On-line Multicasting in Directed Graphs

Work in progress

Michalis Faloutsos   Rajesh Pankaj   Kenneth C. Sevcik

July 8, 1998

Abstract

In this paper we study the problem of on-line multicasting on directed graphs, and we demonstrate the competitiveness of a greedy algorithm in a worst case analysis. We prove that the greedy algorithm performs almost as well as *any* on-line algorithm: the greedy upper bound is very close to the lower bound of any on-line algorithm.

First, we improve on the upper bound of the off-line greedy algorithm on undirected graphs with a simpler than the previous proof.

Assume that A is the measure of asymmetry in a graph and M the size of the multicast group. For highly asymmetric graphs, we prove that the performance of the greedy algorithm is tightly bounded by $\Theta(M)$, and prove that this is also the lower bound for any algorithm. For less asymmetric graphs, the greedy algorithm is bounded above by $O(A\ \log(M))$. For these graphs, we prove that any algorithm performs at least as bad as $O(A\ \log(M)/\log(A))$.

Draft Version 1.0

1 Introduction

In this paper we examine the bounds for the on-line directed multicasting problem, so we explain the three terms that define our problem.

Multicasting can be defined as the distribution of the same information from one to many nodes concurrently. Many emerging applications are of multicast nature, and thus multicasting has become a very active research area. The multicast problem on undirected graphs has been studied quite extensively. It can be stated as follows. Given an undirected weighted graph $G(V, E)$ and a subset of **participant** nodes whose cardinality we denote by M, 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 of at least $O(N3^M)$ [DW72], which raises the need for good approximations with polynomial complexity. 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 algorithms with a WCI bounded by two. [TM80] [KMB81] [RSC86] [Meh88]. In this paper, we focus on a greedy algorithm suggested by Takahashi and Matsuyama [TM80]. In 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 participant that is nearest to some node of the partial multicast tree. We call this algorithm **Nearest Participant First (NPF)**.

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. Furthermore, communication links are not necessarily bidirectional. 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 undirected graphs.

In the on-line multicast problem, destinations join and leave during a multicast session. Considering the join-and-leave problem on undirected graphs, Imase and Waxman [IW91] proved that any on-line algorithm has a worst case performance that can be proportional to the number of nodes in the network. Clearly, this also applies to directed graphs since an undirected graph is a special case of a directed one. In this paper, we focus on the join-only case. For the undirected join-only problem, Bharath-Kumar and Jaffe [BKJ83] and later Imase and Waxman [IW91] proved that the greedy solution is never more expensive than $\log(M)$ times the optimal solution. In a more recent work, Awerbuch and Azar [AA95] study the join-only problem for multiple interleaving sessions on undirected graphs, and propose a $O(\log N \log M)$ competitive algorithm with respect to minimization of the congestion.

The on-line directed multicast problem can be formulates as follows. Assume a directed graph the source of the multicast connection, and a sequence of nodes that we call **destinations** (the destinations and the source are the participants). We want to connect each node of the sequence in order of appearance with a path to the source, one at a time without any knowledge of the number or the identity of the rest of the nodes. Our goal is to minimize the cost of the multicast tree.

1.1 Basic Concepts and Assumptions

In the rest of this paper, we use the following concepts. For the on-line problem, the **worst case inefficiency (WCI)** of a multicast algorithm is defined as the maximum ratio of the cost of the tree of the algorithm over the optimal off-line cost. Also, on-line NPF denotes the on-line version of the above algorithm: at each step we add the new (next in sequence) destination with the minimum cost path to the existing multicast tree.

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*

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

It is worth noting that in our study we focus on the worst case performance of the algorithms. Thus, if we see the on-line problem as an adversarial game between the "algorithm" (us) and the "on-line sequence" (adversary), we examine the case where we are so "unlucky" that our routing decisions are the worst with respect to the future destinations. This corresponds to the case where the adversary is able choose the next destination after seeing our routing decisions. Note, however, that the adversary does not necessarily know the way we make our routing decisions.

