IMPROVED ALGORITHMS FOR CONSTRUCTING HYPERCUBE SP-MULTICASTING TREES

Christopher C. Cipriano and Teofilo F. Gonzalez
Department of Computer Science
University California
Santa Barbara, CA, 93106, USA
email: {ccc,teo}@cs.ucsb.edu

ABSTRACT
Given a source node s and a set of destinations D in the n-cube we study the problem of constructing near-optimal sp-multicast trees. In other words, construct a near-optimal Steiner tree for $\{s\} \cup D$ in which all paths from s to the destination nodes are shortest paths in the n-cube. We discuss known algorithms (`Greedy`, `NGrouping`, and `Clustering`) for the sp-multicast tree problem and identify problem instances for which they perform poorly. We introduce three new algorithms (`MOverlap`, `BEstimate` and `BUp`) that identify structural similarities between the message destinations and avoid the pitfalls of the previously known algorithms. We present an experimental evaluation of all the algorithms over a wide range of problem instances. Our experimentation shows that the new algorithm `BUp` outperforms all the other methods.

KEY WORDS
Heuristics, n-cube, sp-multicasting trees, multicasting, Steiner Trees.

1 Introduction

Multicasting is a communication primitive that allows a node in a network to send a message to multiple destination nodes. There are many ways to implement multicasting. For example, when an e-mail is sent to k destinations, e-mailing systems make k copies of the message and send each copy separately to the destinations (k unicasting operations). This implementation is fine when sending a message to local neighbors or when the messages are small. But when all the destinations are far away from the source and the messages are very large, the above implementation is not an efficient one. In this scenario, sending the message to one of the destinations and then forwarding it from there to the remaining $k-1$ destinations would be a more efficient solution. The communication in this case is modeled by a *tree*. A tree connects (directly or indirectly) the source node to all the destination nodes, and may include other nodes in the network. There are many possible multicasting trees and objective functions. In this paper we consider the multicast tree problem defined over the n-cube network. The problem of generating a multicast tree with the fewest edges is known as the *minimum Steiner tree (MST)* problem[1]. Another type of multicast tree involves a dual objective function where we minimize link usage provided that every path from the source node to each destination node is a shortest path in the original network (i.e. minimize link usage provided the message delay time is minimum). We refer to this problem as the *shortest path multicast(sp-multicast)* problem.

These problems are formally defined below. The Steiner tree (ST) problem is given an undirected graph $G = (V, E)$, a subset of vertices, and $D = \{u_0, u_1, \ldots, u_k\} \subseteq V$, find a subtree $T = (V_T, E_T)$ of G (i.e., $V_T \subseteq V$ and $E_T \subseteq E$) with the least number of edges such that $D \subseteq V_T$.

The sp-multicast tree problem is the Steiner tree decision problem with the added constraint $d_T(u_0, u_i) = d_G(u_0, u_i)$ for $1 \leq i \leq k$, where $d_T(a, b)$ and $d_G(a, b)$ is the number of edges in a shortest path from a to b in T and G, respectively.

Graham and Foulds [7] studied the MST problem for the n-cube in order to determine the possibility of computing specific biological sciences problems in reasonable time. Their work resulted in a complex proof for the NP-Completeness of the decision version of the Steiner tree problem for the n-cube. Later on, a complex transformation and proof was used to establish that the sp-multicast problem for the n-cube is NP-Complete [2, 3]. Recently, Cipriano and Gonzalez [4] presented *simple* transformations and proofs that establish the NP-Completeness of these two problems, and extended these results for the Chord and binomial graph networks.

Formally, an n-*cube* (*hypercube*) consists of 2^n nodes or processors. Every node in the n-cube is represented by an n-bit string and there is an edge between two nodes if their bit representation disagrees in exactly one bit. Two adjacent nodes in the n-cube are said to be *neighbors*. The hypercube is a subnetwork of the Chord and the binomial graph network [1].

For the n-cube graph we refer to the above problems as the n-cube Steiner tree problem and the n-cube sp-multicast tree problem. Without loss of generality we assume that the source node s is always node 0 (represented

[1]Traditionally Steiner tree problems are defined for weighted graphs with the objective being to minimize the total weight of the edges in the tree.

by n 0-bits). There is a trivial algorithm to implement optimum unicasting in the n-cube, when there is only one destination.

