Experimental Comparison of Algorithms for Energy-Efficient Multicasting in Ad Hoc Networks*

Stavros Athanassopoulos, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos

Research Academic Computer Technology Institute and
Dept. of Computer Engineering and Informatics
University of Patras, 26500 Rio, Greece

Abstract. Energy is a scarce resource in ad hoc wireless networks and it is of paramount importance to use it efficiently when establishing communication patterns. In this work we study algorithms for computing energy-efficient multicast trees in ad hoc wireless networks. Such algorithms either start with an empty solution which is gradually augmented to a multicast tree (*augmentation algorithms*) or take as input an initial multicast tree and 'walk' on different multicast trees for a finite number of steps until some acceptable decrease in energy consumption is achieved (*local search algorithms*).
We mainly focus on augmentation algorithms and in particular we have implemented a long list of existing such algorithms in the literature and new ones. We experimentally compare all these algorithms on random geometric instances of the problem and obtain results in terms of the energy efficiency of the solutions obtained. Additional results concerning the running time of our implementations are also presented. We also explore how much the solutions obtained by augmentation algorithms can be improved by local search algorithms. Our results show that one of our new algorithms and its variations achieve the most energy-efficient solutions while being very fast. Our investigations shed some light to those properties of geometric instances of the problem which make augmentation algorithms perform well.

1 Introduction

Wireless networks have received significant attention during the recent years. Especially, ad hoc wireless networks emerged due to their potential applications in battlefield, emergency disaster relief, etc. [15]. Unlike traditional wired networks or cellular wireless networks, no wired backbone infrastructure is installed for ad hoc wireless networks.

A node (or station) in these networks is equipped with an omnidirectional antenna which is responsible for sending and receiving signals. Communication

* This work was partially supported by the European Union under IST FET Project CRESCCO and RTN Project ARACNE.

is established by assigning to each station a transmitting power. In the most common power attenuation model, the signal power falls as $1/r^\alpha$, where r is the distance from the transmitter and α is a constant which depends on the wireless environment (typical values of α are between 1 and 6). So, a transmitter s can successfully send a signal to a receiver t if $P_s \geq \gamma \cdot d(s,t)^\alpha$, where P_s is the power of the signal transmitted, $d(s,t)$ is the Euclidean distance between the transmitter and the receiver, and γ is the receiver's power threshold for signal detection which is usually normalized to 1. In this case, we say that node s establishes a direct link to node t. So, communication from a node s to another node t may be established either directly if the two nodes are close enough and s uses adequate transmitting power, or by using intermediate nodes. Observe that due to the nonlinear power attenuation, relaying the signal between intermediate nodes may result in energy conservation (we use the terms energy and power interchangeably).

A crucial issue in ad hoc wireless networks is to support communication patterns that are typical in traditional networks. These may include broadcasting, multicasting, gossiping (all–to–all communication) and more. Since establishing a communication pattern strongly depends on the use of energy, the important engineering question to be solved is to guarantee a desired communication pattern minimizing the total energy consumption.

An ad hoc wireless network is usually modelled as a complete directed graph $G = (V, E)$, with a non-negative edge cost function $c : E \to R^+$. Intuitively, V is the set of stations or nodes, the edges in E correspond to potential direct links, and the function c denotes the minimum energy required for establishing a direct link between any possible transmitter-receiver pair. Usually, the edge cost function is symmetric (i.e., $c(u,v) = c(v,u)$). An important special case, which usually reflects the real-world situation, henceforth called geometric case, is when nodes of G are points in a Euclidean space and the cost of an edge (u,v) is defined as the Euclidean distance between u and v raised to a fixed power α, i.e. $c(u,v) = d(u,v)^\alpha$. Asymmetric edge cost functions can be used to model medium abnormalities or batteries with different energy levels [12].

Consider a guest network denoted by a graph $H = (U, A)$, with $U \subseteq V$ and $A \subseteq E$. In order to establish the guest network H on the ad hoc network G, we must set the energy level of node u to $\max_{v \in U:(u,v)\in A} c(u,v)$. In other words, the energy level at which node u operates should be such that it can establish as direct links all edges of A directed out of u. The total energy needed for establishing H on G is the sum of the energy levels of all nodes.

The optimization problem we study in this paper can be stated as follows. Given an ad hoc network represented by a graph $G = (V, E)$, with a non-negative edge cost function $c : E \to R^+$, a special node $r \in V$ called *root*, and a set of *terminals* $D \subseteq V - \{r\}$, find a multicast tree, i.e., a tree rooted at r and spanning all the nodes in D, which can be established as a guest network in G with the minimum total energy. This problem is known as Minimum Energy Multicast Tree (MEMT). The special case of MEMT where $D = V - \{r\}$ is known as Minimum Energy Broadcast Tree (MEBT).

In the case of symmetric edge cost functions, Caragiannis et al. [4] present logarithmic (in the number n of stations) approximation algorithms for MEMT and MEBT. These results are asymptotically optimal since MEBT in symmetric graphs has been proven to be inapproximable within a sublogarithmic factor [6]. In [4] it is also shown that, in the case of asymmetric edge cost functions, MEMT is hard to approximate within $O(\log^{2-\epsilon} n)$, while Liang [12] presents an $O(|D|^{\epsilon})$ approximation algorithm. For MEBT in the case of asymmetric edge costs, logarithmic approximation algorithms have been presented in [2, 4]. The algorithm in [4] for the symmetric case of MEMT and the algorithm in [2] for the asymmetric case of MEBT borrow ideas from algorithms for a natural combinatorial problem known as Node Weighted Steiner Tree (NWST) [9, 11].

