Bounds for the On-line Multicast Problem in Directed Graphs

Michalis Faloutsos * Rajesh Pankaj Kenneth C. Sevcik

Abstract

On-line multicasting has received a lot of attention lately, because of its great practical importance. However, most research assumes undirected graphs, which fail to model the directivity of real networks. In this paper, we fill this gap by offering upper and lower bounds for the on-line directed multicast problem in a worst case analysis.

We examine two types of on-line problems: the *join-only* problem where destinations join arbitrarily but stay till the end of the session, and the *join-leave* problem, where destinations can join and leave arbitrarily. We define the asymmetry of a graph to be the maximum ratio of opposite directed edges between a pair of nodes for all node-pairs. For the join-only on-line problem, we prove that a greedy on-line algorithm is $\log(A+1)$-competitive compared to *any* on-line algorithm. For the join-leave problem on both undirected and directed graphs, we prove a new lower bound, which is exponential in the previous undirected bound (the number of joins requests). We, then, show that, for highly asymmetric graphs, the inefficiency is proportional to the asymmetry, i.e., arbitrarily larger than the number of join requests.

Keywords: On-line multicast directed bounds

1 Introduction

In the last years, multicast routing has attracted a lot of attention from the networks community. Many emerging applications are of multicast nature, for example, we can mention video-on-demand, teleconferencing, tele-education, computer supported collaborative work. A multicast connection can substitute many unicast connections of the same information, and this way, we use the network resources more efficiently. Furthermore, in several types of applications the membership can change during the connection. For example, we can think of the case where the participating tele-students join the session at different times. Surprisingly, most theoretical work so far focuses on undirected graphs. Such results can only be considered as an approximation, since the directivity of the traffic in real networks can only be modeled by directed graphs. In this paper, we fill this gap by studying the on-line directed multicasting problem. We demonstrate that the link-asymmetry of the directed graphs makes the problem more difficult to approximate.

The multicast or Steiner tree problem on undirected graphs has been studied quite extensively, and can be stated as follows. Assume an undirected weighted graph $G(V,E)$ and a subset of **participant** nodes whose cardinality we denote by M, while $N = |V|$. We want to find the minimum weight tree that spans the participants. The problem is NP-complete and algorithms for the exact solution have computational complexity exponential in the number of nodes or of the participants [DW72], which raises the need for good approximations with polynomial complexity.

*Designated Contact Author. E-mail: **mfalou@cs.toronto.edu**. Telephone: +416-978-5182.

A common measure of the quality of an approximation is the **worst case inefficiency (WCI)** or **competitive ratio** of a multicast algorithm, which is defined as the maximum ratio of the cost of the tree of the algorithm over the optimal one. There exists a number of 2-competitive off-line algorithms [TM80] [KMB81] [RSC86] [Meh88].

Most previous work assumes undirected graphs. However, in practice, a communications network is usually a directed graph where every bidirectional link corresponds to two opposite directed edges. Recently, Ramanathan [Ram96] proved that a constant-factor approximation algorithm is highly unlikely for the directed version of the problem. Faloutsos et al. [FPS96] prove a tight bound for the greedy multicast algorithm on directed graphs.

The on-line multicast problem has two variations: the **join-leave** case where destinations join and leave during a session, and the **join-only** case where destinations do not quit a session. For both problems, we define M to be the number of the join requests plus one[1]. For the undirected join-leave problem, Imase and Waxman [IW91] prove a lower bound of M, a bound that we increase to 2^M. For the undirected join-only problem, Bharath-Kumar and Jaffe [BKJ83], later Imase and Waxman [IW91] proved that the WCI of the on-line NPF is tightly bounded by $\log(M)$. In a more recent work, Awerbuch and Azar [AA95] study the join-only problem for multiple interleaved sessions on undirected graphs, and propose a $(\log N \log M)$-competitive algorithm with respect to the minimization of congestion. A survey in progress can be found by Awerbuch [Awe96].

The on-line directed multicast problem can be formulated as follows. Assume a directed graph the source of the multicast connection, and a sequence of join-leave requests (the join-only problem has zero leave requests). We want to connect each node of the sequence in order of appearance with a path from the source, one at a time without any knowledge for the rest of the join or leave requests. Of course, when a destination leaves, we prune the edges that do not lead to a destination. THe goal is to minimize the cost of the multicast tree. Finding the optimal solution is hard, as we already said, so, in this paper, we examine how close to the optimal off-line solution we can get.

