An Oblivious Spanning Tree for Single-Sink Buy-at-Bulk in Low Doubling-Dimension Graphs

Srivathsan Srinivasagopalan, Costas Busch, and S.S. Iyengar, *Fellow*, *IEEE*

Abstract—We consider the problem of constructing a single spanning tree for the single-sink buy-at-bulk network design problem for doubling-dimension graphs. We compute a spanning tree to route a set of demands along a graph G to or from a designated sink node. The demands could be aggregated at (or symmetrically distributed to) intermediate edges where the fusion cost is specified by a nonnegative concave function f. We describe a novel approach for developing an oblivious spanning tree in the sense that it is independent of the number and location of data sources (or demands) and cost function at the edges. We present a deterministic, polynomial-time algorithm for constructing a spanning tree in low doubling-dimension graphs that guarantees a $\log^3 D$-approximation over the optimal cost, where D is the diameter of the graph G. With a constant fusion-cost function, our spanning tree gives an $O(\log^3 D)$-approximation for every Steiner tree that includes the sink. We also provide a $\Omega(\log n)$ lower bound for any oblivious tree in low doubling-dimension graphs. To our knowledge, this is the first paper to propose a single spanning tree solution to the single-sink buy-at-bulk network design problem (as opposed to multiple overlay trees).

Index Terms—Spanning tree, buy-at-bulk, network design, approximation algorithm, doubling-dimension graph, data fusion, data structure.

1 Introduction

A typical client-server model has many clients and one server where a subset of the client set wishes to route a certain amount of data to the server at any given time. The set of clients and the server are assumed to be geographically far apart. To enable communication among them, there needs to be a network of cables deployed. Moreover, the deployment of network cables has to be of minimum cost that also minimizes the communication cost among the various network components. This is what we roughly call a typical network design problem. The same problem can be easily applied to many similar practical scenarios such as oil/gas pipelines and telephone network.

The "Buy-at-Bulk" network design considers the economies of scale into account. As observed in [2], in a telecommunication network, bandwidth on a link can be purchased in some discrete units $u_1 < u_2 < \cdots < u_n$ with costs $c_1 < c_2 < \cdots < c_n$, respectively. The economies of scale exhibit the property where the cost per bandwidth decreases as the number of units purchased increases: $c_1/u_1 > c_2/u_2 > \cdots > c_n/u_n$. This property is the reason why network capacity is bought/sold in "wholesale," or why vendors provide "volume discount."

There are different variants of buy-at-bulk network design problems that arise in practice. One of them is "single-sink buy-at-bulk" (SSBB) network design. This SSBB problem has a single "destination" node where all the demands from other nodes have to be routed to. Typically, the demand flows are in discrete units and are unsplittable (indivisible), i.e., the flow follows a single path from the demand node to the destination. These problems are often called "discrete cost network optimization" in operations research.

As mentioned in [3], if information flows from x different sources over a link, then the cost of total information that is transmitted over that link is proportional to $f(x)$, where $f : \mathbb{Z}^+ \rightarrow \mathbb{R}^+$. The function f is called a *canonical* fusion function if it is concave, nondecreasing, $f(0) = 0$, and has the subadditive property $f(x_1 + x_2) \leq f(x_1) + f(x_2)$, $\forall x_1, x_2, (x_1 + x_2) \in \mathbb{Z}^+$. Generally, SSBB problems use the subadditive property to ensure that the "size" of the aggregated data is smaller than the sum of the sizes of individual data. If the set of demand nodes is known in advance and f is constant, then this is a well-known Steiner tree problem.

We study the *oblivious* single-sink buy-at-bulk network design problem with the following constraints: an unknown number of source (or demand) nodes and an unknown concave transportation cost function f. An abstraction of this problem can be found in many applications, one of which is data fusion in wireless sensor networks where constraints such as the number and location of source nodes are assumed unknown or vary over time. Others include design of VLSI power circuitry, Transportation and Logistics (railroad, water, oil, gas pipeline construction), etc. For simplicity, we consider data fusion problems in communication networks. Our solution holds for both data distribution and aggregation problems in doubling-dimension graphs. Informally, a graph has doubling dimension ρ, if there is a smallest ρ such that for every radius $r > 0$, every ball of radius $2r$ can be covered by at most 2^ρ balls of radius r. When ρ is small (constant), the graph is of low doubling dimension.

Doubling-dimension graphs have been used in many different contexts including compact routing in wired

• *The authors are with the Computer Science Department, Louisiana State University, Tower Dr, Baton Rouge, LA 70803.*
E-mail: ssrini1@tigers.lsu.edu, {busch, iyengar}@csc.lsu.edu.

Manuscript received 27 Sept. 2010; revised 15 Feb. 2011; accepted 20 Feb. 2011; published online 17 Mar. 2011.
Recommended for acceptance by J. Chen.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-09-0538.
Digital Object Identifier no. 10.1109/TC.2011.64.

networks [4], [5], [6], hierarchical routing and low-diameter networks [7], [8], traveling salesman, navigability and problems related to modeling the structural properties of the Internet distance matrix for distance estimation [9], [10]. As noted in [11], it has become a key concept to measure the ability of networks to support efficient algorithms or to realize specific tasks efficiently. For wireless networks, this concept has found many uses in solving many distributed communication problems [12], distributed resource management [13], information exchange among producers and consumers [14], and for determining other performance qualities such as energy conservation in wireless sensor networks [15].

1.1 Problem Statement

Assume that we are given a weighted graph $G = (V, E, w)$, with edge weights $w : E \longrightarrow \mathbb{R}_{\geq 1}$, with a sink $s \in V$. We denote w_e to be the weight of edge e. Let $A = \{v_1, v_2, \ldots, v_d\}$, $A \subseteq V$ be the set of demand nodes. Let each node $v_i \in A$ have a nonnegative unit demand. A demand from v_i induces a unit of flow to sink s and this flow is unsplittable. The demands from various demand nodes have to be sent to the destination node s possibly routed through multiple edges in the graph G. This forms a set of paths $P(A) = \{p(v_1), p(v_2), \ldots, p(v_d)\}$, where $p(v_i)$ is the path from $v_i \in A$ to s. The output for a given graph G, sink s, and a set of demand nodes A is a set of paths P from the nodes in A to s. We seek to find such a set of paths with minimal cost with respect to a cost function described below.

There is an arbitrary concave fusion-cost function f at every edge where data aggregate. This f is the same for all the edges in G. Let $p(v)$ be the path taken by a flow from v to s in G. Let $\varphi_e(A) : \{p(v) : e \in p(v) \wedge v \in A\}$ denote the set of paths originating from nodes in A that use an edge $e \in E$. Then, we define the cost of an edge e to be $C_e(A) = f(|\varphi_e(A)|) \cdot w_e$. The total cost of the set of paths is defined to be $C(A) = \sum_e C_e(A)$.

For a given set A of demand nodes in G, the corresponding set of paths $P(A)$ would incur a total cost denoted by $C(A)$. For this set A, there is an optimal set of paths $P^*(A)$ with respect to the total cost denoted by $C^*(A)$. The competitive ratio for the cost of these two sets of paths is given by $\frac{C(A)}{C^*(A)}$.

The oblivious case arises when we do not know the set of demand nodes in advance. So, given a graph $G = (V, E)$ with sink $s \in V$, an *oblivious* algorithm, $\mathcal{A}_{obl}$, must compute a set of paths $P(V)$ which induces $P(A)$ for any set $A \subseteq V$. The competitive ratio of this oblivious algorithm is given by

$$C.R.(\mathcal{A}_{obl}) = \max_{A \subseteq V} \frac{C(A)}{C^*(A)}.$$

We aim to find an oblivious algorithm that minimizes the above competitive ratio. We note that SSBB is NP-Hard as the Steiner tree problem is a special of case of SSBB (when $f(x) = 1$) [16].

1.2 Contribution

We seek to find a spanning tree T rooted at sink s for any doubling-dimension graph G. The spanning tree T we build produces a set of unique paths $P(V)$ from $\forall v \in V$ to the sink s. This T is oblivious since it is independent of the data sources, and can accommodate *any* canonical fusion-cost function. Our approach gives a deterministic, polynomial-time algorithm that guarantees $O(2^{17\rho} \log^3 D)$ competitive ratio for graphs with doubling dimension ρ. Therefore, for low doubling-dimension graphs, we obtain an $O(\log^3 D)$ competitive ratio. When $f(\cdot) = c$, a constant, our spanning tree solution provides an $O(\log^3 D)$-approximation to any Steiner tree that contains the sink s. To our knowledge, these are the first spanning tree solutions to the oblivious SSBB problem and also for the oblivious Steiner tree problem. We also give a lower bound in $n \times n$ grids for the competitive ratio for any oblivious SSBB spanning tree T to be of $\Omega(\log n)$.

It is well known in the research community that tree structures provide a very efficient solution for managing data dissemination and aggregation in large-scale distributed systems. Prominent architectures like the content-based publish subscribe, peer-to-peer communication, muticasting, etc., take advantage of efficient routing in trees and distributed maintenance of the tables in each node of the network.

The motivation for us to build a spanning tree not only comes from the above mentioned advantages and current use, but also because of the fact that it has the most compact form of data structure in the sense that they have the minimum number of edges connecting all the nodes $(n - 1)$. Furthermore, their inherent acyclic property conveniently avoids inefficient use of the network due to unnecessary cyclic data traversal and hence avoids increased costs. Since there are no routing loops formed during the tree construction, any design of routing algorithms on trees is greatly simplified.

We build a spanning tree based on the following technique. We partition the nodes in a hierarchical fashion. The selection of nodes for a given "level" of hierarchy is based on finding d-independent nodes, where d is proportional to that level. Nodes of successive levels are connected by bounded length paths. The intersecting paths that may potentially form cycles are appropriately modified to result in a spanning tree. A modified spanning tree is built from the spanning tree to ensure that all paths have appropriate end nodes. Analysis is done on this modified tree.

To demonstrate the basic techniques and concepts, we initially build an overlay tree and produce a $\log D$ competitive ratio. An overlay tree is a tree where each edge in the tree could be a path in the underlying physical infrastructure. Shortest paths in an overlay tree, when projected to its underlying network, could have several intersections leading to cycles. Our initial overlay tree construction and analysis give an insight for the analysis of the spanning tree that we build subsequently. Since the overlay tree may result in having cycles, our main algorithm for constructing a spanning tree extends the overlay tree algorithm to obtain a competitive ratio of $O(\log^3 D)$.