Although the above cases are very interesting from the theoretical point of view, the most important questions in practice concern the geometric case. This case was first considered in [10] in a slightly different context. Geometric cases of MEBT have received significant attention in the literature. When the nodes are points on a line, MEBT can be optimally solved in polynomial time [3, 7]. The case where the nodes are points in the Euclidean plane has been much studied. In this case, MEBT was proved to be NP-hard in [6]. The first algorithms were proposed in the seminal work of Wieselthier et al. [15]. These algorithms were based on the construction of minimum spanning trees (MST) and shortest path trees (SPT) on the graph representing the ad hoc network. The approach followed in [15] for computing solutions of MEMT was to prune the trees obtained in solutions of MEBT. Experimental results showed that the algorithm *Broadcast Incremental Power* (BIP) outperforms algorithms MST and SPT. In subsequent work Wan et al. [14] study the algorithms presented in [15] in terms of efficiency in approximating the optimal solution. Their main result is an upper bound of 12 on the approximation ratio of algorithm MST. This result implies a constant approximation algorithm for MEMT as well. Slightly weaker approximation bounds for MST have been presented in [6]. In [14], it is also proved that the approximation ratio of BIP is not worse than that of MST, and that other intuitive algorithms have very poor approximation ratio.

However, several intuitive algorithms have been experimentally proved to work very well on random instances of the problem. In [13, 16], algorithms based on shortest paths are enhanced with the *potential power saving* idea and are experimentally shown to outperform most of the known algorithms. In [1], Cagalj et al. introduced a heuristic called *Embedded Wireless Multicast Advantage* (EWMA) for computing efficient solutions to MEBT instances. This algorithm takes as input a spanning tree and transforms it to an energy-efficient broadcast tree by performing local improvements. As we discuss in Section 2, EWMA can be easily converted to work for MEMT as well. Another heuristic called Sweep was proposed in [15]; this also takes as input a tree and transforms it to an energy-efficient tree by performing local improvements. In contrast to these two algorithms, most of the algorithms discussed above are based on the idea of constructing a tree gradually. This means that, starting from an empty solution, a tree is augmented by repeatedly including new structures (i.e., new nodes

and edges) until connectivity from the root to the terminals is established. Another issue of apparent importance is to design algorithms for MEMT that are amenable to implement in a distributed environment (see e.g., [1, 5, 16]).

In this work, we divide algorithms presented in the literature in two categories: *local search algorithms* and *augmentation algorithms*. We describe the general features of both categories and report on the implementation of many algorithms, both existing and new ones. Our implementations include algorithms designed for approximating MEMT in the more general symmetric case as well as algorithms which are more intuitive for the geometric case. Our purpose is to experimentally compare all these algorithms in terms of the energy efficiency of the solutions obtained on random instances of MEMT on the Euclidean plane. An evaluation of the running time of the algorithms is also presented. The rest of the paper is structured as follows. We devote Section 2 to local search algorithms, while augmentation algorithms are discussed in Section 3. The experimental results are presented and commented in Section 4.

2 Local search algorithms

Local search algorithms perform a 'walk' on multicast trees. The walk starts from a multicast tree given as input. In each step, a local search algorithm moves to a new multicast tree obtained by removing some of the edges of the previous one and adding new edges, so that the necessary connectivity properties are maintained. The rule used in each move for selecting the next multicast tree is related to energy. Since local search algorithms require a multicast tree to start walking on, they are usually called after an augmentation algorithm. Typical representatives of this category are the algorithms Prune, EWMA and Sweep.

Algorithm Prune has been extensively used (see e.g., [13, 15]) for obtaining a multicast tree from a broadcast tree. In each step the algorithm performs the following operation. For each leaf which is not a terminal, it removes it from the tree together with its incoming edge. The algorithm terminates when all leaves are terminals. Prune can be easily implemented to run in linear time.

EWMA was proposed in [1] for solving MEBT, where it was assumed that the tree to start with is a minimum spanning tree. However, as it is clear in the following description of the algorithm, it can be used for MEMT and can start with any multicast tree. Starting with a multicast tree, EWMA walks on multicast trees by performing the following two types of changes in each step: (1) outgoing edges are added to a single node v; this node is said to be *extended* and (2) all outgoing edges are removed from some descendants of v; in this case we say that the particular descendants of v are *excluded*. Throughout its execution, EWMA uses three sets C, F and E. Intuitively, C is the set of nodes which have been considered by the algorithm, F is the set of nodes that were extended at least once in some previous step and were never excluded, and E is the set of nodes that were excluded in some previous step. Initially, the algorithm sets $C = \{r\}$ and $F = E = \emptyset$. In each step, EWMA takes as input the multicast tree produced in the previous step together with the sets C, F and E. Define the *gain* of a

node v as the decrease in the energy of the multicast tree obtained by excluding some of the nodes of the tree, in exchange for the increase in node v's energy in order to establish edges to all excluded nodes and their children. If no node in $C - F - E$ has positive gain, the node with the minimum energy is included in F and its children are included in C. Otherwise, the node with maximum gain is included in the set F, the excluded nodes are included in the set E, while both the excluded nodes and their children are included in the set C. The edges of the excluded nodes to their children are removed (changes of type (2)) and outgoing edges from v to all excluded nodes and their children are established (change of type (1)). The new multicast tree together with the updated sets C, F and E are passed as input to the next step. The algorithm terminates when the root and all terminals are contained in C. Our implementation of EWMA has running time $O(n^3)$ in the worst case.