1.1 Basic Concepts and Results

We first present the concepts that research in this area uses, and then present previous and new results. For the on-line problem, the notion of the worst case inefficiency is defined similarly to the off-line case, and the optimal off-line cost is compared with that of the algorithm. In this paper, we study the on-line extension of a greedy algorithm suggested by Takahashi and Matsuyama [TM80]. In the on-line extension of this algorithm, we start from the root and construct the multicast tree by adding one participant at a time. At each step, we connect the new (next in sequence) destination to the existing multicast tree with the minimum cost path. We call this algorithm **on-line Nearest Participant First (NPF)** or just **greedy**. For the off-line undirected problem, NPF has a worst case inefficiency of $2 \cdot (1 - \frac{1}{l})$ where l is the number of leaves in the optimal tree [FPS97]. Recall the naive multicast algorithm, which we call **Shortest Paths**: the tree is the union of the shortest paths between the source and each destination. Clearly, this algorithm can be extended to any on-line problem for any asymmetry of graph. The worst case inefficiency of this algorithm is tightly bounded by $O(M)$ [FPS97].

In a directed graph, we call **opposite** edges the edges between any two nodes. In other words, for $v, w \in V$, (v, w) and (w, v) are opposite edges. We also denote $e(v, w)$ the weight of edge (v, w).

Definition 1 *The maximum asymmetry of a graph $G(V, E)$, A, is defined as*

[1] The addition of one accounts for the source, and we do it for compatibility with the definition of M for the static problem.

$$A = \max_{(v,w)\in E} \left(\frac{e(v,w)}{e(w,v)}, \frac{e(w,v)}{e(v,w)} \right)$$

With the above definition, the undirected case corresponds to a maximum asymmetry of $A = 1$. Note that, without loss of generality, we can assume that there are no multiple edges on the same direction between each pair of nodes: we are only interested in the least cost edge from each direction.

In our study, we examine the worst case performance of the algorithms. We consider deterministic algorithms, and so our worst case is equivalent to the case where we are so "unlucky" that our routing decisions are the worst with respect to the future destinations. Thus, in our examples, once we have routed the previous destination, the on-line sequence will select the "worst" node as the next destination. Furthermore, without loss of generality, we assume that an on-line algorithm creates a tree adding a path for each new destination. One could think that we can create more than one path for one node to facilitate future connections. However, we do not gain anything in the long run, since we can create the extra path the moment we need it and probably only the part of it that is actually needed [Bor].

In this work, we do not consider rerouting. Clearly, rerouting can reduce the cost of the multicast tree, but it is not a trivial task. Rerouting interrupts the communication, and thus, it can be undesirable in many applications and network technologies. Clearly, if we can reroute the whole tree for each new destination, we just have to solve a number of off-line problems. In rerouting problems, we can consider a limited number of global rerouting procedures [IW91] or local rerouting [KK95].

In an overview, the main contribution of this paper is a proof that, for the join-only problem, the greedy algorithm has near-optimal on-line performance. Namely, we show that the greedy algorithm performs as well as *any* on-line algorithm, except for the case of less asymmetric graphs ($A = O(M)$), where the performance may differ by $\log(A+1)$. A second contribution is the proof that for the join-leave problem, the greedy algorithm has large worst case inefficiency. First, we prove that for both undirected and directed graphs, the lower bound is 2^M (previous bound M for undirected graphs [IW91]). Then, for directed graphs, we prove that the greedy competitive ratio is also proportional to the asymmetry and thus, for high asymmetry values, unbounded with respect to the number of the join-requests. Finally, we group all the results for the on-line multicast problem in Table 1. Putting things in perspective, we consider our work a generalization of the work of Imase and Waxman [IW91] on undirected graphs, which inspired our research. Since undirected graphs can be viewed as a special case of directed graphs, our results include their results for asymmetry $A = 1$. In our conclusion, we present all the results in Table 1.

In the first sections of this paper, we discuss the join-only problem, while in section 5, we address the join-leave problem. Namely, in section 2, we present the upper bounds for the on-line NPF algorithm. In section 3, for highly asymmetric graphs (asymmetry of $\Omega(M)$), we prove a lower bound for any on-line algorithm, which is also the upper bound of the greedy algorithm. In section 4, for less asymmetric graphs, we prove a lower bound for any on-line algorithm. In section 5, we examine the lower bound for the join-leave on-line multicast problem for both undirected and directed graphs.

2 The Upper Bound

In this section, we examine the upper bound of the greedy on-line algorithm that we call on-line NPF or simply NPF. First, for any asymmetry value, we prove that NPF is M competitive. Second, we examine graphs of bounded asymmetry, and we prove that NPF is $A \cdot \log(M)$ competitive.

First, we prove that NPF is M-competitive. This is a direct consequence of the greedy nature of the algorithm, and becomes significant for highly asymmetric graphs mainly. Note that in this section, we will use the subscript d in most values that refer to the directed problem, while in the other sections we refer only to directed graphs and so the subscript is not needed.

Theorem 1 *For the on-line directed problem, the cost of the solution of NPF, T_d is bounded*

$$T_d \leq M \cdot OPT_d$$

where M is the number of participants and OPT_d the cost of the optimal solution.