In our work, we make the following assumptions. First, we assume the commonly accepted characteristics of on-line algorithm for multicasting. Namely, an on-line algorithm creates a tree adding a path (not necessarily the shortest one) between each new destination and the partial multicast tree. One can think of algorithms that create non-tree configurations or establish more than a path for each node to facilitate future connections. Such an algorithm may perform better on our examples, but it is probable that other examples can lead to a performance equally bad or worse.

Second, we assume that rerouting is not permitted. If we reroute the whole tree for each new destination, we end up solving a number of off-line problems. However, rerouting is a non trivial task and it can be undesirable in many applications and network technologies. If we allow rerouting, we can consider local rerouting [KK95], or a limited number of global rerouting procedures [IW91]. In the limited rerouting case, an interesting question is to identify when it is worth rerouting.

Finally, without loss of generality, we can assume that there are no multiple edges on the same direction between each pair of nodes.

The rest of this papers is structured as follows. In section 2, we improve slighlty on the upper bound of the off-line greedy algorithm with a much simpler proof than the previous one [TM80]. In section 3, we present the upper bounds for the on-line NPF algorithm. In section 4 and for highly asymmetric graphs (asymmetry of $\Omega(M)$), we show that the upper bound of the greedy algorithm is a lower bound for any on-line algorithm. In section 5, for less asymmetric graphs, we prove that the upper bound of the greedy algorithm is very close to the lower bound of any on-line algorithm. In section 6, we show that, for the greedy algorithm, we can achieve the previous lower bound with a family of graphs smaller ($N = O(M)$) in size than of the ones of the previous section.

2 The Off-line Greedy Bound Revisited

Takahashi and Matsuyama [TM80] prove that the WCI of NPF is bounded by $2(1 - \frac{1}{M})$ compared to the optimal solution. Let us define $r(x) \equiv 2(1 - \frac{1}{x})$. We improve this bound by showing that the worst case complexity is $r(l)$ where l is the number of leaves in the optimal multicast tree.

Our proof will use the proof of KMB [KMB81], so we give an overview of the three steps of the algorithm here. *First*, we construct the complete graph G' of the participants. The cost of each edge (v, w) of G' is set to the cost of the shortest path between v and w. *Second*, we find a minimum spanning tree MST of G' that we denote as $MST_{G'}$. *Third*, we transform the tree back to G and remove any possible cycles.

Theorem 1 *The worst case inefficiency of NPF is bounded by $r(l)$*

$$NPF \leq 2(1 - \frac{1}{l})OPT$$

where l is the number of leaves in the optimal tree.

PROOF. Kou et al [KMB81] prove (as a partial result of their Theorem 1) the following lemma.

Lemma 1 *The cost of the minimum spanning tree of the complete graph G', $MST_{G'}$, satisfies the following formula:*

$$MST_{G'} \leq 2\,(1 - \frac{1}{l})\,OPT$$

For completeness, a sketch of the proof follows.

PROOF. Let the optimal tree have cost OPT and have l leaves. Consider a loop L around the optimal tree in a depth first order, $w_1, ..., w_M$, so that we visit every participant once and pass by every edge exactly twice. Clearly the cost of the loop is $L = 2OPT$. We can decompose L in l simple subpaths that connect one leaf to another. If we remove the maximum cost subpath of the loop f_{max}, we create a walk that we denote by W. Clearly, $f_{max} \geq OPT/l$, and so

$$W = L - d_{max} \leq (1 - \frac{1}{l})\,L = (1 - \frac{1}{l})\,2OPT$$

Note that, in the complete G' graph, the union of the edges (w_i, w_{i+1}) in the order of the walk W corresponds to a spanning tree of the participants with cost at most W. The minimum spanning tree tree of G' is of less cost than any spanning tree, and so, $MST_{G'} \leq W$. ■

Let us construct the auxiliary complete graph G'_m: we begin with a copy of G', and then change the weights of some edges only. Namely, let us consider the execution of NPF on G and denote by $Q = v_1, ..., v_M$ the sequence with which the destinations are chosen, where v_1 is the source. Let us call p_i the cost of the path that we use to connect v_i. In G'_m, we assign weight p_i to the edge (v_{i-1}, v_i).