Sweep was proposed in [15] as a simple heuristic for improving solutions of MEBT obtained by spanning tree and shortest path algorithms. Clearly, it can be used on any multicast tree as well. Sweep works as follows. It first assigns distinct IDs (consecutive integers $0, 1, ...$) to all nodes. Starting from the multicast tree, it proceeds in steps. In the i-th step, it examines the node v_i having ID equal to i. If, for some nodes $v_{i_1}, v_{i_2}, ...$ which are not ancestors of v_i, v_i's energy in the multicast tree is not smaller than the cost of all the edges from v_i to $v_{i_1}, v_{i_2}, ...$, Sweep removes the incoming edges of $v_{i_1}, v_{i_2}, ...$ and adds edges from v_i to $v_{i_1}, v_{i_2}, ...$ in the multicast tree. The algorithm terminates when all nodes have been examined. Although several variations of Sweep seem natural (this is also mentioned in [15]), somewhat surprisingly, none of several variations we implemented is better than Sweep in terms of the energy efficiency of the solution obtained. A simple implementation of Sweep runs in time $O(n^2)$.

3 Augmentation algorithms

Augmentation algorithms build a multicast tree by starting from an empty solution which is gradually augmented until a guest network having directed paths from the root to the terminals is established. Clearly, once such a guest network is available, it can be easily converted to a multicast tree. The solution is augmented in phases. In each phase, an augmentation algorithm adds to the solution a structure. This may be an edge to a node (e.g., in BIP and well known implementations of MST [8]), a set of edges directed out of the same node (e.g., in a variation of BIP called BAIP [15]), a path (e.g., in algorithms presented in [13, 16]), a spider (a special directed graph used in [2] and implicitly in [4]), etc. The structure is selected among all candidate structures so that a local objective is minimized. The local objective is usually related to the energy needed in order to establish the edges of the structure. We devote the rest of this section to the detailed description of several augmentation algorithms. These include existing algorithms in the literature as well as new ones with several variations.

Basic algorithms. Starting from a multicast tree containing only the root, the algorithms *Shortest Path First* (SPF), *Multicast Incremental Power* (MIP), *Dens-*

est Shortest Path First (DSPF) and *Densest Two Shortest Paths First* (D2SPF) augment the multicast tree in phases, until all terminals are included in the tree. In each phase, algorithm SPF adds to the multicast tree the shortest directed path (i.e., the path whose establishment requires the minimum amount of energy) that connects some node of the tree to a terminal out of the tree. Algorithm MIP adds the path requiring the minimum additional energy that connects the tree to a terminal out of the tree. This means that after each phase the cost of the edges directed out of each node v in the path selected in the phase is decreased by the cost of the edge outgoing from v in the path. Algorithm DSPF selects among the minimum additional energy paths from the tree to the terminals out of the tree, the one minimizing the ratio of additional energy over the number of new terminals that are included in the tree. Algorithm D2SPF is similar to DSPF; the difference being that, for any node v and any terminal t both not in the multicast tree, D2SPF considers paths from the tree to t passing through v.

In our implementations, we make extensive use of algorithms for computing shortest paths. Computing shortest paths from a node to all other nodes in a complete directed graph can be done in time $O(n^2)$ using a simple implementation of Dijkstra's algorithm using binary heaps as priority queues [8]. More complex heaps which have been proved to yield faster implementations of Dijkstra's algorithm (e.g., Fibonacci heaps) do not decrease the running time substantially in our case since the graph representing the ad hoc network is complete. SPF, MIP and DSPF run in time $O(mn^2)$, where m is the number of terminals. In each of the at most m phases, the algorithms perform a shortest path computation. The processing of the graph (i.e., the decrease to the edge costs) required in MIP and DSPF does not affect their asymptotic running time compared to SPF. Each phase of D2SPF requires the computation of all-pairs shortest paths which needs time $O(n^3)$ leading to overall running time of $O(mn^3)$.

The potential power saving idea. The basic algorithms do not examine whether establishing a new path could also include nodes which had been included in the multicast tree in previous phases, and could now be connected to the multicast tree as children of some node in the path. In this way, the energy of their previous parent could be decreased. The total decrease is denoted as *potential power saving* ([13, 16]). The algorithms described here are variations of the basic algorithms; their difference being that, when computing the additional energy of a candidate path, they subtract the potential power saving. When a new path is established, the multicast tree is modified accordingly, i.e., nodes previously included in the multicast tree might now be connected as children of nodes in the new path.

Algorithms SP3SF, DSP3SF and D2SP3SF are variations of MIP, DSPF and D2SPF with potential power saving, respectively. Algorithm SP3SF was originally proposed in [13] and, according to our knowledge, computes the most energy-efficient solutions in the setting studied here. Algorithm 2SP3SF is a variation of D2SP3SF where the local objective in each phase is to minimize the additional energy minus the potential power saving. The necessary computations for computing the potential power saving increase the running time of algorithms

SP3SF and DSP3SF to $O(m^2n^2)$, while algorithms 2SP3SF and D2SP3SF run in time $O(mn^3)$. However, in practice the running time of SP3SF and DSP3SF is only slightly worse than that of MIP and DSPF, respectively.