PROOF. Let d_{max} be the maximum distance from the root for all destinations. A greedy solution will never be more expensive than $M \cdot d_{max}$, which is an upper bound on the union of the source-to-destination paths. On the other hand, the optimal solution OPT_d is at least d_{max}. Therefore,

$$\frac{T_d}{OPT_d} \leq \frac{M \cdot d_{max}}{d_{max}} \leq M$$

■

Note that the arguments of the previous theorem hold for the Shortest Paths algorithm. Furthermore, it is easy to prove that the previous bound is tight for the Shortest Paths: assume a star graph with the source in the middle and the destinations at the spikes, and then introduce edges of infinitesimal weight among destinations [FPS97].

Second, we prove an upper bound with respect to the asymmetry of the graph.

Theorem 2 *Assume a weighted directed graph with maximum asymmetry A and a multicast session of M participants. The following holds for the NPF algorithm:*

$$T_d < A \cdot \log(M) \cdot OPT_d$$

where T_d is the cost of the NPF tree and OPT_d the cost of the optimal one.

PROOF. In an overview, we transform the directed graph into an undirected graph in which the weight of each edge is minimum of the two directed edges. Then, we perform a greedy on-line algorithm on the undirected graph. The corresponding directed greedy algorithm is $O(A)$ competitive with respect to the greedy undirected, and the greedy undirected is $O(\log M)$ competitive with respect to the undirected optimal solution. Below we expand the previous steps and examine some details of the proof.

Let $G_d(V, E)$ be the directed graph, and let us create an undirected graph $G_u(V, E_u)$ as follows. For every pair of opposite directed edges in G_d, we create an undirected edge in G_u with the minimum weight of the pair.

$$e_u(v, w) \leq \min\left(e_d(v, w), e_d(w, v)\right) \qquad v, w \in V \tag{1}$$

If OPT_u is the optimal tree on G_u, clearly

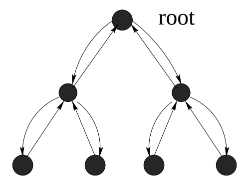

Figure 1: The d-tree : a complete binary directed tree.

$$OPT_u \leq OPT_d \tag{2}$$

Let us execute a variation of NPF on G_u for the on-line problem that we call **restricted** NPF. Namely, we consider connecting each new destination with the shortest path from a previous destination or the source[2]. We call T_u the cost of the resulting tree.

For the undirected on-line problem, Imase and Waxman proved (Lemma 3 [IW91]) that the cost of our restricted greedy solution, T_u, is bounded as follows

$$T_u \leq \log(M) \cdot OPT_u \tag{3}$$

Let us examine closely now the execution of NPF on G_d, and recall that for each destination, we find the shortest path to the existing tree. For each destination, we can use the path that restricted NPF used for the same destination. In the worst case, this path will be A times more expensive. Since each tree is the sum of its paths, the following holds:

$$T_d \leq A \cdot T_u \tag{4}$$

We can combine the right hand side of each formula with the left hand side of the next one in order (2), (3), (4), as follows.

$$T_d \leq A \cdot T_u \leq A \cdot \log M \cdot OPT_u \leq A \cdot \log M \cdot OPT_d$$

This concludes the proof. ∎

We combine the above two theorems in the following corollary.

Corollary 1 *The worst case inefficiency of the on-line greedy solution to the directed multicasting problem is bounded as below*

$$\frac{T_d}{OPT_d} \leq \min(M, A \cdot \log M)$$

At this point, we can quantify the term "highly asymmetric" to imply graphs of asymmetry $\Omega(M)$. Note that $\Omega()$ denotes an asymptotic lower bound.

3 Highly Asymmetric Graphs: a Lower Bound

In this section, we prove that for highly asymmetric graphs the worst case ratio of any on-line algorithm is proportional to the number of destinations. Namely we prove the following theorem.

[2]Note that we use the shortest path between participants and this may cause us to count an edge multiple times in the total cost.

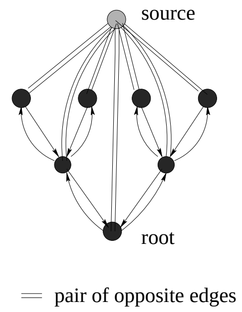

Figure 2: The worst case example for highly asymmetric graphs. Edges from or to the source are of cost c. From the rest, curved edges are of cost c, while their opposite edges are of cost ϵ.

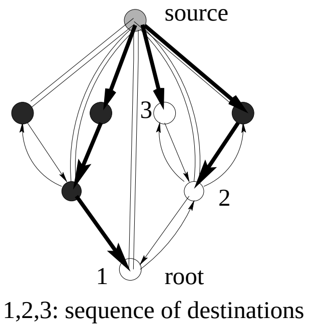

Figure 3: The graph can be interpreted in two ways: (A) The worst case example for highly asymmetric graphs for a possible algorithm, (B)In the next section: The worst case online sequence for graph F_1.

