Extending Greedy Multicast Routing to Delay Sensitive Applications

Ashish Goel* Kameshwar Munagala†

Stanford University
July 29, 1999

Abstract

Given a weighted undirected graph $G(V, E)$ and a subset R of V, a Steiner tree is a subtree of G that contains each vertex in R. In this paper, we present an online algorithm for finding a Steiner tree that simultaneously approximates the shortest path tree and the minimum weight Steiner tree, when the vertices in the set R are revealed in an online fashion. This problem arises naturally while trying to construct source-based multicast trees of low cost and good delay. The cost of the tree we construct is within an $O(\log |R|)$ factor of the optimal cost, and the path length from the root to any terminal is at most $O(1)$ times the shortest path length. The algorithm needs to perform at most one reroute for each node in the tree. Our algorithm extends the results of Khuller *et al.*and Awerbuch *et al.*, who looked at the offline problem [9, 2]. We conduct extensive simulations to compare the performance of our algorithm (in terms of cost and delay) with that of two popular multicast routing strategies: shortest path trees and the online greedy Steiner tree algorithm.

*Dept. of Computer Science, Stanford CA 94305. Research supported by ARO Grants DAAG55-98-1-0170 and ASSERT award DAAG55-97-1-0221. Email: agoel@cs.stanford.edu

†Dept. of Computer Science, Stanford CA 94305. Research supported by ARO Grants DAAG55-98-1-0170 and ASSERT award DAAG55-97-1-0221 and by ONR Grant N00014-98-1-0589. Email: kamesh@cs.stanford.edu

1 Introduction

Online multicast routing is a problem of growing importance with the advent of multimedia applications. Application sensitive multicast routing is critical to the success of these applications. For multicast applications where end to end delay is of overriding importance but the bandwidth requirement is small (such as stock tickers), shortest path routing strategies perform well. A shortest path routing strategy connects the source of the multicast to each receiver using the shortest unicast route from the source to the receiver (or the other way round) in the underlying IP/ATM network. DVMRP [16], CBT [3], and PIM [5] are all examples of deployed routing protocols that use shortest path trees. In shortest path strategies, any sharing of routes by different receivers is incidental; the routing scheme itself makes no effort to share routes.

For applications which require large amounts of bandwidth, but are not delay sensitive, it is important to minimize the total cost (weight) of the multicast tree. The greedy online algorithm for finding small weight Steiner trees proposed by Imase and Waxman [8] has the best possible competitive ratio of $O(\log k)$, where k is the number of multicast receivers. In the worst case, the tree produced by shortest path strategies can be upto a factor k worse than the tree produced by the greedy algorithm. Even when all multicast receivers are chosen randomly from a very large grid, the shortest path algorithm produces trees which are expected to be a factor $\sqrt{k}$ worse than those produced by the greedy algorithm (Theorem 2.2). This demonstrates that shortest path routing strategies will not scale well for bandwidth intensive applications. This claim is also supported by extensive simulations by Doar and Leslie [6] and Waxman [18].

In this paper we focus on applications which are both bandwidth intensive and delay sensitive. Our goal is therefore to simultaneously approximate the shortest path tree and the minimum cost Steiner tree.

Formally, the problem we study is the following. We are given a weighted undirected graph $G = (V, E)$ with a cost function $c : E \to \Re$, and a source node s. Receivers $R = \{v_1, v_2, \ldots, v_k\}$ arrive in an online fashion. Our goal is to maintain a tree T connecting s to all the receivers that have arrived so far. Let $c(T) = \sum_{e \in T} c(e)$ be the total cost of T. Let $d_G(s, v)$ be the shortest path distance from s to v with respect to the cost function c, and let $d_T(s, v)$ be the path length from s to v in T. Also, for a receiver v, let $\mathrm{Stretch}_T(v) = d_T(s, v)/d_G(s, v)$. Now, define

$$\mathrm{Stretch}_T = \max_{v \in R} \mathrm{Stretch}_T(v)$$

and let

$$\mathrm{CostRatio}_T = c(T)/c(T^*),$$

where T^* is the minimum weight Steiner tree on the vertices in $R \cup \{s\}$. An algorithm for this problem is (p, q)-competitive if $\mathrm{Stretch}_T \leq p$ and $\mathrm{CostRatio}_T \leq q$ for the tree T produced by the algorithm. Our goal is to obtain an algorithm that simultaneously guarantees small values of p and q.

