Faster approximation for Symmetric Min-Power Broadcast

G. Calinescu *

Abstract

Given a directed simple graph $G = (V, E)$ and a cost function $c : E \to R_+$, the *power* of a vertex u in a directed spanning subgraph H is given by $p_H(u) = \max_{uv \in E(H)} c(uv)$, and corresponds to the energy consumption required for the wireless node u to transmit to all nodes v with $uv \in E(H)$. The *power* of H is given by $p(H) = \sum_{u \in V} p_H(u)$.

Power Assignment seeks to minimize $p(H)$ while H satisfies some connectivity constraint. In this paper, we assume E is bidirected (for every directed edge $e \in E$, the opposite edge exists and has the same cost), a "source" $y \in V$ is also given as part of the input, and H is required to contain a directed path from y to every vertex of V. This is the NP-Hard Symmetric Min-Power Broadcast problem.

In terms of approximation, it is known that one cannot obtain a ratio better than $\ln |V|$, and at least five algorithms with approximation ratio $O(\ln |V|)$ have been published from 2002 to 2007. Here we take one of them, the $2(1+\ln |V|)$-approximation of Fredrick Mtenzi and Yingyu Wan, and improve its running time from $O(|V||E|)$ to $O(|E| \log^2 |V|)$, by careful bookkeeping and by using a previously-known geometry-based data structure.

1 Introduction

We study the problem of assigning transmission power to the nodes of ad hoc wireless networks to minimize power consumption while ensuring that the given source reaches all the nodes in the network (unidirectional links allowed for broadcast), in the symmetric cost model. This problem takes as input a directed simple graph $G = (V, E)$ and a cost function $c : E \to R_+$. The *power* of a vertex u in a directed spanning simple subgraph H of G is given by $p_H(u) = \max_{uv \in E(H)} c(uv)$, and corresponds to the energy consumption required for the wireless node u to transmit to all nodes v with $uv \in E(H)$. The *power* (or *total power*) of H is given by $p(H) = \sum_{u \in V} p_H(u)$. A "source" (called "root" sometimes in the literature) $y \in V$ is also given as part of the input, and H is required to contain a directed path from y to every vertex of V; we call the problem of minimizing the total power while ensuring this connectivity Symmetric Min-Power Broadcast. Among early work on this problem we mention [22, 24, 11, 23].

This problem is motivated by minimizing energy consumption in a static multi-hop wireless network, where $c(u, v)$ represents the transmission power wireless node u must spend to ensure a packet is received by node v. Our model is that wireless nodes have several levels of transmission power. A packet sent by u with power p is received by all nodes v with $c(u, v) \leq p$. This feature is useful for energy-efficient multicast and broadcast communications.

In some wireless settings, it is reasonable to assume that u and v are embedded in the two-dimensional Euclidean plane, and $c(u, v)$ is proportional to the 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. This is the Euclidean input model.

We do not work in the Euclidean input model, but make a (less-restrictive) "symmetric" assumption that E is bidirected, (that is, $uv \in E$ if and only if $vu \in E$, and the two edges have the same cost).

A survey of Power Assignment problems is given by Santi [20]; like there we only consider centralized algorithms (there is a vast literature on distributed algorithms). Even in the Euclidean input model, Min-Power Broadcast was proven NP-Hard [11], and it was a folklore result in 2000 that Symmetric Min-Power Broadcast is as hard to approximate as Set Cover (this appears in several papers [11, 17, 23, 7, 2, 16]). Based on Feige's hardness result for Set Cover [12], no approximation ratio better than $O(\ln n)$ is possible unless $P = NP$. Here $n = |V|$, and from now on $m = |E|$.

The first $O(\log n)$ approximation algorithm was given by Caragiannis et al. [7] (journal version: [8]). A similar algorithm was presented in [3], and the simplest and best variant (ratio of $2(1 + \ln n)$) of this algorithm was presented by [18] and achieves a $O(mn)$ running time (their analysis claims $O(mn\alpha(mn))$ running time, but one observation can get rid of the inverse Ackermann function α in the analysis). Later, [9] obtains another $O(\log n)$ approximation algorithm, with a complicated algorithms based on [14], that needs multiple calls to Minimum Weight Perfect Matching. Two other algorithms also achieve a $2(1 + \ln n)$-approximation ratio: the Spider algorithm of [4] (which has the same ratio if the input graph is not bidirected) and the Relative

*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA. calinescu@iit.edu. Research supported in part by NSF grant NeTS-0916743.

