Minimum Power Broadcast: fast variants of greedy approximations

***Abstract*—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 most general cost model, the best published approximation ratio is achieved by the "greedy spider" algorithm (Calinescu et al., ESA 2003). We present a variant of this algorithm with running time $O(n^3)$ and the same approximation ratio of $2(1 + \ln n)$, where n is the number of nodes.**

In the restricted "Euclidean" two-dimensional cost model, where the power requirement to transmit from node u to node v is $c_{uv} = \|uv\|^\kappa$ (here $\|uv\|$ denotes the Euclidean distance between the positions of u and v, and κ is a real *constant* dependent on the wireless environment, typically between 2 and 5), the best known approximation ratio is achieved by the "relative greedy" algorithm (Caragiannis et al., ICALP 2007; journal version [8]). We present a variant of this algorithm with running time $O(nm)$ and the same approximation ratio of 4.2 in the Euclidean model, where m is the number of edges in the input graph.

The new variants make use of advanced data structures and/or simple amortized analysis, improving naive variants by a factor of $\Theta(n)$. This improvement allows us to apply these algorithms to large instances (1000-2000 nodes). Our experimental results show that the best output achievable within 100 seconds improves the solution based on minimum spanning tree by an average of up to 15%, and comes within 25% of optimum, in average, on the instances where we can compute the optimum (based on an integer program). The improvement is circa 50% larger compared to what one would get applying existing fast heuristics.

I. Introduction

Ad hoc wireless networks have received significant attention in recent years due to their potential applications in battlefield, emergency disaster relief, and other application scenarios (see, e.g., [12], [38], [42]). Unlike wired networks or cellular networks, no wired backbone infrastructure is installed in ad hoc wireless networks. A communication session is achieved either through single-hop transmission if the recipient is within the transmission range of the source node, or by relaying through intermediate nodes otherwise. We assume that omnidirectional antennas are used by all nodes to transmit and receive signals. Thus, a transmission made by a node can be received by all nodes within its transmission range. This feature is extremely useful for energy-efficient multicast and broadcast communications.

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 [35], 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 [38], [42]. We assume throughout that the wireless network is multi-hop and static.

In this paper, we make an experimental study of the problem of assigning transmission power to the nodes of ad hoc wireless networks to minimize total power consumption during a broadcast session (unidirectional links allowed). Precisely, MIN-POWER BROADCAST has as input a simple directed graph $G = (V, E)$, a vertex $z \in V$ (the source), and a cost function (sometimes called "power requirement") $c : E \to \mathbb{R}^+$. 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 $p(u) \geq c_{uv}$. Let $B(p)$ denote the set of all unidirectional links established between pairs of nodes in V under the power assignment p. MIN-POWER BROADCAST asks for minimizing $\sum_{v \in V} p(v)$ subject to p being *valid*, that is, satisfying the constraint that the directed graph $(V, B(p))$ has a path from the source z 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 bidirected, (that is, $uv \in E$ if and only if $vu \in E$, and if weighted, the two edges have the same cost). In some wireless settings, it is reasonable to assume an *Euclidean* input model, where c_{uv} 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 [36]; 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 [6]), 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 algorithm standpoint, it is easiest to tackle the two-dimensional Euclidean input model, then the three-dimensional input model, followed by the bidirected input model, and the general input model is the hardest. Even in the two-dimensional Euclidean

input model, Min-Power Broadcast was proven NP-Hard [12]. Only the one-dimensional Euclidean input has polynomial-time algorithms.

A fair number of approximation algorithms and heuristics have been proposed and will be discussed in the next subsections. The simplest (and fastest) is the minimum spanning tree (MST) algorithm, and the most sophisticated are the "greedy spider" [6] algorithm[1]and the "relative greedy" [8] algorithm. These two approximation algorithms are however also the slowest, with neither [6] nor [8] making an explicit running time analysis. We have improved their running time, while keeping the approximation ratio.

Precisely, in Section III, we present a variant of the Relative-Greedy algorithm with running time $O(nm)$ and the same approximation ratio of 4.2 in the two-dimensional Euclidean input model, where n is the number of nodes and m is the number of edges in the input graph (m could be as high as $n(n-1)/2$). The space complexity of this algorithm is $O(m+n)$. In Section IV we present a variant of the Greedy-Spider algorithm with running time $O(n^3)$ and the same approximation ratio of $2(1+\ln n)$ in the most general model. The space complexity of this algorithm is $O(mn)$, compared to $O(n^2)$ for the naive variants ($O(m+n)$-space also seems possible with $O(nm(m+n\log n))$ running time).

The new variants make use of existing advanced data structures and/or simple amortized analysis, improving naive variants by a factor of $\Theta(n)$. This improvement allows us to apply these algorithms to large instances (2000 nodes, compared to 210 in [8]). We then conducted extensive experiments on random input data in the two-dimensional Euclidean input model, using these two algorithms and many of the other existing ones (we only excluded ideas that have never been tried on large instances and for which the running time appears to be high).

We also use a post-processing technique as suggested by [40], once after each algorithm. It turns out that the algorithms are varied enough that one should try all the practical ones and pick the best output. Huge instances can be solved by the MST algorithm, and we measure an algorithm, or combination of algorithms, by the percentage improvement over the objective function of the MST algorithm. The improved running times allows adding Greedy-Spider and Relative-Greedy to the combination of algorithms for large instances (500 to 2000 nodes), which in turn increases the improvement over the MST algorithm from an average of circa 9% (on large random instances) to an average of circa 14% (a circa 50% increase in this improvement).

We also tackle obtaining exact solutions, using natural integer programs similar to a bidirected integer linear program first proposed by [34] for Steiner Tree. Using an academic version of CPLEX, we can solve exactly instances with up to 30 nodes in the Euclidean input model. Further experimental study reveals that on instances where we can compute the optimum (random 30 points in two dimensions), the MST-based algorithm is on average 36% bigger than optimum, and the combination of approximation algorithms and fast heuristics is on average 14% bigger than optimum (the improvement over the MST-based algorithm decreases for larger instances).

[1]Actually, [6] suggests the existence of a $1.35\ln n$ approximation algorithm based on the ideas of [20]; this algorithm needs in the worst case $\Omega(n)$ calls to Minimum Weight Perfect Matching. Such running time would be impractical for large instances. A $1.5\ln(n)$ approximation algorithm was claimed by [19]; however this paper has errors. These may be fixable at the expense of making $\Omega(n)$ calls to Minimum Weight Perfect Matching or an $O(m)$ increase in the running time.