We perform simulation to compare the cost of the spanning tree with trees from several prior related work and a few well-known trees (Minimum Spanning Tree (MST) and Shortest-Paths Tree (SPT)). For comparison, we generate the trees and costs by simulation using NetworkX

TABLE 1
Our Results and Comparison with Previous Results for Data-Fusion Schemes

Related Work	Algorithm Type	Graph Type	Oblivious Function f	Oblivious Sources	Approx Factor	Tree Type
Lujun Jia *et al.* [30]	Deterministic	Random Deployment	×	✓	$O(\log n)$	One Overlay
Lujun Jia	Deterministic	Arbitrary Metric	×	✓	$O(\frac{\log^4 n}{\log\log(n)})$	Universal Steiner Tree (Overlay)
et al. [31]	Deterministic	Doubling Metric	×	✓	$O(\log(n))$	Universal Steiner Tree (Overlay)
Ashish Goel *et al.* [3]	Randomized	General Graph $\triangle$-inequality	✓	×	$O(\log k)$	One Overlay
Ashish Goel *et al.* [32]	Probabilistic	General Graph	✓	×	$O(1)$	Multiple Overlay
Anupam Gupta	Randomized	General Graph	✓	✓	$O(\log^2 n)$	Multiple Overlay
et al. [33]	Randomized	Low Doubling	✓	✓	$O(\log n)$	Multiple Overlay
This paper	Deterministic	Low Doubling Dimension	✓	✓	$O(\log^3 D)$	One Spanning

n is the total number of nodes in the topology, k is the total number of source nodes. Note that our work gives a spanning tree and others provide an overlay tree that may have cycles.

[17]. The simulations corroborate the analytical results and show that the oblivious spanning tree (OST) provides very competitive costs and in fact provides better costs than the well-known trees.

1.3 Related Work

1.3.1 Non-Oblivious SSBB

There has been a lot of research work in the area of approximation algorithms for network design. Since network design problems have several variants with several constraints, only a partial list has been mentioned in the following paragraphs.

SSBB problems have been primarily considered in both Operations Research and Computer Science literatures in the context of flows with concave costs. SSBB problem was first introduced by Salman et al. [16]. They presented an $O(\log n)$-approximation for SSBB in euclidean graphs by applying the method of Mansour and Peleg [18]. Bartal's tree embeddings [19] can be used to improve their ratio to $O(\log n \log\log n)$. An $O(\log^2 n)$-approximation was given by Awerbuch and Azar [20] for graphs with general metric spaces. Bartal [21] further improved this result to $O(\log n)$. Guha et al. [22] provided the first constant factor approximation to the problem, whose ratio was estimated to be around 9,000 by Talwar [23].

Some other special cases of the problem have also constant factor approximations. Algorithms by Kumar et al. [24] and Gupta et al. [25] provide constant factor approximation algorithms for the rent-or-buy variation of the problem. They provide a 76.8-approximation algorithm for the splittable-SSBB problem. Talwar [23] proposed an LP rounding approach for the SSBB problem with an approximation ratio of 216. Jothi and Raghavachari [26] provide an improvement over Talwar's with a 145.6-approximation and guaranteeing that each flow follows a single path to the sink. Their work also proposes a technique for the splittable-flow SSBB problem which reduces the previous best ratio of 72.8 to α_K which is less than 65.49 for all K-types of cables (each type has a specified capacity and cost per unit length).

Another variant is the "capacitated" buy-at-bulk network design problem where each edge (link) of the network has an upper bound on the amount of demand flows it can route through it. This problem is otherwise known as *network loading* problem. Many heuristic and branch-cut approaches have been used to solve such problems. Frangioni and Gendron [27] show that a nontrivial 0-1 reformulation of the Multicommodity Network Design (MCND) provides the same LP bound obtained by adding exponentially many residual capacity inequalities to the LP relaxation of the general integer formulation. Gendron et al. [28] provide a survey of methods that solve MCND, particularly through LP relaxations. The methods highlighted are the simplex-based cutting plane algorithms, Lagrangian relaxation, and heuristics. Öncan [29] provides a fast approximate reasoning algorithm, which is based on the Esau-Williams savings heuristic and fuzzy logic rules to solve this problem.

1.3.2 Oblivious SSBB

Below, we present the related work in oblivious SSBB and Table 1 summarizes most of these results and compares our work with their's. What distinguishes our work with the others' is the fact that we provide a spanning tree while others provide an overlay tree that may have cycles.

Goel and Estrin [3] build an overlay tree on a graph that satisfies the triangle inequality. Their technique is based on a maximum matching algorithm that guarantees $(1+\log k)$-approximation, where k is the number of sources. Their solution is oblivious with respect to the fusion-cost function f. An overlay tree, if projected to a graph, may not be a tree (could have cycles). In a related paper [32], Goel and Post construct (in polynomial time) a set of overlay trees from a given general graph such that the expected cost of a tree for any f is within an $O(1)$-factor of the optimum cost for that f.

Jia et al. [30] build a Group Independent Spanning Tree (GIST) Algorithm that constructs an overlay tree for randomly deployed nodes in a euclidean two-dimensional plane. The tree (that is oblivious to the number of data sources) simultaneously achieves $O(\log n)$-approximate fusion cost and $O(1)$-approximate delay. However, their solution assumes a constant fusion-cost function. We summarize and compare the related work in Table 1.

Jia et al. [31] provide approximation algorithms for TSP, Steiner Tree, and set cover problems. They present a polynomial-time $(O(\log(n)), O(\log(n)))$-partition scheme for general metric spaces. An improved partition scheme for doubling metric spaces is also presented that incorporates constant-dimensional euclidean spaces and growth-restricted metric spaces. The authors present a polynomial-time algorithm for Universal Steiner Tree (UST) that achieves polylogarithmic stretch with an approximation guarantee of $O(\log^4 n/\log\log(n))$ for arbitrary metrics and derive a logarithmic stretch, $O(\log(n))$ for any doubling, euclidean, or growth-restricted metric space over n vertices.

Gupta et al. [33] develop a framework to model *oblivious network design* problems and give algorithms with polylogarithmic competitive ratio. They develop oblivious algorithms that approximately minimize the total cost of routing with the knowledge of aggregation function, the class of load on each edge, and nothing else about the state of the network. Their results show that if the aggregation function is summation, their algorithm provides an $O(\log^2 n)$ competitive ratio and when the aggregation function is max, the competitive ratio is $O(\log^2 n \log\log n)$. The authors claim to provide a deterministic solution by derandomizing their approach. But, the complexity of this derandomizing process is unclear.

Chuzhoy et al. [34] consider the Fixed Charge Network Flow (FCNF) problem and show that this problem and several other basic network design problems cannot be approximated better than $\Omega(\log\log n)$ unless $NP \subseteq DTIME(n^{O(\log\log\log n)})$. They show that this inapproximability threshold holds for the Priority-Steiner Tree problem, single-sink Cost-Distance problem, and the single-sink FCNF problem.

A lower bound for the summation aggregation function is provided in the online Steiner tree problem by Imase and Waxman [35]. This provides a $\Omega(\log n)$ competitive ratio for planar graphs. However, the specific planar graph they used is not of low doubling dimension. For this reason, we provide an alternative lower bound for low doubling graphs, in particular for two-dimensional grids.

1.4 Organization

In the next section, we present some definitions and notations used throughout the rest of the paper. Section 3 provides the description and analysis of an overlay tree which will be useful for the analysis of the spanning tree that we build later. In Section 4, we describe a spanning tree algorithm. Section 5 contains the modified spanning tree construction algorithm. Section 6 provides the analysis of the modified spanning tree as well as the main theorem of this paper. Section 7 discusses the lower bound analysis. In Section 8, we briefly describe our simulation results comparing our tree with several well-known trees. Finally, we discuss our contribution and future work in Section 9.

2 Definitions

Consider a weighted graph $G = (V, E, w)$, $w : E \longrightarrow \mathbb{R}_{\geq 1}$. Let $s \in V$ be the sink node. For any two nodes $u, v \in V$, let $\mathrm{dist}(u, v)$ denote the *distance* between u, v (measured as the total weight of the shortest path that connects u and v). Given a subset $V' \subseteq V$, we denote $\mathrm{dist}(u, V')$ the smallest distance between u and any node in V'. Let D denote the *diameter* of G, that is, $D = \max_{u,v \in V} \mathrm{dist}(u, v)$. For any path p denote its length (number of edges) as $|p|$.

A set of nodes I is said to be *a d-independent set* if for each pair $u, v \in I$, $u \neq v$, $\mathrm{dist}(u, v) \geq d$. Given a set of nodes $H \subseteq V$ and parameter d, we define *Maximal Independent Set of G for distance d* as $I = MIS(G, H, d)$ to be an arbitrary maximal d-independent set of nodes in G such that $H \subseteq I$. Note that, to begin with, the nodes in the given set H must also be d-independent. $MIS(G, H, d)$ can be constructed in polynomial time with a simple greedy algorithm.

Given a graph $G = (V, E)$, the *r-neighborhood* of any vertex $u \in V$ denoted $N(u, r)$, is defined as the set of nodes whose distance is at most r from u; namely, $N(u, r) = \{v | \mathrm{dist}(u, v) \leq r\}$. The r-neighborhood of a set of vertices $V' \in V$ denoted by $N(V', r)$, is defined as the set of nodes whose distance is at most r from any node in v'. We adapt the definition of doubling-dimension graph from [36] and [37].

Definition 2.1 (Doubling Dimension of a Graph). *The doubling dimension of a graph G is the smallest ρ such that every r-neighborhood is a subset of the union of at most 2^ρ sets of $r/2$-neighborhoods. If ρ is constant, then we say that G is of low doubling dimension.*

Observation 2.2. *For a graph with doubling dimension ρ, any 1-neighborhood contains at most 2^ρ nodes. Any 2^k-neighborhood, can be covered by at most $2^{(k-l)\rho}$ number of 2^l-neighborhoods, where $k \geq l \geq 0$.*

Lemma 2.3. *In any 2^k-neighborhood, the size of any 2^l-independent set of nodes does not exceed $2^{(k-l+3)\rho}$, where $k \geq l \geq 0$.*

Proof. Let U be 2^k-neighborhood of a node v. Let I be a 2^l-independent set of nodes in the 2^k-neighborhood of a node v. If $0 \leq l \leq 2$, then $|I| \leq |U| \leq 2^{(k+1)\rho} \leq 2^{(k-l+3)\rho}$ (from Observation 2.2). If, $l \geq 3$, from Observation 2.2, U can be covered by at most $2^{(k-l+3)\rho}$ number of 2^{l-3}-neighborhoods. Therefore, we have that $|I| \leq 2^{(k-l+3)\rho}$.□

3 Overlay Tree

We describe how to construct an overlay tree from a connected graph $G = (V, E)$. This will be useful for the design and analysis of the spanning tree algorithm.