Lemma 2 *The following inequality holds for the cost of minimum spanning trees of G' and G'_m*

$$MST_{G'_m} \leq MST_{G'}$$

PROOF. We will prove that G'_m is "lighter" than G', or more accurately, every edge of G'_m is at most as heavy as the corresponding edge of G'. We only need to examine the edges that differ in weight. Consider the edge (v_{i-1}, v_i) that has cost c' in G' and cost $c'_m = p_i$ in G'_m (the joining cost of v_i in NPF on G). Note that $p_i \leq c'$, because when v_i joins the tree node v_{i-1} belongs to the partial tree built by NPF, and p_i is the minimum cost path. ■

Lemma 3 *The union of the edges $T'_m = \{(v_{i-1}, v_i) : i = 2, ..., M\}$, where v_1 is the source, constitutes a minimum spanning tree for G'_m.*

PROOF. One way to find the minimum spanning tree of G'_m is to execute NPF on G'_m. A detailed comment: in the case of equal-cost edges, we bias NPF to choose the edge of T'_m among them. We claim that NPF on G'_m, will choose the destinations in the same order and create the T'_m tree. We can prove it by induction on the node index.

For $k = 2$. It holds trivially.

For $k = r$. We accept that the first $r - 1$ edges of T'_m are chosen in the minimum spanning tree, which now consists of the nodes $S_r = (v_0, ..., v_{r-1})$.

For $k = r + 1$. We want to prove that v_{r+1} will be connected next with the edge p_i to v_r. Observe that, at the same step, NPF on G chose p_i as the minimum cost path. Note that unconnected nodes, $S - S_r$, have the same distances from the connected participants, S_r, in both NPF executions. Thus, edge (v_r, v_{r+1}) with cost p_i is the minimum cost edge (or one of them) and NPF on G'_m will select it. ■

Combining the two lemmas above and the trivila observation that NPF on G has cost $NPF = MST_{G'_m}$ we get:

$$NPF = MST_{G'_m} \leq MST_{G'} \leq 2\,(1 - \frac{1}{l})\,OPT$$

This concludes the theorem. ■

A direct consequence of the above theorem is that $NPF \leq 2\,(1 - \frac{1}{M})\,OPT$, since $M \geq l$.

Finally, it is worth noting that this bound is tight [TM80] (Theorem 2).

3 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 $A\ \log(M)$ competitive.

First, we prove that $T_d \leq M\, OPT_d$. This bound comes as 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 becomes obsolete.

Theorem 2 *For the on-line directed problem, the cost of the solution of NPF, T_d, is bounded*

$$T_d \leq M\, OPT_d$$

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 $M\, d_{max}$, which is a bound of the union of the source-to-destination paths. On the other hand, the optimal solution OPT_d is at least d_{max}. Therefore,

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

■

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

Theorem 3 *Assume a weighted directed graph with maximum asymmetry A and a multicast session of M participants. The following holds for the NPF algorithm:*

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

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 with weights those of the minimum of the two directed edges. Then, we perform a greedy on-line algorithm on the undirected graph. The 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. Some crucial points of the proof are discussed in the following expansion of the above steps.

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 edges in G_d, we create an undirected edge in G_u with the minimum weight of the pair.

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

If OPT_u is the optimal tree on G_u, clearly

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

Let us execute a variation of NPF on G_u for the on-line problem that we call **restricted** NPF. Namely, we consider joining a destination to the tree only to destinations or the source[1]. We call T_u the cost of the resulting tree.

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

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

[1]Note that we use the shortest path between participants and this may lead to count an edge multiple times in our cost.

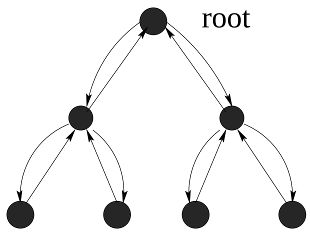

Figure 1: The d-tree : a complete binary directed tree.

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:

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

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.

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

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 is bounded as below*

$$\frac{T}{OPT} = O(\min(M, A\,\log M))$$

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

4 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.

Theorem 4 *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 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).