A. Approximation Algorithms and Heurstics

Min-Power Broadcast was first studied by Wieselthier et al. [42]. They proposed three heuristics but do not prove approximation ratios. The first heuristic, SPT, uses a shortest-path tree from the source z. The second heuristic, MST, uses a minimum spanning tree. In both cases the resulting undirected graph is oriented away from the source, and power is assigned accordingly. The third heuristic, called BIP (broadcasting incremental power), is a Prim-like heuristic which starts with an outgoing branching consisting of the source, and iteratively adds an arc connecting the set of nodes currently reached from the source to an unreached node, with minimum increase in the total power given by the selected arcs.

For the two-dimensional Euclidean input model, Wan et al. [41] study the approximation ratios of the above three heuristics. An instance was constructed in [41] to show that the approximation ratio of SPT is as large as $\frac{n}{2}-o(1)$. On the other hand, MST has a constant approximation ratio [12], [41]. The approximation ratio of BIP is at most the ratio of MST [41]. A sequence of further papers [4], [17], [16], [32], [15], [2] improved the analysis of the MST algorithm to 6. In three dimensions, the MST algorithm also has constant approximation ratio [33]. The running time of the MST algorithm is $O(m+n)$ in bidirected input graphs (where it has approximation ratio $\Theta(n)$), and $O(n\ln n)$ in the two-dimensional Euclidean input model. Thus it can be used on huge instances; it was experimentally analyzed in [18].

It is straightforward to give a variant of BIP running in $O(n^2)$; its approximation ratio is between 4.598 [3] and 6. Thus BIP is a practical algorithm. The best approximation ratio in the two-dimensional Euclidean input model is 4.2 and is achieved by the Relative-Greedy algorithm of [8].

In the bidirected input model, a standard reduction from Set Cover shows that no approximation ratio better than $O(\ln n)$ is possible unless $P=NP$ (using [14]). This reduction was known in 2000 and appears in several papers [12], [29], [9], [40], [27], [6]. The first $O(\ln n)$ approximation algorithm was given by Caragiannis et al. [9]. Similar algorithms were presented in [28], [40], and the simplest and best variant (ratio of $2(1+\ln n)$) of this algorithm was presented by [31] and achieves a $O(mn)$ running time (their analysis claims $O(mn\alpha(m,n))$ running time, but one observation can get rid of the inverse Ackermann function α in the analysis). This algorithm and the crux of its proof of approximation ratio can be traced back to [44] which is why we call it Hypergraph-Greedy. Later, [11] obtains another $O(\ln n)$ approximation algorithm, with a complicated algorithms based on [20], that

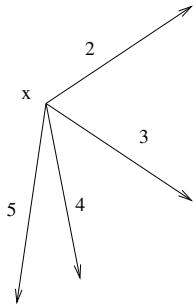

Fig. 1. A star with center x and four arcs, of power $\max\{2,3,4,5\} = 5$

needs in the worst case $\Omega(n)$ calls to Minimum Weight Perfect Matching. Relative-Greedy [8] also has approximation ratio of circa $2(1+\ln n)$ in this model.

In the general (directed) model, [6] were the first to publish a $O(\ln n)$-approximation: the Greedy-Spider algorithm (adapted from [26]). In a simultaneous submission [10] (journal version: [11]) also obtains a $O(\ln n)$-approximation, with a worse constant in front of $\ln n$, and with what appears to be a large running time.

Two other heuristics that do not have a good approximation ratio are fast enough and make sense. In [40], [25], a localized reduction in power is proposed. We adopt this as a post-processing for all of our algorithms (to do so after each iteration of various algorithms is too time consuming). Algorithm 2 of [31], which we call Greedy-Broom, is somehow similar to BIP, but chooses a directed star instead of an arc, minimizing the ratio of the power of the star by the number of newly reached nodes. It has a $O(mn)$ running time. We do not consider the algorithms of [21], [23], [43], [45], [1], [30] as they seem unsuitable for large instances.

II. Preliminaries

When discussing approximation ratios, natural logarithm ($\ln$) was used. For set X and element v, we may use $X+v$ to denote $X \cup \{v\}$ and $X-v$ to denote $X \setminus \{v\}$. For a set of arcs/edges X, we write $c(X) = \sum_{uv \in X} c_{uv}$.

For $u \in V$ and $r \in \{c_{uv} \mid uv \in E\}$, let $S(u,r)$ be the *directed 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.

Many algorithms described above use all the possible stars, and there are $O(m)$ of them (for each vertex u, the number of stars with center u is u's degree in the input graph).

The Hypergraph-Greedy algorithm [31] 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. At the end, it re-orients if needed the edges of H to lead away from the source z and thus obtain an out-going arborescence from the source (this idea, first applied in [9], only works if the input graph is bidirected, and typically loses a factor of 2 in the approximation ratio). See [31] for the $O(mn)$ variant. Recently, Calinescu [5] has obtained an $O(m \ln^2 n)$ variant of this algorithm using geometric data structures.

III. The Relative-Greedy algorithm

The Relative-Greedy algorithm of [8] (using ideas from [46]) keeps an undirected spanning tree T (initially, the minimum spanning tree in G according to cost c), and then does greedy local improvements as follows: For a set of vertices X, define the *swap-set* of X in T, $W_T(X)$, to be the maximum-cost set of edges of T such that, removing $W_T(X)$ from T leaves $|X|$ components, each containing exactly one vertex of X. Recall that $V(S)$ is the set of vertices of a star. Choose a star $S = S(u,r)$ to maximize the ratio $c(W_T(V(S)))/p(S)$ and add it to the set of stars (which starts empty), provided that this ratio exceeds 2. Remove $W_T(X)$ from T, and add "fake" edges: all the undirected version of the arcs of S, with the new cost 0, to obtain the tree used for the next iteration. Repeat choosing a new star as long as such a star exists. Once no star has ratio exceeding 2, take the graph consisting of the "real" edges of T and the arcs of all the selected stars, and just as for other algorithms above, re-orient if needed all its edges to obtain an out-going arborescence from the source z.

One needs to find the next star at most n times, and the challenge is to compute $W_T(V(S))$ and $c(W_T(V(S)))$ for every S. One naive method (as in Problem 23-1 of [13]) takes $O(n^2)$ per star. This can be improved to $O(n)$ per star by using the following known fact: For any spanning tree T, set $X \subseteq V$, and $v \in V \setminus X$, the swap-set $W_T(X+v)$ consists of $W_T(X)$ plus one edge. The stars S with the same center u do have the property that each has one more vertex compared to the previous one, if they are sorted by r.

The running time can be further improved to $O(\ln n)$ amortized time per star, using the linking-and-cutting trees data structure of Sleator and Tarjan (based on Splay Trees; see [39]). We do better (details to follow), with the main operation being: once the tree T is changed into T' by the addition of a fake edge and the removal of a "true" edge, our algorithm finds the edge in $W_{T'(X+v)} \setminus W_{T'(X)}$ (for each of the relevant X and v) in constant time using some precomputation of the tree T', and the knowledge accumulated and saved for T.