The overlay tree $T = (V_T, E_T)$ is built as follows: let $\kappa = \lceil \log D \rceil$, where D is the diameter of graph G. The overlay tree T consists of $\kappa + 1$ levels of node sets, $V_T = I_0 \cup \cdots \cup I_\kappa$, which are selected in a top-down manner. The root of T is s and $I_\kappa = \{s\}$. Given I_{i+1}, we define $I_i = MIS(G, I_{i+1}, 2^i)$. The leaves of T are all the nodes in G, namely, $I_0 = V$. Members of I_i are also called *leaders* at level i. Note that some leaders could belong to multiple levels (e.g., the sink s is a member of all levels). For any node $u \in I_i$, $i < \kappa$, its parent in T is chosen to be a leader in $I_{i+1} \cap N(u, 2^{i+2} - 2)$ which is closest to s (a parent is guaranteed to exist due to the maximal independent set property of I_{i+1}).

For every edge $(u,v) \in E_T$, where $u \in I_i$ and $v \in I_{i+1}$, we select one of the shortest paths from u to v to be the *designated path* from u to v to represent edge (u,v). In case $u = v$, the designated shortest path has length zero. For any node v, the tree T defines a unique path $q(v) = (e_0, e_1, \ldots, e_{\kappa-1}) \in T$ from the leaf v to the root s. The path $q(v)$ is translated to a unique path $p(v) = (p_0(v), p_1(v), \ldots, p_{\kappa-1}(v))$ from v to s in G by replacing each edge $e_i \in q(v)$ with the respective designated shortest path $p_i(v)$. We will refer to $p_i(v)$ as the *layer-i subpath* of $p(v)$.

3.1 Basic Properties of Overlay Tree

For each node $u \in I_i$, let Z_i^u denote all the leaves in T which appear in the subtree of T rooted at u at level i. The overlay tree T naturally defines a hierarchical partition of G because for any $v \neq u$, $Z_i^u \neq Z_i^v$ and for all $y \in G$, $y \in Z_i^x$ for any x.

We will use the following parameters for the analysis of overlay trees. Please note that the same set of parameters with appropriately modified values will be later used in Section 6 for the modified tree analysis.

$$\begin{aligned}
\mu_i &= 2^{i+2} // \text{upper bound on } |p_i(u)| \\
\delta_i &= 2^{i+2} \ // \text{upper bound on the radius of } Z_i^u \\
\phi_i &= 2^i // \text{lower bound on dist } (s, Z_i^u), u \neq s \\
\xi_i &= 2\delta_i + 2\phi_i \ // \text{coloring radius} \\
\chi &= 2^{7\rho} // \text{coloring of } I_i \text{ with radius } \xi_i.
\end{aligned}$$

For each path $p_i(v)$, we have $|p_i(v)| \leq 2^{i+2} - 2 < \mu_i$, and hence, we obtain

Observation 3.1. *For any node $v \in V$, $|p_i(v)| < \mu_i$.*

Lemma 3.2. *For any $v \in Z_i^u$, $\mathrm{dist}(v,u) < \delta_i$.*

Proof. Let $p'(v) = (p_0(v), p_1(v), \ldots, p_{i-1}(v))$ be the respective path in the overlay tree from v to u. From Observation 3.1, $|p_j(v)| < \mu_j = 2^{j+2}$. Thus, $|p'(v)| = \sum_{j=0}^{i-1} |p_j(v)| < \sum_{j=0}^{i-1} 2^{j+2} < 2^{i+2} = \delta_i$. □

Lemma 3.3. $N(s, 2^i - 1) \subseteq Z_i^s$.

Proof. Consider a node $v \in Z_i^s$, with $v \neq s$. Suppose that $v \in I_j$, where $j < i$. Let ℓ_{j+1} denote the parent of v. According to the parent selection criterion, $\ell_{j+1} \in I_{j+1} \cap N(v, 2^{j+2} - 2)$ and ℓ_{j+1} is closest to s.

We first show that if $v \in N(s, 2^i - 1)$, then $\ell_{j+1} \in N(s, 2^i - 1)$. We only need to show that $B = I_{j+1} \cap N(s, 2^i - 1) \neq \emptyset$. Let r_v denote the shortest path from v to s. If $|r_v| \leq 2^{j+2} - 2$, then $s \in B$, and $B \neq \emptyset$. Suppose that $|r_v| > 2^{j+2} - 2$. Take a node $x \in r_v$ such that $\mathrm{dist}(x,v) = 2^{j+1} - 1$. Let r_x denote the subpath of r_v from x to s. If we consider a neighborhood $N(x, 2^{j+1} - 1)$, then, there is a node $y \in I_{j+1}$ such that $y \in N(x, 2^{j+1} - 1)$ and $dist(x,y) \leq 2^{j+1} - 1$. Let r_y denote the shortest path from y to s. We have that $|r_y| \leq |r_x| + 2^{j+1} - 1 = |r_v|$. Consequently, $y \in B$, and $B \neq \emptyset$.

We can easily see that if $v \in I_{i-1}$ and $v \in N(s, 2^i - 1)$, then the parent of v is s, and thus $v \in Z_i^s$. Using an induction on $j = i-1, \ldots, 0$, we obtain that if $v \in I_j$ and $v \in N(s, 2^i - 1)$, then $v \in Z_i^s$. Consequently, when we consider $j = 0$, we obtain that $N(s, 2^i - 1) \subseteq Z_i^s$. □

From Lemma 3.3, we obtain the following corollary:

Corollary 3.4. *For any $u \in I_i$, $u \neq s$, $\mathrm{dist}(s, Z_i^u) \geq \phi_i$.*

Let $X_i = (I_i, E_{X_i})$, be a graph such that for any two $u, v \in I_i$, $(u,v) \in E_{X_i}$ if and only if $\mathrm{dist}(u,v) \leq \xi_i$.

Lemma 3.5. *Graph X_i admits a vertex coloring with at most χ colors.*

Proof. Let $v \in I_i$. The nodes adjacent to v in I_i are the set $Y = N(v, \xi_i) \cap I_i$. Since I_i is a 2^i-independent set, and $\xi_i = 2\delta_i + 2\phi_i = 2^{i+3} + 2^{i+1} \leq 2^{i+4}$, from Lemma 2.3, we obtain $|Y| \leq 2^{((i+4)-i+3)\rho} = 2^{7\rho}$. Consequently, graph X_i has degree at most $2^{7\rho} - 1$, and by a greedy algorithm, it can be colored with at most $\chi = 2^{7\rho}$ colors. □

3.2 Competitive Analysis of Overlay Tree

Let $A \subseteq V$ denote an arbitrary set of source nodes. Let $C^*(A)$ denote the cost of the of the optimal path set from A to s. Let $C(A)$ denote the cost of the paths given by the overlay tree T. We will bound the competitive ratio $C(A)/C^*(A)$.

The cost $C(A)$ can be bounded as a summation of costs from the different layers as follows: for any edge e, let $\varphi_{e,i}(A) = \{p_i(v) : (v \in A) \wedge (e \in p_i(v))\}$ be the set of layer-i subpaths that use edge e. Recall that the fusion-cost function $f : \mathbb{Z}^+ \to \mathbb{R}^+$ is concave, nondecreasing, and has the subadditive property $f(x_1 + x_2) \leq f(x_1) + f(x_2)$, $\forall x_1, x_2, (x_1 + x_2) \in \mathbb{Z}^+$ where $f(0) = 0$. Denote by $C_{e,i}(A) = f(|\varphi_{e,i}(A)|) \cdot w_e$ the cost on the edge e incurred by the level-i subpaths. Since f is subadditive, we get $C_e(A) \leq \sum_{i=0}^{\kappa-1} C_{e,i}(A)$. Let $C_i(A) = \sum_{e \in E} C_{e,i}(A)$ denote the cost incurred by the layer-i subpaths. Since $C(A) = \sum_{e \in E} C_e(A)$, we have that

$$C(A) \leq \sum_{i=0}^{\kappa-1} C_i(A). \tag{1}$$

Let $A_i^u = A \cap Z_i^u$. We obtain the following lower bound on $C^*(A)$:

Lemma 3.6. *For any ξ_i-independent set $I' \subseteq I_i$, $C^*(A) \geq R(I')$, where $R(I') = \sum_{u \in I' \setminus s} f(|A_i^u|) \cdot \phi_i$.*

Proof. From Lemma 3.2, any node in A_i^u is at distance at most $\delta_i - 1$ from u. Since any pair $u, v \in I' \setminus \{s\}$, $u \neq v$, is at least $\xi_i = 2\delta_i + 2\phi_i$ distance apart, any two nodes $x \in A_i^u$ and $y \in A_i^v$ are at least $2\phi_i$ distance apart. From Corollary 3.4, $s \notin N(A_i^u, \phi_i - 1)$. Let $Y(A_i^u)$ be the set of edges with one node in $N(A_i^u, \phi_i - 1)$ and the other outside $N(A_i^u, \phi_i - 1)$. The set $Y(A_i^u)$ forms a cut that has to be crossed by the paths in A_i^u in order to reach s. The smallest cost for crossing the cut is when the paths of A_i^u are combined through the fusion function f. Therefore, each path from A_i^u requires length at least ϕ_i in order to reach s. Thus, we have that the optimal cost of sending the demands from A_i^u to s is at least $f(|A_i^u|) \cdot \phi_i$. Since for each $u \in I' \setminus s$, the respective cuts are disjoint, we obtain: $C^*(A) \geq \sum_{u \in I' \setminus s} f(|A_i^u|) \cdot \phi_i$. □

Lemma 3.7. *$C_i(A) \leq Q_i$, where $Q_i = \sum_{u \in I_i \setminus \{s\}} f(|A_i^u|) \cdot \mu_i$.*

Proof. Note that $\varphi_{e,i}(A) = \bigcup_{u \in I_i} \varphi_{e,i}(A_i^u)$. Since f is subadditive, for any edge e,

$$C_{e,i}(A) = f(|\varphi_{e,i}(A)|) \cdot w_e \leq \sum_{u \in I_i} f(|\varphi_{e,i}(A_i^u)|) \cdot w_e.$$

Since for $e \in p_i(u)$, $|\varphi_{e,i}(A_i^u)| = |A_i^u|$, and for $e \notin p_i(u)$, $|\varphi_{e,i}(A_i^u)| = 0$, using Observation 3.1, we obtain

$$C_i(A) \leq \sum_{u \in I_i} f(|A_i^u|) \cdot |p_i(u)| \leq \sum_{u \in I_i \setminus \{s\}} f(|A_i^u|) \cdot \mu_i.$$

□

Lemma 3.8. $C_i(A) \leq C^*(A) \cdot \chi \cdot \mu_i/\phi_i$.