Khuller *et al.*looked at the offline version of this problem (the receiver set R is known in advance), and gave an $(O(1), O(1))$-approximation [9], extending the work of Awerbuch *et al.* [2]. Our problem can also be looked upon as a variation of the online shallow-light Steiner tree problem. In the shallow-light Steiner tree problem, there is a cost $c(e)$ and distance $d(e)$ associated with each edge e. Given a set of receivers R (which appear online), a source s, and a distance bound Δ, the goal is to find a tree T connecting all vertices in $R \cup \{s\}$, which is cheapest in terms of cost metric c (lightness) and where the distance $d_T(s, v)$ in terms of metric d is bounded by Δ. Our problem is easier since the metrics c and d are the same. The shallow-light Steiner tree problem was studied by [11, 4] in an offline setting. They gave an algorithm which violates Δ by at most a constant factor, approximates the cost by a poly-logarithmic factor, and which runs in quasi-polynomial time. In the online setting, various heuristics have been proposed [10, 14, 7], but each of these heuristics suffers from either a super-polynomial running time, or poor bounds on costs and distances.

Our main result is the *DSG* (Delay Sensitive Greedy) Algorithm, which is $(O(1), O(\log k))$-competitive, where $k = |R|$. We need to do a small amount of rerouting – each node in R gets rerouted at most once (A node is said to be rerouted when its parent pointer in the multicast tree changes). The weight of the tree constructed by the DSG algorithm is within $O(1)$ of the weight of the greedy online Steiner tree. Since the greedy Steiner tree has the best possible competitive ratio in terms of weight of the tree upto constant factors [8], we are within constant factor of the optimum in terms of both the weight and stretch. The running time of DSG is $O(|R|)$, over and above the running time of the greedy algorithm, assuming that all pairs shortest paths between the source and the receivers have been precomputed. The algorithm also has low communication overhead and is easy to implement in a distributed setting. Our algorithm is modeled after the elegant algorithm in [9] which we adapt to an online setting. Like their algorithm, our algorithm provides a continuous tradeoff between the weight of the tree and the stretch – we can obtain trees with smaller weight by relaxing the stretch, and vice-versa. We believe that our algorithm is simple enough to lend itself, like the greedy algorithm, to efficient implementations.

We simulate our DSG algorithm for Waxman networks [18] and grid networks. Our simulation results (Section 4) show that our algorithm's performance is comparable to the greedy algorithm in terms of the overall cost of the tree, and comparable to the shortest path algorithm in terms of the end to end delay. Also, the actual results observed in simulations are significantly better than the worst case approximation guarantees we prove in this paper. Further, most nodes in R never get rerouted.

2 Background: The Greedy Algorithm

The greedy algorithm [8, 1] for constructing small weight Steiner trees online does the following. Suppose we have already constructed a tree T. Let s be the source of the multicast tree and R be the set of already connected receivers. When a new node v requests to join the set R, the algorithm finds the node in T that is the closest to v, and attaches v to T via this node.

Assume that $|R| = k$. Let T_G denote the tree constructed by the greedy algorithm, T^* denote the shortest (optimal) tree, and T_S denote a shortest path tree. For any tree T, let $C(T)$ denote its total cost. Imase and Waxman [8] proved the following theorem.

Theorem 2.1. **[8]** $C(T_G) \leq \lceil \log k \rceil \, C(T^*)$. *Further, no algorithm can guarantee a factor better than* $\frac{\lceil \log k \rceil}{2}$.

In contrast, it is possible for $C(T_S)$ to be as large as $(k-1)C(T^*)$ in the worst case. The above result should make a clear case for the superiority of the greedy algorithm for minimizing the total cost of the multicast tree. However, one might object that the pathological networks which result in the large difference in the performance of the greedy algorithm and the shortest path approach do not occur in practice. To address this concern, we study the greedy and the shortest path algorithms in the following simple model. We assume that we have a completely connected network, and the receivers and the source for the multicast are drawn randomly from within a unit square, and the distance between nodes is the Euclidean distance.

Theorem 2.2. *The expected cost of the tree T^* is $\Theta(\sqrt{k})$. The expected cost of the tree T_G is also $\Theta(\sqrt{k})$. The expected cost of the tree T_S is $\Theta(k)$.*

The distance between two randomly chosen points on a grid is $\Theta(1)$; using linearity of expectations we immediately obtain $T_S = \Theta(k)$. $T^* = \Theta(\sqrt{k})$ is implicit in [17]; $T_G = \Theta(\sqrt{k})$ follows from [17] and [15]. Thus even when the multicast nodes are chosen probabilistically from a simple network (ie. the grid), the performance difference between the greedy algorithm and a shortest path approach is of the order of $\sqrt{k}$.