The algorithm keeps the following information, for current tree T. Fix in the discussion vertex $u \in V$. For this u, let $v_1, \ldots, v_{d'(u)}$ be the neighbors of u in G, sorted in non-decreasing order by c_{uv_i}. Here $d'(u)$ stands for the degree of u in the graph G. For convenience, set $v_0 = u$ and $d' = d'(u)$. Let $S_1, \ldots, S_{d'}$ be the stars with center u, where $V(S_j) = v_0, v_1, \ldots, v_j$ and the arcs connecting u to each v_i are included. Note that these stars with center u are sorted such that their power is non-decreasing and $p(S_i) = c_{uv_i}$. For convenience, define S_0 to be the one-vertex graph with only v_0. Let $A_j := W_T(V(S_j))$. We include for completeness a claim implicit in several earlier papers (such as [46]) - for intuition we mention that there is submodularity hidden

when computing swap-sets, which have an interpretation as a maximum weight independent set in a matroid (Example 44.1.1 from [37]).

Claim III.1 (Folklore)**.** *Assume all the tree weights are non-negative.*

1) *For any $j = 1, 2, \ldots, d'$, we have $|A_j| = j$.*
2) *A_j can be computed by the following procedure. Start with $A = \emptyset$, and find (if not possible, stop, and output A) any two vertices of $V(S_j)$ that are in the same connected component of $(V, E(T) \setminus A)$, then find the maximum-weight edge on the unique simple path of T between these two vertices, and add it to A.*
3) *For any $j = 1, 2, \ldots, d'$, A_j can be obtained from A_{j-1} by the following procedure: the graph $(V, E(T) \setminus A_{j-1})$ has j connected components, each containing one vertex of $V(S_{j-1})$. Find the component with v_j, and then find e, the maximum-weight edge of T on its unique simple path from v_j to the unique vertex of $V(S_{j-1})$ in this component. Then $A_j = A_{j-1} + e$.*

We omit the proof. Based on Claim III.1, the algorithm "stores" the sets A_j by keeping a pointer to the edge $e_j := A_j \setminus A_{j-1}$, using an array of size d'. This is done for every vertex u (as the center of the stars).

With this information, in time $O(m)$, one can go through all the stars S (for each u, use the order $S_1, \ldots, S_{d'(u)}$) to compute $c(W_T(V(S)))$ and identify the star S that maximizes the ratio $c(W_T(V(S)))/p(S)$. Next, the algorithm should remove $W_T(X)$ from T, and add "fake" edges: all the undirected version of the arcs of S, with the new cost 0, in order to obtain the tree used for the next iteration. Call u the center of S, and let j be such that $S = S_j$ (from the list $S_1, \ldots, S_{d'(u)}$ mentioned above). Recall that $A_j = W_T(V(S_j))$. If instead of removing A_j from T and adding the fake edges from S_j (all at once), we execute sequentially, for $i = 1$ to j, the step that adds a fake edge of cost 0 and endpoints u and v_i, and removes e_i from T (recall that $e_i = A_i \setminus A_{i-1}$), then one can check that we have exactly the same effect.

Lemma III.2. *For all $i = 1, \ldots, j$, e_i becomes the largest-weight edge on the unique simple path in the tree from u to v_i, before it is removed and replaced by uv_i.*

Proof: For any $q = 1, 2, \ldots, i$, e_q is the largest-weight edge on the unique simple path P from v_q to some v_l, for $l \in \{0, \ldots, q-1\}$, in $(V, E(T) \setminus A_{q-1})$, and the unique simple path from u to v_q before e_q is removed from T and replaced by uv_q consists of P followed by uv_l (or just P, if $l = 0$), and uv_l has weight 0. ∎

To obtain a better bound on the running time, we change the algorithm such that it does not change T in the case that e_j is a fake edge (we are unable to rule out e_j being a fake edge). This is based on the following claim, whose proofs we omit.

Claim III.3. *Let $T(i)$ be the tree in the original algorithm after the i^{th} iteration. Let $T'(i)$ be the tree in the Relative-Greedy algorithm after the i^{th} iteration. Let A_j be the swap set of S_j in $T(i)$, A'_j be the swap set of S_j in $T'(i)$. Then $c(A_j) = c(A'_j)$ for any S_j and any i.*

Thus our variant chooses the same stars as the original algorithm [8]. Also in their approximation-ratio analysis, the power of the output is bounded by twice the power of the selected stars plus the cost of the remaining "true" edges of the tree, and we have exactly the same quantity, since in our variant the tree only differs from the tree of the original variant in fake edges. Thus our variant has the same approximation ratio of 4.2 in the two-dimensional Euclidean input model.

Since we do not remove any fake edges, there are in total at most $n-1$ times when we change the tree. There are also at most $n-1$ stars that are added by the algorithm (each must introduce at least one fake edge), and thus in total at most $(n-1)^2$ times when some fake edge is considered (even if not added). One also has to construct the data structures (the arrays, one for each u, with e_j in position j) for the initial tree T. This can be easily accomplished in time $O(mn)$ using ideas developed later for maintaining the data structures. We omit the details; note that we allocate much more time to this initialization than the $O(m)$ allocated to an update - and indeed a running time of $O(m \lg n)$ in total seems possible using the linking-and-cutting trees data structure of Sleator and Tarjan (based on Splay Trees; see [39]).

Based on this discussion, once we establish how to maintain our data structures in time $O(m)$ whenever an edge of T is replaced by a fake edge, we obtain a $O(mn)$ running time for this variant of the Relative-Greedy algorithm. Thus from now we concentrate on the following "main" operation: given tree T, edge xx' of T and fake edge yy' of cost 0, recompute the data structures for the tree T' obtain from T by deleting xx' and adding yy'. One can check that, for one tree, these data occupies $O(m+n)$ space. We also note that once the data structures for T' are computed, we have no need to keep the data of T. Thus the overall space is $O(m+n)$.

For fixed center u, as above, we have A_j as the swap-set of $V(S_j)$ before T is transformed in T'. We are computing A'_j, the swap set of S_j in T'. Note that x, x', y, y' and T are not related to u and j. We know from Claim III.1 that A'_{j+1} has exactly one more edge than A'_j , and we aim to identify this edge, which we call e'_{j+1}. Recall that v_j is the only vertex in $V(S_j) \setminus V(S_{j-1})$. Let Q be the the connected component of $(V, E(T) - xx')$ that contains x and let Q' be the the connected component of $(V, E(T) - xx')$ that contains x' . Rename y, y', the endpoints of the fake edge, such that $V(Q)$ contains y (and $V(Q')$ contains y'). Note that Q is also the connected component of $(V, E(T) - yy')$ that contains x.

