2011

Oblivious buy-at-bulk network design algorithms

Srivathsan Srinivasagopalan
Louisiana State University and Agricultural and Mechanical College

OBLIVIOUS BUY-AT-BULK NETWORK DESIGN ALGORITHMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Srivathsan Srinivasagopalan
M.S., University of Texas, Dallas, 1999
B.S., Madras University, 1997

May, 2011

Acknowledgments

In many ways, the best part of my years at LSU has been the amazing people that I have had the opportunity to meet, to collaborate with, and to get to know personally. Without their ideas, their enthusiasm, and their support, this thesis would never have happened.

Working towards a Ph.D. is a long meandering road where one has to have several people's help and advice. First and foremost, I would like to thank Prof. Costas Busch and Prof. S.S. Iyengar. Even more than their indispensable advice and technical skill, it has been their enthusiasm and positive attitude that has made this thesis possible. From Dr. Busch, I learn the art and philosophy of why rigorous proofs are important and how to go about proving things with all the gory details. I will especially treasure the many hours spent with him in arguments over very abstract problems and in searching for that one elusive counter-example that would destroy our algorithm. Dr. Iyengar's infectious enthusiasm for research and his irrational belief in my abilities has often made his office my first recourse when in doubt. I would also like to thank the other members of my thesis committee, Prof. Seung-Jong Park, and Prof. Supratik Mukhopdhyay. They have provided valuable feedback and interesting perspectives on the ideas contained in this thesis.

Finally, let me conclude by thanking my parents Rajam Srinivasagopalan and Nalini Srinivasagopalan, and my brothers Venkat Sreenivas and Pat Srinivas for their help, encouragement and support. I would like to thank my wife Harini for everything; without her, none of this would have been possible. For the past 4 years, she has made my academic life a lot more easier with her unconditional support.

Table of Contents

Abstract

Large-scale networks such as the Internet has emerged as arguably the most complex distributed communication network system. The mere size of such networks and all the various applications that run on it brings a large variety of challenging problems. Similar problems lie in *any* network - transportation, logistics, oil/gas pipeline etc where efficient paths are needed to route the flow of demands. This dissertation studies the computation of efficient paths from the demand sources to their respective destination(s).

We consider the buy-at-bulk network design problem in which we wish to compute efficient paths for carrying demands from a set of source nodes to a set of destination nodes. In designing networks, it is important to realize economies of scale. This is can be achieved by aggregating the flow of demands. We want the routing to be *oblivious*: no matter how many source nodes are there and no matter where they are in the network, the demands from the sources has to be routed in a near-optimal fashion. Moreover, we want the aggregation function f to be unknown, assuming that it is a concave function of the total flow on the edge. The total cost of a solution is determined by the amount of demand routed through each edge. We address questions such as how we can (obliviously) route flows and get competitive algorithms for this problem. We study the approximability of the resulting buy-at-bulk network design problem.

Our aim is to find minimum-cost paths for all the demands to the sink(s) under two assumptions: (1) The demand set is unknown, that is, the number of source nodes that has demand to send is unknown. (2) The aggregation cost function at intermediate edges is also unknown. We consider different types of graphs (doubling-dimension, planar and minor-free) and provide approximate solutions for each of them. For the case of doubling graphs and minor-free graphs, we construct a single spanning tree for the single-source buy-at-bulk network design problem. For the case of planar graphs, we have built a set of paths with an asymptotically tight competitive ratio.

Chapter 1

Introduction

1.1 Network Design

Network Design is an active research area in the intersection of Combinatorial Optimization and Theoretical Computer Science that focuses on problems arising in the realm of modern communication networks.

A typical instance of a network design problem has a directed or undirected graph $G = (V, E)$ that has non-negative edge costs c_e for all $e \in E$. The objective is to compute a minimum-cost subgraph H of G that satisfies a certain given criteria. For example, the objective would be to find a set of minimum-cost paths to a single sink (this is a minimum-cost spanning tree problem) or it could be about finding minimum-cost set of arcs in a directed graph such that every vertex can reach every other vertex (this is the *minimum-cost strongly connected subgraph* problem). There are a large number of practical applications for this abstract model of network design. Optimal communication networks, publish/subscribe systems, VLSI chip design etc are a few examples. More are given in section 1.5.