Greedy algorithm of [6] (which also achieves the best known approximation ratio, of 4.2, in the Euclidean input model).

All these algorithms use greedy methods, mostly adopted from the Steiner Tree problems and its variants (precisely, [15, 26, 14]). Most of these papers do not explicitly analyze the running time of the presented algorithms (and none, as given, is faster than $O(mn)$). We set to achieve the same $2(1 + \ln n)$-approximation ratio with an improved running time.

For this, we give a faster variant of the "Hypergraph-Greedy" algorithm of Mtenzi and Wan [18]. It turns out this algorithm is a special case of the greedy method for Polymatroid Cover of [25] (a simpler analysis in [13]), and we use this observation to give an alternative proof of its approximation ratio. For some readers, the direct proof of [18] may be more enlightening; we just point out in this paper that the proof is a special case of the [13] proof. We achieve a running time of $O(m \log^2 n)$ by careful book-keeping and by using a data structure of [5].

The next section presents the Hypergraph-Greedy algorithm of [18] with our notation, and gives an alternative, shorter proof of its approximation ratio, based on [25] and [7]. Then, in Section 3, we describe how to use a known data structure. Section 4 combines several data structures with careful book-keeping and analysis to obtain the improved running time.

2 Algorithm Description and Approximation Ratio

Given a directed edge uv, its *undirected version* is the undirected edge with endpoints u and v; for a set of directed edges F, we denote by $\widehat{F}$ the multiset of edges that are the undirected version of the edges of F.

For $u \in V$ and $r \in \{c(uv) \mid uv \in E\}$, let $S(u, r)$ be the *directed star* (or, simply, *star*) consisting of all the arcs uv with $c(uv) \leq r$. We call u the center of S and note that r is the power of $S(u, r)$. For a directed star S, let $p(S)$ denote its power, let $E(S)$ be its set of arcs, and define $V(S)$, its set of vertices, to be its center plus the heads of its arcs. See Figure 1 for an example. The algorithm treats $V(S)$ as a hyperedge in a hypergraph with vertex set V.

The algorithms described use all the possible stars, and there are $O(m)$ of them (for each vertex, the number of stars is its degree in the input graph). In the first phase, the algorithm keeps a set of stars (initially empty), giving a set of arcs H. It then selects the next star such that to maximize the decrease in the number of weakly connected components in (V, H) divided by the power of the star (see Figure 2). The first phase stops when (V, H) is weakly connected. At this moment, the second phase of the algorithm constructs a spanning tree in the undirected version of (V, H) (for example,

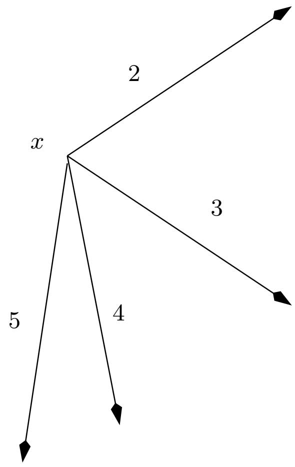

Figure 1: A star with center x and four arcs, of power $\max\{2, 3, 4, 5\} = 5$.