Theorem 3 *Given the size of the multicast group, M, and an on-line algorithm, there exists a directed graph, and a sequence of nodes for which the performance of the algorithm is bounded from below by*

$$\frac{T}{OPT} = \Omega(M)$$

where T is the cost of the tree of the algorithm and OPT is the optimal off-line cost.

PROOF. We will first present the intuition behind the graphs that we use. We want to create graphs that can be "bad" for the routing decision of any algorithm. For this, we have to ensure that, the on-line algorithm for every destination must have two routing choices at least. In addition, for either routing choice, the arrival sequence must have the possibility to be equally bad. In other words, we need "two identical paths" for every arriving destination, and this suggests a binary tree configuration (see Fig.1).

Definition 2 *A* **d-tree** *is a complete binary tree that has pairs of opposite directed edges instead of undirected edges.*

Assume a d-tree with $M-1$ distinct levels (see Fig.1). We assign weight c to the edges "pointing away" from the root (curved edges), and weight ϵ to the opposite edges (straight edges) We will use the notion of a "child" as in the undirected trees. Let the source be a node outside the d-tree and connected to all the nodes of the d-tree with cost c (see Fig.2). We can assume that the opposite of the edges "from" the source are of cost c.

For the on-line sequence, we choose first the root of the tree, then we choose a child of the previous destination that does not belong to the multicast tree (see Fig.3). Given this, the destinations form a path from the d-tree root to the source. In more detail, for each destination, the algorithm may choose a path along the d-tree, or some other way that will contain a direct edge to the source or the previous destination. This way, every destination is connected with a cost of at least c, and the total cost becomes

$$T = (M-1) \cdot c$$

As we already mentioned, there exists a path from the root of the d-tree to the source that contains all the destinations. Therefore, the optimal cost is

$$OPT = c + (M-2) \cdot \epsilon$$

If we choose $\epsilon = O(c/M)$, and use the two previous equations, we get:

$$\frac{T}{OPT} = \frac{(M-1) \cdot c}{c + O(c)} = \Omega(M)$$

■

As for the asymmetry of the previous graph, we can see that the maximum asymmetry appears at the edges of the d-tree : $\frac{c}{\epsilon}$. Given our previous choice of ϵ, the asymmetry of the graph is at least as large as $\Omega(M)$.

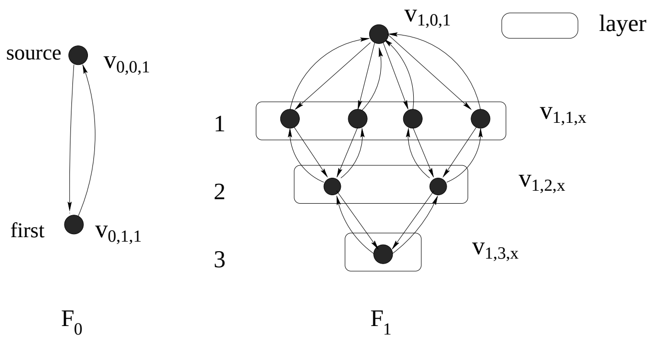

Figure 4: The family of F_g with $k = 2$: F_0 and F_1

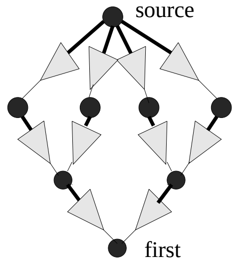

Figure 5: The family of F_g with $k = 2$: F_2. The triangles represent d-trees whose root is a node of the previous generation. The thick edges represent many pairs of edges that we merge for the clarity of the layout.

4 Less Asymmetric Graphs: a Lower Bound

The performance of greedy algorithms improves as the asymmetry decreases. As we saw, less asymmetric graphs ($A = O(M)$), have an upper bound ($A \log M$) for the worst case inefficiency of the greedy algorithm. Two questions are being raised: how tight this upper bound is (for our greedy algorithm), and, whether we can expect other on-line algorithms to do better. Our answer to both questions is that no on-line algorithm can perform better than $A \frac{\log M}{\log(A+1)}$ in a worst case analysis.

Theorem 4 *Given the size of the multicast group, M, a bound on maximum asymmetry $A = O(M)$, and an on-line algorithm, there exists a graph and a sequence of nodes for which the performance of the algorithm is bounded from below by*

$$\frac{T}{OPT} = \Omega\left(A \cdot \frac{\log M}{\log(A+1)}\right)$$

where T is the cost of the tree of the algorithm and OPT is the optimal cost.

PROOF. The proof consists of two parts. First, we describe how we construct a graph given the asymmetry and multicast group size, and then, we define the on-line sequence of nodes.

PART I: Graph construction.

Before the detailed description, we present the intuition behind the family of graphs that we denote by F_g. The building block of our graphs is the d-tree that we defined earlier. The d-tree guarantees that the on-line algorithm has two identical routing choices for each destination.