As mentioned by Anupam Gupta *et al.* [GK10], many practically relevant instances of network design problems are NP-hard and thus are likely intractable. The research work presented in this dissertation focuses on approximation algorithms as one possible way of circumventing this impasse. Approximation algorithms have been used widely and for a long time. They are known to be very efficient (i.e., they run in polynomial time) and provide solutions to instances of many different optimization problems whose objective values are close to those of their respective optimum solutions. More specifically, the problems discussed in this work are minimization problems. In this context, we say that an algorithm is an α-approximation for a

given problem if the ratio of the cost of an approximate solution computed by the algorithm to that of an optimum solution is at most α over all instances.

A typical client-server model has many clients and multiple servers where a subset of the client set wishes to route a certain amount of data to a subset of the servers at any given time. The set of clients and the servers 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 as a typical network design problem. The same problem can be easily applied to many similar practical scenarios such as oil/gas pipelines and the Internet.

There has been a lot of research on approximation algorithms in the last 30 years, particularly in the area of network design algorithms. During this period, many different approaches have been explored and exploited to design algorithms and for their analysis.

The minimum spanning tree problem has been studied for at least a century, and it is clearly one of the most prominent network design problems. This earliest known algorithm for this problem was developed by Boruvka [Bor26], and since then, a number of techniques have been developed and used to design increasingly sophisticated algorithms.

1.1.1 Steiner Tree Problems

As mentioned by Stefan Voßin [Vo6], one of the oldest mathematical problems related to network design may be formulated as follows: *Given three points A,B and C in the plane, find a point P such that the sum of its distances to the three given points is minimal.*

Connecting a given set of points at minimum cost may be rated as one of the most important problems in telecommunications network design. For that matter, it can be regarded as one of the core problems in networks of *any* kind - transportation/logistics, power circuitry in VLSI

chips etc. There are number of variations to this problem and most of them have immediate practical applications. One of them may be formulated in metric spaces as well as in graphs: Given a weighted graph, the Steiner tree problem in graphs requires to determine a minimum cost subgraph spanning a set of specified vertices. This subgraph could use vertices other than the specified vertices for interconnection. This problem is viewed as combinatorial optimization problem in telecommunications.

This problem, though it sounds simple, is at the core of many network design problems. Many researchers and mathematicians have contributed to solving this problem including Fermat (in 1640) and Steiner (1835). There are several variants of the Steiner tree problem:

- The Euclidean Steiner problem (metric space version)
- The rectilinear Steiner problem
- The Steiner problem in graphs

1.2 Oblivious Buy-at-Bulk Network Design

The "Buy-at-Bulk" network design considers the economies of scale into account. As observed by Chekuri *et al.* in [CHKS06], in a telecommunication network, bandwidth on a link can be purchased in some discrete units $u_1 < u_2 < \cdots < u_j$ with costs $c_1 < c_2 < \cdots < c_j$ respectively. The economies of scale exhibits the property where the cost per bandwidth decreases as the number of units purchased increases: $c_1/u_1 > c_2/u_2 > \ldots c_j/u_j$. 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" network design (SSBB). This SSBB problem has a single "destination" node where all the demands from other nodes has to be routed to. The generalized form of the buy-at-bulk problem is where there are multiple demands from sources

to destinations, and it is commonly referred as *Multi-Sink Buy-at-Bulk* (MSBB). 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 its destination. These problems are often called "discrete cost network optimization" in operations research.