Claim III.4. $|A'_j \setminus A_j| \leq 1$.

Proof: Use the first procedure (second point) of Claim III.1 to construct A_j using only pairs of vertices of $V(S_j)$ that are both in $V(Q)$, or both in $V(Q')$. We select this way $|V(S_j) \cap V(Q)| - 1$ edges for pairs of vertices of $V(S_j) \cap V(Q)$, unless $V(S_j) \cap V(Q) = \emptyset$, in which case no such edge is

selected. Similarly, we select this way $|V(S_j) \cap V(Q')| - 1$ edges for pairs of vertices of $V(S_j) \cap V(Q')$, unless $V(S_j) \cap V(Q') = \emptyset$, in which case no such edge is selected. In total, we select either $|V(S_j)| - 2$ or $|V(S_j)| - 1$ edges.

We select exactly the same edges constructing A'_j using the same procedure. Thus $|A_j \cap A'_j| \geq |V(S_j)| - 2$, and with $|A_j| = |A'_j| = |V(S_j)| - 1$, the claim follows. ■

Both A'_j and A_{j+1} are used when computing e'_{j+1}. In fact, we do not access the full A'_j and A_{j+1}, but only the (at most three) elements of $A_{j+1} \setminus A'_j$ and $A'_j \setminus A_{j+1}$. If A'_j is removed from T', there will be exactly one node $v \in V(S_j)$ that is connected to y and y' in T'; this vertex is useful and the algorithm calls it v_c. However, if the edge yy' is selected in A'_j, v_c is not used by the algorithm any more, and the fact that it is not correctly defined does not matter. Moreover, the tree T' is pre-processed (in time $O(n)$) such that, for any v, v', with $v \in V(Q)$ and $v' \in V(Q')$, the maximum cost of an edge on the unique simple path in T' from v to v' can be found in constant time. Then one can obtain e'_{j+1} in constant time, using Algorithm 1. In the algorithm, the input consists of v_c, and tree edges f and f' whose meaning is as follows: if $A'_j = A_j$, then $f = f'$ is some arbitrary edge in A_j; otherwise $f = A_j \setminus A'_j$ and $f' = A'_j \setminus f$. The algorithm also makes use of the following queries, which are all answered in constant time (with a precomputation that takes time $O(n)$, and which we also describe below).

1) $max\_to\_y\_root(v)$ returns, for a vertex v, the maximum weight edge of T' on the unique simple path from v to y. If $v = y$, it returns NULL, which has, for convenience, negative weight. This quantity is stored in an array of size n which is computed in a preorder traversal of T' when rooted at y (if we process node v, a child of v', then $max\_to\_y\_root(v)$ is either $max\_to\_y\_root(v')$ or the edge vv'). A look-up in the table takes constant time.
2) $max\_T'(v, v')$ returns, for a vertices v and v' such that one of them is in $V(Q)$ and the other in $V(Q')$, the maximum weight edge of T' on the unique simple path from v to v'. It is done in constant time by calling $max\_to\_y\_root(v)$ and $max\_to\_y\_root(v')$ and comparing the results.
3) $max\_to\_x\_root(v)$ returns, for a vertex v, the maximum weight edge of T on the unique simple path from v to x. If $v = x$, it returns NULL, which has, for convenience, negative weight.
4) $v\_descendant\_of\_e$, for input vertex v and edge e, tests if e on the path from v to y in T'. For this, T' is preprocessed, in time $O(n)$, to answer in constant time Least Common Ancestor queries (see [22]).

After the code is executed, one has to reset $v_c = v_{j+1}$ if $v_c\_descendant\_of\_e'_{j+1}$. A long proof considering all the nine cases shows that Algorithm 1 correctly computes A'_j for every j. Preprocessing is $O(n)$, and applies to all stars S with all the possible centers. Thus in time $O(\sum_{u \in V} d'(u)) = O(m)$ we can obtain $W_{T'}(V(S))$ for all the stars S with all the centers. Based on the discussion of this section, we obtain one of our main results:

```
Computes e'_{j+1} and updates f, f', v_c.
if f = f' then
    if v_c, v_{j+1} ∈ V(Q), or v_c, v_{j+1} ∈ V(Q') then
        e'_{j+1} ← e_{j+1} // Case I
    end
    else
        if xx' = e_{j+1} then
            g ← max_T'(v_c, v_{j+1})
            e'_{j+1} ← g // Case II
        end
        else
            e'_{j+1} ← e_{j+1} // Case III
        end
        f' ← e'_{j+1}
        f ← e_{j+1}
    end
end
else if f' = e_{j+1} then
    if v_c, v_{j+1} ∈ V(Q), or v_c, v_{j+1} ∈ V(Q') then
        e'_{j+1} ← f
        f' = f // Case IV
    end
    else
        g ← max_T'(v_c, v_{j+1})
        e'_{j+1} ← g
        f' ← g // Case V
    end
end
else if f is not on the same side as v_{j+1} then
    e'_{j+1} ← e_{j+1} // Case VI
end
else if e_{j+1} is not on the same side as v_{j+1} then
    e'_{j+1} ← f
    f ← e_{j+1} // Case VII
end
else
    if e_{j+1} is the largest edge from v_{j+1} to x then
        e'_{j+1} ← max(e_{j+1}, f)
        f ← min(e_{j+1}, f) // Case VIII
    end
    else
        e'_{j+1} = e_{j+1} // Case IX
    end
end
```

Algorithm 1: Filling one entry in the data structure of T'

Theorem III.5. *For Min-Power Broadcast, there exists an algorithm with running time $O(mn)$ and approximation ratios 4.2 in the two-dimensional Euclidean input model and $2(1 + \ln n)$ in the bidirected input model.*

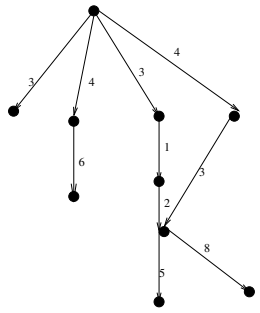

Fig. 2. A spider with four legs, weight $\max\{3,4,3,4\}+6+(1+2+5)+(3+8)=29$ and power 25.

Input: A directed graph $G=(V,E)$, cost function c_{uv} and a source vertex z
Output: A directed spanning graph H (seen as a set of arcs, with $V(H)=V$) such that in H there is a path from the source z to every vertex of V.