3 The DSG (*Delay Sensitive Greedy*) Algorithm

3.1 Description

We are given an undirected network G and with weights $d(e)$ on each edge e, and a pair (s, R) of source and receivers (this set grows in an online fashion). The tree T constructed by the DSG algorithm should have $\text{Stretch}_T = O(1)$, and the weight of the tree should be within an $O(1)$ factor of the weight of the greedy online tree.

Let T be the tree constructed so far for receiver set R. We root the tree T at s, and assume all edges in T are directed towards the root. Let β be the maximum permissible value of the ratio $d_T(s,v)/d_G(s,v)$ over all $v \in R$. Let α be any number between 1 and β (ie. $1 < \alpha < \beta$). The algorithm accepts α and β as parameters.

For any node v in tree T, let $\text{parent}(v)$ denote the parent of v in T.

The basic idea behind the DSG algorithm is quite simple. When a request to join R arrives, the requesting node v is first connected to the node in T which is the closest to v (this is just the greedy algorithm). As soon as it happens that $d_T(s,v) > \beta d_G(s,v)$ for the newly added receiver v, we reroute v so that it obeys the constraint $d_T(s,v) \leq \alpha d_G(s,v)$. Since $\alpha < \beta$, the above constraint is more strict than the one we are required to satisfy. Along with v some other nodes on the path, along T, from v to s may also get rerouted; these rerouted nodes also satisfy the more stringent constraint. No node gets rerouted more than once; most nodes never get rerouted (see Fig. 5).

We now present the DSG algorithm in detail. Since this is an online algorithm, we only need to specify what happens when a new receiver v requests to join the existing tree T. For any node $t \in T$, $\text{parent}(t)$ is the next node in the path along T from t to the source. We say that a node got rerouted when its parent changes. The algorithm works in four steps:

(1) JOIN: Let u be the node in T closest to the new receiver v. Augment the existing tree T by adding the shortest path from v to u. Notice that this is the same as the greedy algorithm.

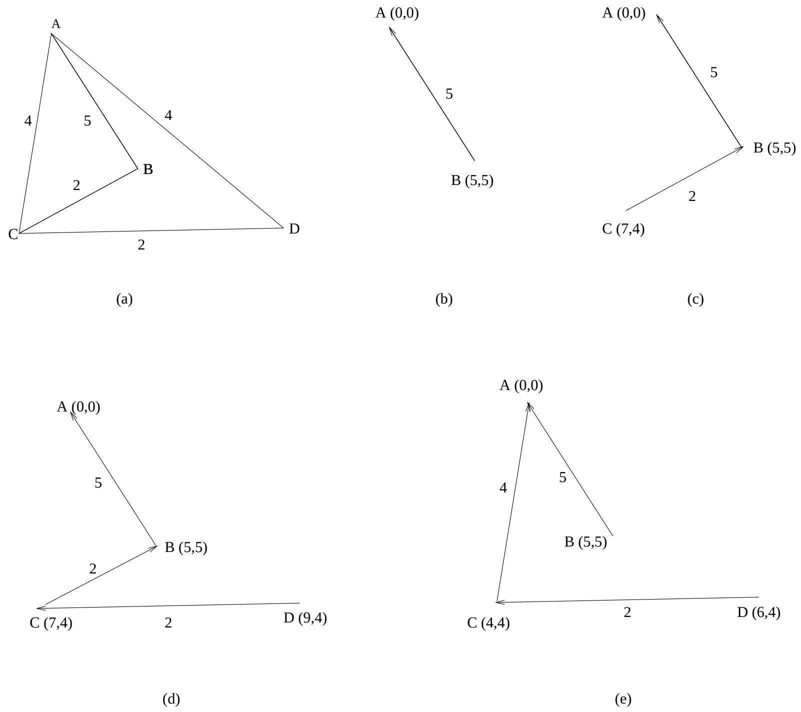

Figure 1: Execution of the Algorithm on a four node graph. The numbers in parentheses next to any node v indicate the current values of $d_T(s,v)$ and $d_G(s,v)$ respectively.

(2) CHECK**:** Check whether $d_T(s,v) \leq \beta \cdot d_G(s,v)$. If so, exit. If not, goto Step (3).

(3) CUT**:** Starting from v, traverse parent pointers in T and locate the first node v' such that $d_T(s,v') \leq \alpha \cdot d_G(s,v')$. "Cut" the path just downstream of v'. Let P be the path from v' to v, excluding v'. Direct path P from v to v'.

(4) RELAX**:** This is the main step where the DSG algorithm deviates from the greedy strategy. Let w be the child of v' on path P. Traverse path P starting at w and ending at v. For every node t on P encountered during the traversal, perform the following operation. If $d_T(s,t) > \alpha \cdot d_G(s,t)$, add the shortest path from s to t to the tree T, and delete the edge connecting t to parent(t) from T. This step is called the *Rerouting Step.*

The algorithm works even if each branching point in the multicast tree is a receiver. Therefore it can be run at the application layer or over a set of enabled routers, much like the MBone [13].