NWST-based algorithms. We now present six algorithms which borrow ideas from approximation algorithms for NWST presented in [9, 11]. In fact, the algorithm of [4] for the symmetric case of MEMT reduces instances of MEMT to instances of NWST. Although, the size of the resulting instance of NWST is polynomial in theory, even for small instances of MEMT the corresponding instance of NWST is intractable to be solved with the algorithms of [9, 11] in practice. The algorithms presented below apply ideas from [9, 11] to the MEMT instance directly. A similar algorithm has been designed for MEBT in [2]. All these algorithms use the idea of gradually augmenting a guest network by repeatedly adding spiders or forks of small density.

A *spider* is a directed graph consisted of a node called *head* and a set of directed paths called *legs*, each of them from the head to the nodes called *feet* of the spider. The definition allows legs to share nodes and edges. The weight of a spider is the maximum cost of the edges leaving the head plus the sum of costs of the legs, where the cost of a leg is the sum of the cost of its edges without the edge leaving the head. A *fork* is a spider having a *center* node that is reached through a shortest directed path from the head so that the subgraph of the fork induced by removing the edges of this path and all nodes in the path but the center is a spider having the center as its head. We call this spider the *subspider* of the fork. The weight of the fork is the cost of the edges on the directed path from the head to the center plus the weight of its subspider. We say that a node u can be connected *for free* to a path, if the cost $c(v_i, u)$ of the edge from a node v_i in the path to u is smaller than the cost of the edge directed out of v_i in the path.

Algorithm *Densest Spider First* (DSF) establishes a guest network H in which the root and the terminals are contained in the same loosely connected component. Loose connectivity of a directed graph means that its undirected counterpart (i.e., the graph obtained by substituting directed edges by undirected ones) is connected. Then, for establishing a multicast tree in G, the algorithm computes a tree directed from the root to the terminals in the supergraph of H containing all the edges in H and their opposite-directed edges. The algorithm assigns indices to nodes to keep track of which nodes belong to the same loosely connected component in H. Each index is a non-negative integer; nodes having the same finite index are loosely connected and nodes having different or infinite indices are not connected at all. Initially, the index of the root is 0, terminals have finite, positive and distinct integer indices, while all other nodes have infinite index. The algorithm proceeds in steps until the root and all terminals have index 0. In each step, the algorithm finds a node v and a spider having v as its head and nodes having pairwise distinct, different than v and not infinite indices as its feet, so that the ratio of the weight of the spider over the number of its feet (called the density of the spider) is minimized. Let $i_1, i_2, ..., i_k$ be the indices of the nodes in the spider in non-decreasing order. For each node in the

graph with index i_1 or i_2, ..., or i_k the algorithm sets its index to i_1 and adds the edges of the spider to H. Algorithms DSF-2 and DSF-3 are slight variations of DSF; the only difference being that the node selected as the head of the spider in each step is constrained to have finite index in DSF-2 and index 0 in DSF-3.

In our implementations, these algorithms first perform a preprocessing to compute, for each node and each possible energy level of this node, the shortest paths from this node to all other nodes. This requires time $O(n^4)$ and dominates the asymptotic running time of the algorithms. Once the length of all shortest paths has been computed, then computing the minimum density spider having as head a particular node with a particular energy level can be done in time $O(m)$. Hence, the minimum density spider in each of the at most m phases is computed in time $O(mn^2)$.

Algorithm *Densest Incremental Spider First* (DISF) is a variation of DSF-3. In each step, after the minimum density spider has been selected, for each node u in the spider, the cost of the edges directed out of u in G is decreased by the maximum cost of the edges directed out of u in the spider. Intuitively, the weight of the spider in algorithms DSF, DSF-2 and DSF-3 is an upper bound on the energy required to establish the edges of the spider. In DISF, the weight of the spider computed in each step is an upper bound on the additional energy required to establish the edges of the spider, given that edges that were included in H in previous steps have already been established. Due to the update in the edge cost function required in each phase of algorithm DISF, a similar preprocessing to that used in DSF is required in each phase. This needs time $O(mn^4)$ and dominates the asymptotic running time of the algorithm.

Algorithm *Densest Fork First* (DFF) is a variation of DISF. In each step a fork having as head a node of index 0 minimizing the ratio of the weight of the fork over the number of feet is added to the guest network H. Finally, algorithm DFF-2 is similar to DFF. The main difference is that the number of feet plus the number of non-zero finite indices of nodes which can be connected for free to the path from the head to the center is used in the denominator of the local objective. The running time of algorithms DFF and DFF-2 is asymptotically the same with that of DISF. In each phase the time required for preprocessing asymptotically dominates the time required for computing the densest fork.

Constrained and iterative versions. In many of the above algorithms, more than one terminals can be added to the multicast tree in each phase. It may be the case that adding a structure with many new terminals in a phase worsens the final solution. We have implemented constrained versions of the algorithms described above. which take as input an *augmentation constraint parameter* denoting the maximum number of new terminals allowed to be included in each phase and constrain appropriately the space of candidate structures. Iterative versions of algorithms DSPF, DSP3SF, SP3SF, D2SPF, D2SP3SF, 2SP3SF, DISF, DFF and DFF-2 repeatedly run their constrained versions for all possible values of the augmentation constraint parameter and output the best solution. These algorithms are called iDSPF, iDSP3SF, iSP3SF, iD2SPF, iD2SP3SF, i2SP3SF, iDISF, iDFF and iDFF-2, respectively. Clearly, an iterative version of an algorithm

is superior to its unconstrained version in terms of energy efficiency; however, it may require running time proportional to the number of terminals times the running time of the unconstrained version.