We construct the graph in a recursive manner, until we reach the size we want (the terminating condition will be specified later). We start from a pair of nodes, and at each step we replace the edges of adjacent nodes with a d-tree of depth k (eventually we choose k equal to A). We call **generation** (denoted by g) the number of the steps of the graph.

It is already clear that the graph construction involves the addition of a large number of d-trees. Therefore, in identifying nodes, we cannot use the depth of a d-tree, since this depth of a d-tree is relative to the given d-tree. Given that we need an identifier relative to the whole graph, we extend the concept of depth to an estimate of the distance from the source (see Fig.4).

Definition 3 *The* **layer** *of a node in the graph F_g is equal to the length (number of edges) of the shortest path from the source to the node.*

We use the definition of the length of the path in terms of edges.

A node of the F_g graphs is denoted by $v_{g,l,x}$, where g is the generation, l is the index of the layer as previously defined, and x an index that identifies nodes of the same layer (see Fig.4). In most cases, it will not be necessary to define values for x, and for this, x will imply the appropriate index.

The recursive creation procedure. We outline briefly the procedure. We start with a pair of nodes one of which is the source (see Fig.4). For every generation, we insert a binary tree between every pair of nodes of the previous generation (see Fig.4 and Fig.5). The edges are consistently asymmetric; from the source and "down" they are light (straight edges), while the opposite edges are heavy (curved edges). As we will see later, the weights of the edges are set in such a way that preserve the distances: all paths between the two nodes of F_0 are of equal cost.

Initial Graph. The initial graph, F_0 has two nodes, $v_{0,0,1}$ and $v_{0,1,1}$ (see Fig.4). We consider $v_{0,0,1}$ to be the source of the multicast session. Node $v_{0,1,1}$ is denoted as **First**. There are two directed edges between the nodes:

$$e(v_{0,0,1}, v_{0,1,1}) = \rho_0 \tag{5}$$

$$e(v_{0,1,1}, v_{0,0,1}) = A \cdot \rho_0 \tag{6}$$

Creating the Next Generation. Given the graph F_{g-1}, we create the graph of the next generation F_g. We substitute every pair of opposite edges in F_{g-1} by a d-tree(see Fig.4 and Fig.5). Assume the pair of nodes $v_{g-1,l,x}$ and $v_{g-1,l+1,x}$ of F_{g-1}. Node $v_{g-1,l+1,x}$ becomes the root of the new d-tree. Node $v_{g-1,l,x}$ is connected with all the leaves of the tree. The new nodes introduced in the generation g are denoted as W_g, while the set of all the nodes in generation g is denoted by V_g.

Note that the "old" nodes can be identified with indices of the new generation. The new layer indices are $k+1$ times the previous ones:

$$v_{g-1,i,x} = v_{g,i\ (k+1),x}$$

The weights of the edges between adjacent nodes are set as follows. In the equations below, x replaces indices of adjacent nodes.

$$e(v_{g,l,x}, v_{g,l+1,x}) = \rho_g \tag{7}$$

$$e(v_{g,l+1,x}, v_{g,l,x}) = A \cdot \rho_g \tag{8}$$

$$\rho_g = \frac{\rho_0}{(k+1)^g} \tag{9}$$

We describe some important properties of the previous graphs. First, it is trivial to notice that, by construction, the asymmetry of the graph is bounded by A. Second, when we insert a d-tree between two nodes, the distance (cost of shortest path) between them does not change. Actually, the weights of the edges are chosen to maintain the distances between nodes. Generalizing the above, we make the following observation.

Observation 1 *All paths between the initial pair of nodes are of equal cost.*

Finally, we want to relate the number of layers with the generation index. Using induction on g, we can prove the following relation between the layers and the generation of a graph.

Observation 2 *The number of layers, l, of graph F_g is exponential in the generation index g*

$$l = (k+1)^g + 1$$

PART II: The on-line sequence of destinations.

Given a graph F_g, we choose one node from each layer along a path from the First to the source. Namely, we choose the nodes in order of increasing generation number, and within the same generation in order of decreasing layer. For example, we present the beginning of the sequence of nodes (listed by generation):

- $v_{0,1,x}$
- $v_{1,k,x}, v_{1,k-1,x}, \ldots, v_{1,1,x}$
- $v_{2,m,x}, v_{2,m-1,x}, \ldots, v_{2,1,x}$, where $m = k \cdot (k+1)$, except for the nodes that correspond to destinations of the previous generation (that have already joined).

As we already mentioned, the number of destinations we choose is equal to the number of layers. Using Observation 2, the relation between the number of generations and the number of participants is given below.

Observation 3 *If the number of participants of the graph F_g is M, the maximum number of generations g_m is tightly bounded by*

$$g_m = \Theta\left(\frac{\log M}{\log(k+1)}\right)$$

Without loss of generality, we can assume that the number of participants, M, is a power of $k+1$ minus one.