Proof. From Lemma 3.5, graph X_i accepts a vertex coloring with at most χ colors. Let I_i^j denote the set of nodes of X_i which receive color $j \in \Psi = \{1, \ldots, \chi\}$. Note that $I_i = \sum_{j \in \Psi} I_i^j$, and $I_i^j \cap I_i^k = \emptyset$ for any $j \neq k$. Let $Q_i^j = \sum_{u \in I_i^j \setminus \{s\}} f(|A_i^u|) \cdot \mu_i$. We have that $Q_i = \sum_{j \in \Psi} Q_i^j$. Let $Q_i^{j^*} = \max_{j \in \Psi} Q_i^j$. Thus, $Q_i \leq |\Psi| \cdot Q_i^{j^*} \leq \chi \cdot Q_i^{j^*}$. From Lemma 3.7, we have that $C_i(A) \leq Q_i \leq \chi \cdot Q_i^{j^*}$. Further, from Lemma 3.6, $C^*(A) \geq R(I_i^{j^*}) = Q_i^{j^*} \cdot \phi_i/\mu_i$. Consequently, $C_i(A) \leq C^*(A) \cdot \chi \cdot \mu_i/\phi_i$. □

Since A is chosen arbitrarily, the following theorem follows immediately from (1) and Lemma 3.8:

Theorem 3.9 (Oblivious Competitive Ratio of Overlay Tree). *The oblivious competitive ratio of the overlay tree T is $C.R.(T) \leq \chi \cdot (1 + \log D) \cdot \max_i\{\mu_i/\phi_i\}$.*

From Theorem 3.9, we immediately obtain the following corollary when we replace the values of the parameters.

Corollary 3.10. *The oblivious competitive ratio of the overlay tree T is $C.R.(T) = O(2^{7\rho} \cdot \log D)$.*

4 Spanning Tree Construction

We start with an informal description of the construction of the spanning tree. We build the tree in a hierarchical manner that has $\kappa = O(\log D)$ levels. A formal description appears in Algorithm 1. The terms and notations used here are the same as defined for the overlay tree construction.

The construction of the hierarchical levels of independent nodes is top down. I_i is computed by $MIS(G, I_{i+1}, 2^i)$, for $0 \leq i \leq \kappa - 1$. I_i will contain all the 2^j-independent nodes of higher levels j, $i < j \leq \kappa$ as well as a 2^i-independent set of nodes. We enforce the constraint that $s \in I_i$ for every I_i. Note that each node $v \in I_i \setminus I_{i+1}$ has to be within distance $2^{i+2} - 2$ to at least one node in I_{i+1} (otherwise, v must be a member of I_{i+1}).

Paths are also constructed in a top-down fashion. The path from any level i, denoted $p_i(v)$, starts at some leader v at level i and ends at a leader at level $i+1$. The set of all paths at level i is denoted as P_i and the set of all paths of all levels is denoted by $P = \{P_{\kappa-1}, P_{\kappa-2}, \ldots, P_2, P_1, P_0\}$. The path computation is detailed in the function `FindPath`.

The main objective of `FindPath` function is to ensure that any node u at level i is in $N(s, 2^i - 1)$ and that all the nodes in that neighborhood fall inside the subtree Z_i^s rooted at s at level i. The function `FindPath` enforces this condition by computing paths that have the following properties:

1. If there is a node u at level $i \leq j + 3$, a shortest path to s is directly built.
2. If there is a node u at level $i > j + 3$ and is close to a fixed ring r_k, then it finds an $(i+1)$-level leader inside the $(2^k - 1)$-ring. Once a leader is chosen, a special path $p_i(u)$ is built from u to ℓ_{i+1}. Path $p_i(u)$ is built such that for each node $v \neq u$ on $p_i(u)$, $\text{dist}(v, s) \leq \text{dist}(u, s)$. The existence of such a leader ℓ_{i+1} is guaranteed.

The Function `FindPath` ensures that if path $p_i(u)$ crosses a fixed ring r_k, then the path does not cross back and goes outside r_k. In order to satisfy this property, `FindPath` guarantees to find a leader *inside* r_k. Hence, any path from a node that is inside $N(s, 2^i - 1)$ stays within that neighborhood. This guarantees that $N(s, 2^i - 1) \subseteq Z_i^s$. Details are in Lemma 6.3.

Algorithm 1: Spanning Tree

Input: Graph G with sink s.
Output: A spanning tree T_s.

```
1  P ← ∅; I_κ ← {s} ;                          // κ ← ⌈log D⌉
2  P^reg ← ∅; P^pr ← ∅ ; // List of regular and
   pruned paths
3  foreach level i = κ − 1 to 0 do
4      I_i ← MIS(G, I_{i+1}, 2^i);
5      foreach v ∈ I_i do
6          p_i(v) ← FindPath(v, i);
7          if p_i(v)
           intersects any path at level > i at point u
           then
               // Prune path p_i(v) by removing
                  segment from u to ℓ
8              p'_i(v) ← path segment from v to u;
9              P_i^pr ← P_i^pr ∪ p'_i(v);
10         else
11             P_i^reg ← P_i^reg ∪ p_i(v);
12         end
13     end
14 end
15 P ← ⋃_{i=0}^{i=κ−1} P_i^reg ∪ ⋃_{i=0}^{i=κ−1} P_i^pr;
16 return T_s ;          // Formed by paths in P
```

When paths for all levels are built, the resulting structure may not be a tree. It could result in a graph that might have intersecting paths. Define *regular* paths as paths that do not intersect any (higher level) path on their way to their end nodes. The paths of $P_{\kappa-1}$, are regular paths, since there were no higher level paths to intersect and are included in $P_{\kappa-1}^{reg}$.

Define *pruned paths* as those paths that intersect paths of higher level. If a path $p_i(v)$ intersects a path $p_j(v')$ $(j > i)$ along its way to ℓ_{i+1}, $p_i(v)$ is pruned from the intersection point to its destination. Such paths are included in P_i^{pr}. This pruning of intersecting paths ensures the structural property of a spanning tree (see Fig. 1).

Note that regular paths of the same level could intersect and continue on different directions to reach a common leader. In this case, one of the paths is modified to use the same segment as the other after the intersection point. Another scenario is when two paths (say from u and v of level i) intersect at m and proceed to their respective end nodes x and y. In this case, either v or u will choose a common leader and appropriately modify its path. In both

these scenarios, the resulting paths remain regular and avoid cycles when they overlap. Note that in both the cases, the path segments, after intersection, should have the same length. We have not mentioned this aspect in Algorithm 1.

The spanning tree algorithm executes in polynomial time with respect to the size of the graph.

Function FindPath(u, j)

Input: Node u at level j.
Output: A path $p_j(u)$, that connects u to $\ell_{j+1} \in I_{j+1}$.

Let r_k be fixed rings with radius $2^k - 1$ around s, $\forall k \leq \kappa$ and $k > j + 3$;
if $\text{dist}(u, s) \leq 2^{j+3} - 1$ **then**
  $\ell_{j+1} \leftarrow s$;
  $p_j(u) \leftarrow$ Shortest path from u to ℓ_{j+1};
  return $p_j(u)$;
end
Let r_k be the first fixed ring intercepted by the shortest path from u to s;
if $\text{dist}(u, r_k) \leq 2^{j+2} - 2$ **then**
  Let y be the intersection point on the ring r_k with the shortest path from u to s ;
  // $\text{dist}(u, y) \leq 2^{j+2} - 2$
  Let q_1 be a path segment from u to y;
  Let x be a point on the shortest path from u to s and $\text{dist}(y, x) = 2^{k-1} - 1$;
  Let q_2 be a path segment from y to x;
  $u' \leftarrow v \in N(s, 2^k - 1) \cap I_{j+1}$ and $\text{dist}(x, v) \leq 2^{k-1} - 1$;
  Let q_3 be a path segment from x to u';
  $p_j(u) \leftarrow q_1 + q_2 + q_3$;
  return $p_j(u)$;
end
if $\text{dist}(u, r_k) > 2^{j+2} - 2$ **then**
  Let x be a point on the shortest path from u to s and $\text{dist}(u, x) = 2^{k-1} - 1$;
  Let q_1 be a path segment from u to x;
  $u' \leftarrow v \in N(s, 2^k - 1) \cap I_{j+1}$ and $\text{dist}(x, v) \leq 2^{k-1} - 1$;
  Let q_2 be a path segment from x to u';
  $p_j(u) \leftarrow q_1 + q_2$;
  return $p_j(u)$;
end

5 Modified Tree Construction

The pruned paths in the spanning tree T will not have leaders as end nodes. To ensure that end nodes of all paths are leaders, we modify T to $\overline{T}$. The main goal is to merge pruned paths to form longer paths whose end nodes are leaders in some level. We then find "pseudoleaders" $\overline{I}_i$ among the intermediate nodes in the merged paths that serve as end nodes for these pruned paths.

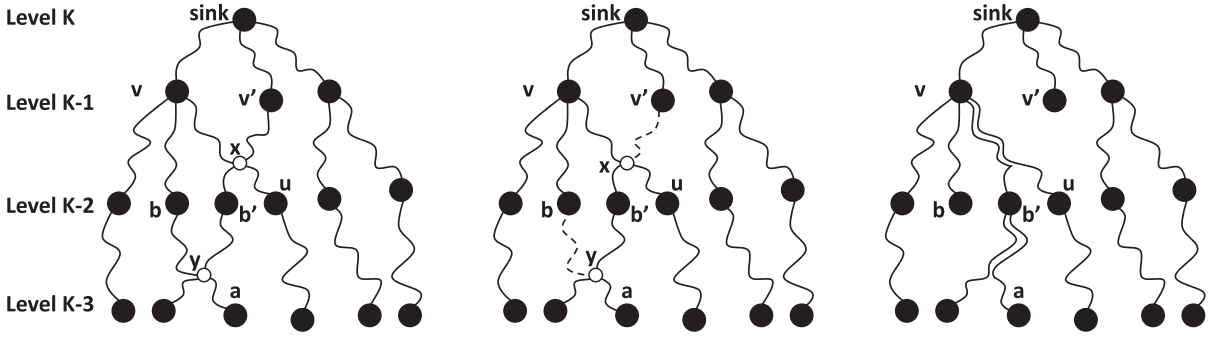
We begin with an overview of the modified tree construction. We construct $\overline{T}$ from T by assigning alternate leaders to those paths whose "upper" sections have been pruned. We first begin by assigning *levels* to all the nodes of regular paths by `AssignLevels` function in AssignLevels and including those paths in $\overline{T}$. Then, we begin a top-down, level-by-level process where we "modify" the pruned paths by extending the pruned paths to their newly assigned alternate leaders. Note that a modified path could be a concatenation of multiple pruned paths. Then, we assign levels to the nodes of the recently modified path as well and include this modified path in $\overline{T}$. The end of this process results in a modified tree $\overline{T}$. A more formal description appears in Algorithm 2 `Modified Tree`.

Algorithm 2: Modified Tree

Input: Spanning Tree T rooted at s.
Output: A modified tree $\overline{T}$.