As mentioned by Goel and Estrin [GE03], 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, non-decreasing, $f(0) = 0$ 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 and MSBB 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. In the case of SSBB, 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* buy-at-bulk network design problem (SSBB and MSBB) with the following constraints: an unknown set of demands and an unknown concave fusion cost function f. In other words, we describe a novel approach for developing an oblivious spanning tree (or set of paths) in the sense that it is independent of the number and location of demand sources and cost function at the edges. An abstraction of this problem can be found in many applications, one of which is data fusion in wireless sensor networks where data from sensors is aggregated over time in multiple sinks. Other application include Transportation & Logistics (railroad, water, oil, gas pipeline construction) etc. Many of these problems are formulated as networks on a plane that can be mapped to planar graphs.

1.2.1 General Problem Statement

In the following paragraphs, a general problem statement is given. More specific problem statements for appropriate graph types and scenarios are provided in subsequent chapters.

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 non-negative unit demand. In the SSBB case, a demand from v_i induces a unit of flow to sink s and this flow is unsplittable. For the MSBB case, each unit demand $d_i = (s_i, t_i)$ induces an unsplittable unit of flow from source node $s_i \in V$ to destination node $t_i \in V$. Let $A = \{d_1, d_2, \ldots, d_r\}$ be a set of demands that are routed through paths in G. It is possible that some paths may overlap.

The demands from various demand nodes have to be sent to their respective destination node(s) possibly routed through multiple edges in the graph G. This flow of demands 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 their destination(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 demand aggregates. This f is the same for all the edges in G. Let $p(v)$ be the path taken by a flow from v to its destination 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, the positions of those nodes and the fusion-cost function 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.

1.3 Definitions

We provide several common definitions here that will be used in later chapters. Some terminologies used later in the each of the chapters would be specific to that chapter and defined in appropriate sections in that chapter.

Consider a weighted graph $G = (V, E, w)$, where $w : E \to \mathbb{Z}^+$. Let $s \in V$ be the sink node. For any two nodes $u, v \in V$ let $\text{dist}(u, v)$ 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 $\text{dist}(u, V')$ the smallest distance between u and any node in V'. Let D denote the *diameter* of G, that is, $D = \max_{u,v \in V} \text{dist}(u, v)$. For any path p denote its length (number of edges) as $|p|$ or $\text{len}(p)$. For any path p in G let the length be $\text{len}(p) = \sum_{e \in p} w_e$, that is, the sum of the weights of the edges in p.

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, $N(u, r) = \{v \mid \text{dist}(u, v) \leq r\}$. 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 Nieberg and Gupta *et al.* [Nie06, GKL03].

For any two nodes $u, v \in V$, their *distance* $\text{dist}(u, v)$ is the length of the shortest path that connects the two nodes in G. The *diameter* D is the length of the longest shortest path in G. The radius of a node v is $\text{rad}(v) = \max_{u \in V}(\text{dist}(v, u))$. The *radius* of G is defined as

$\text{rad}(G) = \min_v(\text{rad}(v))$. We denote by $N_k(v)$ the *k-neighborhood* of v which is the set of nodes distance at most k from v. For any set of nodes $S \subseteq V$, we denote by $N_k(S)$ the k-neighborhood of S which contains all nodes which are within distance k from any node in S.

A set of nodes $X \subseteq V$ is called a *cluster* if the induced subgraph $G(X)$ is connected. Let $Z = \{X_1, X_2, \ldots, X_k\}$ be a set of clusters in G. For every node $v \in G$, let $Z(v) \subseteq Z$ denote the set of clusters that contain v. The *degree* of v in Z is defined as $\beta_v(Z) = |Z(v)|$, which is the number of clusters that contain v. The degree of Z is defined as $\beta(Z) = \max_{v \in V} \beta_v(Z)$, which is largest degree of any of its nodes. The radius of Z is defined as $\text{rad}(Z) = max_{X \in Z}(\text{rad}(X))$.

1.4 Related Work

1.4.1 Oblivious Network Design

Below, we present the related work on oblivious network design and Table 3.1 summarizes some results and compares our work with their's. What distinguishes our work with the others' is the fact that we provide a set of paths for the MSBB problem while others provide an overlay tree for the SSBB version.