In more detail, for each destination, say $v_{g,l,x}$, the on-line algorithm has to select one of the two paths to the source or the edge to the previously connected destination. After the routing decision is made, we choose a node of the next (smaller by one) layer , $v_{g,l-1,x}$, that does not belong in the selected path and is adjacent to the previous destination $v_{g,l,x}$ (see Fig.3). If $v_{g,l,x}$ is connected to the previous destination, we can choose either of the adjacent nodes. In the next, generation we choose nodes from d-trees that are between already connected nodes. This way, it is easy to see the following.

Observation 4 *Given the above on-line sequence on graph F_g, there exists a path from the source to the First node that contains all the destinations.*

For the above on-line sequence, we will find a **lower bound** on the cost for any on-line algorithm that we will denote T. As we said above, there exists a path from source to First that contains all the destinations. Recalling Observation 1, the cost of the optimal solution is given below

$$OPT = \rho_0 \tag{10}$$

At this point, we set k equal to A. We did not do this earlier in order to distinguish the two parameters.

$$k = A \tag{11}$$

Given that from each layer we select a destination, k is less than M. Therefore, Equation 11 suggests an **upper bound on the asymmetry of the graph**:

$$A \leq M \tag{12}$$

Calculation of the Cost (Lower Bound). The lower bound T of the cost of the any on-line algorithm is equal to the sum of the lower bounds of the costs of each generation that we denote by T_g :

$$T = T_0 + \sum_{g=1}^{g_m} T_g \tag{13}$$

It is easy to see that $T_0 = \rho_0$. We will now calculate the value of a lower bound (T_1) of the destinations added in F_1, and then generalize the result for the other generations. In F_1, a destination can either connect to the source or to another destination (see Fig.3). Given the asymmetry (see Eq.11), the minimum cost connection for a destination is the path to the source. This way, the cost of joining a node is at least as large as the cost of that path. Note that all paths are edge-disjoint, because of the way we choose our destinations. Therefore, the sum of all these paths is a lower bound of the cost of this generation:

$$T_1 = \sum_{i=1}^{k} i \cdot \rho_1 = \frac{k(k-1)}{2} \cdot \rho_1 = \frac{k(k-1)}{2} \cdot \frac{\rho_0}{k+1} \tag{14}$$

For the cost of generation g, it is not difficult to see that for every added d-tree , we can calculate the cost in a similar way and we will end up with an expression similar to the one above only with ρ_g instead of ρ_1. At each step, the number of these d-trees (that contain destinations) is equal to the number of F_{g-1} destinations minus one. Using Observation 2, we have $(k+1)^g$ such trees. Thus, the total cost of generation g is given below:

$$T_g = (k+1)^{g-1} \cdot \rho_g \cdot \frac{k(k-1)}{2} = \frac{\rho_0}{k+1} \cdot \frac{k(k-1)}{2} \tag{15}$$

It is interesting to note that T_g is constant for all the generations. Given that $k = A$ (Eq.11), the total cost from Eq.13 becomes:

$$T = T_0 + \sum_{g=1}^{g_m} T_g = \rho_0 + \frac{\log M}{\log(k+1)} \cdot \frac{\rho_0}{k+1} \cdot \frac{k(k-1)}{2} = \Theta\left(A \cdot \frac{\log M}{\log(A+1)}\right) \cdot \rho_0 \tag{16}$$

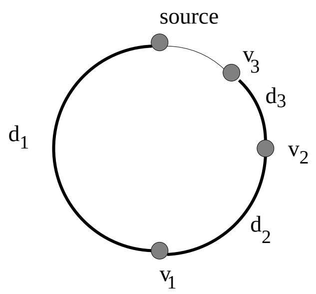

Figure 6: A bad case example for undirected and less asymmetric directed graphs for 3 join requests.

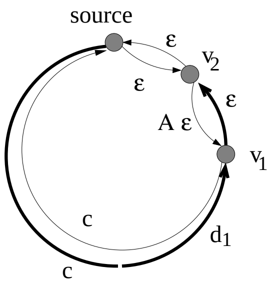

Figure 7: A bad case example for highly asymmetric directed graphs.

The above equation concludes the proof, if we recall that $OPT = \rho$ (Eq.10), and that T is a lower bound for any on-line algorithm. ■

Note that the two bounds correspond to different kinds of graphs: highly asymmetric ($\Omega(M)$) and less asymmetric ($O(M)$) ones. The high asymmetry bound does not hold for less asymmetric graphs and vice versa. For asymmetry exactly M, the two bounds are equal. Given this, lower bound for any graph is the minimum of the two bounds.

Corollary 2 *(Lower Bound) Given the size of the multicast group, M, and a directed graph of maximum asymmetry, A, the performance of any on-line greedy algorithm is bounded from below as follows*

$$\frac{T}{OPT} = \Omega\left(\min\left(M, A \cdot \frac{\log M}{\log(A+1)}\right)\right)$$

Recalling the upper bound from corollary 1, $O\left(\min\left(M, A \log M\right)\right)$ we can see that upper and lower bound differ only by the term $\log(A+1)$ in the second clause.