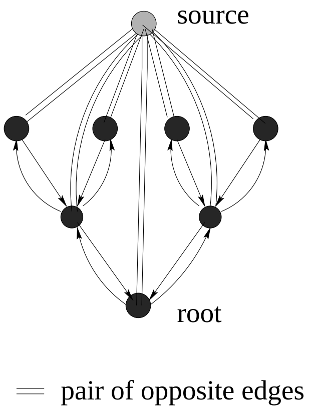

Figure 2: The worst case example for highly asymmetric graphs. Edges from or to the source are of cost c. From the rest, curved edges are of cost c, while their opposite edges are of cost ϵ.

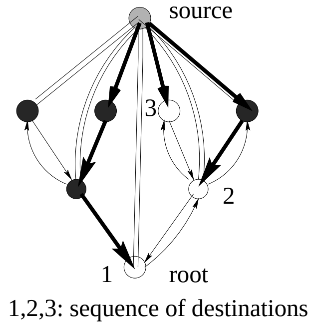

Figure 3: The graph can be interpreted in two ways: (A) The worst case example for highly asymmetric graphs for a possible algorithm, (B) The worst case online sequence for graph F_1 on the discussion of less asymmetric graphs.

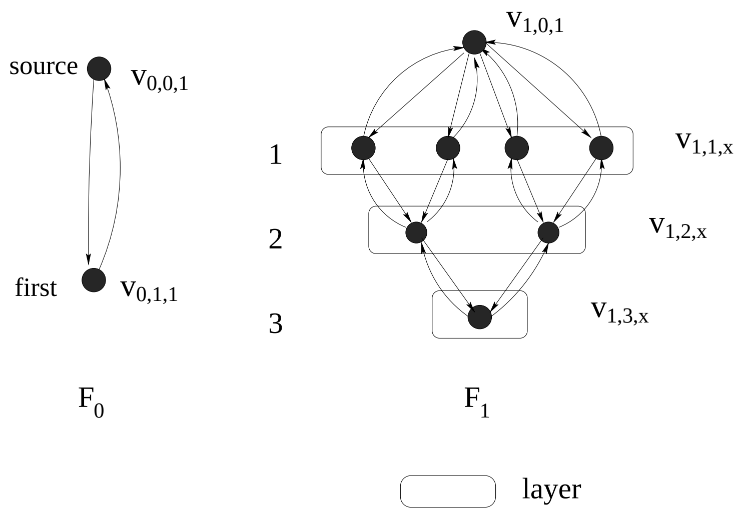

Figure 4: The family of F_g with $k = 2$: F_0 and F_1

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

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

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

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

If we choose $\epsilon = O(c/M)$, the two previous equations conclude the proof. ∎

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 $\Omega(M)$.

5 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

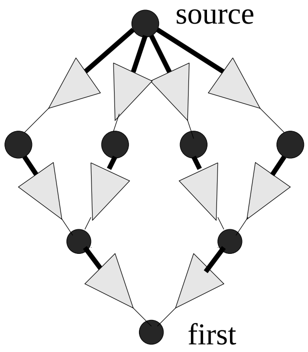

Figure 5: The family of F_g with $k = 2$: F_2. The triangles represent d-trees whose root is a node of the previous generation. The thick edges represent many pairs of edges that we merge for the clarity of the layout.

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 any on-line algorithm can perform at least as bad as $A\frac{\log M}{\log A}$ in a worst case analysis.

Theorem 5 *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 below by*

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

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 can not use the depth of a d-tree, since this depth of a d-tree is relative to the given tree 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 all paths from the source to the First 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**. We considered two directed edges

$$e(v_{0,0,1}, v_{0,1,1}) = \rho_0 \tag{5}$$
$$e(v_{0,1,1}, v_{0,0,1}) = A\ \rho_0 \tag{6}$$

Creating the Next Generation. Given the graph $Fg-1$, we create the graph of the next generation F_g. We substitute every pair of opposite edges in $Fg-1$ 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 $Fg-1$. 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.

$$\begin{aligned} e(v_{g,l,x}, v_{g,l+1,x}) &= \rho_g & (7)\\ e(v_{g,l+1,x}, v_{g,l,x}) &= A\,\rho_g & (8)\\ \rho_g &= \frac{\rho_0}{(k+1)^g} & (9) \end{aligned}$$

We describe some important properties of the previous graphs. First, it is trivial to notice that, by the 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. The following lemma holds.