$\overline{T} \leftarrow \phi$; // $T = P = \{P_{\kappa-1}, P_{\kappa-2}, \ldots, P_1, P_0\}$
// Assign Levels to all nodes in all regular paths in T.
$i \leftarrow \kappa - 1$; // start from second level from top
while $i \geq 0$ **do**
  foreach $p_i(v) \in P_i^{reg}$ **do**
    // v and w are the start and end nodes of path p_i
    $H \leftarrow \{v, w\}$; // v is at same level as that of i.
    `AssignLevels` ($p_i(v)$, H, i);
    $\overline{T} \leftarrow \overline{T} \cup p_i(v)$;
  end
  $i \leftarrow i - 1$;
end
// Pruned paths in $\overline{T}$ - Modify paths and assign levels.
$i \leftarrow \kappa - 2$;
while $i > 0$ **do**
  foreach $p_i(u) \in P_i^{pr}$ **do**
    $\overline{p}_i(u) \leftarrow$ `ModifyPath`$(p_i(u), p_j(v))$; // $p_i(u)$ intersects $p_j(v)$, $j > i$ and v' be the elected pseudo-leader. $p_j(v)$ may be a modified path itself.
    $\overline{T} \leftarrow \overline{T} \cup \overline{p}_i(u)$;
    $H \leftarrow \{u, v'\}$; // u and v' are the start and end nodes of $\overline{p}_i(u)$.
    `AssignLevels` ($\overline{p}_i(u)$, H, i);
  end
  $i \leftarrow i - 1$;
end
return $\overline{T}$;

Define `AssignLevels`$(p_i(v), H, i)$, where H is a pair of end nodes of $p_i(v)$, to assign levels to all the nodes of $p_i(v)$ by identifying maximal independent nodes (excluding the end nodes of $p_i(v)$). This is given in more detail in the function `AssignLevels`. Levels are assigned in the range $(i-1)$ to 0. A modified path is connected to an alternate leader called *pseudoleader* by the function `ModifyPath`$(p_i(u), p_j(v))$ which chooses the nearest level-$(i+1)$ node on $p_j(v)$ from the intersection point. The existence of a pseudoleader in any given path $p_j(v)$, $j > i$, is justified by the Lemma 5.1.



(a) Intersecting paths (b) Pruned paths y, b and x, v'. (c) Modified Paths $p_{a,b'}$ and $p_{b',v}$

Fig. 1. Pruning and tree modification.

Function AssignLevels($p_i(v), H, i$)

Input: Path $p_i(v)$, set of end-nodes H of $p_i(v)$, level i.
Output: Assignment of levels to all nodes in $p_i(v)$.

$L_\lambda \leftarrow \phi$; // Set of 2^λ-independent nodes
for $\lambda \leftarrow (i-1)$ **to** 0 **do**
  // Find 2^λ-independent nodes at levels $\lambda = (i-1), (i-2), \ldots, 1, 0$.
  $L_\lambda \leftarrow MIS(p_i(v), H, 2^\lambda)$;
  Assign level λ to nodes in L_λ.
end

Function ModifyPath($p_i(u), p_j(v)$)

Input: Paths $p_j(v)$ and $p_i(u)$ where $p_i(u)$ intersects $p_j(v)$ and $j > i$
Output: A modified path $\overline{p}_i(u)$.

// Let $p_i(u)$ start from $u \notin p_j(v)$ and intersect at $y \in p_j(v)$ along its path to its leader ℓ_{i+1}.
$v' \leftarrow$ Identify a level-$(i+1)$ node $v' \in p_j$ that is close to y and in the direction of s;
$p_i^a(u) \leftarrow$ subpath from u to y in $p_i(u)$;
$p_i^b(y) \leftarrow$ subpath from y to v' in $p_j(v)$;
$\overline{p}_i(u) \leftarrow p_i^a(u) + p_i^b(y)$; // Concatenate $p_i^a(u)$ and $p_i^b(y)$.
return $\overline{p}_i(u)$;

Lemma 5.1 (Presence of a Pseudoleader). *The* `ModifyPath`$(p_i(u), p_j(v))$ *function guarantees selection of an* $(i+1)$*-level pseudoleader.*

Proof. Suppose path $p_i(u)$ intersects a higher level path $p_j(v)$, $i < j$. Let the start node of p_i be u and let the end node of $p_j(v)$ be w. Note that a path $p_j(v)$ goes from level j to level $j+1$. There could be two cases for the presence of a pseudoleader in $p_j(v)$. If level of w is $i+1$, then w itself acts as a pseudoleader for u. If level of w is greater than $i+1$, then $p_j(v)$ must have some nodes (within its end nodes) that have been assigned to level $i+1$ (by the `AssignLevels` function). Hence, in either case, a pseudoleader is guaranteed to be found in $p_j(v)$ for u. □

Consider that we are at some level i where $0 \le i \le \kappa - 1$ and suppose that there are several pruned paths in P_i. Let $p_i(u) \in P_i$ be one such path and let $y \in p_j(v)$ be the intersection point, where $j > i$. A *pseudoleader*, v', is chosen on $p_j(v)$ using `ModifyPath` $(p_i(u), p_j(v))$ in ModifyPath. This pseudoleader is chosen in such a way that it is closer to both s and y. Such a leader is always guaranteed to exist because the connection from a pruned path occurs to a modified path that has already elected new pseudoleaders toward the direction of s. Note that this may alter I_j to $\overline{I}_j$ by replacing the original leader by the pseudoleader. The path $p_i(u)$ is extended from y to v' and this new extended path, denoted by $\overline{p}_i(u)$, replaces $p_i(u)$ in the modified tree $\overline{T}$. The upper bound on the length of $\overline{p}_i(u)$ is given by Lemma 6.1. Once a new path $\overline{p}_i(u)$ is established, all the nodes in it are assigned levels using (`AssignLevels`$(\overline{p}_i(u), H, i)$), where H is the set of end nodes of $\overline{p}_i(u)$). This procedure of modifying pruned paths, replacing the old pruned paths by new, extended, *modified* paths, and assigning levels to all nodes in those paths is repeated for all levels down to 0. The resulting tree is a modified tree with normal leaders and pseudoleaders for respective types of paths.

Fig. 1 gives an example of intersecting path and its modification to reach a pseudoleader and form a modified path. At level $\kappa - 2$, we see there is a path from u to v. The path from b' to v' intersects the former path at x. This path is pruned from the point of intersection x till v' and a new connection is made from x to v, resulting in a new path from b' to v.

6 Analysis of Modified Tree

We will analyze the performance of the modified tree $\overline{T}$. The analysis is similar to the analysis of the overlay tree in Section 3. We will focus on finding in $\overline{T}$ the respective values of the parameters μ_i, δ_i, ϕ_i, ξ_i, and χ given in Section 3.1. With these values, we can immediately apply the results of Section 3.2 to obtain a competitive ratio of $\overline{T}$.

The modified tree $\overline{T}$ naturally defines a hierarchical partition of G. This tree has κ levels of pseudoleaders $\overline{I}_0$ to $\overline{I}_\kappa = s$. For each node $u \in \overline{I}_i$, let $\overline{Z}_i^u$ denote all the leafs in $\overline{T}$ which appear in the subtree of $\overline{T}$ rooted at u at level i. For our analysis, we will use the following parameters:

$\overline{\mu}_i = 2^{i+3}$//upper bound on $|\overline{p}_i(u)|$
$\overline{\delta}_i = 2^{i+3}$//upper bound on the radius of $\overline{Z}_i^u$
$\overline{\phi}_i = 2^i$//lower bound ondist $(s, \overline{Z}_i^u), u \neq s$
$\overline{\xi}_i = 2\overline{\delta}_i + 2\overline{\phi}_i$//coloring radius
$\overline{\chi} = 2^{17\rho} \log^2 D$//coloring of $\overline{I}_i$ with radius ξ_i.

A path $\overline{p}_i^j(v)$ could be intersected by multiple lower level paths. Even though the leaders at a level i are sufficiently far off, due to intersection by other paths, the leader at level i might be close to many leaders of lower level paths. However, the number of such leaders that are close is *limited*. Lemmas 6.5, 6.6, and 6.7 establish the maximum number of pseudoleaders in a given neighborhood.

Lemma 6.1. $|\overline{p}_i(u)| < \overline{\mu}_i$.

Proof. Consider a path $p_i(u) \in T$ that starts at $u \notin p_j(v)$, $(j > i)$, and intersects another path $p_j(v)$ at $y \in p_j(v)$. Since $p_i(u)$ is a pruned path, its length from u to the intersection point y is at most $2^{i+2} - 3$ (if it was $2^{i+2} - 2$ or more, point y would have been its original leader). `ModifyPath` will attempt to seek an $(i+1)$-level node (pseudoleader) on $p_j(v)$ that is close to y and in the direction of s (Lemma 5.1). Note that y itself cannot be the pseudoleader for u because, if it was, then $p_i(u)$ would not have been a pruned path. The distance from y to a pseudoleader v' on $p_j(v)$ would be at most $2^{i+2} - 2$ because if this distance was more than $2^{i+2} - 2$, we would have found another pseudoleader v'' that is 2^{i+1} distance away from v' and closer to y. This is due to the presence of (2^{i+1})-independent set nodes on this path $p_j(v)$ computed by `AssignLevels`. Note that y cannot be an end node of $p_j(v)$ and v' could be one of the end nodes of $p_i(v)$. Hence, the length of $\overline{p}_i(u)$, denoted by $\overline{\mu}_i$, could be at most $(2^{i+2} - 3) + (2^{i+2} - 2) < 2^{i+3}$. Note that $p_j(v)$ itself could be a stretched pruned path and the upper bound holds irrespective of the length of $p_j(v)$. □

Lemma 6.2. *For any* $v \in \overline{Z}_i^u$, $\text{dist}(v, u) < \overline{\delta}_i$.

Proof. Consider a path $\overline{p}_i(v) \in \overline{Z}_i^u$. In the worst case, this path could be a concatenation of several modified paths, ranging from level 0 to $i-1$. The total length of $\overline{p}_i(v)$ would be equal to the sum of maximum lengths of each of those segments: $\sum_{j=0}^{i-1}(2^{i+2}) < 2^{i+3}$. □

Lemma 6.3. $N(s, 2^i - 1) \subseteq \overline{Z}_i^s$.

Proof. Consider a node $v \in N(s, 2^i - 1)$, $v \neq s$. Suppose that $v \in \overline{I}_j$, where $j < i$. Let $\overline{\ell}_{j+1}$ denote the parent of v. This parent $\overline{\ell}_{j+1}$ could be a pseudoleader on a modified path $\overline{p}_j(v)$.