Goel *et al.* [GE03] build an overlay tree on a graph that satisfies triangle-inequality. Their technique is based on 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. In a related paper [GP09], Goel *et al.* 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. A recent improvement by Goel [GP10] provides the first constant guarantee on the simultaneous ratio of $O(1)$.

Jia *et al.* [JNRS06] build a Group Independent Spanning Tree Algorithm (GIST) that constructs an overlay tree for randomly deployed nodes in an Euclidean 2 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 3.1.

Lujun Jia *et al.* [JLN+05] 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. They provide a lower bound of $\Omega(\log n/\log\log n)$ for UST that holds even when all the vertices are on a plane.

Gupta *et al.* [GHR06] develop a framework to model *oblivious network design* problems (MSBB) and give algorithms with poly-logarithmic approximation 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 a $O(\log^2 n)$ approximation ratio and when the aggregation function is max, the approximation 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.

1.4.2 Non-Oblivious Network Design

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 here. The "single-sink buy-at-bulk" network design (SSBB) problem has a single "destination" node where all the demands from other nodes have to be routed to. Network design problems have been primarily considered in both Operations

Research and Computer Science literatures in the context of flows with concave costs. The single-sink variant of the problem was first introduced by Salman *et al.* [SCRS00]. They presented an $O(\log n)$-approximation for SSBB in Euclidean graphs by applying the method of Mansour and Peleg [MP94]. Bartal's tree embeddings [Bar94] can be used to improve their ratio to $O(\log n \log\log n)$. A $O(\log^2 n)$-approximation was given by Awerbuch *et al.* [AA97] for graphs with general metric spaces. Bartal *et al.* [Bar98] further improved this result to $O(\log n)$. Guha [GMM01] provided the first constant-factor approximation to the problem, whose ratio was estimated to be around 9000 by Talwar [Tal02]. The constant has been further improved by Grandoni and Rothvoss [GR10].

1.5 Significance of the Research

Connectivity and facilities location are two important topics in network design with applications in data communication, transportation, product planning, and VLSI designs. There are two issues concerning these two topics: design and optimization. They involve combinatorial design and combinatorial optimization. No polynomial time algorithms are known for the design and optimization for problems such as Steiner tree problems, topology network design, nonlinear assignment problems, problems in facilities location and allocation and network problems appearing in VLSI design.

Buy-at-Bulk Network Design Problem has numerous practical applications. A brief list of applications is provided below.

VLSI Power Circuitry: The exponential scaling of feature sizes in semiconductor technologies has side-effects on layout optimization, related to effects such as interconnect delay, noise and crosstalk, signal integrity, parasitics effects, and power dissipation, that invalidate the assumptions that form the basis of previous design methodologies and tools.

In a microprocessor, there are several components that need power. To minimize power usage and heat generation, microprocessors work by activating only those components that need to work while others are inactive. So, at any instant of time, only a subset of components must be powered by a *single* power circuitry. Also, this single circuit that connects all the components must be of near-optimal length for all demand scenarios. The smaller the length of the wires, the lower the IR-Drop (power dissipation).

Wireless Sensor Networks: Distributed Wireless Sensor Networks collect and send information to a sink via multiple hops in the network. During this multihop relay of information, it gets aggregated with other information at the fusion points (nodes). Typically, sensor networks applications may care only about aggregate information (eg., average temperature, humidity etc). An important aspect in such networks is the dynamism in the set of sources that needs to send data. At various instances of time, different set of sources might have data to send to the sink. Since wireless sensor nodes are energy constrained, they are incapable of computing an optimal tree for every instance. In such cases, one needs to build a single tree to route data to the sink.

Publish-Subscribe Systems: In the publish/subscribe (pub/sub) communication paradigm, publishers and subscribers interact in a decoupled fashion. Publishers publish their messages through logical channels and subscribers receive the messages they are interested in by subscribing to the appropriate services, which deliver messages through these channels. Designing an overlay network for publish/subscribe communication in a system where nodes may subscribe to many different topics of interest is of fundamental importance. For scalability and efficiency, it is important to keep the degree of the nodes in the publish/subscribe system low.