3.2 An Example

The DSG algorithm is best illustrated by means of this simple example (see Fig. 1). Let G be the 4 node graph shown in Fig. 1(a). We choose $\alpha = 1.6$ and $\beta = 2$, and let $s = A$.

We consider the sequence of addition of node B, followed by C, and finally D. When node B is added (Fig. 1(b)), it is connected to A, and so, parent$(B) = A$, and

$d_T(A, B) = 5$. When node C is added (Fig. 1(c)), it is connected in the greedy fashion to its closest neighbor in T, in this case, B. Therefore parent$(C) = B$ and $d_T(A, C) = 7$. Since $d_T(A, C)$ is less than β times $d_G(A, C)$, no rerouting takes place. When node D is added (Fig. 1(d)), it is first connected greedily to its closest node, C, setting parent$(D) = C$ and $d_T(A, D) = 9$. However $d_T(A, D) > \beta \cdot d_G(A, D)$, so we need to reroute it.

We start from D and follow parent pointers finding the first node v' satisfying the condition $d_T(A, v') \leq \alpha \cdot d_G(A, v')$. Note that C violates the α requirement, while B does not; hence with reference to the notation used in the description of the algorithm in steps 3 and 4, $v' = B$, $w = C$, $v = D$, and our path P consists of the nodes C and D. We reroute $w = C$ in Fig. 1(e) using the shortest path from C to A. This makes parent$(C) = A$, and $d_T(A, C) = 4$. This causes $d_T(A, D)$ to be set to 6 in sub-step (2) of step (4) of the algorithm. Since D now satisfies the α condition, it is not rerouted. The final tree is shown in Fig. 1(e).

3.3 Competitive Analysis

We now prove that the tree produced by the DSG algorithm is within a constant factor of the weight of the greedy tree, while all path lengths get dilated by at most a constant factor. We define a reroute as the change in the parent pointer of an existing node in the tree.

Theorem 3.1. *No node in the tree T generated by DSG gets rerouted more than once.*

Proof. Note that when a node v gets rerouted, the following two conditions always hold:

1. Just before rerouting, $d_T(s, v) > d_G(s, v)$. This is because the reroute is always along the shortest path to the source.
2. After rerouting, $d_T(s, v) = d_G(s, v)$, and this value remains the same as long as v is in T.

This implies that once a node gets rerouted, condition 1 is no longer satisfied for it, and so, it cannot get rerouted again. □

In practice, most nodes never get rerouted (see Fig. 5).

Theorem 3.2. *The cost of the tree produced by DSG is at most $\frac{\alpha(1-\alpha+2\beta)}{(\alpha-1)(\beta-\alpha)}$ times the cost of the tree generated by the greedy algorithm.*

Proof. Our analysis is similar to that used by Khuller *et al.*[9] for showing the existence of shallow spanning trees in any graph.

If the RELAX step does not reroute, the DSG algorithm adds exactly the path that would be added by the greedy algorithm. If this step actually results in a relaxation, we will show that the new paths added in that step are at most a constant factor multiple of the old path. This will prove the result.

Consider the path P that was relaxed in one particular RELAX step of the algorithm. Let us focus our attention on this step. For the purpose of analysis, let us include v' in the path. Let $v_1, v_2, \ldots, v_k$ be the sequence of nodes along P for which we were forced to add shortest paths to

s. Note that $v_1 = w$. Further note that the following inequalities are easily deducible from the algorithm:

1. $d_T(s, v') + d_G(v', v_1) > \alpha d_G(s, v_1)$.
2. $d_G(s, v_{i-1}) + L_P(v_{i-1}, v_i) > \alpha d_G(s, v_i)$ for $i = 2, 3, \ldots, k$, where $L_P(v_{i-1}, v_i)$ is the distance between nodes v_{i-1} and v_i along path P.