We observe that all the nodes in $N(s, 2^i - 1)$ use internal special paths to s due to `FindPath` algorithm. This is because a path from a node v to its leader is always toward s. A pseudoleader $\overline{\ell}_{j+1}$ for a modified path can be found within $2(2^{i+2} - 2)$ distance from v such that $\overline{\ell}_{j+1}$ is within $N(s, 2^i - 1)$ and closer to sink s, due to Lemma 6.1. Since the pseudoleader of v is found inside $N(s, 2^i - 1)$, $v \in \overline{Z}_i^s$. By induction on $j = i - 1, \ldots, 0$, we obtain that if $v \in \overline{I}_j$ and $v \in N(s, 2^i - 1)$, then $v \in \overline{Z}_i^s$. Consequently, when we consider $j = 0$, we obtain that $N(s, 2^i - 1) \subseteq \overline{Z}_i^s$. □

From Lemma 6.3, we obtain the following corollary:

Corollary 6.4. *For any* $u \in \overline{I}_i$, $u \neq s$, $\text{dist}(s, \overline{Z}_i^u) \geq \overline{\phi}_i$.

Lemma 6.5 (Max Path Segments). *The total number of path segments* $p(v) \in T$ *at level* i *or higher that cross* $N(x, 2^{i+5})$ *is at most* $2^{10\rho} \cdot (\kappa - i + 1)$.

Proof. We know, by construction, that the length of a path $p_{i+j}(v) \in T$ is at most 2^{i+j} where $0 \leq j \leq (\kappa - i)$ and that there is at most one leader $\ell_{i+j} \in I_i$ within $N(x, \frac{2^{i+j}}{2})$. Since we are looking at the number of path segments $p_{i+j}(v)$ that go through $N(x, 2^r)$, where $r = i + 5$, consider a large neighborhood $N(x, (2^{i+j} + 2^r))$ and determine the number of neighborhoods of radius $\frac{2^{i+j}}{2}$; $N(x, \frac{2^{i+j}}{2})$. If $r < (i + j)$, then $(2^{i+j} + 2^r) < 2 \cdot 2^{i+j}$. From Lemma 2.3, the number of path segments at level i or higher that cross $N(x, 2^r)$ is at most $2^{\rho((i+j+1)-(i+j-1)+3)} = 2^{5\rho}$. If $r \geq (i + j)$, then $(2^{i+j} + 2^r) < 2 \cdot 2^r = 2^{r+1}$. From Lemma 2.3, the number of path segments at level i or higher that cross $N(x, 2^r)$ is at most $2^{\rho((r+1)-(i+j-1)+3)} = 2^{\rho(r-i+5)}$. Since $r = i + 5$, $\max(2^{4\rho}, 2^{\rho(r-i+5)}) = \max(2^{4\rho}, 2^{10\rho}) = 2^{10\rho}$. For all paths that span the levels from i to κ, the total number of path segments that cross $N(x, 2^{i+j-1})$ is equal to $2^{10\rho} \cdot (\kappa - i + 1)$. □

Lemma 6.6 (Max Modified Paths in a Path Segment). *Consider a path segment* $p(v) \in T$ *that crosses* $N(x, 2^{i+5})$. *The total number of modified paths* $\overline{p}(v) \in \overline{T}$ *at level* i *or higher that use nodes in* $p(v) \cap N(x, 2^{i+5})$ *is at most* $2^{7\rho} \cdot (\kappa - i + 1)$.

Proof. Let $Q = p(v) \cap N(x, 2^r)$, where $r = i + 5$. From Lemma 6.1, we know that the maximum length of any modified path $\overline{p}_{i+j}(v)$ would be 2^{i+j+3}. To find the total number of modified paths $\overline{p}_{i+j}(v)$ that passes through Q, we consider a larger neighborhood $N(x, 2^{i+j+3} + 2^r)$ and find the number of $N(y, 2^{\frac{i+j+3}{2}})$ that would cover the larger neighborhood. Note that each $\overline{p}_{i+j}(v)$ has start node in I_{i+j}. If $r < (i + j + 3)$, then $(2^{i+j+3} + 2^r) < 2 \cdot 2^{i+j+3} = 2^{i+j+4}$. By Lemma 2.3, the number of path segments at level i or higher that cross $N(x, 2^r)$ is at most $2^{\rho((i+j+4)-(i+j+2)+3)} = 2^{5\rho}$. If $r \geq (i + j + 3)$, then $(2^{i+j+3} + 2^r) < 2 \cdot 2^r = 2^{r+1}$. From Lemma 2.3, the number of path segments at level i or higher that cross $N(x, 2^r)$ is at most $2^{\rho((r+1)-(i+j+2)+3)} = 2^{\rho(r-i+2)}$. We consider $\max(2^{4\rho}, 2^{\rho(r-i+2)}) = \max(2^{4\rho}, 2^{7\rho}) = 2^{7\rho}$ for our analysis. Since $j \in [0, (\kappa - i)]$, the total number of paths that would cross $N(x, 2^{i+j+2})$ is equal to $2^{7\rho} \cdot (\kappa - i + 1)$. □

Lemma 6.7. *The total number of pseudoleaders at level* i, *which are inside* $N(x, 2^{i+5})$ *is at most* $2^{17\rho} \cdot (\kappa - i + 1)^2$.

Proof. From Lemma 6.5, there are $2^{10\rho} \cdot (\kappa - i + 1)$ path segments $p_{i+j}(v) \in T$, $j \geq 0$, crossing $N(x, 2^r)$, where $r = i + 5$. From Lemma 6.6, each such path segment can have multiple modified path segments at level i or higher passing through it $(\leq 2^{7\rho} \cdot (\kappa - i + 1))$, the total number of modified path segments that cross $N(x, 2^r)$ would be at most $2^{17\rho} \cdot (\kappa - i + 1)^2$. This gives also an upper bound to the number of pseudoleaders at level i or higher. □

Let $\overline{X}_i = (\overline{I}_i, \overline{E}_{\overline{X}_i})$, be a graph such that for any two $u, v \in \overline{I}_i$, $(u, v) \in \overline{E}_{\overline{X}_i}$ if and only if $\text{dist}(u, v) \leq \overline{\xi}_i$.

Lemma 6.8. *Graph* $\overline{X}_i$ *admits a vertex coloring with at most* $\overline{\chi} = 2^{17\rho} \cdot (\kappa - i + 1)^2$ *colors.*

Proof. Let $v \in \overline{I}_i$. The nodes adjacent to v in $\overline{I}_i$ are the set $Y = N(v, \overline{\xi}_i) \cap \overline{I}_i$. Since $\overline{I}_i$ is a 2^i-independent set, and $\overline{\xi}_i = 2\overline{\delta}_i + 2\overline{\phi}_i \leq 2 \cdot 2^{i+3} + 2 \cdot 2^i = 2^{i+4} + 2^{i+2} \leq 2^{i+5}$. From Lemma 6.7, we obtain $|Y| \leq 2^{17\rho} \cdot (\kappa - i + 1)^2$.

Consequently, graph $\overline{X}_i$ has degree at most $[2^{17\rho} \cdot (\kappa - i + 1)^2] - 1$, and by a greedy algorithm, it can be colored with at most $\overline{\chi} = 2^{17\rho} \cdot (\kappa - i + 1)^2 \leq 2^{17\rho} \log^2 D$ colors. □

Now, the remaining part of the analysis identical to that in Overlay Tree (3.2), where instead of the parameters μ_i, δ_i, ϕ_i, ξ_i, and χ, we use $\overline{\mu}_i$, $\overline{\delta}_i$, $\overline{\phi}_i$, $\overline{\xi}_i$, and $\overline{\chi}$. We derive the competitive ratio of the modified tree as below.

Theorem 6.9 (Oblivious Competitive Ratio of Modified Tree). *The oblivious competitive ratio of the modified tree $\overline{T}$ is $C.R.(\overline{T}) \leq \overline{\chi} \cdot (1 + \log D) \cdot \max_i\{\overline{\mu}_i/\overline{\xi}_i\}$.*

From Theorem 6.9, we immediately obtain the following corollary when we replace the values of the parameters:

Corollary 6.10. *The oblivious competitive ratio of the modified tree $\overline{T}$ is $C.R.(\overline{T}) = O(2^{17\rho} \log^3 D)$.*

7 Lower Bound

We now present an overview of the technique used for computing the lower bound. The lower bound given by Imase and Waxman [35] doesn't work in our case. Their technique works for nonlow doubling-dimension planar graphs. Therefore, we give a new lower bound for the spanning tree construction for low doubling-dimension graphs.

For our study, we consider a special class of planar graphs commonly called grid graphs or lattice graphs. A grid graph G is a euclidean $n \times n$ graph for some positive integer n where the nodes are situated at each of the n^2 grid points. Any two vertices are connected by an edge if and only if their euclidean distance is one unit and a node has at most four neighbors. For example, see Fig. 4.

Let there be an arbitrary tree T that spans the grid vertices. Assume that the root r of the tree T is one of the corners of the grid. We compare the cost of a path from a set of grid vertices to the root r to the cost of the tree path of those vertices.

We show that there exists a vertical (or horizontal) line in the grid that contains pairs of nodes whose distances in T sum to $\theta(n \log n)$, whereas, the shortest path along the grid vertices would be $\Omega(n)$.

Define a U^x-Path as a path between any two adjacent nodes in an $n \times n$ grid. Define a *reference* node to a U^x-Path as one of its end nodes. All the distances in any U^x-Path will be measured from its respective reference node.

A U^x-Path could extend at least $x/2 - 1$ distance from its reference node. A U^x-Path has the following properties:

1. The total length of the path is at least $x - 1$.
2. The U^x-Path has a node that is $x/2$ away from its reference node. In other words, the path will intersect a node in its $x/2$-radius from one of its end nodes. Informally, we call it "width."

Consider any two adjacent nodes u and v (with respect to G) that form a U^x-Path. Let u be its reference node. Let there be a node $p \in U^x$-Path such that $\mathrm{dist}(u, p) \geq x/2 - 1$. If the vertical distance of node p from u is greater than or equal to the horizontal distance of it from u, then we say that the U^x-Path is *vertical*. Otherwise, it is *horizontal*. We shall refer to such paths as V-Paths and H-Paths, respectively.

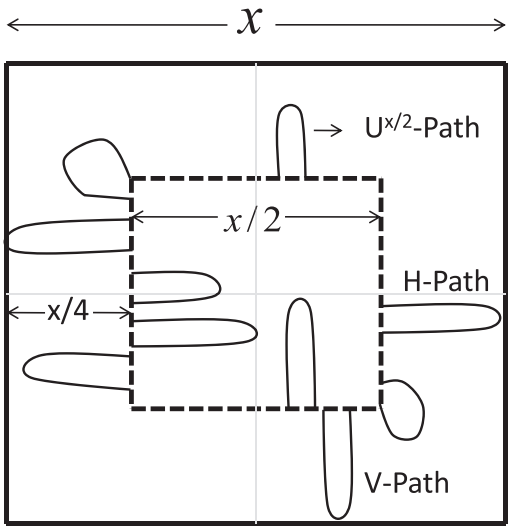


Fig. 2. $U^{x/2}$-Paths originating from an $x/2 \times x/2$ subgrid centered in an $x \times x$ subgrid of G.