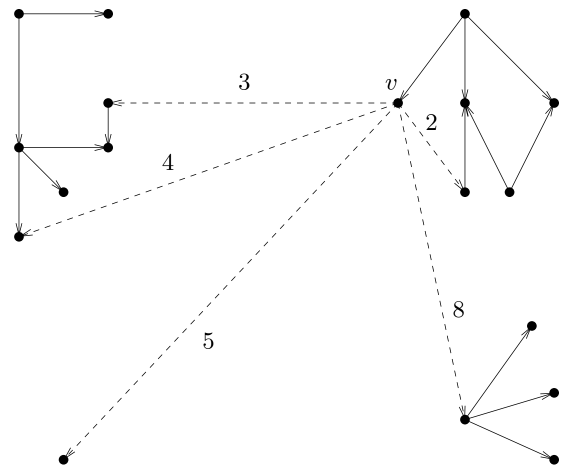

Figure 2: The current H is drawn as solid segments, with arrows indicating the direction of the edges. The directed edges with tail v are drawn as dashed segments. The star with center v and power 2 has one edge, and it does not decreases the number of weakly connected components of H. The star with center v and power 3, with two edges, decreases the number of weakly connected components of H by 1. The star $S(v, 4)$ has three edges and also achieves a reduction of 1. $S(v, 5)$ achieves a reduction of 2, and $S(v, 8)$ achieves a reduction of 3. Among the stars with center v, the algorithm would choose $S(v, 5)$ as the next star.

by breadth-first search or depth-first search), and it re-orients if needed the edges of this tree to lead away from y (the vertex given in the input as the source), and thus obtains a feasible output. This re-orientation, first applied in [7], only works if the input graph is bidirected.

2.1 Approximation Ratio

We first cast the problem in a different setting. A polymatroid $f : 2^N \to \mathbb{Z}^+$ on a ground set N is a non-decreasing (monotone) integer-valued submodular function. A function f is monotone iff $f(A) \leq f(B)$ for all $A \subseteq B$. A function f is submodular iff $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq N$. Polymatroids generalize matroids which have the additional condition that $f(\{i\}) \leq 1$ for all $i \in N$. We call a subset $A \subseteq N$ *spanning* iff $f(A) = f(N)$.

Assume each element $j \in N$ has a weight w_j. Define $f_A(B) = f(A \cup B) - f(A)$ and $t = \max_{i \in N} f(\{i\})$. The greedy algorithm of Wolsey [25] find a H_t-approximation to the minimum weight polymatroid spanning set, where H_t is the t^{th} harmonic number, $\sum_{i=1}^{t}(1/i)$, which is known to be at most $1 + \ln t$. This algorithm, a generalization of Chvatal's algorithm [10] for Set Cover, starts with $B = \emptyset$, and as long as $f(B) < f(N)$, adds to B the element $j \in N$ that maximizes $f_B(\{j\})/w_j$.

In our setting, define N to be the set of all stars, and for a set B of stars, define $f(B)$ to be the size of a maximal forest in $\bigcup_{S \in B} \widehat{E(S)}$. We do have a polymatroid: as explained in [21], Example 44.1(a), the rank function of a matroid (in our case, the graphic matroid, where a set of edges of an undirected graph is independent iff it is a forest) produces a polymatroid. Note that $f(N) = n - 1$ and a spanning set in the polymatroid corresponds to a set of stars whose arcs form a spanning weakly connected subgraph of G. Note also that, if for a set of stars B, we let $co(B)$ be the number of connected components of $(V, \bigcup_{S \in B} \widehat{E(S)})$, we have $f(B) = |V| - co(B)$. Then $f_B(S) = f(\{S\} \cup B) - f(B) = |V| - co(B \cup \{S\}) - (|V| - co(B)) = co(B) - co(B \cup \{S\})$, which is the decrease in the number of weakly connected components in (V, H) when H, given by $\bigcup_{S' \in B} E(S')$, is replaced by $\bigcup_{S' \in B} E(S') \cup E(S)$. With the weight of a star defined to be its power, the algorithm of [18] is the greedy algorithm for polymatroids.

Let OPT be an optimum solution of the instance at hand. Without increasing total power or decreasing connectivity, add, if needed, to OPT every arc vu with $c(vu) \leq p_{OPT}(v)$. For each $v \in V$, call the star $S = S(v, p_{OPT}(v))$ a star of OPT. Since OPT contains a path from the source y to every other vertex of G, we have that the stars of OPT form a spanning set in the polymatroid above. Thus, using [25], the collection of stars $\mathcal{A}$ selected by the first phase of the algorithm satisfies:

$$\sum_{S \in \mathcal{A}} p(S) \leq (1 + \ln n) \sum_{S \text{ star of } OPT} p(S) \leq (1 + \ln n)\, opt, \qquad (1)$$

where $opt = p(OPT)$.