Lemma 4 *The number of layers, l, of graph F_g is exponential in g*

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

PROOF. It can be proven by induction on g. ■

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 $m = k\,(k+1)$, 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 Lemma 4, the relation between the number of generations and the number of participants is given below.

Corollary 2 *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 2 *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 between the two parameters easier.

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

The above equation defines an upper bound on the asymmetry of the graph. Since from each layer we select a destination, k is less than M. Thus, the asymmetry is also bounded by M

$$A = O(M) \tag{12}$$

The lower bound T of the cost of the greedy NPF 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:

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

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 Lemma 4, we have $(k+1)^g$ such trees. Thus, the total cost of generation g is given below:

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

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:

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

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 3 *(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 below as follows*

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

Recalling the upper bound from corollary 1, $\emptyset\left(\min\left(M, A\log M\right)\right)$ we can see that they differ by the term $\log A$ 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].

6 A Greedy On-line Lower Bound

In the previous section, we saw a family of graphs for the worst case inefficiency of any on-line algorithm. However these graphs have exponentially more nodes than participants $N = o(2^M)$. For the greedy algorithm, this is not a requirement. In this section, we construct a family of graphs with $N = O(M)$ and show that the greedy on-line algorithm reaches its lower bound.

6.1 The Family of Directed Graphs

We define the family of graphs G_g, where index g is called the **generation** of the graph. Higher generations are made incrementally from previous generation graph, and the maximum generation is denoted g_m. The nodes of generation g are denoted as $v_{g,x}$, where x identifies the node within the generation and we call it **index** of the node. Note that nodes may have different indexes in different generations. For example, node 1 of generation 1 becomes node 2 of generation 2, $v_{1,1} = v_{2,k+1}$.

The intuition behind these family of graphs is that edges from lower indexes to higher indexes are cheap, while their opposite edges are A times more expensive.

Initial Graph. G_0 has two nodes, $v_{0,0}$ and $v_{0,1}$. There are two directed edges between them:

$$(v_{0,0}, v_{0,1}) = \rho_0 - \epsilon, \quad (v_{0,1}, v_{0,0}) = A\,\rho_0$$

Note that ϵ is an arbitrarily small constant that breaks the ties between paths of equal length. Given this, the asymmetry of these two edges is (asymptotically) A.

Creating the Next Generation. Given graph G_{g-1}, between every pair of consecutive-index nodes $v_{g-1,i}$ and $v_{g-1,i+1}$, we will insert a chain of k nodes which we call **d-chain**. The procedure is described in more detail below.

1. We assign the names of the new generation to the nodes of the previous generation:

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

2. Between each pair of nodes of the previous generation, say $v_{g-1,i}$ and $v_{g-1,i+1}$, we add a k-chain. The new nodes introduced in the generation g are denoted as W_g, while the total number of nodes at generation g is denoted as V_g.

   (a) We connect the ends of the d-chain with the pair of the previous generation, and we arrange the weights as follows. The "cheap" edges that we denote $\boldsymbol{\rho_{g,j}}$:

$$(v_{g,j}, v_{g,j+1}) = \rho_{g,j}, \quad j = i, i+1, ..., i\,k$$

   The opposite edges that we denote as $\mathbf{a_{g,j}}$ have a higher cost:

$$a_{g,j} = (v_{g,j+1}, v_{g,j}) = A\,\rho_{g,j}$$

   We can choose $\rho_{g,i}$ to be independent of i and we can denote it as ρ_g

$$\rho_{g,i} = \rho_g = \frac{\rho_0}{(k+1)^g}, \quad \forall i, \quad g = 1, .., g_m \tag{17}$$

(b) We connect the d-chain with the nodes of the previous generation pair that is closest to the source $v_{g-1,i}$:

$$(v_{g-1,i}, v_{g,j}) = \sum_{m=0}^{j-1} \rho_m - \epsilon$$

We will denote such an edge as s-**edge**. The opposite edge of an s-edge have a higher weight:

$$(v_{g,j}, v_{g-1,i}) = A\ (v_{g-1,i}, v_{g,j})$$

The following lemmas capture interesting properties of these graphs concerning their maximum asymmetry and their size.