Lemma 7.1. *In an $x \times x$ subgrid of G, there is at least one U^x-Path in T with its end nodes in the perimeter of the subgrid.*

Proof. For contradiction, let us suppose that all the pairs of nodes in the subgrid have a U^x-Path of length at most $x - 1$. This formation will lead to two observations. The center (a square of unit length) of the subgrid will not be reached by any of the paths. This will result in a cycle. This leads to a contradiction. Hence, there must be at least one U^x-Path that is longer than $x - 1$. □

Define an *x-class* to be a decomposition of G into $x \times x$ subgrids where two adjacent subgrids share a common edge. The number of such subgrids would be n^2/x^2. There will be $\log n$ classes of such subgrids based on the value of x, $(= n, n/2, n/4, \ldots, 1)$.

Let $U^{x/2}$-*Core* be an $x/2 \times x/2$ subgrid centered within an $x \times x$ subgrid of G as given in Fig. 2. We observe that the $U^{x/2}$-Paths from adjacent node pairs along the perimeter of the $U^{x/2}$-Core would extend either internally or externally to a maximum distance (width) of $x/4$. The minimum distance they would extend will be $x/8$.

Each $x \times x$ subgrid will have either an H-Path or a V-Path in it, as shown in Fig. 3. This identifies the "type" of that subgrid (namely, H-Type or V-Type). Consider a certain x-class decomposition of G. There will be a mix of H-Type and V-Type subgrids totaling n^2/x^2 subgrids that constitutes this decomposition. If the number of H-Type subgrids is larger ($>n^2/2x^2$) than the number of V-Type subgrids, then we say that the x-class decomposition is of type H. Otherwise, it is of type V. Therefore, out of the $\log n$ classes of decomposition of G, some of them will be "H-Type" and some will be "V-Type." Without loss of generality, assume that the majority is of H-Type.

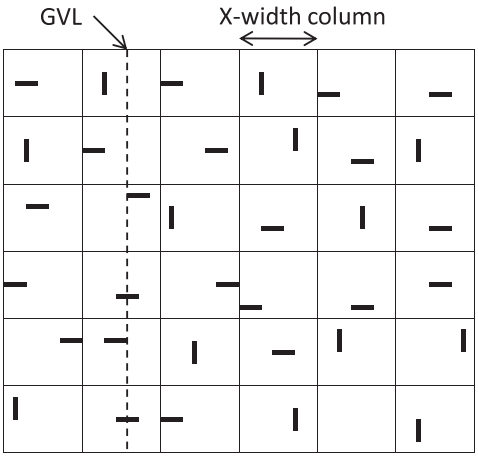


Fig. 3. An example of a GVL in a grid where each $x \times x$ subgrid has either an H-Path or a V-Path.

Consider an H-Type x-class of G. Define *x-width column* as one of the columns in G where G is divided into several columns of width x. Consider a vertical line $\ell \in G$ of length n. This line will span n/x subgrids. Those n/x subgrids will possibly be a mixture of H-Type and V-Type subgrids. Observe that ℓ will intersect zero or more ($\leq n/x$) H-Paths present in those subgrids. We say that ℓ is a "good vertical line" for the x-class (GVL_x) if it intersects a constant ($n/2x$) number of H-Paths at a position less than or equal to $3/4$th of the "width" of those H-Paths measured from their respective end nodes. The constraint associated with the intersection point on the H-Path is to ensure that the length of the U-Path from the intersection points still remains significantly long.

Lemma 7.2 gives the total number of GVLs in G. We choose the one that intersects the largest number of H-Paths (c_1 is the largest among all) and refer to that line as GVL^*_x. For each of the $\log n$ classes of subgrids, there will be a respective GVL^*_x (or a GHL^*_x if the class is a V-Type).

Lemma 7.2. *The total number of GVLs in an x-class of G is $3n/128$.*

Proof. Consider an H-Type x-class decomposition of G. The "width" of any H-Path in a subgrid is at least $x/8$. Hence, the number of vertical lines that can intersect such an H-Path is $x/8$. But a GVL would intersect only within $3/4$th of the width of any H-Path. On an average, in an x-width column, there will be $\frac{n}{2x}$ H-paths. And, by pigeonhole principle, on an average, at least half of the columns in G will have average number of H-Paths. Therefore, the total number of GVLs in G for x-class will be $\frac{n}{2x} \cdot \frac{1}{2} \cdot \frac{x}{8} \cdot \frac{3}{4} = \frac{3n}{128}$. □

A GVL for a class $n/2^k$ will have 2^k such pairs of vertices. Each pair of these vertices forms an H-Path of length $\theta(n/2^k)$. Now, we shift our focus to finding one GVL for all the $\log n$ classes. To find such a line, we first find GVLs for all the individual classes $n, n/2, n/4, \ldots, 1$. We form an overlay of all such GVLs and find the one that overlaps all the classes. Such a GVL would be the line that would have pairs of nodes that has U-paths of all the different lengths, and each path would contribute a length of n.

Lemma 7.3. *There is a GVL (denoted by GVL^*) that is common to a constant fraction of the total number of horizontal classes.*

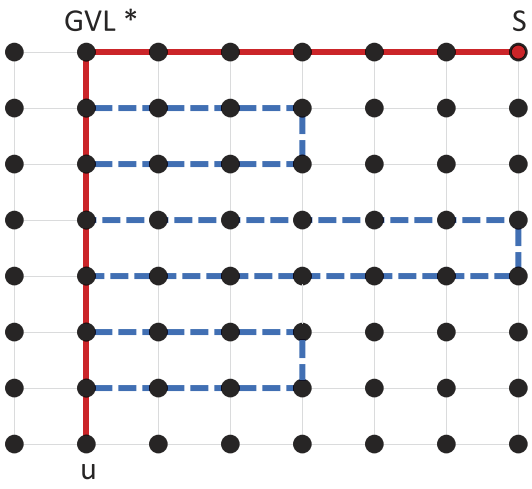


Fig. 4. Paths in an $n \times n$ grid.

Proof. The number of classes that are of type H is at least $\frac{\log n}{2}$. The number of GVLs in all the $\frac{\log n}{2}$ classes will be $\frac{3n}{128}\frac{\log n}{2} = \frac{3n \log n}{256}$. Therefore, the number of GVL^*s that overlaps a constant number of these classes would be $\frac{\frac{3n \log n}{256}}{n} = \frac{3 \log n}{256}$. This proves the existence of at least one GVL^*. □

Now, we are ready to present the central theorem of this section.

Theorem 7.4. *There exists a set S of nodes in G such that 1) S constitutes $\theta(n)$ nodes, 2) optimal tree T^* for S has cost $O(n)$, and 3) the induced subtree $T(S)$ has $\Omega(n \log n)$ cost.*

Proof. From Lemma 7.3, we observe that GVL^* crosses H-Paths that belong to different (a constant number of) x-classes. For an arbitrary class x_i, it will have $\theta(n/x_i)$ paths of length $\theta(n/x_i)$. An example of this scenario can be seen in Fig. 4. Since there will be a constant number of classes ($\geq \log n/2$) that belong to H-Type, the total cost of the induced paths will be $x_i(n/x_i) + x_j(n/x_j) + \cdots = \theta(n \log n)$. Hence, the least cost along the tree path would be $\Omega(n \log n)$.

Note that there will be overlaps in the H-Paths from different classes. An H-Path from an x_i-class can contain an H-Path from an x_j-class where $x_i > x_j$. The overlaps can go further such that an H-Path from an x_i-class can contain one or more H-Paths from classes that are smaller than x_i. In effect, the number of overlaps will halve the number of H-paths of smaller classes and hence, the effective path length is half of its contribution. □

From Lemma 7.4, we obtain the following corollary:

Corollary 7.5. *In any $n \times n$ grid, any spanning tree T will have $C.R.(T) = \Omega(\log n)$.*

8 Simulation Results

We simulated our algorithm, denoted by Oblivious Spanning Tree and compared its performance (fusion cost) with GRID_GIST [30] and other common trees such as MST and SPT. We used an $n \times n$ grid topology for our simulation using NetworkX [17]. $n \times n$ grids are a special case of doubling-dimension graphs and they fall under a variation of the Steiner tree problem called "Rectilinear Steiner

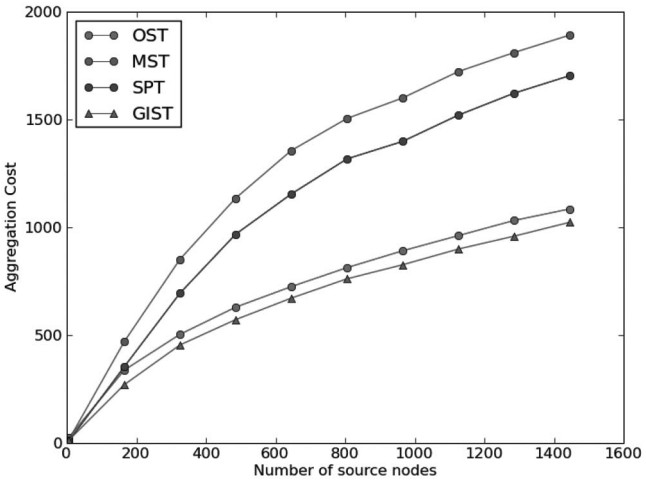


Fig. 5. Fusion cost for varying set of source nodes in a 1,600-node grid.

Problem" (RSP) where the tree structure has only vertical and horizontal lines that interconnect all points and is proved to be NP-Complete [38]. Since calculating a minimum weight tree structure in an n x n grid topology (a doubling-dimension graph) is essentially an RSP, the problem we are addressing is NP-Hard.

We build a single spanning tree in a grid with $n^2 = 1{,}600$ nodes. We simulate it for random sets of data sources, up to 1,445, that are randomly placed. The random data sets (of known size) are generated using Python's random sampling method without replacement from the given population. Note that GRID_GIST is a special algorithm designed for grids and ours is a generalized algorithm. Hence, GRID_GIST performs slightly better than OST (in Fig. 5).

9 Conclusions and Future Work

We provide a spanning tree algorithm for a variant of the single-sink buy-at-bulk network design problem in low constant doubling-dimension graphs. Contrary to many related works where the source-destination pairs were already given, or when the source set was given, we assumed the obliviousness of the set of source nodes. Moreover, we considered an unknown fusion-cost function at every edge of the tree. We presented nontrivial upper and lower bounds for the cost of the set of paths in the spanning tree. We have demonstrated that a simple, deterministic, polynomial-time algorithm based on appropriately defined distance-based independent sets can provide single spanning tree for data fusion. We have shown that this algorithm guarantees $(\log^3 D)$-approximation. As part of our future work, we are looking into the same problem on planar graphs, arbitrary graphs, and also the general buy-at-bulk network design problem.