Since any sp-multicast tree is a Steiner tree we know that an optimum n-cube sp-multicast tree has at least as many edges as an optimum n-cube Steiner tree. For most problem instances it has more. There are well known approximation algorithms for the Steiner tree problems [8, 5] for arbitrary graphs (including the n-cube). The simplest approximation algorithm begins by constructing a complete graph G' over the set of destination nodes and the source node. The weight of an edge joining vertices i and j is equal to the number of edges in a shortest path from node i to node j in the original graph. Then a minimum cost spanning tree is constructed over the complete graph G'. The edges in the spanning tree are translated back to shortest paths in the original graph and if the resulting structure is not a tree, superfluous edges are eliminated. The resulting tree has a number of edges that is not more than twice the number of edges in an optimum Steiner tree for the original graph. Therefore, the number of edges in the suboptimal Steiner tree constructed by the above procedure divided by two is a lower bound for the number of edges in an optimum sp-multicast tree. We will refer to this algorithm as the min-cost spanning tree algorithm (`MCST`).

2 Previous Heuristics

In this section we discuss previous algorithms (`Oblivious`, `Greedy`, `NGrouping` and `Clustering`) for the n-cube sp-multicasting problem. We discuss problem instances for which all of these algorithms generate sp-multicast trees that are far from optimum. Then we identify all the minimization opportunities that these algorithms ignore, and present in the next section, new algorithm that avoid these pitfalls. In Section 4 we present the results of our empirical evaluation of the solutions generated by all the algorithms discussed in this paper.

To simplify the presentation we discuss the algorithms in terms of constructing an sp-multicasting tree for the n-cube, rather than as an algorithm that preprocesses the input and then performs the routing as in Refs. [6, 12, 10, 11]. This simplifies the exposition. There is no loss of generality, since it is possible to transform our offline algorithms to algorithms that preprocess the input and then perform the routing top down.

The simplest algorithm, referred to as the `Oblivious` algorithm by Fujita [6], constructs the tree from the source node s. Let D be the set of all the destinations. The tree constructed is stored in the global structure T consisting of nodes and edges. The structure T is initialized to $\emptyset$ and a call is made to procedure `Oblivious`(s,D). The procedure, `Oblivious`(r, S), below specifies the details of the tree construction process.

```
Algorithm Oblivious(r, S);
  // Source node: r, Set of destinations: S
  T ← T ∪ {r};
  S = S \ {r};
  if S is empty then return T;
  Let r_1, r_2, ..., r_k be any minimal set[2] of neighbors of r
      such that every destination d ∈ S has a shortest path
      to node r that visits at least one of the nodes r_i;
  Assign each destination d to set S_i, where i is the
      smallest integer such that destination d has a shortest
      path to r going through node r_i;
  Add edge {r, s_i} to T for 1 ≤ i ≤ k;
  Invoke algorithm Oblivious(r_i, S_i), for 1 ≤ i ≤ k;
end of algorithm Oblivious;
```

The `Oblivious` algorithm generates trees with at most $(n-1) * |D| + 1$ edges, since every destination can have at most $n-1$ 1-bits, except for one which may have n 1-bits. One can prove sharper upper bounds, but for our purpose this bound suffices. A lower bound for an optimum tree is $|D|$, as every destination requires at least one edge in the tree. It follows that the ratio of $\hat{f}/f^* \leq n - 1 + 1/|D|$, where $\hat{f}$ is the number of edges in the tree generated by the `Oblivious` algorithm and f^* is the number of edges in an optimum tree. Though, one cannot find examples for which the bound is tight.

Lan, Esfahanian, and Ni [9] developed an improved version of the `Oblivious` algorithm that has been referred to as algorithm `LEN`, but which we refer to in this paper as algorithm `Greedy`. In practice Algorithm `Greedy` generates sp-multicast trees that in general have a near-optimal number of edges, but for many problem instances the trees generated are far from optimal [6]. The difference between algorithm `Greedy` and the `Oblivious` algorithm is that from all the possible choices for $(r_1, r_2, \ldots, r_k)$ one selects the one that results in set S_1 having the largest number of destinations, then S_2 has the largest number of remaining destinations and so on. In other words, the first branch taken is along one of the most popular bits. Note that if two or more bits are among the most popular, then one can select either of the most popular bits to branch off. One can easily show that algorithm `Greedy` can be implemented to take $O(n^2 d)$ time, where n is the number of dimensions in the hypercube and d is the number of destinations.