The value of $d_T(s, v')$ in the above inequalities is that at the beginning of the RELAX step. Adding the inequalities enumerated above, we get

$$d_T(s, v') + L_P(v', v_1) + \Sigma_{i=2}^{k} L_P(v_{i-1}, v_i) > (\alpha - 1)\Sigma_{i=1}^{k} d_G(s, v_i) \tag{1}$$

Let L_{new} denote the total length of the new paths added during the relaxation phase. Note that all the new paths are shortest paths, and their total length is clearly at most $\Sigma_{i=1}^{k} d_G(s, v_i)$. Using this fact, equation 1 can easily be simplified to:

$$L_P(v', v) + d_T(s, v') > (\alpha - 1) L_{new} \tag{2}$$

Note that since the greedy algorithm adds unrelaxed paths, the path it has added for inserting the vertices w through v on the path P is precisely the length of the path P including the edge (w, v'). Therefore, this phase of the greedy algorithm adds weight $L_P(v', v)$. The DSG algorithm will add paths of length at most $L_{new} + L(v', v)$ in the same phase. Since no vertex on P will ever get relaxed again, if we bound L_{new} in terms of $L(v', v)$, the same bound holds of the weight of the tree the DSG algorithm constructs in terms of the weight of the greedy online tree.

Observe that the following inequalities hold for the vertices v and v':

1. $d_T(s, v') < \alpha d_G(s, v')$,
2. $\beta d_G(s, v) < d_T(s, v') + L_P(v', v)$, and
3. $d_G(s, v') < d_G(s, v) + L_P(v, v')$, by triangle inequality.

From these, we deduce:

$$d_T(s, v') < \frac{\alpha(1+\beta)}{\beta - \alpha} \cdot L_P(v', v). \tag{3}$$

From equations 2 and 3, it follows that

$$L_{new} < \frac{\beta(1+\alpha)}{(\alpha-1)(\beta-\alpha)} \cdot L_P(v', v). \tag{4}$$

Therefore,

$$L_{new} + L_P(v', v) < \frac{\alpha(1 - \alpha + 2\beta)}{(\alpha-1)(\beta-\alpha)} \cdot L_P(v', v). \tag{5}$$

As observed above, the weight of the tree the DSG algorithm produces is at most $\frac{L_{new}}{L_P(v', v)} + 1$ times the weight of the greedy online tree. This proves the theorem. □

The factor in the above theorem is minimized when α is chosen to be $\frac{\beta\sqrt{\beta+1}+\beta\sqrt{2}}{\beta\sqrt{2}+\sqrt{\beta+1}}$. In our simulations, this factor is much smaller than that predicted in Theorem 3.2.

Theorem 3.3. *Let T be the tree generated by DSG. For all $v \in R$, $d_T(s,v) < \beta d_G(s,v)$.*

Proof. The proof follows trivially from the observation that we reroute v if it does not satisfy the condition stated in the theorem. □

Theorems 3.2 and 3.3 allow us to trade off the cost of the tree with the stretch in a controlled fashion, by changing parameters α and β. From the simulations we can see that for realistic networks, the guarantees for the cost are much better than these worst case predicted values. The tradeoff between end to end delay and overall cost is clearly discernible from our simulation results.

Theorem 3.4. *Let the greedy algorithm take total time τ to add all nodes of the graph to the multicast tree. Then, the DSG algorithm takes total time $O(n+\tau)$ to perform the same task assuming that all pairs shortest paths between the source and all receivers have been precomputed. Here, n is the number of nodes in the graph.*

The proof of theorem 3.4 is simple; each parent pointer in the tree gets visited at most once. If we allow only the receivers to be the branch points in the tree, the additional time changes to $O(|T|+\tau)$.

4 Simulation Studies

We have simulated the DSG algorithm on two classes of graphs, the Waxman networks and the rectangular grid graphs.

4.1 Waxman Networks

In this model [18] we choose n nodes uniformly at random from a square Cartesian grid on the plane. For every pair of nodes (u,v), we add an edge in G between them using the probability function:

$$P_e(u,v) = \frac{k\bar{e}}{|G|}\gamma \cdot exp(\frac{-d(u,v)}{\delta L})$$

where $d(u,v)$ is the Euclidean distance between u and v, L is the diagonal of the grid, γ and δ are parameters in the range $(0,1]$, $\bar{e}$ is the mean degree of each node, $|G|=n$ is the number of nodes in the graph, and k is a constant. The cost of an edge is simply its Euclidean length.

In our case, the values of the parameters are: $\gamma = 0.2$, $\delta = 0.25$, $\bar{e} = 4$, $n = 1000$ and $k = 25$. In other words, we will be considering 1000 node graphs with mean degree 4. The factors δ and γ are chosen to make the graph resemble geographical maps of major nodes on the Internet [6].