4 Experimental results

In this section we discuss the outcome of our experimentation with the algorithms presented in the previous sections. Results of our experiments are depicted in Tables 1-5. We have also implemented a few more broadcasting algorithms (e.g., variations of algorithms BIP and BAIP) and performed experiments with them by running algorithm Prune to their solutions in order to obtain multicast trees. These results are certainly inferior to those presented below and will not be further discussed. In addition, we have considered variations of the algorithms which are appropriate to implement in a distributed setting. The corresponding experimental results are usually worse than those of the centralized algorithms. We will not discuss distributed implementation issues here; we prefer to focus on centralized implementations since they give lower bounds on the energy efficiency of the solutions and the overall work required in their distributed implementations.

The algorithms were executed on geometric instances of the problem of different size. Input instances consist of nodes corresponding to points with uniformly random coordinates in $[0, 5)$. A node is randomly selected to be the root and terminals are selected uniformly at random without replacement among all other nodes. For each instance, we use the term *group* to denote the set of terminals together with the root.

Basic algorithms and algorithms with potential power saving were executed on instances of size 100 for groups of size $10, 20, ..., 100$ and values 2 and 4 for α. Table 1 shows the performance of the basic algorithms and algorithms with potential power saving with respect to the energy efficiency. For each group size, 100 random instances with the particular group size are constructed and all algorithms are executed on these random instances. The energy values shown in Table 1 for each algorithm in its executions on a particular group size, is the average energy of the multicast trees computed by the algorithm in its execution on all random instances of the particular group size.

All potential power saving algorithms perform better than basic algorithms (this had been experimentally observed for SP3SF and MIP in [13, 16]). This difference is larger in the case $\alpha = 2$. Interestingly, D2SPF is at most 1% worse than SP3SF in this case. D2SP3SF outperforms all potential power saving algorithms in the case $\alpha = 2$; it is significantly better than SP3SF and 2SP3SF and slightly better than DSP3SF. Also, 2SP3SF is marginally better than SP3SF. In the case $\alpha = 4$, SP3SF and 2SP3SF seem to produce the most energy-efficient solutions (with DSP3SF being marginally worse). In our experiments, we observed that algorithms using two concatenated shortest paths as structures are always much slower than algorithms using as structures single shortest paths (see e.g., Table 5). This is interesting (and somewhat surprising) since the asymptotic running

time of all algorithms using the potential power saving idea is essentially the same for large group sizes.

$\alpha = 2$	10	20	30	40	50	60	70	80	90	100
SPF	5.27	7.19	8.49	9.51	10.07	10.56	11.26	11.55	12.09	12.45
MIP	4.98	6.78	7.96	8.85	9.47	9.89	10.52	10.84	11.26	11.61
DSPF	4.93	6.64	7.83	8.66	9.22	9.61	10.17	10.40	10.73	11.07
D2SPF	5.01	6.59	7.80	8.56	9.06	9.46	10.00	10.22	10.53	10.86
SP3SF	4.92	6.59	7.66	8.50	9.03	9.38	9.90	10.10	10.49	10.79
DSP3SF	4.87	6.51	7.59	8.43	8.88	9.25	9.76	10.01	10.26	10.63
2SP3SF	4.90	6.56	7.62	8.45	8.99	9.32	9.89	10.07	10.39	10.74
D2SP3SF	4.96	6.50	7.60	8.34	8.87	9.18	9.69	9.90	10.18	10.49
iDSPF	4.89	6.55	7.70	8.52	9.07	9.44	9.97	10.23	10.56	10.92
iD2SPF	4.87	6.49	7.61	8.41	8.89	9.30	9.78	10.05	10.39	10.71
iSP3SF	4.92	6.59	7.66	8.50	9.03	9.38	9.90	10.10	10.48	10.79
iDSP3SF	4.81	6.37	7.38	8.16	8.58	8.94	9.41	9.63	9.92	10.28
i2SP3SF	4.90	6.55	7.62	8.45	8.99	9.32	9.88	10.07	10.38	10.73
iD2SP3SF	4.81	6.38	7.40	8.16	8.64	8.98	9.47	9.68	9.96	10.25
$\alpha = 4$	10	20	30	40	50	60	70	80	90	100
SPF	1.51	1.97	2.46	2.65	2.89	2.99	3.19	3.25	3.40	3.61
MIP	1.48	1.93	2.41	2.59	2.82	2.93	3.10	3.18	3.32	3.52
DSPF	1.49	1.93	2.41	2.61	2.82	2.93	3.11	3.19	3.32	3.52
D2SPF	1.49	1.94	2.42	2.60	2.81	2.91	3.10	3.18	3.29	3.52
SP3SF	1.47	1.91	2.36	2.55	2.74	2.85	3.02	3.08	3.21	3.41
DSP3SF	1.46	1.92	2.38	2.55	2.75	2.87	3.04	3.12	3.25	3.43
2SP3SF	1.46	1.90	2.36	2.55	2.74	2.85	3.02	3.09	3.21	3.41
D2SP3SF	1.48	1.92	2.38	2.55	2.75	2.86	3.03	3.12	3.22	3.41
iDSPF	1.47	1.91	2.38	2.56	2.78	2.89	3.06	3.13	3.27	3.46
iD2SPF	1.47	1.90	2.37	2.55	2.77	2.87	3.05	3.12	3.24	3.44
iSP3SF	1.47	1.91	2.36	2.55	2.74	2.85	3.02	3.08	3.21	3.41
iDSP3SF	1.45	1.88	2.33	2.50	2.70	2.81	2.96	3.04	3.16	3.34
i2SP3SF	1.46	1.90	2.36	2.55	2.74	2.85	3.02	3.09	3.21	3.41
iD2SP3SF	1.46	1.89	2.34	2.51	2.71	2.82	2.99	3.06	3.17	3.35

Table 1. Comparison of basic algorithms, algorithms using the potential power saving idea and their iterative versions on random instances with 100 nodes, $\alpha = 2$ and $\alpha = 4$ and group sizes $10, 20, ..., 100$.