(1) Initialize $H=\emptyset$
(2) While H has at least one unhit component
  2.1 Find the spider S which minimizes $w(S)/(sf(S))$ w.r.t. H
  2.2 Set $H \leftarrow H \cup S$
(3) For all vertices v assign $p(v)=\max_{vu\in H} c_{vu}$

Fig. 3. The Greedy-Spider algorithm for Min-Power Broadcast with asymmetric costs

IV. The Greedy-Spider Algorithm

Recall that the input here is a directed graph. Part of the description below is copied and/or adapted from [6]. The algorithm starts iteration i with a directed graph H_i, seen as a set of arcs with vertex set V. The strongly connected components of H_i which do not contain the source z and have no incoming arc are called *unhit components*. The algorithm stops if no unhit components exists, since in this case the source can reach every vertex in H_i. Otherwise, a weighted structure called a *spider* (details below) is computed such that it achieves the biggest reduction in the number of unhit components divided by the weight of the spider. The algorithm then adds the spider (seen as a set of arcs) to H_i to obtain H_{i+1}.

Definition IV.1. *A* spider *is a directed graph consisting of one vertex called* head *and a set of non-trivial 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. As opposed to stars, 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*.

Definition IV.2. *The* shrink factor *of a spider S with head h with respect to H, denoted by $sf(S)$, is:*

1) *the number of representatives among its feet if h is reachable (where, by convention, a vertex is reachable from itself) from the source z or if h is not reachable from any of its feet, or*
2) *the number of representatives among its feet minus one, otherwise.*

The algorithm of [6] appears in Figure 3, with small differences in notation.

A method for finding the spider S which minimizes $w(S)/(sf(S))$ w.r.t. H is described in [6], without a proof of correctness. A running time is not given there, but a naive implementation, with some careful precomputation, is still $\Omega(n^3)$. Up to n spiders may be added to H by the algorithm, and to achieve an overall $O(n^3)$ requires amortized analysis, described next.

The new variant. We directly present the new implementation of finding the optimal spider, with competely new arguments that the correct spider is found. As in [6], we search for *powered* spiders (the power of the head is given). Instead of minimizing the ratio $w(S)/(sf(S))$, we prefer to use below maximizing the ratio $(sf(S)/w(S))$, which we call the *M-ratio* of a spider. Using Floyd-Warshall, all pairs shortest paths are precomputed and stored in $SP[i][j]$. This takes time $O(n^3)$.

As before, for every u, the vertices $v_1,\ldots,v_{d'(u)}$ are the neighbors of u in G, sorted in non-decreasing order by c_{uv_i}. As in the original version [6], we try all possible heads and all possible discrete power values for the head. That is, for each node u and each i, we have a data structure of size n (several variables and arrays of length n, actually), which is designed to compute the spider with the maximum M-ratio w.r.t H among those with head u and such that u has power c_{uv_i}; that is, with powered head u,i.

In the following u is fixed. The *children* of the powered head u,i are $u,v_1,\ldots,v_i$. Call $v_0=u$. Also as in the original variant (with precise definitions), for every vertex v, let $d[v]$ be the length of a shortest path from a child of the spider to v ($d[v]=0$, if v itself is a child). $d[v]=\min_{k=0}^{i} SP[v_k][v]$. This is computed, for fixed u and all $i=0,1,\ldots,d'(u)$, as follows. For powered head $u,0$, $d[v]=SP[v_0][v]$ for all $v\in V$. In the next iteration, we try to update $d[v]$ by $SP[v_1][v]$. Now $d[v]$ is correct for powered head $u,1$. Repeat for $2,\ldots$, until we finish powered head $u,d'(u)$. The total time used here is $O(n^3)$, including some additional book-keeping.

Fix i as well so we discuss powered head u,i. All the nodes v are sorted in increasing order of $d[v]$, and their indices kept in an array $I[j]$; we call this order u,i-sorted. u itself is the first in I (first position). Another array I^{-1} keeps for each $x\in V$, its index in I; that is, the j with $I[j]=x$.

The list R contains the representatives that cannot reach u, and is kept implicitly, as follows: arrays $Prev[j]$ and $Next[j]$ keep the previous and next index of I; that is, such that

$I[Next[j]]$ is the node that follows $I[j]$ in in the u,i-sorted R, and $Prev[j]$ is the node that preceeds $I[j]$ in the u,i-sorted R. We also keep $\overline{R}$, a list that contains all representatives, implicitly as R is kept above, using arrays $\overline{Prev}[j]$, $\overline{Next}[j]$.

Also, the algorithm keeps three variables, q_{opt}, S, and j_{opt}, which are meant to describe the following mathematical indices: Let $x_1, x_2, \ldots, x_{|R|}$ be the elements of R, u,i-sorted. Define $P := c_{uv_i}$ and

$$M_q := \frac{q}{P + \sum_{l=1}^{q} d[x_l]} \tag{1}$$

Then q_{opt} should be $\arg\max_{q\in\{1,2,\ldots,|R|\}} M_q$, S should be $\sum_{l=1}^{q_{opt}} d[x_l]$, and j_{opt} should be that $I[j_{opt}] = x_{q_{opt}}$ (or in other words, $j_{opt} = I^{-1}[x_{q_{opt}}]$). Due to space limitations, our later proof that these variables stored are indeed what they should be is omitted.

The algorithm also keeps another three variables, $\overline{q_{opt}}$, $\overline{S}$, and $\overline{j_{opt}}$, which are meant to describe the following mathematical indices: Let $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_{|\overline{R}|}$ be the elements of $\overline{R}$, u,i-sorted. Define

$$\overline{M}_q := \frac{q-1}{P + \sum_{l=1}^{q} d[\overline{x}_l]} \tag{2}$$

Then $\overline{q_{opt}}$ should be $\arg\max_{q\in\{1,2,\ldots,|\overline{R}|\}} \overline{M}_q$, $\overline{S}$ should be $\sum_{l=1}^{\overline{q_{opt}}} d[\overline{x}_l]$, and $\overline{j_{opt}}$ should be that $I[\overline{j_{opt}}] = \overline{x}_{\overline{q_{opt}}}$ (or in other words, $\overline{j_{opt}} = I^{-1}[\overline{x}_{\overline{q_{opt}}}]$).