Note that in the examples for both lower bounds, M has to be $O(\log N)$, in other words, N must be exponential in M. Namely, we need a large number of nodes to "confuse" any on-line algorithm. However, for the greedy algorithm, we can create graphs with a similar procedure, where N does not have to be exponential in M [FPS97].

5 The Join and Leave Problem

In this section we study the join-leave on-line multicast problem, in which participants can join and leave anytime during the session. First, we improve the lower bound for the undirected case [IW91]. Then, we use that lower bound to show that the greedy algorithm can fail dramatically on directed graphs.

First, we consider undirected graphs. Let us indicate as $M(t)$ the number of the participants as a function of the time t and $M-1$ the number of join requests. Imase and Waxman [IW91] proved that the competitive ratio of the greedy algorithm for a given time t can be as large as M, and thus arbitrarily larger than the current number of participants $M(t)$. Here, we prove that the competitive ratio can be exponential to M with a careful weight assignment on the topology they used.

Theorem 5 *The worst case inefficiency of the greedy algorithm in the on-line join-leave multicast problem on undirected graphs with M join requests is bounded below by*

$$\frac{T}{OPT} \geq 2^{M-1}$$

PROOF. Suppose that a network is a cycle of M nodes $v_0, ..., v_{M-1}$ in that order and let v_0 be the source (see Fig.6). Assume that the sum of all the weights of the cycle is C. Consider that the weight of the edges between nodes $d_i = e(v_i, v_{i-1})$, $i = 1, ..., M-1$ decreases by half as follows: $d_1 = \frac{C}{2}$, $d_2 = \frac{C}{4}$ and in general $d_i = \frac{C}{2^i}$. We distinguish two phases: the joining and the leaving phase.

Joining Phase. Assume that the nodes join in increasing index $\{v_1, ..., v_{M-1}\}$ In the greedy algorithm, each destination joins with the previous destination in the sequence[3].

Leaving Phase. We can assume that all destinations leave the session except for v_{M-1}. So, in the greedy tree, we have the destination v_{M-1} connected to the source along the longest part of the cycle.

$$T = C - d_{M-1} = C - \frac{C}{2^{M-1}} \tag{17}$$

Clearly the optimal cost is given below

$$OPT = d_{M-1} = \frac{C}{2^{M-1}} \tag{18}$$

By taking the ratio $\frac{T}{OPT}$, we complete the proof. ■

Now let us consider directed graphs and examine how the asymmetry of the directed graphs, can degrade the efficiency of the solutions. Since an undirected graphs corresponds to a directed graphs of $A = 1$, the bound of Theorem 5 holds: the inefficiency can be exponential to M. Furthermore, we can prove that the inefficiency can be proportional to the asymmetry A.

Theorem 6 *The worst case inefficiency of the greedy algorithm in the on-line join-leave problem on a directed graph of maximum asymmetry A is bounded below by*

$$\frac{T}{OPT} \geq A$$

[3]Note that each destination is equi-distant from the source and the previous destination. To break these ties we can introduce infinitesimal differences in the weights, which were omitted here for the sake of simplicity.

PROOF. Intuitively, using the asymmetry we don't need a large M to create the previous bad case example. Let us assume the source and two destinations v_1, v_2 (see Fig.7). The weight of the edges between v_1 and the source are c both ways. The weights of the edges between v_2 and the source are ϵ. The asymmetry appears only between v_1 and v_2:

$$e(v_1, v_2) = \epsilon \quad e(v_2, v_1) = A \cdot \epsilon \tag{19}$$

First, we connect v_1. We can choose c to be of less cost than the path from the source to v_1 that includes v_2, which is $(A+1) \cdot \epsilon$:

$$c = A \cdot \epsilon \tag{20}$$

When connecting v_2, the greedy algorithm chooses edge (v_1, v_2), whose cost we can assume to be slightly (infinitesimally) smaller than the cost of $(source, v_2)$.

It is easy to see that when we remove v_1 the cost of the greedy tree is:

$$T = c + \epsilon \overset{(Eq.20)}{\Longrightarrow} T = (A+1) \cdot \epsilon \tag{21}$$

For the second formula we used Eq.20. Clearly, the optimal solution is:

$$OPT = \epsilon$$

Thus, the competitive ratio becomes:

$$\frac{T}{OPT} = A + 1 \tag{22}$$

■

The significance of the above theorem is that when the asymmetry is infinite the worst case inefficiency is unbounded in terms of the number of requests or the asymmetry. Below, we combine the two lower bounds for the directed graphs.