The simulation results are summarized in figures 2,3, 4 and 5. The curves corresponding to the DSG algorithm are labeled by the pair (α,β) that we use as parameters in the execution (see section 3.1). We run the DSG algorithm on ten random graphs each for three distinct pairs of

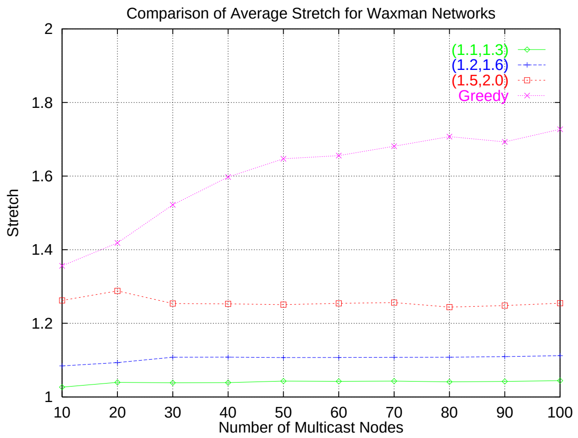

Figure 2: Comparison of Average Stretches for Waxman Networks with 1000 nodes. The numbers in parentheses are the values of α and β used by DSG.

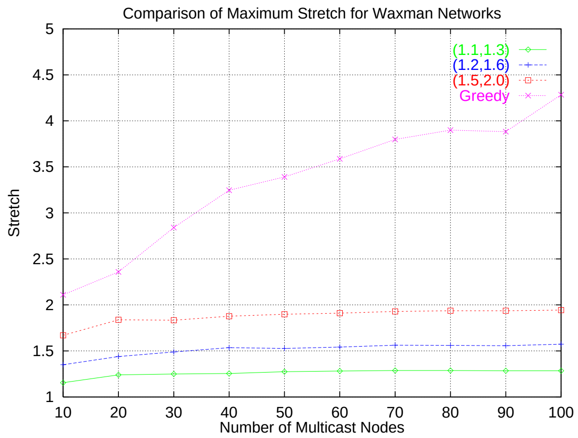

Figure 3: Comparison of Maximum Stretches for Waxman Networks with 1000 nodes. The numbers in parentheses are the values of α and β used by DSG.

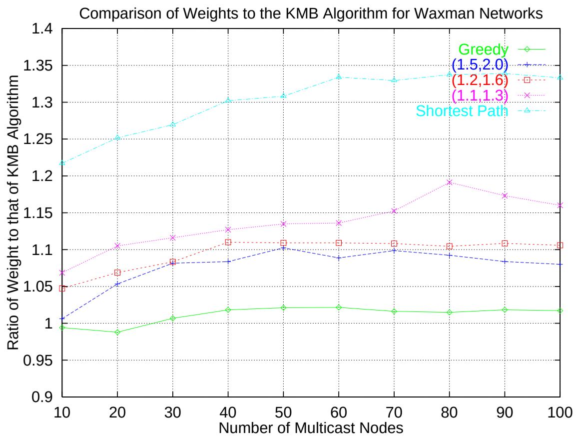

Figure 4: Comparison of Weights for Waxman Networks with 1000 nodes. The numbers in parentheses are the values of α and β used by DSG.

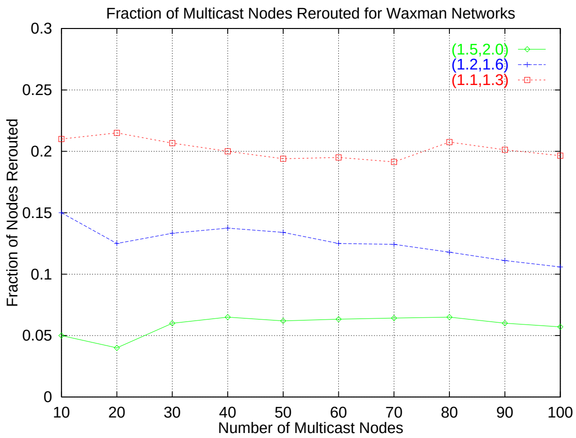

Figure 5: Fraction of the Multicast Nodes which are rerouted by the DSG algorithm for Waxman Networks with 1000 nodes. The numbers in parentheses are the values of α and β used by DSG.

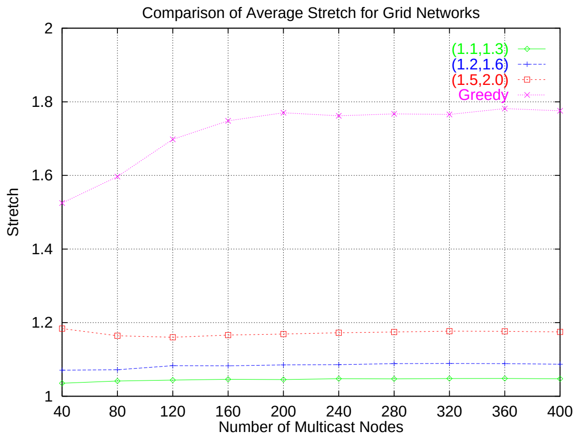

Figure 6: Comparison of Average Stretches for 200×200 Grid Networks. The numbers in parentheses are the values of α and β used by DSG.