The algorithm will keep $\widehat{q_{opt}}$, $\widehat{S}$, and $\widehat{j_{opt}}$ when u is reachable from z. They are meant to describe the following mathematical indices: Define

$$\widehat{M}_q := \frac{q}{P + \sum_{l=1}^{q} d[\overline{x}_l]} \tag{3}$$

Then $\widehat{q_{opt}}$ should be $\arg\max_{q\in\{1,2,\ldots,|\overline{R}|\}} \widehat{M}_q$, $\widehat{S}$ should be $\sum_{l=1}^{\widehat{q_{opt}}} d[\overline{x}_l]$, and $\overline{j_{opt}}$ should be that $I[\widehat{j_{opt}}] = \overline{x}_{\widehat{q_{opt}}}$ (or in other words, $\widehat{j_{opt}} = I^{-1}[\overline{x}_{\widehat{q_{opt}}}]$)

Lemma IV.1. *For every q, there exists a spider with powered head u,i and M-ratio at least M_q and a spider with powered head u,i and M-ratio at least $\overline{M}_q$. If u is reachable from the source, there exists a spider with powered head u,i and M-ratio at least $\widehat{M}_q$.*

We omit the straightforward proofs of this lemma, the next lemma, and the following theorem.

Lemma IV.2. *Fix powered head u,i. If u is not reachable from z, and if the spider maximizing the M-ratio for powered head u,i does not have any feet that can reach the head, then $M_{q_{opt}} = \max_{q\in\{1,2,\ldots,|R|\}} M_q$ provides such a spider. If u is not reachable from z, and if the optimal spider for powered head u,i has at least one foot that can reach the head, then $\overline{M}_{\overline{q_{opt}}} = \max_{q\in\{1,2,\ldots,|\overline{R}|\}} \overline{M}_q$ provides such a spider. If u is reachable from z, then $\widehat{M}_{\widehat{q_{opt}}} = \max_{q\in\{1,2,\ldots,|\overline{R}|\}} \widehat{M}_q$ provides the optimal spider for powered head u,i.*

Theorem IV.3. *Assume that q_{opt} is indeed such that $M_{q_{opt}} = \max_{q\in\{1,2,\ldots,|R|\}} M_q$, $\overline{q_{opt}}$ is indeed such that $\overline{M}_{\overline{q_{opt}}} = \max_{q\in\{1,2,\ldots,|\overline{R}|\}} \overline{M}_q$, and $\widehat{M}_{\widehat{q_{opt}}} = \max_{q\in\{1,2,\ldots,|\overline{R}|\}} \widehat{M}_q$. The maximum M-ratio of a spider w.r.t. H equals $\max\{M_{q_{opt}}, \overline{M}_{\overline{q_{opt}}}\}$ over all possible u,i when u is not reachable from z, or $\widehat{M}_{\widehat{q_{opt}}}$ over all possible u,i when u is reachable from z.*

We use the following properties of M_q, previously known in similar settings.

Lemma IV.4 (folklore)**.** *Assume $2 \le q \le |R|-1$. If $M_{q+1} \ge M_q$, then $M_q \ge M_{q-1}$. If $M_{q-1} \ge M_q$, then $M_q \ge M_{q+1}$. Assume $2 \le q \le |\overline{R}|-1$. If $\overline{M}_{q+1} \ge \overline{M}_q$, then $\overline{M}_q \ge \overline{M}_{q-1}$. If $\overline{M}_{q-1} \ge \overline{M}_q$, then $\overline{M}_q \ge \overline{M}_{q+1}$. If $\widehat{M}_{q+1} \ge \widehat{M}_q$, then $\widehat{M}_q \ge \widehat{M}_{q-1}$. If $\widehat{M}_{q-1} \ge \widehat{M}_q$, then $\widehat{M}_q \ge \widehat{M}_{q+1}$.*

As the algorithm progresses, more arcs are added to H and as result we have fewer strongly connected components, and fewer unhit strongly connected components. The algorithm can easily be careful to never select new representatives, and thus R (which is $R = R(u,i)$ for the powered head u,i) is shrinking. $\overline{R}$ is also shrinking. Whenever a vertex y becomes a non-representative, we go to each u and each i and as we describe starting with the next paragraph, in constant amortized time, modify the data structure for the u,i-powered head to satisfy the conditions mentioned above. There are $O(n^2)$ powered heads, and thus one vertex becoming a non-representative takes $O(n^2)$ time, for a total of $O(n^3)$.

Modifying the data structure, for a given, fixed u,i, is as follows. Using I^{-1}, find the j such that $I[j] = y$. We have three cases: $j < j_{opt}$, $j = j_{opt}$, and $j > j_{opt}$. In all cases we adjust $Prev[Next[j]] = Prev[j]$ and $Next[Prev[j]] = Next[j]$; in other words, we remove j from the doubly-linked list that implicitly stores $R(u,i)$.

For the purpose of the proof, let R' be $R(u,i)$ without the element y (same sorted order). Also, define M'_l (for $l = 1, \ldots, |R'|$) to be the M-ratio using the new list R'. If $j > j_{opt}$, there is nothing else left to do. Indeed, we do have that S and q_{opt} are correct, and we prove below that $M_{q_{opt}}$ still has the highest M-ratio. Indeed, we have that $M'_{q_{opt}} \ge M'_{q_{opt}-1}$ as $q_{opt}-1$ and q_{opt} refer to the same sublists, respectively. And we still have that $M'_{q_{opt}} \ge M'_{q_{opt}+1}$, since even if the $q_{opt}+1$st element of R' differs versus R, the change resulted in an increase of $P + \sum_{l=1}^{q_{opt}+1} d[x_l]$. Correctness follows in this case from Lemma IV.4.

If $j < j_{opt}$, we set $j' = j_{opt}$, and if $j = j_{opt}$, then we set $j' = Prev[j_{opt}]$. In both cases, we set $q' = q_{opt} - 1$, and $S' = S - d[I[j]]$. Note that $M'_{q'}$ is correctly computed as $q'/(P + \sum_{l=1}^{q'} d[x_l])$. We then compare $M'_{q'}$ with $M'_{q'+1}$, where $M'_{q'+1}$ can be computed in constant time using $Next$ links. If $M'_{q'} \ge M'_{q'+1}$ we stop with $j_{opt} = j'$, $q_{opt} = q'$, and $S = S'$; else we update $q' = q'+1$, $j' = Next[j']$, and $S' = S' + d[I[j']]$, and continue by comparing $M'_{q'}$ with $M'_{q'+1}$. We omit the arguments, based on Lemma IV.4 that this method is correct; note however that it takes time $O(1)$ plus the number of times when $M'_{q'} < M'_{q'+1}$. Each such comparison increases $Next[j_{opt}]$ by at least one; note also that $Next[j_{opt}]$ can never decrease. Thus the aggregate cost

of such comparisons is $O(n)$ for every powered head u, i, for a total of $O(n^3)$. This means that the overall cost for updating the data structure after $n-1$ representatives becoming non-representative is $O(n^3)$.

Similar arguments are used for $\overline{M'}_{q'}$ and $\widehat{M'}_{q'}$.