Corollary 3 *The worst case inefficiency of the greedy algorithm in the on-line join-leave problem on directed graphs is bounded below by*

$$\frac{T}{OPT} \geq \max(A, 2^{M-1})$$

Note that, in contrast to the lower bounds in the previous sections, we take the maximum of the two bounds. The difference is that in both worst case examples of this section we did not make assumptions on the asymmetry. This way, both examples apply to all directed graphs independent of their maximum asymmetry.

6 Conclusions

In this paper, we studied the performance of a greedy (NPF) and the Shortest Paths algorithm for the on-line multicast problem on directed graphs in a worst case analysis. In brief, for the join-only problem the greedy bounds are better than that of the Shortest Paths. However, for the join-leave problem, this is reversed: the worst case inefficiency of the greedy algorithm sky-rockets, while that of the Shortest Paths is not affected by the premature leaving.

In more detail, for the join-only on-line problem, we prove that the greedy algorithm has $\log(A+1)$-competitive on-line performance. In other words, the greedy performs as well as *any*

•	Undirected			Directed		
Bound Type	Static	Join	Join-Leave	Static	Join	Join-Leave
Short.Paths	M	M	M	M	M	M
Greedy Up	2	$\log M$	–	$\min(A, M)$	$\min(M, A \cdot \log M)$	–
Greedy Low	2	$\log M$	2^M	$\min(A, M)$	$\min(M, A \cdot \frac{\log M}{\log A+1})$	$\max(A, 2^M)$

Table 1: The worst case bounds of the greedy and Shortest Paths algorithms for the various multicast problems.

on-line algorithm. For the join-leave problem, first, we prove that for both undirected both directed graphs, the lower bound is 2^M (previous bound M for undirected graphs [IW91]). Then, for directed graphs, we prove that the greedy competitive ratio can also be proportional to the asymmetry, and thus, a lower bound is $\max(2^M, A)$. Given the large lower bounds for the join-leave problem, we find upper bounds to be of limited interest.

In Table 1, we present the previous and the new bounds of the worst case inefficiency for the greedy and the Shortest Paths algorithm for all previously mentioned problems. Note that the bounds for the Shortest Paths are tight, while for the greedy algorithm, we have separate lines for the upper and lower bounds. Table 1 shows that, in the off-line and join-only case, the greedy algorithm has an advantage over the Shortest Paths one. However, the greedy algorithm fails dramatically in the join-leave problem. The explanation is that the greedy approach is based on the idea that at each step we "make our best choice", which can be "proven wrong" by the ability of the destinations to leave. On the other hand, the performance of the Shortest Paths algorithm is tightly bounded by $O(M)$ for all problems. The reason is that the route of one destination is not influenced by the other destinations. This way, the functional and topological variations of the problems do not affect the worst case performance.

Future Work. For the join-only problem, for less asymmetric graphs, we see a discrepancy between the lower and upper bound of the greedy algorithm. It is the intuition of the authors that the upper bound could probably be reduced. The rationale, is that if we consider only one destination, it could be the case that all the edges towards it are of the higher cost edges (of each opposite pairs). In this case, even the optimal solution would have to use the same high cost edges, so the difference is not going to be proportional to the asymmetry.

Acknowledgments. The authors would like to thank Yossi Azar for suggesting the much simpler proof that we present here for Theorem 2. Special thanks to Panayotis Tsaparas and Allan Borodin, whose tough reviews and keen comments improved the paper significantly.

References

[AA95] Baruch Awerbuch and Yossi Azar. Competitive multicast routing. *Wireless Networks*, 1:107–114, 1995.

[Awe96] Baruch Awerbuch. *Online Selective Multicast and Maximal Dense Trees: A survey*, 1996. Available from the author's homepage: http://www.cs.jhu.edu/ baruch/MULTICAST/.

[BKJ83] K. Bharath-Kumar and J.M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Trans. on Communications*, 31:343–351, 1983.

[Bor] *Private conversation with Allan Borodin.*

[DW72] S.E. Dreyfus and R.A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[FPS96] Michalis Faloutsos, Rajesh Pankaj, and Kenneth C. Sevcik. Multicasting in directed graphs. *18th Biennial Symposium on Communications, June 2-5*, pages 29–32, 1996.

[FPS97] Michalis Faloutsos, Rajesh Pankaj, and Kenneth C. Sevcik. Tight(er) bounds for multicasting on directed graphs. (available from: http://www.cs.utoronto.ca/ mfalou/), 1997.

[IW91] M. Imase and B.M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4:369–384, 1991.

[KK95] J. Kadirire and G. Knight. Comparison of dynamic multicast routing algorithms for wide-area packet switched (ATM) networks. *Proc. IEEE INFOCOM*, pages 212–219, 1995.

[KMB81] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica, Springer-Verlag*, 15:141–145, 1981.

[Meh88] Kurt Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, pages 125–128, 1988.

[Ram96] S. Ramanathan. An algorithms for multicast tree generation in networks with asummetric links. *Proc. IEEE INFOCOM '96*, pages 337–344, 1996.

[RSC86] V. J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.