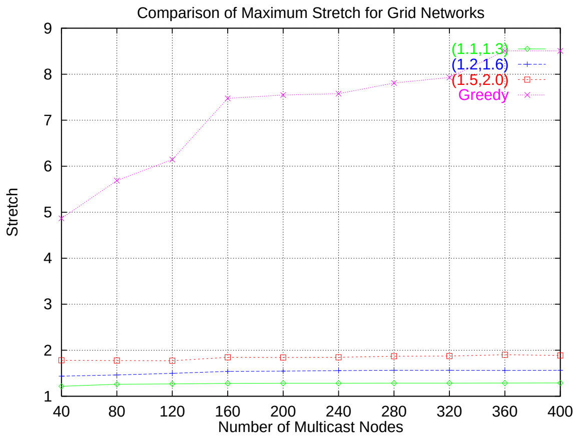

Figure 7: Comparison of Maximum Stretches for 200×200 Grid Networks. The numbers in parentheses are the values of α and β used by DSG.

(α, β) values (shown in the figures), and take the average for each pair. The source is chosen from the pool of nodes at random. We add randomly chosen nodes to the multicast tree in increments of 10, till we have added 100 nodes, which is 10% the size of the network. Readings are taken after every addition of 10 nodes.

In figures 2 and 3, we compare the average and maximum stretches that the DSG algorithm produces versus that produced by the greedy algorithm. In figure 4, we compare the weights of the trees generated by the DSG algorithm to those generated by the greedy algorithm and the shortest path algorithm. The y-axis gives the ratio of the weight to the weight of the tree generated by a commonly used heuristic known as the KMB Algorithm [12]. Finally, in Fig. 5, we plot the fraction of the nodes which are rerouted by the DSG algorithm.

The simulation results clearly support the claims made in previous sections. The weight of the shortest path trees is at least 20% away from the optimum, while the weight of the trees produced by the DSG algorithm and the greedy algorithm are very close to optimal. The DSG algorithm has bounded maximum stretch, but the average stretch is very close to one. This compares well with the stretch of the shortest path tree. The greedy algorithm, however, does very poorly in this regard. Observe also that there is a tradeoff between the weight of the tree our algorithm produces, and its stretch. By tweaking the values of α and β, we can reduce the stretch at the expense of increasing weight, or vice versa. The DSG algorithm guarantees that no node is rerouted more than once. But, the results clearly show that the fraction of nodes actually rerouted is much smaller (below 20%). This fraction can be reduced by increasing the values of α and β.

4.2 Grid Graphs

Here, we consider a rectangular grid of size $n \times n$. All points on the grid are nodes in the graph, and the only edges are horizontal or vertical edges between adjacent grid points. All edges have the same length.

We run the DSG algorithm on 200×200 grids. The source is chosen at random. We add

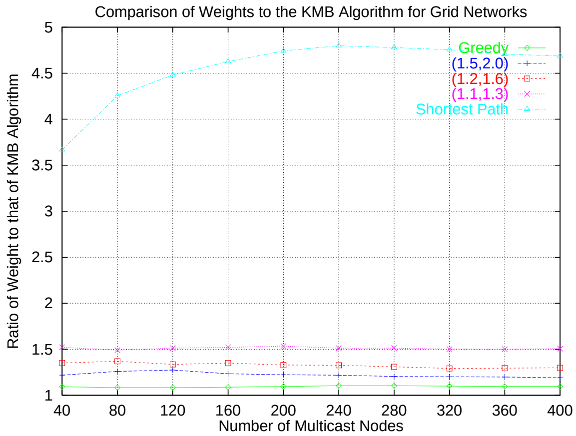

Figure 8: Comparison of Weights of the Trees for 200×200 Grid Networks. The numbers in parentheses are the values of α and β used by DSG.