Let H be obtained by keeping an arbitrary subtree of $\bigcup_{S' \in \mathcal{A}} \widehat{E(S')}$ and orienting the edges away from the source y. For vertex $u \in V$, we denote by u' its parent in this outgoing arborescence. Also, we denote by $\tilde{c}(S)$ the center of star S. Now, using the argument of [8], we have:

$$\begin{aligned} p(H) &= \sum_{u \in V} p_H(u) \\ &\leq \sum_{u \in V} \left(\sum_{S \in \mathcal{A} | u = \tilde{c}(S)} p(S) + \sum_{S \in \mathcal{A} | u = \tilde{c}(S)'} p(S) \right) \\ &\leq 2 \sum_{S \in \mathcal{A}} p(S), \end{aligned}$$

where we use that a star $S \in \mathcal{A}$ appears at most twice in the middle summation: once for the center of S, and once for the parent in H of the center of S. Combined with Inequality 1, we obtain the desired approximation ratio.

3 The data structure used

A *Rel-Max* data structure stores a list of items i, sorted by their *cost* c_i (non-decreasing). A query is finding the j maximizing j/c_j. The update consists of, given i, remove the i^{th} item from the list (this changes the position of the items k, for $k > i$). Calinescu and Qiao [5] present an implementation for a generalization of *Rel-Max* queries/updates In their data structure, each item also has a "coverage" f_i, non-decreasing in i, and one must find the j maximizing f_j/c_j, while the update consist of re-setting, for given i and $delta > 0$, for all $k \geq i$, $f_k = f_k - \delta$. Their approach is based on keeping upper convex hulls. See Figure 3 for some intuition.

It seems they re-invented some ideas from [19], also concerned with keeping convex hulls, under different update operations; [1] being a more recent work on this topic. [5] obtains an initialization/preprocessing time of $O(l \log^2 l)$, a query time of $O(\log l)$, and an update time of $O(\log^2 l)$, where l is the number of items in the initial list.

4 Book-keeping

One needs to find the next star at most n times, and the main challenge is to compute the star that maximizes the decrease in the number of weakly connected components in (V, H) divided by the power of the star. The method of [18] is to try all $O(m)$ stars, and with careful bookkeeping one gets a $O(mn)$-time algorithm. Our goal is $O(m \log^2 n)$.

For every u, let $v_1^u, \ldots, v_{d(u)}^u$ be the neighbors of u in G, sorted in non-decreasing order by $c(uv_i^u)$. Let $S_j(u)$ be the star with center u and power $c(uv_j^u)$.

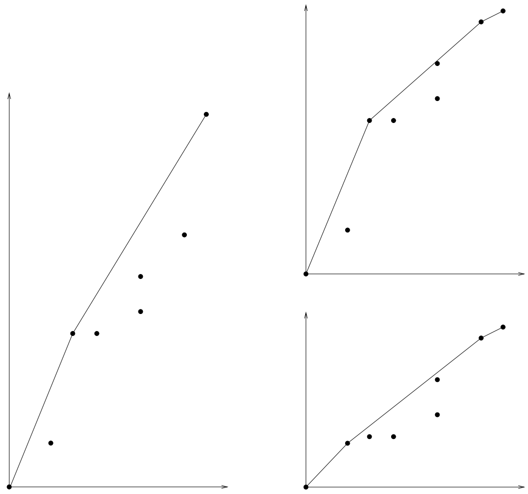

Figure 3: Points P_i have coordinates (c_i, f_i). On the left, an example of points P_i, with the upper convex hull drawn. Add for convenience P_0 with coordinates $(0, 0)$. The answer to the query is the neighbor of P_0 on the upper hull. On the right, top, the points P_i after f_7 is updated (decreased), with the upper convex hull drawn. On the right, bottom, the points P_i after an update with $i = 2$, causing the second coordinate to drop for points $P_2 \dots P_7$. The upper convex hull is also drawn.

