtained from H_i, there is a valid spider S such that*

$$\frac{w(S)}{r(S)} \le \frac{opt}{u(H_i)}.$$

Also in [4], the *shrink factor* $sf(S)$ of a spider S is defined, and the definition implies that $sf(S) \ge r(S) - 1$. Lemma 1 of [4] states:

Lemma 7. *For a spider S (seen as a set of arcs), $u(H_i \cup S) \le u(H_i) - sf(S)$.*

Based on this we conclude:

$$a_i \le a_{i-1} - r(S_i) + 1. \tag{2}$$

Lemma 8. *Phase one stops. When Phase 1 stops at iteration t, we have $\sum_{j=1}^{t} w(S_j) \le (1+\ln n)opt$, and moreover $a_t \le opt$.*

Proof: Since we only add valid spiders, each iteration adds some arcs: those entering the unhit components of the representatives among the feet of the spider, except maybe for the unhit component that contains the head of the spider (but then, the spider is valid, so we cannot have only this except case).

In iteration i, S_i is defined to be the spider that minimizes $w(S)/r(S)$ with respect to H_{i-1}, Thus $r(S_i) \ge a_{i-1}\frac{w(S_i)}{opt}$. We have the following relations:

$$a_1 \le a_0(1 - \frac{w(S_1)}{opt}) + 1$$

$$a_2 \le a_0(1 - \frac{w(S_1)}{opt})(1 - \frac{w(S_2)}{opt}) + 1 + (1 - \frac{w(S_2)}{opt})$$

$$a_i \le a_0 \Pi_{j=1}^{i}(1 - \frac{w(S_j)}{opt}) + \sum_{j=0}^{i-1}(1 - \frac{1}{opt})^j \tag{3}$$

The second term of the RHS of this equation is strictly less than opt.

Assume we run the Phase 1 until, after q iterations, $\sum_{j=1}^{q} w(S_j) \ge (\ln n)opt$ and $\sum_{j=1}^{q-1} w(S_j) < (\ln n)opt$. In a first case, the condition $a_i \le \frac{\sum_{j=1}^{i} w(S_j)}{\ln n} + 1$ was met at iteration $t < q$. We have that $\sum_{j=1}^{t} w(S_j) < (\ln n)opt$, and we also deduce that $a_t < \frac{(\ln n)opt}{\ln n} + 1 = opt + 1$; as a_t and opt are integers, we conclude that $a_t \le opt$.

In the second case, iteration q is executed. The first term of the RHS of Equation (3) for $i = q$ is at most $a_0 e^{-\ln n} = \frac{a_0}{n} \le 1$. Combined with the second term, we obtain that $a_q < 1 + opt$; as a_q and opt are integers, we conclude that $a_q \le opt$. Moreover, $\sum_{j=1}^{q} w(S_j) \ge (\ln n)opt$ and therefore $a_q \le 1 + \frac{\sum_{j=1}^{q} w(S_j)}{\ln n}$. This means that q is the last iteration. We know that the q^{th} spider we pick has $w(S_q) \le opt$ (by Lemma 6 and $r(S_q) \le a_{q-1}$). Therefore we have $\sum_{j=1}^{q} w(S_j) < (1 + \ln n)opt$.

□

Lemma 9. *The edge-weighted Steiner tree instance from Phase 2 admits a solution T of cost at most $opt + a_t$.*

Proof: Let T' be an s-rooted outgoing arborescence inside OPT; the $p(T') \le opt$. Prune T' such that all its leafs are representatives of the unhit components of H_t; note that there are at most a_t leafs. For every node of u that is either a leaf of T' or an internal node T' that has $p_{T'}(u) = 1$, associate to u the arc vv' of T' such that $c_{v,v'} = 1$ and there is a path (that can be trivial) from v' to u in T' such that all the arcs of this path have cost 0, assuming such arc vv' exists. Then every arc of T' that has cost 1 is associated to such a vertex u. This shows that $c(T') \le p(T') + a_t$. T is the undirected version of T'. □

Theorem 10. *The algorithm have approximation ratio $\ln n + 1 + 2\lambda$, where λ is the approximation ratio of edge weighted undirected Steiner tree.*

Proof: With t denoting the last iteration of Phase 1, we have that the power of the output satisfies

$$p(H) \le \sum_{i=1}^{t} w(S_i) + c(A), \tag{4}$$

as indeed, if u is such that $p_H(v) = c_{v,u}$, then either $vu \in A$ (and uv is not put in H because of A, since A was oriented away from the roof, and therefore each edge of the undirected A is counted at most once) or vu is one arc of a spider and $c_{v,u}$ is counted in the weight of that spider.