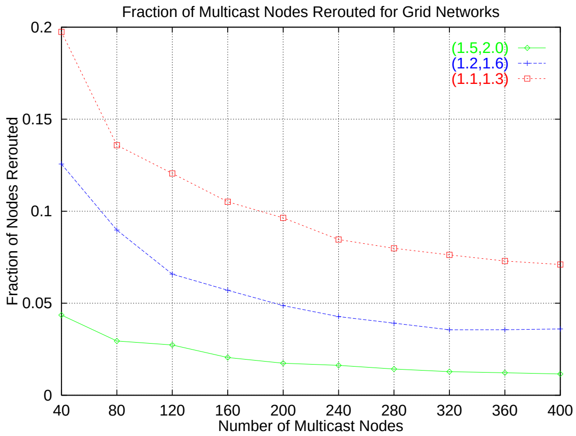

Figure 9: Fraction of the Multicast Nodes which are rerouted by the DSG algorithm for 200×200 Grid Networks. The numbers in parentheses are the values of α and β used by DSG.

nodes chosen at random in increments of 40, till we reach 400 nodes, which is 1% the total number of nodes. The paths we add between any two nodes are made to closely resemble straight line-segments, *i.e.*, for connecting any node to any other node, we use a uniform step-like path. For this purpose, we use the standard *Bresenham's Algorithm* from computer graphics.

These networks closely resemble the simple completely connected network model described in section 2. The paths we use for joining nodes to each other are nearly straight lines. The length of the lines is measured in the l_1 norm (since the distance between two nodes is the *Manhattan Distance*), instead of the usual Euclidean norm. But, this does not change the analysis given in section 2. Our results are summarized in figures 6, 7, 8, and 9. It is clear that the greedy algorithm produces trees with much worse stretches, and the shortest path algorithm produces trees with much worse weights than the corresponding trees for Waxman Networks. This conforms with the analysis presented in section 2, where it was shown that the weight of the shortest path tree could be $\Theta(\sqrt{k})$ times the greedy solution, where k is the number of nodes in the multicast. These results reinforce the fact that even for simple networks, the greedy and shortest path algorithms could perform very badly in terms of the delays and total cost respectively.

5 Conclusions

We present an online algorithm that produces Steiner trees with small weight as well as stretch. Our technique extends the greedy construction of [8] and that of Khuller *et al.* [9]. The algorithm has a small running time and communication overhead, and guarantees a total cost of the tree within $O(\log |R|)$ times the optimal, and per-receiver stretch of $O(1)$. We need to perform some amount of rerouting whenever a node gets added or deleted from the system. Each receiver gets rerouted at most once, and in our simulations most receivers never get rerouted. Extending the DSG algorithm for the case when we have two different metrics on the edges remains an interesting open problem.

References

[1] N. Alon and Y. Azar. On-line steiner trees in the euclidean plane. *Discrete and Computational Geometry*, 10(2):113–121, 1993.

[2] B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensitive analysis of communication protocols. *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 177–87, 1990.

[3] A. Ballardie, P. Francis, and J. Crowcroft. Core based trees(cbt) - an architecture for scalable inter-domain multicast routing. *ACM SIGCOMM*, pages 85–95, 1993.

[4] M. Charikar, C. Chekuri, T. Cheung, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed steiner problems. *Ninth Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 192–200, 1998.

[5] S. Deering, D.L. Estrin, D. Farinacci, C. Jacobson, V. Liu, and L. Wei. The pim architecture for wide-area multicast routing. *IEEE/ACM Trans on Networking*, 4(2):153–152, April 1996.

[6] M. Doar and I. Leslie. How bad is naive multicast routing. *IEEE INFOCOM*, pages 82–89, 1992.

[7] Sung-Pil Hong, Heesang Lee, and Bum Hwan Park. An efficient multicast routing algorithm for delay-sensitive applications with dynamic membership. *Proceedings IEEE INFOCOM*, pages 1433–40, 1998.

[8] M. Imase and B. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, August 1991.

[9] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning and shortest path trees. *Algorithmica*, 14(4):305–321, 1994.

[10] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–92, June 1993.

[11] G. Kortsarz and D. Peleg. Approximating shallow-light trees. *Proceeding of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 103–110, 1997.

[12] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15:141–45, 1981.

[13] M. Macedonia and D. Brutzman. Mbone provides audio and video across the net. *IEEE Computer*, pages 30–36, April 1994.

[14] M. Parsa, Qing Zhu, and J.J. Garcia-Luna-Aceves. An iterative algorithm for delay-constrained minimum-cost multicasting. *IEEE/ACM Transactions on Networking*, 6(4):361–74, 1998.

[15] Y.T. Tsai and C.Y. Tang. An on-line algorithm for computing low distance spanning trees on euclidean space. *Computer Science and Informatics*, 22(2):3–16, 1992.

[16] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. *Internet RFC 1075*, 1988.

[17] N.B. Wang and R. Chang. An upper bound for the average length of the euclidean minimum spanning tree. *International Journal of Computer Mathematics*, 30(1-2):1–12, 1989.

[18] B. Waxman. Routing of multipoint connections. *IEEE J. on Sel. Areas in Commun.*, 6(9):1617–22, 1988.