Algorithm `Greedy` has the same upper and lower bounds as the ones for the `Oblivious` algorithm, but in practice it behaves much better. Fujita [6] constructed a set of problem instances for which $\hat{f}/f^* = n/2 - o(n)$. Thus, for some problem instances the worst case bound is not too different from the one for the `Oblivious` algorithm.

Sheu and Yang [12] modified algorithm `Greedy`. The algorithm, which we refer to as algorithm `NGrouping` (Neighbor Grouping), tends to generate better solutions than algorithm `Greedy`. The idea is to

[2] Note that the set is not necessarily the set with the least number of nodes.

group together destinations that are neighbors of each other. So instead of having a set of destinations, we have a set of trees of destinations $D = (T_1, T_2, \ldots)$, where each node in each tree T_i has one fewer 1-bit than its children and each node is a neighbor of each of its children (in the n-cube). Then one applies algorithm `Greedy` to the root nodes of all the trees. When one reaches the root of a tree in an invocation (i.e., r is the root of one of the trees in S), one replaces such tree in S by the set of trees rooted at the root's children. It is important to clarify that when selecting the most popular bit, one selects it by considering only the roots of the trees in S. The preprocessing can be implemented to take $O(nd^2)$ time. Therefore the overall time complexity for the algorithm is $O(n^2d + nd^2)$ time.

The motivation behind this algorithm is that if we construct a path P from s to the root of tree T_i, one can just append the tree T_i to the path P and that will be the best possible tree for the set of destinations in T_i given the path P. This is because the number of edges added to P is equal to the number of destinations in the tree minus one.

As pointed out in Ref. [10] the above procedure does not always generate a tree! It generates tree-like structures, but some nodes may end up with more than one parent. Though, these structures are easy to transform into trees via a simple post processing procedure.

Algorithm `Clustering` introduced by Lu, Fan, Dou, and Yang [10, 11] is an extension of algorithm `NGrouping`. One constructs trees as in the algorithm `NGrouping`, but the trees are more general. A destination d can be made the child of destination a if there is a shortest path P from s to d that visits destination a, and the path P from a to d does not include another destination in D. The resulting trees depend on the node ordering used to construct the trees. The way we apply this procedure is as follows. At each step we consider one destination at a time in non-increasing order of their shortest distance to the source s. I.e., from farthest to closest to s. When considering destination d, if there are several nodes that could be considered as destination a, we select the one that is closest to d and in case of ties we pick any of the closest ones. The resulting forest of destinations tend to include fewer trees than those produced by algorithm `NGrouping`. Then we apply algorithm `Greedy` in the same way as we did in algorithm `NGrouping`, but using "weighted popularity". I.e., the contribution of the bits of the root of each tree T_i are weighted by the number of destinations in the tree T_i. As you can see in Figure **??**, the algorithm generates a tree for the example given in Figure **??**. In fact Lu, Fan, Dou, and Yang [10] have shown that algorithm `Clustering` always generates a tree. The preprocessing can be implemented to take $O(nd^2)$ time. Therefore the overall time complexity for the algorithm is $O(n^2d + nd^2)$ time.

A careful analysis of one of Fujita's [6] instances, that makes algorithm `Greedy` perform poorly, shows that algorithm `NGrouping` generates trees with a number of edges that is about $(0.4n - o(n)) * f^*$. This instance can be easily modified so that algorithm `Clustering` generates trees

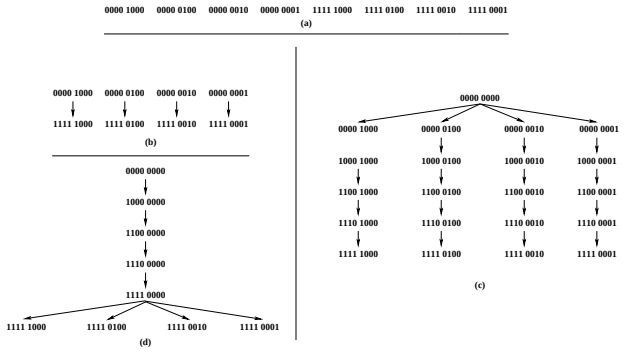

Figure 1. (a) Set of destinations D. (b) Trees generated by algorithm `Clustering`. (c) Sp-multicast tree generated by algorithm `Clustering`. (d) Optimal sp-multicast tree.

with a number of edges that is just about $(0.4n - o(n)) * f^*$. In Figure 1 we present a simple problem instance that when extended to the n-cube makes algorithm `Clustering` generate sp-multicast trees with about $0.16n * f^*$ edges.