In general, NWST-based algorithms are slow. This has been already justified in Section 3 where we discuss their running time (see also Table 5). This fact did not allow us to perform large experiments. Our experiments with instances of the problem with size 40 (see Table 2) show that NWST-based algorithms are inferior to most of the algorithms discussed above. DSF, DSF-2 and DSF-3 are not much slower than D2SP3SF and 2SP3SF but their solutions are much worse in terms of energy efficiency (in particular in the case $\alpha = 2$). DISF is slightly better in the case $\alpha = 2$ and rather worse in the case $\alpha = 4$. Its running time is huge. DFF and DFF-2 are even slower but seem to be the best among the NWST-based algorithms in terms of energy efficiency in their solutions. Overall, all NWST-based algorithms are worse than algorithms using the potential power saving idea. Algorithms DISF, DFF and DFF-2 include some of those properties which make basic algorithms with potential power saving perform well, i.e., they augment a multicast tree containing the root by including in it new terminals in each phase. Unfortunately, the local objective used in NWST-based

algorithms contains the weight of spiders or forks and this may not always be proportional to the energy. In addition, we see no clear way of incorporating the potential power saving (or a similar) idea to NWST-based algorithms. Recall that NWST-based algorithms are variations of DSF which was designed to efficiently approximate optimal solutions of MEMT in the more general symmetric case. The performance of NWST-based algorithms indicates that the particular geometric version of the problem we consider here has certain properties which are better exploited by algorithms with simple and intuitive local objectives.

$\alpha = 2$	10	20	30	40
MIP	7.16	9.61	10.99	11.97
SP3SF	6.94	9.09	10.28	11.18
DSP3SF	6.83	8.87	9.97	11.04
DSF	8.49	11.29	12.31	12.57
DSF-2	8.11	10.54	11.62	12.57
DSF-3	8.12	10.40	11.28	11.95
DISF	7.72	10.06	11.00	11.79
DFF	7.20	9.63	10.69	11.79
DFF-2	7.19	9.54	10.66	11.72
iDISF	6.94	9.15	10.18	10.95
iDFF	7.06	9.47	10.50	11.55
iDFF-2	7.04	9.39	10.44	11.53

$\alpha = 4$	10	20	30	40
MIP	5.66	8.22	9.25	10.00
SP3SF	5.55	7.96	8.98	9.66
DSP3SF	5.51	7.87	9.06	9.70
DSF	6.52	9.13	9.99	10.27
DSF-2	5.83	8.36	9.50	10.27
DSF-3	6.15	8.91	9.99	10.65
DISF	6.01	8.76	9.74	10.49
DFF	5.74	8.37	9.38	10.11
DFF-2	5.71	8.33	9.37	10.11
iDISF	5.57	8.05	9.11	9.79
iDFF	5.67	8.20	9.33	10.05
iDFF-2	5.65	8.20	9.32	10.00

Table 2. Comparison of NWST-based algorithms and their iterative versions with algorithms MIP, SP3SF and DSP3SF on random instances with 40 nodes, $\alpha = 2$ (left) and $\alpha = 4$ (right) and group sizes $10, 20, 30$ and 40.

We also investigated iterative versions of our algorithms. By the definition of these algorithms, it is clear that they always perform better than their unconstrained counterparts in terms of energy efficiency at the expense of multiplying the running time with the group size. In practice, in most of our implementations, the running time of iterative algorithms is closer to the running time of their unconstrained counterparts (see Table 5). iDSP3SF computes the most efficient solutions with respect to energy efficiency. iD2SP3SF is slightly worse than iDSP3SF in terms of energy efficiency but its running time is enormous. The only iterative algorithm with running time comparable to that of iDSP3SF is iSP3SF which does not actually improve its unconstrained counterpart SP3SF. Overall, the solutions obtained by iDSP3SF are up to 5% (and about 2%) better than those of iSP3SF in the case of $\alpha = 2$ (and $\alpha = 4$, respectively). The corresponding results are depicted in Tables 1. The solutions obtained by iterative versions of NWST-based algorithms are usually much worse while their running time is huge. It is interesting, however, that the solutions obtained by iDISF significantly improve the results obtained by DISF. Unfortunately, this seems to be the slowest among all algorithms we implemented.

Our next investigation probably answers why iDSP3SF is superior to iSP3SF while DSP3SF and SP3SF compute solutions of comparable energy efficiency. In Table 3 we present the performance of constrained versions of algorithms DSP3SF and SP3SF (additional results for the constrained version of DSPF are also pre-

sented). It is clear that the constrained version of SP3SF computes solutions of almost the same energy regardless of the augmentation constraint parameter, while this is not the case for DSP3SF. This indicates that, given an instance of the problem, many different augmentation constraint parameter values are possible to give the best solution of iDSP3SF with respect to energy efficiency, while the solution obtained by iSP3SF is marginally better than the solution obtained by the constrained version of SP3SF with augmentation parameter constraint equal to 1.