Lemma 5 *The maximum asymmetry of the above graph goes to A as ϵ goes to zero.*

PROOF. By construction. ∎

Lemma 6 *The number of nodes of graph G_g is exponential in g*

$$|V_g| = (k+1)^g + 1$$

PROOF. We prove the lemma by induction on g.

For $g = 1$. It holds since we start with only one pair of nodes in V_0, and so $V_1 = k + 2$.

For $g = m$. We accept the induction hypothesis

$$|V_m| = (k+1)^m + 1$$

For $g = m + 1$. The G_m graph has $|V_m| - 1$ pairs of nodes with consecutive indexes. For every such pair we add k nodes to create graph G_{m+1}, and so the total number of nodes we add is given below:

$$|W_{m+1}| = k\ (|V_m| - 1) = k\ (k+1)^m$$

Using the above equation and the hypothesis of the induction, the total number of nodes at $m + 1$ generation is given below

$$|V_{m+1}| = |V_m| + |W_{m+1}| = (k+1)^m + 1 + k\ (k+1)^m = (k+1)^{m+1} + 1$$

This concludes the induction and the lemma. ∎

The above lemma has the following direct consequence.

Corollary 4 *If the total number of nodes of the graph G_g is M, then the total number of generations is tightly bounded by*

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

6.2 The Lower Bound

Using the above family of graphs, we can prove the lower bound for an on-line greedy algorithm on a directed graph of bounded asymmetry.

Theorem 6 *Given the size of the multicast group, M, and a bound on maximum asymmetry A, we can construct a graph and an on-line sequence of nodes for which the performance of a greedy on-line algorithm is lower bounded by*

$$\frac{T}{OPT} \geq A \frac{\log M}{\log(A+1)}$$

PROOF. Let us consider an online problem on graph G_g and assume the following multicast session. The source is node $v_{0,0}$ and the destinations arrive in the following sequence : First, we choose the nodes of the smallest generation number and within a generation the ones of highest index. In other words, the nodes come in decreasing index and, then, in increasing generation number. Note that nodes have multiple names, but they are considered only the first time.

As an example, we present beginning of the sequence (listed by generation):

- $v_{0,1}$
- $v_{1,k}, v_{1,k-1}, ..., v_{1,1}$
- $v_{2,x}, v_{2,x-1}, ..., v_{2,1}$, where $x = k\,(k+1)$ except the multiples of $(k+1)$

Given the above sequence of connection requests we will calculate what is the worst case inefficiency of a greedy on-line algorithm. Without loss of generality, we will assume that the number of participants, M, is a power of $k+1$ minus one.

Furthermore, we choose k to be equal to A:

$$k = A \tag{18}$$

The optimal off-line cost. It is relatively easy to see that the optimal off-line solution is a chain connecting all the nodes using the "cheap" edges from lower to higher indexes for the last generation.

$$OPT = \sum_{i=0}^{M} \rho_{g_m,i} \tag{19}$$

The on-line cost. We will calculate the cost of the greedy algorithm. We calculate the cost that is added by the nodes of each generation g, and we denote this cost as $\mathbf{T_g}$.

Generation 0. In this step, we add node $v_{0,1}$ costs

$$T_0 = \rho_0 - \epsilon$$

Generation 1. In this step, we add k nodes. The cost of adding node $v_{1,i}$, denoted as $\mathbf{nodecost}(\mathbf{v_{1,i}})$, where $j = k, ..., 1$ is:

$$nodecost(v_{1,i}) = \min(a_i, s_i) = \min\left(A\,\rho_{1,i}, \sum_{n=0}^{i-1} \rho_{1,n} - \epsilon\right) \tag{20}$$