Figure 1 will helps us identify important optimization opportunities that algorithm `Clustering` misses to catch and therefore it cannot generate near-optimal solutions all of the time. All of the above three algorithms suffer from the same type of problems. Instances like the one in Figure 1 have subsets of destinations with very similar bit patterns. When algorithms do not identify these large patterns, individual trees for each of these destinations may be generated, rather than just generating a tree for the nodes with similar bit patterns. The algorithms in the next section identify, up to a certain extent, and exploit this information to generate near-optimal sp-multicast trees.

3 New Algorithms

In this section we discuss our algorithms (`MOverlap`, `BEstimate` and `BUp`) for the n-cube sp-multicasting problem. Our algorithms try to avoid the pitfalls of the previously known algorithms. In the next section we compare the performance of all the algorithms discussed in this paper.

Algorithm `MOverlap`(r, S) is like algorithm `Oblivious`(r, S) except that the pairs $(r_1, S_1), (r_2, S_2), \ldots$ are selected differently. First we find a pair of nodes P in S with the Maximum bit-pattern Overlap (i.e., the pair of nodes with the largest number of identical bits). Then we add to set P one by one the destinations in $S - P$ with the largest number of bits identical to all the current destinations in P. Then we select a 1-bit that is common to all the destinations in P but is a 0-bit in r. We define r_1 as the neighbor of r that differs from r in the bit just identified, and we let S_1 be set P. We apply the above procedure to $S - S_1$ to generate the pair (r_2, S_2) and continue until all the destinations in S have been partitioned into the sets $S_1, S_2, \ldots S_k$, for some

integer k. One can easily show that the this algorithm can be implemented to take $O(n^2d^2)$ time.

Algorithm `BEstimate`(r, S) is a more elaborate version of algorithm `MOverlap`(r, S). The idea behind the algorithm `BEstimate` is to consider each pair of destinations in S at a time and find the bits in common between them. Then we select a subset of destinations, $S' \subseteq S$, on which one can transition on a permutation of the bits in common. We then estimate, by procedures defined below, the cost of the tree that includes the transition on a permutation of the bits in common. We then select the pair that we estimate will produce the best tree. For that pair, the first bit in the transition is used to determine the node r_1. The set of destinations is now partitioned into two sets which we call S_1 (this is simply S' for the pair selected) and $S - S_1$. Similarly we identify r_2 and S_2, r_3 and S_3, and so forth. Then we apply the algorithm recursively to each of these pairs. Below you will find a more concrete description of the algorithm. The sp-multicast tree constructed by the algorithm will be represented by the set T (of nodes and edges). Initially T is empty.

```
Algorithm BEstimate(r, S);
  // r is the source node and S is the set of destinations;
  T ← T ∪ {r};
  S = S \ {r};
  k = 0
  while S ≠ ∅ do
    k = k + 1;
    for every pair p of destinations in S do
      Define the set of nodes P as all the destinations that
        are "similar" to the nodes in p;
      The structure in common between the destinations
        in p and P partitions the set of destinations
        into equivalence classes.
      Estimate the number of edges in the trees for each
        equivalence class, and use that as estimate the
        cost of the resulting tree.
    endfor
    Select the pair of destinations p that has the best
      estimate on the resulting tree and let P be the
      set of nodes "similar" to p;
    Let r_i be a node that is neighbor of r and all the 1-bits
      in r_i are 1-bits in p and P;
    Assign p and P to set S_i
    Delete all elements in S_i from set S;
  endwhile
  Add edge {r, r_i} to T for 1 ≤ i ≤ k;
  Invoke algorithm BEstimate(r_i, S_i), for 1 ≤ i ≤ k;
end of algorithm BEstimate;
```

One of the steps that has been left open in our algorithm is how to estimate the number of edges in an sp-multicast tree for a source node and a set of destinations. There are many ways we can do this. The first one is by using any of procedures defined before, e.g., `Greedy`, `NGrouping`, `Clustering`, and `MOverlap`. Another is by running two or more of these algorithm and using the best of the best of these solutions. In the next section we discuss the quality of the solutions generated by these different possibilities. The time complexity for algorithm `BEstimate` is $O(n^2d^2 + nd^3c)$, where c is the time complexity of the algorithm used to provide the estimate. Algorithm `BEstimate` with the best solution of `Greedy` and `MOverlap` (which we will refer to as algorithm `BEstimate`(Best of `Greedy` and `MOverlap`), has overall time complexity $O(n^3d^5)$.