$\alpha = 2$, group size: 50	1	2	3	4	5	6	7	8	9	10	> 10
DSPF	9.46	9.36	9.32	9.27	9.26	9.25	9.24	9.23	9.24	9.23	9.22
SP3SF	9.04	9.03	9.03	9.03	9.03	9.03	9.03	9.03	9.03	9.03	9.03
DSP3SF	9.04	9.02	8.91	8.88	8.85	8.85	8.86	8.86	8.88	8.88	8.87
$\alpha = 2$, group size: 100	1	2	3	4	5	6	7	8	9	10	> 10
DSPF	11.61	11.43	11.32	11.25	11.21	11.18	11.15	11.15	11.12	11.11	11.07
SP3SF	10.83	10.80	10.80	10.79	10.79	10.79	10.79	10.79	10.79	10.79	10.79
DSP3SF	10.83	10.85	10.74	10.71	10.70	10.64	10.68	10.64	10.64	10.67	10.62
$\alpha = 4$, group size: 50	1	2	3	4	5	6	7	8	9	10	> 10
DSPF	2.82	2.83	2.82	2.82	2.82	2.82	2.82	2.82	2.82	2.82	2.82
SP3SF	2.74	2.74	2.74	2.74	2.74	2.74	2.74	2.74	2.74	2.74	2.74
DSP3SF	2.74	2.75	2.74	2.75	2.75	2.75	2.76	2.76	2.76	2.75	2.75
$\alpha = 4$, group size: 100	1	2	3	4	5	6	7	8	9	10	> 10
DSPF	3.52	3.52	3.52	3.53	3.53	3.52	3.52	3.52	3.52	3.52	3.51
SP3SF	3.41	3.41	3.41	3.41	3.41	3.41	3.41	3.41	3.41	3.41	3.41
DSP3SF	3.41	3.43	3.43	3.42	3.43	3.42	3.43	3.43	3.42	3.43	3.42

Table 3. The energy efficiency of constrained versions of algorithms DSPF, SP3SF and DSP3SF on random instances with 100 nodes, $\alpha = 2$ and $\alpha = 4$, and group sizes 50 and 100 for different augmentation constraint parameter values. The last column contains the energy of the best solution for parameter values greater than 10.

The effect of local search algorithms on solutions obtained by the augmentation algorithms we have implemented is important in the case $\alpha = 2$ while it seems to be marginal in the case $\alpha = 4$ (see Table 4). Such algorithms do not add significant overhead to the overall running time and usually lead to much better solutions. EWMA seems to be appropriate for the case $\alpha = 2$ and in particular for broadcasting instances for which it was originally designed, while Sweep seems to be slightly better in the case $\alpha = 4$. The improvement in solutions of augmentation algorithms achieved after running EWMA and/or Sweep is larger for the augmentation algorithms which are worse in terms of the energy efficiency and smaller for those algorithms which output more efficient solutions. Usually, running repeatedly local search algorithms can improve a solution further. This improvement starts to be negligible after the first two or three executions. An interesting question is whether running EWMA and/or Sweep after the unconstrained version of an algorithm is better than its iterative version. We observed that this is the case for algorithms SP3SF and 2SP3SF (we have already seen that iterative versions of these algorithms do not significantly improve on the energy efficiency of the solutions), but this is not clear for algorithms DSPF, DSP3SF,

D2SPF and D2SP3SF, even if we compare these algorithms followed by several calls to EWMA and Sweep with their iterative versions. Also, running EWMA after e.g., iDSP3SF improves the solutions further. This discussion implies that local search algorithms cannot substitute iterative algorithms; however they can be used to slightly improve their performance with respect to energy efficiency of the solutions obtained.

In conclusion, iDSP3SF followed by EWMA seems to give the most energy-efficient solutions. Algorithms SP3SF and DSP3SF followed by local search algorithms provide a good compromise between energy efficiency and running time.

$\alpha = 2$	10				50				100			
SPF	5.27	5.13	5.15	5.04	10.07	9.55	9.40	9.06	12.45	11.76	11.00	10.77
MIP	4.98	4.91	4.92	4.87	9.47	9.09	9.06	8.82	11.61	10.93	10.72	10.46
DSPF	4.93	4.87	4.89	4.85	9.22	8.98	8.90	8.75	11.07	10.70	10.51	10.31
iDSPF	4.89	4.84	4.85	4.81	9.07	8.82	8.79	8.63	10.92	10.57	10.37	10.19
DSP3SF	4.87	4.87	4.84	4.83	8.88	8.88	8.74	8.71	10.63	10.63	10.30	10.23
iDSP3SF	4.81	4.81	4.79	4.78	8.58	8.58	8.48	8.46	10.28	10.28	10.05	10.02
SP3SF	4.92	4.92	4.89	4.89	9.03	9.02	8.85	8.79	10.79	10.78	10.39	10.32
iSP3SF	4.92	4.92	4.89	4.89	9.03	9.02	8.85	8.79	10.79	10.78	10.39	10.31
$\alpha = 4$	10				50				100			
SPF	1.51	1.49	1.50	1.48	2.89	2.81	2.85	2.79	3.61	3.49	3.53	3.45
MIP	1.48	1.47	1.48	1.47	2.82	2.75	2.79	2.74	3.52	3.41	3.49	3.40
DSPF	1.49	1.48	1.48	1.47	2.82	2.76	2.80	2.76	3.52	3.42	3.49	3.42
iDSPF	1.47	1.46	1.47	1.46	2.78	2.73	2.76	2.73	3.46	3.37	3.43	3.36
DSP3SF	1.47	1.47	1.46	1.46	2.75	2.75	2.75	2.74	3.43	3.43	3.43	3.42
iDSP3SF	1.45	1.45	1.45	1.45	2.70	2.69	2.69	2.69	3.34	3.34	3.34	3.33
SP3SF	1.47	1.47	1.46	1.46	2.74	2.74	2.74	2.73	3.41	3.41	3.40	3.39
iSP3SF	1.47	1.47	1.46	1.46	2.74	2.74	2.74	2.73	3.41	3.41	3.40	3.39