After the first phase, Lemma 8 gives that the total spiders' weight is at most $(1 + \ln n)opt$. In the second phase, we add the arc cost of at most $\lambda * (opt + a_t) \leq 2\lambda opt$. In total, the ratio is at most $\ln n + 1 + 2\lambda$. $\square$

The current best $\lambda = \ln 4$ [2], and therefore we obtain an approximation ratio of at most $4 + \ln n$.

V. Conclusions and Future Work

Compared to previous work, we improved the running time of MIN-POWER k-UNICAST by a factor of k and the approximation ratio of MIN-POWER BROADCAST WITH POWER LEVELS $\{0, 1\}$ by a constant factor (and within a additive term of the best possible ratio).

For MIN-POWER BROADCAST WITH TWO POWER LEVELS in an Euclidean setting, a constant approximation ratio may be achievable, as it is known since [24] for the Euclidean setting when the power levels are completely adjustable.

Acknowledgment. The work of Gruia Calinescu, Samuel Baugh, and David Rincon-Cruz was supported in part by NSF grant REU Site: BigDataX: From Theory to Practice in Big Data Computing at Extreme Scales. The work of Kan Qiao was performed while at Illinois Institute of Technology.

References

[1] E. Althaus, G. Calinescu, I. Mandoiu, S. Prasad, N.Tchervenski, and A. Zelikovsky. Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks. *Wireless Networks*, 12(3):287–299, 2006.

[2] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, February 2013.

[3] Mario Cagalj, Jean-Pierre Hubaux, and Christian Enz. Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues. In *ACM Mobicom*, pages 172–182, 2002.

[4] G. Calinescu, S. Kapoor, A. Olshevsky, and A. Zelikovsky. Network lifetime and power assignment in ad-hoc wireless networks. In *Proc. 11th European Symphosium on Algorithms*, pages 114–126, 2003.

[5] Gruia Călinescu and Howard J. Karloff. Sequential dependency computation via geometric data structures. *Comput. Geom.*, 47(2):141–148, 2014.

[6] Gruia Călinescu and Kan Qiao. Minimum power broadcast: Fast variants of greedy approximations. In *11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2014, Philadelphia, PA, USA, October 28-30, 2014*, pages 479–487. IEEE, 2014.

[7] I. Caragiannis, M. Flammini, and L. Moscardelli. An exponential improvement on the mst heuristic for minimum energy broadcasting in ad hoc wireless networks. *Networking, IEEE/ACM Transactions on*, 21(4):1322–1331, Aug 2013.

[8] Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. New results for energy-efficient broadcasting in wireless networks. In Prosenjit Bose and Pat Morin, editors, *ISAAC*, volume 2518 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 2002.

[9] A. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca. On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In *18th Annual Symposium on Theoretical Aspects of Computer Science, LNCS 2010, 2001*, pages 121–131, 2001.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2. edition, 2001.

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[12] U. Feige. A threshold of $\ln n$ for approximating set cover. *JACM*, 45:634–652, 1998.

[13] S.K. Ghosh. Energy efficient broadcast in distributed ad hoc wireless networks. In *Computational Science and Engineering, 2008. CSE '08. 11th IEEE International Conference on*, pages 394–401, 2008.

[14] Sudipto Guha and Samir Khuller. Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets. *Information and Computation*, 150:57–74, 1999.

[15] S. Krumke, R. Liu, E. Lloyd, M. Marathe, R. Ramanathan, and S.S. Ravi. Topology control problems under symmetric and asymmetric power thresholds. In *Proc. Ad-Hoc Now*, pages 187–198, 2003.

[16] Weifa Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 112–122. ACM Press, 2002.

[17] Fredrick Mtenzi and Yingyu Wan. The minimum-energy broadcast problem in symmetric wireless ad hoc networks. In *Proceedings of the 5th WSEAS international conference on Applied computer science*, ACOS'06, pages 68–76, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS).

[18] T.S. Rappaport. *Wireless Communications: Principles and Practices*. Prentice Hall, 1996.

[19] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, 2005.

[20] S. Singh, C.S. Raghavendra, and J. Stepanek. Power-aware broadcasting in mobile ad hoc networks. In *Proceedings of IEEE PIMRC*, 1999.

[21] Anand Srinivas and Eytan Modiano. Finding minimum energy disjoint paths in wireless ad-hoc networks. *Wireless Networks*, 11(4):401–417, 2005.

[22] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125145, 1974.

[23] R.E. Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.

[24] P.-J. Wan, G. Călinescu, X.-Y. Li, and O. Frieder. Minimum energy broadcast routing in static ad hoc wireless networks. In *Proc. IEEE INFOCOM*, pages 1162–1171, 2001.

[25] J.E. Wieselthier, G.D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proc. IEEE INFOCOM*, pages 585–594, 2000.