Before we present our next algorithm, we introduce the following notation and convention. For any node d in the n-cube which may or may not be a destination in D, define $dist(d)$ as the length of a shortest path from s to d. For this algorithm we define the set S to contain all the destinations D plus the source node s.

The above algorithms generate the sp-multicast tree top-down, i.e., from node s to all the destinations. Algorithm `BUp` constructs the tree Bottom-Up. First we find a node $x \in S$ with largest $dist(x)$ value. Then we find a destination node, y, in S closest to x. If y is at a distance one from x we just add the edge $\{y, x\}$ to the tree and delete node x from S. Otherwise we add the edge (x, x') that takes x closer to y in the direction of s to the tree, delete node x from S, and add the new node x' to S. If $dist(y)$ is equal to $dist(x)$ we add the edge (y, y') that takes y closer to x in the direction to s to the tree and delete node y from S. If node x' is different from node y' we add node y' to S. The specific details are given below.

```
Algorithm BUp (s, D)
  S ← D ∪ {s};
  T ← {S};
  loop
    Let x be a node whose dist(x) value is maximum
      amongst all nodes in S;
    if dist(x) == 0 then break
    Find the node y ∈ S − {x} closest to x;
    if (dist(y) == dist(x) − 1) then {delete x from S;
          add edge {y, x} to T; }
    else
      {Let b be a position such that x has a 1-bit and
        y has a 0-bit;
       Let x' be x and set the b^th bit of x' to zero;
       Add the node x' and edge {x', x} to T;
       S ← (S − {x}) ∪ {x'};
       if (dist(y) equals dist(x)) then
         {Let c be a position such that y has a 1-bit and
           x has a 0-bit
          Let y' be y and set the c^th bit of y' to zero;
          Add edge {y', y} to T and delete y from S;
          if (x' != y') then {Add the node y' to T;
                S ← S ∪ {y'}; }
       }
  forever
End of Algorithm BUp
```

When we apply algorithm BUp to the instance given in Figure 1(a), it generates the best possible tree (Figure 1(d)). One can easily show that the time complexity for this algorithm is $O(n^2d^2)$. With respect to the time complexity bound, all the algorithms take about the same amount time. The only exception is algorithm BEstimate, which is the slowest.

4 Experimental results

The quality of the solutions generated by all the algorithms was evaluated, except for algorithm Oblivious which is clearly inferior. For comparison purposes we also evaluated the performance of algorithm MCST. Note that this procedure generates a Steiner tree that may or may not be an sp-multicast tree. But the Steiner tree that it generates has a number of edges that is at most twice of those in an optimum Steiner tree. We know that an optimum sp-multicast tree has at least as many edges as an optimum Steiner tree. Therefore, we know the best possible sp-multicast tree has at least as many edges as half the number of edges in the Steiner tree generated by the MCST.

In the first round of experiments we compared the performance of algorithms BUp, MCST, Greedy, NGrouping, Clustering, MOverlap and BEstimate. Algorithm BEstimate was run with several estimators. The estimators used were algorithms Greedy, NGrouping, Clustering, MOverlap, as well as combinations of two or more of these methods. In this first round we run each of our methods on 10,000 randomly generated problem instances on an n cube with d destination nodes. The value for $n \in \{8, 9, 10, 12, 14, 16, 18, 20, 22\}$ and the value for $d \in \{25, 35, 45, 55, 65, 75, 85, 95\}$. Thus each method was run about 720,000 times.