In such systems, users publish or subscribe to information and such information flowing through network can be aggregated. If a publisher produces web pages, the content distribution network replicates web pages to many locations so consumers can access at higher speed.

Another instance is the typical web-proxy installation problem where an ISP needs to determine how many web-proxies need to be installed at what places to properly serve its customers. Furthermore, it has to determine what contents are needed to be pushed into those proxies and at what rate they must be refreshed (if needed).

Oil/Gas Pipelines: There is a cost in laying oil/gas pipes to connect various stations/cities. Naturally, the larger the capacity of a pipe and the greater the number of consumers using the pipe, the cheaper would be price to pay for using the pipe (by the consumers). Hence, to build an optimal pipeline, buy-at-bulk network design principles comes into play.

Data-Center Networks: Cloud Computing is quickly being adopted by various industries and customers alike despite apparent issues in security and maintenance. A key factor in the performance of cloud-computing is the network efficiency of the associated data-centers (DC). Data centers are located geographically apart to serve customers in all regions. The inter-DC network bandwidth poses a high-risk in performance (goodput) if the network is not properly designed. This problem boils down to properly decomposing the network graph such that customers at geographically well-separated regions are well-served.

Chapter 2

Doubling-Dimension Graphs

2.1 Overview

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 non-negative 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 demand sources 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 a $O(\log^3 D)$-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.

2.1.1 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 non-negative 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 demand aggregates. 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 $f(x) = 1$) [SCRS00].

Definition 2.1.1 (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 2.1.2. *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 2.1.3. *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 2.1.2). If, $l \geq 3$, from Observation 2.1.2, U can be covered by at most $2^{(k-l+3)\rho}$ number of 2^{l-3}-neighborhoods. Therefore, have that $|I| \leq 2^{(k-l+3)\rho}$. □

We consider building an oblivious spanning tree for doubling dimension graphs. Doubling dimension graphs has been used in many different contexts including compact routing in wired networks [AGGM06, KRX08], traveling salesman, navigability and problems related to modeling the structural properties of the Internet distance matrix for distance estimation [KSW09, Fra07]. As noted by Fraigniaud [FLL06], it has become a key concept to measure the ability of network 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 [KMW05], distributed resource-management [GGMZ09], information exchange among producers and consumers [FGNW06], and for determining other performance qualities such as energy-conservation in wireless sensor networks [PP06].

2.1.2 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 demand 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 a $O(\log^3 D)$ competitive ratio. When $f(\cdot) = c$, a constant, our spanning tree solution provides a $O(\log^3 D)$-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 demand 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 gives 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 and Shortest-Paths Tree). For comparison, we generate the trees and costs by simulation using NetworkX [HSS08]. The simulations corroborate the analytical results and show that the oblivious spanning tree provides very competitive costs and in fact provides better costs than the well known trees.

2.2 Definitions

A set of nodes I is said to be *a d-independent set* if for each pair $u, v \in I$, $u \neq v$, $\text{dist}(u, v) \geq d$. 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.

2.3 Technique Used

Our spanning tree construction is based on the following techniques. We partition the nodes in a hierarchical fashion. The selection of nodes for a given 'level' of hierarchy is based on their mutual distances proportional to the level. Nodes of successive levels are connected by shortest paths. The intersecting paths are appropriately modified to result in a spanning tree. A modified tree is built from the spanning tree to ensure that all paths have appropriate end-

nodes. Analysis is done on this modified tree.

2.4 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 (eg., 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 $p(v)$.

2.4.1 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 2.7 for the modified tree analysis.

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

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

Observation 2.4.1. *For any node $v \in V$, $|p_i(v)| < \mu_i$.*

Lemma 2.4.2. *For any $v \in Z_i^u$, $\text{dist}(v, u) < \delta_i$.*

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 2.4.1, $|p_j(v)| < \mu_j = 2^{j+2}$. 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 2.4.3. $N(s, 2^i - 1) \subseteq Z_i^s$.