In each iteration, we add to H all the arcs in at most n shortest-paths; thus the algorithms only attempts to add to H at most n^2 arcs (and for every such arc, if it already in H, we do not do it again). We use the data structure of [24] to keep all the reachability data of H, with an an overall running time of $O(n^3)$. Based on the discussion of this section, we obtain one of our main results:

Theorem IV.5. *For Min-Power Broadcast, there exists an algorithm with running time $O(n^3)$ and approximation ratio $2(1+\ln n)$ in the directed/asymmetric input model.*

V. Simulation results

We implemented the following algorithms: the MST-based algorithm, BIP, SPT (shortest-paths tree), RG (Relative-Greedy), HG (Hypergraph-Greedy), GS (Greedy-Spider), and an IP-based algorithm using the free CPLEX Optimization Studio Academic Research Edition 12.2. We also wrote a post-processing heuristic that makes the solution minimal (goes through the vertices and reduces the power of each vertex as much as possible while keeping the same connectivity). We write MST-P to denote the MST algorithm, followed by this post-processing, and similarly have BIP-P, SPT-P, RG-P, HG-P, and GS-P. All the code, other than CPLEX, was written in C++ and compiled with `gpp`, and run on a Intel(R) Core(TM) i7-2310M CPU @ 2.10GHz desktop with 16G memory. We checked using a separate program that our experimental outputs are out-connected from the source.

We use $\kappa = 2$ in our experiments in the two dimensional plane. Tables I and II give the percent improvement over MST and the runtimes for the compared algorithms. For each instance size, we generated 50 uniformly random instances, and 50 instances using the propellant setting of the GenSeN topology generator [7]. In the table, the size of an instance is followed by r if uniformly random, and p if propellant. CPU is used to denote the CPU-time measured in seconds.

The results show that the exact algorithm has a practical running time up to 30 nodes, and produces an average improvement over MST of 22-39%. Post-processing is useful and fast enough. The Shortest-Paths based algorithm underperforms frequently. By itself, the Relative-Greedy algorithm does best in average, but we run into instances where its output is worse than the MST output; this can be explained since Relative-Greedy improves over the cost of the minimum spanning tree, and the power of a tree could be much less than it's cost (up to $(1/6)$-th, in the two-dimensional Euclidean input model).

Our data shows that the algorithms are varied enough that one should try all the practical ones and pick the best output. Table II includes MIN-E, which is obtained by running all the existing fast algorithms (including post-processing), and picking the best (the running time is pretty much the sum of the running times), and MIN-A, which is obtained the same way using our two new fast versions, plus all the existing fast algorithms. For 2000 nodes, the Greedy-Spider algorithm was too slow, and only a heuristic based on it (no proven approximation ratio) was included in MIN-A. The improvement over MST of MIN-A is in the range 13-24%, and is circa 50% more than the improvement of MIN-E.

n	MST-P		BIP		BIP-P		SPT-P		RG-P	
	%	CPU	%	CPU	%	CPU	%	CPU	%	CPU
20r	1.23	0.00	6.45	0.00	8.13	0.00	-3.13	0.00	10.34	0.00
20p	3.13	0.00	7.88	0.00	8.45	0.00	6.12	0.00	7.77	0.00
25r	5.65	0.00	8.77	0.00	8.98	0.00	5.33	0.00	9.14	0.00
25p	7.46	0.00	6.45	0.00	8.35	0.00	8.06	0.00	9.36	0.00
30r	8.17	0.00	4.31	0.00	5.33	0.00	3.66	0.00	10.66	0.00
30p	6.43	0.00	7.22	0.00	8.45	0.00	5.01	0.00	9.31	0.00
40r	5.34	0.00	10.31	0.00	11.14	0.00	4.52	0.00	10.11	0.00
40p	8.34	0.00	6.58	0.00	7.34	0.00	4.03	0.00	16.71	0.00
50r	4.66	0.00	7.71	0.00	8.23	0.00	3.03	0.00	15.34	0.00
50p	6.53	0.00	7.11	0.00	8.32	0.00	2.18	0.00	11.88	0.00
60r	3.89	0.00	10.66	0.00	11.54	0.00	6.21	0.00	13.98	0.00
60p	5.76	0.00	8.44	0.00	10.14	0.00	-4.45	0.00	10.56	0.00
70r	5.41	0.00	6.53	0.00	7.91	0.00	-6.38	0.00	15.32	0.00
80r	4.12	0.00	7.86	0.00	10.53	0.00	-5.12	0.00	12.37	0.00
90r	6.23	0.00	8.98	0.00	9.28	0.00	-4.57	0.00	14.43	0.00
100r	5.12	0.00	7.24	0.00	10.31	0.00	-3.64	0.00	12.13	0.00
200r	4.14	0.00	8.13	0.00	10.12	0.00	-0.21	0.00	12.47	0.00
500r	4.56	0.00	4.14	0.00	8.94	0.00	-3.85	0.00	12.88	1.12
1000r	5.12	0.00	6.64	0.00	7.16	0.00	-1.71	0.00	13.13	4.78
2000r	3.14	0.00	6.15	0.00	7.45	0.00	-3.41	0.00	11.45	35.67

TABLE I
AVERAGE PERCENT IMPROVEMENT OVER THE MST (%) AND RUNTIME IN SECONDS (CPU) FOR THE COMPARED ALGORITHMS.

n	IP		HG-P		MIN-E		GS-P		MIN-A	
	%	CPU	%	CPU	%	CPU	%	CPU	%	CPU
20r	28.3	1.4	7.34	0.00	12.13	0.00	5.42	0.00	16.02	0.00
20p	27.4	2.3	8.23	0.00	9.08	0.00	6.33	0.00	13.57	0.00
25r	34.5	21.0	9.13	0.00	14.13	0.00	6.52	0.00	17.93	0.00
25p	35.6	23.1	7.57	0.00	13.11	0.00	12.65	0.00	22.06	0.00
30r	35.7	69.4	5.68	0.00	12.30	0.00	9.67	0.00	20.66	0.00
30p	39.4	71.5	8.43	0.00	14.33	0.00	7.56	0.00	19.01	0.00
40r	—	—	5.78	0.00	15.39	0.00	11.45	0.00	17.83	0.00
40p	—	—	9.13	0.00	14.26	0.00	8.85	0.00	21.31	0.00
50r	—	—	7.34	0.00	15.27	0.00	7.36	0.00	19.73	0.00
50p	—	—	12.13	0.00	13.35	0.00	9.74	0.00	17.88	0.00
60r	—	—	9.34	0.00	12.45	0.00	11.26	0.00	24.78	0.00
60p	—	—	8.56	0.00	15.15	0.00	8.36	0.00	19.45	0.00
70r	—	—	6.45	0.00	14.34	0.00	9.36	0.00	21.14	0.00
80r	—	—	7.31	0.00	13.64	0.00	9.73	0.00	23.92	0.00
90r	—	—	8.56	0.00	12.42	0.00	8.63	0.00	20.11	0.00
100r	—	—	8.57	0.00	13.62	0.00	11.85	0.00	18.01	0.00
200r	—	—	9.45	0.00	12.42	0.00	11.65	1.22	17.10	2.34
500r	—	—	8.15	3.11	9.25	3.11	10.52	11.93	13.7	16.57
1000r	—	—	7.83	10.12	9.26	10.34	11.52	15.81	14.61	31.94
2000r	—	—	7.46	90.42	9.23	91.04	—	—	15.43	126.71

TABLE II
AVERAGE PERCENT IMPROVEMENT OVER THE MST (%) AND RUNTIME IN SECONDS (CPU) FOR THE COMPARED ALGORITHMS.