Figures 2a and 2b depict the results for the experimentation in the 8-cube, and 22-cube, respectively. The graphs depict the quality of the solutions generated by the methods relative to the BUp method. The x-axis represents the quality of the solution generated by the BUp method. The other curves depict the performance of the different methods. Lets consider the curve for the Clustering method in Fig. 2a. For each experiment with 25 destinations we computed the number of nodes, c, in the tree generated by the Clustering method and the number of nodes, b, in the tree generated by the BUp method. Then the relative performance of the clustering method with respect to the BUp method was computed as $(c-b)/b$. Let a be the average relative performance of the clustering method over all the experiments. Then the point $(25, a)$ is part of the curve representing the the performance of the Clustering method. In other words, the x-axis represents the number of destinations and the Y-axis is the relative performance of the method with respect to the BUp method. Note that we measure performance with respect to the number of nodes in the tree generated by the methods. A point above the the x-axis means that the BUp method generates trees with fewer nodes and a point below means that the BUp method generates trees with more nodes.

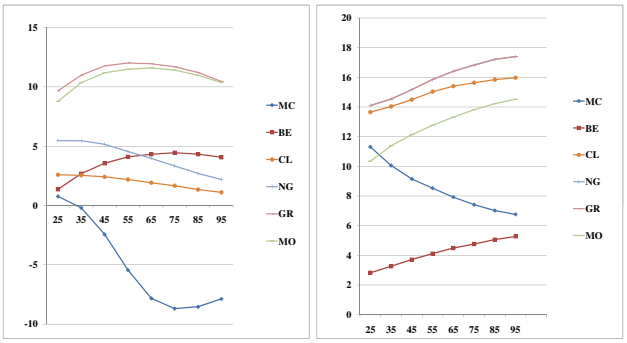

Figure 2. We use the following notation: MCST (MC), BEstimate (BE), Clustering (CL), NGrouping (NG), Greedy, (GR), and MOverlap (MO). The x-axis represents the number of destinations and the y-axis is the relative performance of the method. (a) Experiments for the 8-cube (left). (b) Experiments for the 22-cube (right).

The first observation is with respect to MOverlap and Greedy (See Figure 2). The performance of these two methods was about the same on the 8-cube, but MOverlap was about 4% better on the 22-cube. However, it is important to note that on a large range of problem instances each of these two methods was about 13% better than the other. I.e., on a large set of problem instances MOverlap was better than Greedy by about 13%, and on another large set of problem instances Greedy was better than MOverlap by about 13%.

When we run BEstimate(Best of Greedy and MOverlap) it outperformed both BEstimate(Greedy) and BEstimate(MOverlap) by about 4% to 5%. The observations given in the previous paragraph justify this result. BEstimate(Best of NGrouping and MOverlap) and BEstimate(Best of Clustering and MOverlap) were slightly worse than BEstimate(Best of Greedy and MOverlap), with no significant difference on the 22-cube. BEstimate (Best of BUp and MOverlap) was about 2% worse that BEstimate(Best of Greedy and MOverlap). In order to simplify Figure 2 we only show the results for BEstimate(Best of Greedy and MOverlap) as the representative of the BEstimate method.

In the 8-cube (Fig. 2a) one can see that Clustering outperforms NGrouping which outperforms Greedy. In the 22-cube (Fig. 2c) NGrouping and Greedy are indistinguishable and slightly worse than Clustering. The reason for this is that fewer destinations can be grouped together as we increase the number of nodes in the hypercube while keeping the the number of destination nodes fixed. The performance of BEstimate(Best of NGrouping and MOverlap) is better in the 22-cube than in the 8-cube. The MCST method is the best one in the 8-cube and it is actually better than the

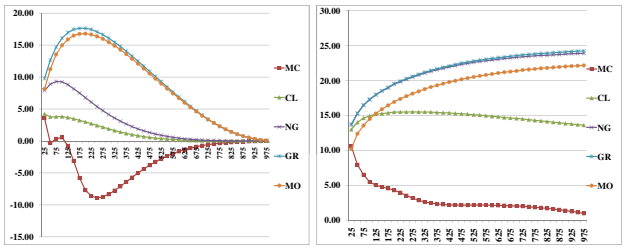

Figure 3. We use the following notation: `MCST` (MC), `Clustering` (CL), `NGrouping` (NG), `Greedy`, (GR), and `MOverlap` (MO). The x-axis represents the number of destinations and the y-axis is the relative performance of the method. (a) Experiments for the 10-cube (left). (b) Experiments for the 20-cube (right).