We keep the following three data structures. Let $Q_1, \dots, Q_l$ be the vertex sets of the current weakly connected components of (V, H). We keep the components by having an explicit representative vertex in each, that is, an array *comp*$[v]$ stores the representative of the component containing vertex v. We keep l binary search trees B_i (for $i \in \{1, 2, \dots, l\}$), one for each component. The tree B_i keeps, sorted by ID, the nodes of Q_i together with the nodes u such that an edge $uv \in E$ exists with $v \in Q_i$; in this case we also store the smallest j such that $v^u_j \in Q_i$. For each component with vertex set Q_i, we explicitly keep $|Q_i|$ and a linked list of its vertices.

For each u, we keep a list L_u of items j, each corresponding to the edge uv^u_j and of value $c_j = c(uv^u_j)$, sorted in non-decreasing order by c_j. We only keep the item j if there exists a Q_i with $u \notin Q_i$ and j is smallest among those k with $v^u_k \in Q_i$. As an example, in Figure 1, we only keep items 2, 4, and 5 corresponding to the edges of cost 3,, 5, and 8. Then it is easy to check that the star $S_j(u)$ has endpoints in exactly $l + 1$ sets Q_i, where j is the l^{th} item in L_u. Moreover, among the stars with center u and endpoints in exactly $l + 1$ sets Q_i, one with minimum power is $S_j(u)$, where j is the l^{th} item in L_u. Notice also that in this situation, l is the decrease in the number of weakly components if $E(S_j(u))$ is added to H. We also keep, for each u, the value $z_u = \min_{j \in L_u} l_j / c_j$, where l_j is the position of item j in L_u, and the item j_u that achieves this minimum.

We also keep a binary max-heap with all $u \in V$ having as key the value z_u. With these data structures, we can find the star that maximizes the decrease in the number of weakly components of H, divided by the power of the star, if we pick an u with maximum z_u and then use $S_{j_u}(u)$. Finding $S_{j_u}(u)$ is then done in constant time.

Now we describe how the data structures are maintained when some $S_{j_u}(u)$ is added to the set of selected stars. Let l_u be such that j_u is the l_u^{th} item in L_u, and let $j_1, \dots, j_{l_u}$, be such that item j_i is the i^{th} item in L_u. Let Q_{k_0} be the vertex set of the component of u, and Q_{k_i} be the vertex set of the component of v_{j_i}. The way we keep L_u implies that these components are distinct.

The effect of adding $S_j(u)$ to H is the merging into one of the components $Q_{k_0}, Q_{k_1}, \dots, Q_{k_l}$. The algorithm will make these merges one by one, first Q_{k_0} with Q_{k_1}, then the result with Q_{k_2} (if $l \geq 2$), and so on.

Consider such a merge between Q_r and Q_s, and assume by symmetry that $|Q_r| \leq |Q_s|$. We merge Q_r into Q_s; that is Q_s will be the resulting component. First, for each vertex in Q_r, we add it to the list of vertices of Q_s and change its representative to the representative of Q_s. The running time is $O(n \log n)$ over all the merges, since if we spend time on vertex v, v will become part of a component that has at least twice as many vertices as the component of v before the merge.

Second, we traverse (inorder) the binary tree B_r, and for each v in the tree we proceed as described in the four cases below. In Case 1, $v \notin B_s$; then we insert v in B_s together with the j-index (if any) it has in B_r. In Case 2, $v \in B_s$ and $v \in Q_r$; then the v from B_s also has an index j such that $w^v_j \in Q_s$ and such that j is the only item in L_v among those k with $w^v_k \in Q_s$. We update B_s to mark that $v \in Q_s$. We also remove item j from L_v, updating if necessary, z_v, l_v, and the binary max-heap which keeps vertices u with keys z_u. Case 3 is when $v \in Q_s$ (and thus $v \in B_s$), and $v \notin Q_r$ (recall that $v \in B_r$); in this case, the v in B_r also has an index j such that $w^v_j \in Q_r$ and such that j is the only item in L_v among those k with $w^v_k \in Q_r$. We also remove item j from L_v, updating if necessary, z_v, l_v, and the binary max-heap which keeps vertices u with keys z_u. We update B_s to mark that $v \in Q_s$. Case 4 is when $v \in B_s$, but $v \notin (Q_r \cup Q_s)$; then we have two indices j (from the v in B_r) and j' (from the v in B_s), such that $w^v_j \in Q_s$, and $w^v_{j'} \in Q_r$, and such j is the only item in L_v among those k with $w^v_k \in Q_s$, and such that j' is the only item in L_v among those k with $w^v_k \in Q_r$. We will keep the smaller of j, j' for the v in B_s, and remove the larger of j, j' from L_v, updating if necessary, z_v, l_v, and the binary max-heap which keeps vertices u with keys z_u.