The total cost of adding the nodes of this step is

$$T_1 = \sum_{i=1}^{k} nodecost(v_{1,i}) \tag{21}$$

The optimal tree cost is

$$OPT = \rho_0 - \epsilon \tag{22}$$

Given our choices of $\rho_{g,i}$ and k, the $nodecost()$ can be simplified as follows. The edge a_i will always be greater than s_i, and thus, Eq.20 becomes

$$nodecost(v_{1,i}) = \sum_{n=0}^{i-1} \rho_{1,n} - \epsilon = i\,\rho_1 - \epsilon$$

Therefore, T_1 is estimated below. Note that we omit the factor that corresponds to ϵ.

$$T_1 = \sum_{i=1}^{k} nodecost(v_{1,i}) = \sum_{i=1}^{k} i\,\rho_1 = \frac{k(k-1)}{2}\,\rho_1 \tag{23}$$

Generation g. We want to calculate the cost of adding the nodes of generation g T_g. For each pair of nodes of graph of the previous generation we insert a d-chain, and we follow a similar procedure as for T_1.

Thus, for each such pair, we have a cost of $\rho_g \frac{k(k-1)}{2}$, and in total we have:

$$T_g = (k+1)^{g-1}\,\rho_g\,\frac{k(k-1)}{2} \tag{24}$$

Using Eq.17, we can substitute ρ_g in the above equation and get:

$$T_g = \frac{\rho_0}{k+1}\,\frac{k(k-1)}{2} \tag{25}$$

Note that T_g is independent of the generation g. This is not surprising since we essentially repeat the the cost calculation of generation 1 for each d-chain we insert. The relative cost of each such d-chain is a fraction of the total cost, but we have many such insertions. In more detail, there are $(k+1)^{g-1}+1$ nodes in $g-1$ generation, there are $(k+1)^{g-1}$ such pairs that are connected with an edge of ρ_{g-1} cost.

Given the added cost for each generation by Eq.25 and the number of generations by Corollary.4, the total cost of the on-line algorithm, T, is

$$T = T_0 + \sum_{g=1}^{g_m} T_g = \rho_0 + \frac{\log M}{\log(k+1)}\,\frac{\rho_0}{k+1}\,\frac{k(k-1)}{2} \tag{26}$$

Recall that we chose $k = A$. The inefficiency ratio for this case is:

$$\frac{T}{OPT} = \frac{T_0 + \sum_{g=1}^{g_m} T_g}{\rho_0} = 1 + \frac{\log M}{log(A+1)}\,\frac{1}{A+1}\,\frac{A(A-1)}{2} \Rightarrow \tag{27}$$

$$\frac{T}{OPT} = \Theta\left(\frac{\log M}{\log(A+1)}A\right) \tag{28}$$

The above equation concludes the proof of the theorem. ■

7 Conclusions

In this paper, we studied the performance of a greedy algorithm (NPF) for the on-line multicast problem on directed graphs in a worst case analysis. We distinguish two cases: highly asymmetric graphs with asymmetry of $o(M)$, and less asymmetric graphs. We prove upper and lower bounds for each case. We conclude that the upper bound of the greedy algorithm differs only by a logarithmic factor ($\log A$ in the less asymmetric graphs) from the lower bound of any on-line algorithm. Consequently, the greedy algorithm is fairly competitive, and even the best on-line algorithm would not perform much better in the worst case.

Given the lower and upper boundary discrepancy, we expect that either the upper bound can be reduced or the lower bound can be increased. It is the intuition of the authors that the upper bound could probably be reduced. If we consider only one destination, it could be the case that all the edges towards it are the higher cost edges (of the 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.

In our study, we use worst case analysis. However, it would be interesting to study the performance of the greedy algorithm for the average case. The average case of the topology of a graph is ill defined, so research could focus on the average on-line sequence. Furthermore, randomized algorithms could introduce interesting results.

ACKNOWLEDGMENTS. The authors would like to thank Yossi Azar for suggesting the much simpler proof that we present here for the upper bound theorem on less asymmetric graphs, and Panayotis Tsaparas for his keen comments and corrections.

References

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

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

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

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

[FPS97] M. Faloutsos, R. Pankaj, and K. 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] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, pages 125–128, 1988.

[Ram96] S. Ramanathan. An algorithm for multicast tree generation in networks with asymmetric 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.