Acknowledgments

Supported by US National Science Foundation (NSF) grants 0963793 and 0846081. An earlier version of our work with preliminary results in the context of data aggregation in graphs of low doubling dimension appeared as a brief announcement in [1].

References

[1] S. Srinivasagopalan, C. Busch, and S.S. Iyengar, "Brief Announcement: Universal Data Aggregation Trees for Sensor Networks in Low Doubling Metrics," *Proc. Fifth Int'l Workshop Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS '09)*, pp. 151-152, July 2009.

[2] C. Chekuri, M.T. Hajiaghayi, G. Kortsarz, and M.R. Salavatipour, "Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design," *Proc. Ann. IEEE Symp. Foundations of Computer Science (FOCS '06)*, pp. 677-686, 2006.

[3] A. Goel and D. Estrin, "Simultaneous Optimization for Concave Costs: Single Sink Aggregation or Single Source Buy-at-Bulk," *Proc. Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '03)*, pp. 499-505, 2003.

[4] I. Abraham, C. Gavoille, A.V. Goldberg, and D. Malkhi, "Routing in Networks with Low Doubling Dimension," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '06)*, p. 75, 2006.

[5] G. Konjevod, A.W. Richa, D. Xia, and H. Yu, "Compact Routing with Slack in Low Doubling Dimension," *Proc. 26th Ann. ACM Symp. Principles of Distributed Computing (PODC '07)*, pp. 71-80, http://doi.acm.org/10.1145/1281100.1281113, 2007.

[6] G. Konjevod, A.W. Richa, and D. Xia, "Dynamic Routing and Location Services in Metrics of Low Doubling Dimension," *Proc. Int'l Symp. Distributed Computing (DISC '08)*, pp. 379-393, 2008.

[7] H.T.-H. Chan, A. Gupta, B.M. Maggs, and S. Zhou, "On Hierarchical Routing in Doubling Metrics," *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '05)*, pp. 762-771, http://portal.acm.org/citation.cfm?id=1070432.1070540, 2005.

[8] T.-H.H. Chan and A. Gupta, "Small Hop-Diameter Sparse Spanners for Doubling Metrics," *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithm (SODA '06)*, pp. 70-78, http://doi.acm.org/10.1145/1109557.1109566, 2006.

[9] J. Kleinberg, A. Slivkins, and T. Wexler, "Triangulation and Embedding Using Small Sets of Beacons," *J. ACM*, vol. 56, no. 6, pp. 1-37, 2009.

[10] P. Fraigniaud, "The Inframetric Model for the Internet," technical report, 2007.

[11] P. Fraigniaud, E. Lebhar, and Z. Lotker, "A Doubling Dimension Threshold $\theta(\log \log n)$ for Augmented Graph Navigability," *Proc. Ann. European Symp. (ESA)*, pp. 376-386, 2006.

[12] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "On the Locality of Bounded Growth," *Proc. Ann. ACM Symp. Principles of Distributed Computing (PODC '05)*, pp. 60-68, 2005.

[13] J. Gao, L.J. Guibas, N. Milosavljevic, and D. Zhou, "Distributed Resource Management and Matching in Sensor Networks," *Proc. Int'l Conf. Information Processing in Sensor Networks (IPSN '09)*, pp. 97-108, Apr. 2009.

[14] S. Funke, L. Guibas, A. Nguyen, and Y. Wang, "Distance-Sensitive Information Brokerage in Sensor Networks," *Proc. Int'l Conf. Distributed Computing in Sensor Networks (DCOSS '06)*, pp. 234-251, 2006.

[15] S.V. Pemmaraju and I.A. Pirwani, "Energy Conservation via Domatic Partitions," *Proc. Int'l Symp. Mobile Ad Hoc Networking and Computing (MobiHoc '06)*, pp. 143-154, 2006.

[16] F.S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian, "Approximating the Single-Sink Link-Installation Problem in Network Design," *SIAM J. Optimization*, vol. 11, no. 3, pp. 595-610, 2000.

[17] A.A. Hagberg, D.A. Schult, and P.J. Swart, "Exploring Network Structure, Dynamics, and Function Using NetworkX," *Proc. Seventh Python in Science Conf. (SciPy '08)*, pp. 11-15, Aug. 2008.

[18] Y. Mansour and D. Peleg, "An Approximation Algorithm for Minimum-Cost Network Design," technical report, Inst. of Science, Rehovot 1998.

[19] Y. Bartal, "Competitive Analysis of Distributed Online Problems - Distributed Paging," PhD dissertation, 1994.

[20] B. Awerbuch and Y. Azar, "Buy-at-Bulk Network Design," *Proc. Ann. Symp. Foundations of Computer Science (FOCS '97)*, pp. 542-547, 1997.

[21] Y. Bartal, "On Approximating Arbitrary Metrices by Tree Metrics," *Proc. Ann. ACM Symp. Theory of Computing (STOC '98)*, pp. 161-168, 1998.

[22] S. Guha, A. Meyerson, and K. Munagala, "A Constant Factor Approximation for the Single Sink Edge Installation Problems," *Proc. ACM Symp. Theory of Computing (STOC '01)*, pp. 383-388, 2001.

[23] K. Talwar, "The Single-Sink Buy-at-Bulk lp Has Constant Integrality Gap," *Proc. Ninth Int'l IPCO Conf. Integer Programming and Combinatorial Optimization*, pp. 475-486, 2002.
[24] A. Kumar, A. Gupta, and T. Roughgarden, "A Constant-Factor Approximation Algorithm for the Multicommodity Rent-or-Buy Problem," *Proc. 43rd Symp. Foundations of Computer Science (FOCS '02)*, p. 333, 2002.
[25] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden, "Approximation via Cost Sharing: Simpler and Better Approximation Algorithms for Network Design," *J. ACM*, vol. 54, no. 3, p. 11, 2007.
[26] R. Jothi and B. Raghavachari, "Improved Approximation Algorithms for the Single-Sink Buy-at-Bulk Network Design Problems," *J. Discrete Algorithms*, vol. 7, no. 2, pp. 249-255, 2009.
[27] A. Frangioni and B. Gendron, "0-1 Reformulations of the Multicommodity Capacitated Network Design Problem," *Discrete Applications Math.*, vol. 157, no. 6, pp. 1229-1241, 2009.
[28] B. Gendron, T.G. Crainic, and A. Frangioni, "Multicommodity Capacitated Network Design," *Telecomm. Network Planning*.
[29] T. Öncan, "Design of Capacitated Minimum Spanning Tree with Uncertain Cost and Demand Parameters," *Information Sciences: An Int'l J.*, vol. 177, no. 20, pp. 4354-4367, 2007.
[30] L. Jia, G. Noubir, R. Rajaraman, and R. Sundaram, "Gist: Group-Independent Spanning Tree for Data Aggregation in Dense Sensor Networks," *Proc. IEEE Int'l Conf. Distributed Computing in Sensor Systems (DCOSS)*, pp. 282-304, 2006.
[31] L. Jia, G. Lin, G. Noubir, R. Rajaraman, and R. Sundaram, "Universal Approximations for Tsp, Steiner Tree, and Set Cover," *Proc. 37th Ann. ACM Symp. Theory of Computing (STOC '05)*, pp. 386-395, 2005.
[32] A. Goel and I. Post, "An Oblivious o(1)-Approximation for Single Source Buy-at-Bulk," *Proc. Ann. IEEE Symp. Foundations of Computer Science*, pp. 442-450, 2009.
[33] A. Gupta, M.T. Hajiaghayi, and H. Räcke, "Oblivious Network Design," *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithm*, pp. 970-979, 2006.
[34] J. Chuzhoy, A. Gupta, J.S. Naor, and A. Sinha, "On the Approximability of Some Network Design Problems," *ACM Trans. Algorithms*, vol. 4, no. 2, pp. 1-17, 2008.
[35] M. Imase and B.M. Waxman, "Dynamic Steiner Tree Problem," *SIAM J. Discrete Math.*, vol. 4, no. 3, pp. 369-384, http://link.aip.org/link/?SJD/4/369/1, 1991.
[36] T. Nieberg, "Independent and Dominating Sets in Wireless Communication Graphs," PhD dissertation, Univ. of Twente, Apr. 2006.
[37] A. Gupta, R. Krauthgamer, and J.R. Lee, "Bounded Geometries, Fractals, and Low-Distortion Embeddings," *Proc. Ann. IEEE Symp. Foundations of Computer Science (FOCS '03)*, p. 534, 2003.
[38] M.R. Garey and D.S. Johnson, "The Rectilinear Steiner Tree Problem is Np-Complete," *SIAM J. Applied Math.*, vol. 32, no. 4, pp. 826-834, http://www.jstor.org/stable/2100192, 1977.

Srivathsan Srinivasagopalan received the BE degree in computer science from SRM Engineering College, Chennai, and the MS degree in computer science from the University of Texas, Dallas. He is currently working toward the PhD degree in the Computer Science Department at LSU. His current research interests are in network design algorithms and approximation schemes. His other interests are in large-scale distributed networks, graph theory, wireless sensor networks, and game theory. He has more than seven years of work experience in various telecom industries.

Costas Busch received the BSc degree in 1992 and the MSc degree in 1995 in computer science from the University of Crete, Greece. He received the PhD degree in computer science from Brown University in 2000. He is currently an assistant professor in the Computer Science Department at LSU. His research interests are in the following areas: theory of distributed computing, distributed algorithms and data structures, design and analysis of communication protocols for wireless, sensor, and optical networks, data directories for wireless sensor networks, data streaming algorithms, and algorithmic game theory. He has several publications in theoretical computer science conferences and journals including Symposium on Theory of Computing (STOC), *Journal of the ACM*, and *SIAM journal on Computing*. He has also served in the program committees of MOBIHOC, IPDPS, ICDCS, and DCOSS. His research is funded by the US National Science Foundation (NSF).

S.S. Iyengar is currently a professor and the chair of the Computer Science Department at Louisiana State University. His publications include six textbooks, five edited books, and more than 400 research papers. His research interests include high-performance algorithms, data structures, data fusion, intelligent systems, and distributed sensor networks. He is serving as an editor of several IEEE Journals and is the founding editor-in-chief of the *International Journal of Distributed Sensor Networks*. His research has been funded by the US National Science Foundation (NSF), US Defense Advanced Research Projects Agency (DARPA), US Department of Energy (DoE), NASA, ONR, and other agencies. He is a fellow of the ACM, a fellow of the IEEE, a fellow of the AAAS, and a fellow of the SDPS.