Table 4. Effects of local search algorithms on augmentation algorithms for random instances with 100 nodes, $\alpha = 2$ and $\alpha = 4$, and group sizes $10, 50$ and 100. The four columns for each group size denote the energy of the algorithm, the algorithm followed by Sweep, the algorithm followed by EWMA, and the algorithm followed by executions of EWMA, Sweep and EWMA, respectively.

Algorithm	Time	Algorithm	Time	Algorithm	Time
SPF	12 msec	iDSPF	95 msec	DSF-3	1.2 sec
MIP	14 msec	iD2SPF	6 sec	DISF	11.6 sec
DSPF	6 msec	iSP3SF	250 msec	DFF	15.7 sec
D2SPF	265 msec	iDSP3SF	130 msec	DFF-2	18.1 sec
SP3SF	22 msec	i2SP3SF	10.7 sec	iDISF	349 sec
DSP3SF	6 msec	iD2SP3SF	6.7 sec	iDFF	137.6 sec
2SP3SF	1 sec	DSF	1.2 sec	iDFF-2	194.8 sec
D2SP3SF	280 msec	DSF-2	1.2 sec		

Table 5. Running time of the algorithms on random instances of 40 nodes and group size 40.

References

1. M. Cagalj, J.P. Hubaux and C. Enz. Minimum-Energy Broadcast in All-Wireless Networks: NP-completeness and Distribution Issues. In *Proc. of the 8th ACM International Conference on Mobile Networking and Computing (Mobicom '02)*, pp. 172–182, 2002.
2. G. Călinescu, S. Kapoor, A. Olshevsky and A. Zelikovsky. Network Lifetime and Power Assignment in Ad-Hoc Wireless Networks. In *Proc. of the 11th Annual European Symposium on Algorithms (ESA '03)*, LNCS 2832, Springer, pp. 114–126, 2003.
3. I. Caragiannis, C. Kaklamanis and P. Kanellopoulos. New Results for Energy-Efficient Broadcasting in Wireless Networks. *In Proc. of the 13th Annual International Symposium on Algorithms and Computation (ISAAC '02)*, LNCS 2518, Springer, pp. 332–343, 2002.
4. I. Caragiannis, C. Kaklamanis and P. Kanellopoulos. Energy-Efficient Wireless Network Design. *In Proc. of the 14th Annual International Symposium on Algorithms and Computation (ISAAC '03)*, LNCS 2906, Springer, pp. 585–594, 2003.
5. J. Cartigny, D. Simplot, I. Stojmenovic. Localized minimum energy broadcasting in ad hoc networks. In *Proc. of IEEE INFOCOM 2003*, 2003.
6. A.E.F. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca. On the Complexity of Computing Minimum Energy Consumption Broadcast Subgraphs. In *Proc. of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS '01)*, LNCS 2010, Springer, pp. 121–131, 2001.
7. A.E.F. Clementi, M. Di Ianni, R. Silvestri. The Minimum Broadcast Range Assignment Problem on Linear Multi-Hop Wireless Networks. *Theoretical Computer Science*, 299 (1-3), pp. 751–761, 2003.
8. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein. Introduction to Algorithms. *The MIT Press*, Second Edition, 2001.
9. S. Guha and S. Khuller. Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets. *Information and Computation*, 150(1), pp. 57–74, 1999.
10. L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power Consumption in Packet Radio Networks. *Theoretical Computer Science*, 243(1-2), pp. 289–305, 2000.
11. P.N. Klein and R. Ravi. A Nearly Best Possible Approximation Algorithm for Node-Weighted Steiner Trees. *Journal of Algorithms*, 19(1), pp. 104–115, 1995.
12. W. Liang. Constructing Minimum-Energy Broadcast Trees in Wireless Ad Hoc Networks. In *Proc. of 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC '02)*, pp. 112–122, 2002.
13. P. Mavinkurve, H.Q. Ngo and H. Mensa. MIP3S: Algorithms for Power-conserving Multicasting in Wireless Ad Hoc Networks. In *Proc. of the 11th IEEE International Conference on Networks (ICON '03)*, 2003.
14. P.-J. Wan, G. Călinescu, X.-Y. Li, and O. Frieder. Minimum-Energy Broadcasting in Static Ad Hoc Wireless Networks. *Wireless Networks*, 8(6), pp. 607–617, 2002.
15. J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In *Proc. of IEEE INFOCOM 2000*, pp. 585–594, 2000.
16. V. Verma, A. Chandak and H.Q. Ngo. DIP3S: A Distributive Routing Algorithm for Power-Conserving Broadcasting in Wireless Ad Hoc Networks. In *Proc. of the Fifth IFIP-TC6 International Conference on Mobile and Wireless Communications Networks (MWCN '03)*, pp. 159–162, 2003.