To analyze the overall time of this updates, consider this: each directed edge uv_j^u appears in L_u initially, but will only be removed once, with a time of $O(\ln^2 n)$. This time is also enough for updating z_u after this removal, and updating the position of u in the max-heap after z_u changes.

When we merge B_r in B_s above, other than removals from L_v's, we spend $O(\log n)$ per element of B_r, to find and if necessary insert it in B_s. Say we process a $v \in B_r$. If v appears in B_r without a j, or in other words, $v \in Q_r$, then we charge this $O(\log n)$ to v. Vertex v can be charged at most $\lg n$ times this way, as each time it belongs to a component with at least twice as many vertices. If v appears in B_r with a j, then we are in the following case: there a vertex $w_j^v \in Q_r$, the head of a directed edge vw_j^v. We charge the time spent to directed edge vw_j^v. Notice that w_j^v will belong to a component twice the size, and thus edge vw_j^v can be charged at most $\lg n$ times. Each charge is $O(\lg n)$, thus we spend $O(\lg^2 n)$ per vertex and per directed edge of v.

In conclusion, the running time of the Hypergraph-Greedy algorithm, implemented with these data structures is $O(m \log^2 n)$.

References

[1] G.-S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. IEEE FOCS*, pages 617–626, 2002.

[2] M. Cagalj, J.-P. Hubaux, and C. Enz. Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues. In *Proc. ACM Mobicom*, pages 172–182, 2002.

[3] M. Cagalj, J.-P. Hubaux, and C. Enz. Energy-efficient broadcasting in all-wireless networks. *Wirel. Netw.*, 11(1-2):177–188, 2005.

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

[5] G. Calinescu and K. Qiao. Asymmetric topology control: Exact solutions and fast approximations. In *Proc. IEEE INFOCOM*, pages 783–791, 2012.

[6] I. Caragiannis, M. Flammini, and L. Moscardelli. An Exponential Improvement on the MST Heuristic for Minimum Energy Broadcasting in Ad Hoc Wireless Networks. In *Proc. ICALP*, pages 447–458, 2007.

[7] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. New results for energy-efficient broadcasting in wireless networks. In *Proc. ISAAC*, pages 332–343, 2002.

[8] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. A logarithmic approximation algorithm for the minimum energy consumption broadcast subgraph problem. *Inf. Process. Lett.*, 86(3):149–154, 2003.

[9] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Energy-efficient wireless network design. *Theor. Comp. Sys.*, 39(5):593–617, 2006.

[10] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operation Research*, 4:233–235, 1979.

[11] A. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca. On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In *Proc. STACS*, pages 121–131, 2001.

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

[13] G. Baudis and C. Gropl and S. Hougardy and T Nierhoff and H. J. Prömel. Approximating minimum spanning sets in hypergraphs and polymatroids. In *Proc. ICALP*, 2000.

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

[15] P. Klein and R.Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19:104–115, 1995.

[16] 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.

[17] W. Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *Proc. ACM MOBIHOC*, pages 112–122. ACM Press, 2002.

[18] F. Mtenzi and Y. Wan. The minimum-energy broadcast problem in symmetric wireless ad hoc networks. In *Proc. WSEAS ACOS*, pages 68–76, 2006.

[19] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981.

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

[21] A. Schrijver. *Combinatorial Optimization.* Springer, 2003.

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

[23] P.-J. Wan, G. Calinescu, X.-Y. Li, and O. Frieder. Minimum Energy Broadcast Routing in Static Ad Hoc Wireless Networks. *Wireless Networks*, 8(6):607–617, 2002.

[24] 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.

[25] L.A. Wolsey. Analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2:385–392, 1982.

[26] A. Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. Technical Report CS-96-06, Department of Computer Science, University of Virginia, 1996.