`BUp` method. However in the 22-cube it is outperformed by the `BEstimate`(Best of `NGrouping` and `MOverlap`) and `BUp` methods.

In the second round of experiments we compared the performance of `BUp`, `MCST`, `Greedy`, `NGrouping`, `Clustering`, and `MOverlap`. Note that since the `BUp` method was better in the first set of experiments than the `BEstimate` method, and the `BEstimate` method is significantly slower than all the other methods, we decided not to evaluate its performance in the second round. We run each of our methods on 10,000 randomly generated problem instances on an n cube with d destination nodes. The value for $n \in \{10, 12, 14, 16, 18, 20\}$ and the value for $d \in \{25, 50, 75, 100, \ldots, 950, 975\}$. Thus each method was run about 3,200,000 times.

The same observations that we made for Fig. 2 can be made for the Fig. 3. In Fig. 3(a) we observe that as the number of destinations becomes large compared to the nodes in the n-cube, all the methods perform equally and generate near optimum solutions. When the number of destinations is small, is when one can observe the greatest differences between the performance of the methods. Algorithm `BUp` outperforms the `Clustering` method by less than 5% in the 8-cube. However this increases to about 16% in the 20-cube. On larger n-cubes one observes larger differences. In a wide range of applications the number of destinations is small and that is where our methods are significantly better than the previous methods. The figures show that on average the `BUp` method is always better than the previous methods. However, one can also say that for almost all problems instances it outperforms all the other methods. There are very few cases when it generates a solution that is worse than the one generated by some of the other methods, but when that occurs the difference is minimal.

5 Conclusions

We introduce three new algorithm to construct near-optimal sp-multicast trees. We demonstrate experimentally that our methods outperform previous algorithms for this problem. In particular, the fastest of our methods (algorithm `BUp`) outperforms all other methods. It is not known whether or not algorithm `BUp` is a constant ratio approximation algorithm. This is the most important open problem in this research area.

References

[1] Angkun, T., Bosilca, G., Vander Zanden, B., and Dongarra, J., "Optimal Routing in Binomial Graph Networks," PDCAT'07, 363 –369, 2007.

[2] Choi, H.-A., and Esfahanian, A. H., "On Complexity of a Message-Routing Strategy for Multicomputer Systems," *LNCS*, 484, 170 – 181, Springer-Verlag, 1991.

[3] Choi, H.-A., Esfahanian, A. H., and Houck, B., "Optimal Communication Trees with Application to Hypercube Multiprocessors," *Graph Theory, Combinatorics, and Applications*, 1, 245 – 264, 1991.

[4] Cipriano, C.C., and Gonzalez, T. F., Multicasting in the Hypercube, Chord and Binomial Graphs, TR 2009-09, UCSB, May 2009.

[5] Du, D.-Z., and Wu, W., Approximations for Steiner Minimum Trees, in *Handbook of Approx. Alg. and Metaheuristicts*, ed. T. Gonzalez, Chapter 42, Chapman and Hall/CRC, 2007.

[6] Fujita, S., A Note on the Size of a Multicast Tree in Hypercubes, *IPL*, Vol 54, pp. 223 – 227, 1995.

[7] Graham, R. L., and Foulds, L. R., "Unlikelihood that Minimal Phylogenies for a Realistic Biological Study can be Constructed in Reasonable Computation Time", *Math. Biosciences*, 60, 133 – 142, 1982.

[8] Hwang, F. K., Richards, D. S., and Winter, P., *The Steiner Tree Problem* North-Holland, 1992.

[9] Lan, Y., Esfahanian, A. H., and Ni, L. M., Multicast in Hypercube Multiprocessors, *J. of Par. and Dist. Comp.*, Vol 8, pp. 30 – 41, 1990.

[10] Lu, S., Fan, BH, Dou, Y., and Yang, XD, Clustering Multicast on Hypercube Network, HPCC-2006, *LNCS*, 4208, pp. 61 – 70, 2006.

[11] Lu, SL and Yang, XD, A Clustering Model for Multicast on Hypercube Network, GPC-2008, *LNCS*, 5036, pp. 211 – 221, 2008.

[12] Sheu, S.-H. and Yang, C.-B., Multicast Algorithms for Hypercube Multiprocessors, *J. of Par. and Dist. Comp.*, 61, pp. 137 – 149, 2001.