VI. Conclusions and Future Work

This study of the problem of assigning transmission power to the nodes of ad hoc wireless networks to minimize total power consumption during a broadcast session showed that a combination of fast approximation algorithms and heuristics can be applied to large instances and improve the total power consumption compared to the output of the well known MST-based algorithm. Our new (faster) algorithms make this improvement circa 50% bigger compared to existing fast heuristics.

We plan to use the multi-thread capability using multi-core processors to further speed up the heuristics (most greedy algorithms allow some parallelization) and the IP-based algorithms (CPLEX did use four threads).

REFERENCES

[1] S. Al-Shihabi, P. Merz, and S. Wolf. Nested partitioning for the minimum energy broadcast problem. In Vittorio Maniezzo, Roberto Battiti, and Jean-Paul Watson, editors, *Learning and Intelligent Optimization*, volume 5313 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, 2008.

[2] C. Ambühl. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *Proceedings of 32th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1139–1150. Springer Verlag, 2005.

[3] J. Bauer, D. Haugland, and D. Yuan. New results on the time complexity and approximation ratio of the broadcast incremental power algorithm. *Inf. Process. Lett.*, 109(12):615–619, May 2009.

[4] H. Cai and Y. Zhao. On approximation ratios of minimum-energy multicast routing in wireless networks. *Journal of Combinatorial Optimization*, 9(3):243–262, 2005.

[5] G. Călinescu. Faster approximation for symmetric min-power broadcast. In *CCCG*. Carleton University, Ottawa, Canada, 2013.

[6] 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.

[7] T. Camilo, J. Silva, A. Rodrigues, and F. Boavida. Gensen: A topology generator for real wireless sensor networks deployment. In *Proc. SEUS'07*, pages 436–445, 2007.

[8] 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.

[9] I. Caragiannis, C. Kaklamanis, and P. 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.

[10] I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Energy-efficient wireless network design. In *ISAAC*, 2003.

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

[12] 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.

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

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

[15] M. Flammini, A. Klasing, R.and Navarra, and S. Perennes. Tightening the upper bound for the minimum energy broadcasting. *Wirel. Netw.*, 14(5):659–669, October 2008.

[16] M. Flammini, R. Klasing, A. Navarra, and S. Perennes. Improved approximation results for the minimum energy broadcasting problem. *Algorithmica*, 49(4):318–336, 2007.

[17] M. Flammini, A. Navarra, R. Klasing, and S. Pérennes. Improved approximation results for the minimum energy broadcasting problem. In *DIALM-POMC*, pages 85–91, 2004.

[18] M. Flammini, A. Navarra, and S. Perennes. The "real" approximation factor of the mst heuristic for the minimum energy broadcasting. In *Proceedings of the 4th international conference on Experimental and Efficient Algorithms*, WEA'05, pages 22–31, Berlin, Heidelberg, 2005. Springer-Verlag.

[19] 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.

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

[21] S. Guo and O. Yang. Minimum-energy multicast in wireless ad hoc networks with adaptive antennas: Milp formulations and heuristic algorithms. *IEEE Transactions on Mobile Computing*, 5(4):333–346, April 2006.

[22] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[23] H. Hernández, C. Blum, and G. Francès. Ant colony optimization for energy-efficient broadcasting in ad-hoc networks. In *Proceedings of the 6th international conference on Ant Colony Optimization and Swarm Intelligence*, ANTS '08, pages 25–36, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] T. Ibaraki and N. Katoh. On-line computation of transitive closures of graphs. *Inf. Process. Lett.*, 16(2):95–97, 1983.

[25] R. Klasing, A. Navarra, A. Papadopoulos, and S. Prennes. Adaptive broadcast consumption (abc), a new heuristic and new bounds for the minimum energy broadcast routing problem. In NikolasM. Mitrou, Kimon Kontovasilis, GeorgeN. Rouskas, Ilias Iliadis, and Lazaros Merakos, editors, *NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, volume 3042 of *Lecture Notes in Computer Science*, pages 866–877. Springer Berlin Heidelberg, 2004.

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

[27] 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.

[28] D. Li, X. Jia, and H. Liu. Energy efficient broadcast routing in static ad hoc wireless networks. *IEEE Trans. Mobile Comput*, 3:144–151, 2004.

[29] W. 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.

[30] M. Min, A.F. O'Brien, and S.Y. Shin. Improved psor algorithm for minimum power multicast tree problem in wireless ad hoc networks. *Int. J. Sen. Netw.*, 8(3/4):193–201, October 2010.

[31] F. Mtenzi and Y. 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).

[32] A. Navarra. Tighter bounds for the minimum energy broadcasting problem. In *Proceedings of the Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, WIOPT '05, pages 313–322, Washington, DC, USA, 2005. IEEE Computer Society.

[33] A. Navarra. 3-d minimum energy broadcasting. In Paola Flocchini and Leszek Gsieniec, editors, *Structural Information and Communication Complexity*, volume 4056 of *Lecture Notes in Computer Science*, pages 240–252. Springer Berlin Heidelberg, 2006.

[34] T. Polzin and S.V. Daneshmand. On Steiner trees and minimum spanning trees in hypergraphs. *Oper. Res. Lett.*, 31(1):12–20, 2003.

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

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

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

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

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

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

[41] 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.

[42] 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.

[43] S. Wolf and P. Merz. Evolutionary local search for the minimum energy broadcast problem. In *Proceedings of the 8th European conference on Evolutionary computation in combinatorial optimization*, EvoCOP'08, pages 61–72, Berlin, Heidelberg, 2008. Springer-Verlag.

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

[45] D. Yuan, J. Bauer, and D. Haugland. Minimum-energy broadcast and multicast in wireless networks: An integer programming approach and improved heuristic algorithms. *Ad Hoc Netw.*, 6(5):696–717, July 2008.

[46] 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.