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 $\text{dist}(x, v) = 2^{j+1} - 1$. 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 2.4.3, we obtain the following corollary:

Corollary 2.4.4. *For any $u \in I_i$, $u \neq s$, $\text{dist}(s, Z_i^u) \geq \phi_i$.*

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 $\text{dist}(u,v) \leq \xi_i$.

Lemma 2.4.5. *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 is 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 2.1.3, 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. □

2.4.2 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, non-decreasing 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:

$$C(A) \leq \sum_{i=0}^{\kappa-1} C_i(A). \tag{2.1}$$

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

Lemma 2.4.6. *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 2.4.2, any node in A_i^u is at distance at most $\delta_i - 1$ from u. Since any pair $u, v \in I' \setminus \{s\}$, $u \neq v$, are 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 2.4.4, $s \notin N(A_i^u, \phi_i - 1)$. 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 2.4.7. *$C_i(A) \leq Q_i$, 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 2.4.1

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

Proof. From Lemma 2.4.5, 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 2.4.7, we have that $C_i(A) \leq Q_i \leq \chi \cdot Q_i^{j^*}$. Further, from Lemma 2.4.6, $C^*(A) \geq R(I_i^{j^*}) = Q_i^{j^*} \cdot \phi_i/\mu_i$. 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 Equation 2.1 and Lemma 2.4.8:

Theorem 2.4.9 (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 2.4.9, we immediately obtain the following corollary when we replace the values of the parameters.

Corollary 2.4.10. *The oblivious competitive ratio of the overlay tree T is $C.R.(T) = O(2^{7\rho} \cdot \log D)$.*

2.5 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.

Algorithm 1: Spanning Tree

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

$P \leftarrow \emptyset$; $I_\kappa \leftarrow \{s\}$; // $\kappa \leftarrow \lceil \log D \rceil$
$P^{reg} \leftarrow \emptyset$; $P^{pr} \leftarrow \emptyset$; // List of regular and pruned paths
foreach *level* $i = \kappa - 1$ **to** 0 **do**
  $I_i \leftarrow MIS(G, I_{i+1}, 2^i)$;
  foreach $v \in I_i$ **do**
    $p_i(v) \leftarrow$ `FindPath`(v, i);
    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
      $p'_i(v) \leftarrow$ path segment from v to u;
      $P_i^{pr} \leftarrow P_i^{pr} \cup p'_i(v)$;
    else
      $P_i^{reg} \leftarrow P_i^{reg} \cup p_i(v)$;
    end
  end
end
$P \leftarrow \bigcup_{i=0}^{i=\kappa-1} P_i^{reg} \cup \bigcup_{i=0}^{i=\kappa-1} P_i^{pr}$;
return T_s ; // Formed by paths in P

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 falls inside the subtree Z_i^s rooted at

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

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 a $(i+1)$-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 go 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 2.7.3.

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 Figure 2.1).

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 avoids 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.

2.6 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 'pseudo-leaders' $\overline{I}_i$ 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` .

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 *pseudo-leader*

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 pseudo-leader in any given path $p_j(v)$, $j > i$, is justified by the Lemma 2.6.1.

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 $p_j(v)$;
$\bar{p}_i(u) \leftarrow p_i^a(u) + p_i^b(y)$; `// Concatenate` $p_i^a(u)$ `and` $p_i^b(y)$.
return $\overline{p}_i(u)$;

Lemma 2.6.1 (**Presence of a Pseudo-Leader**). *The* `ModifyPath`$(p_i(u), p_j(v))$ *function guarantees selection of a* $(i+1)$*-level pseudo-leader.*

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 pseudo-leader in $p_j(v)$. If level of w is $i+1$, then, w itself acts as a pseudo-leader 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 pseudo-leader is guaranteed to be found in $p_j(v)$ for u. □

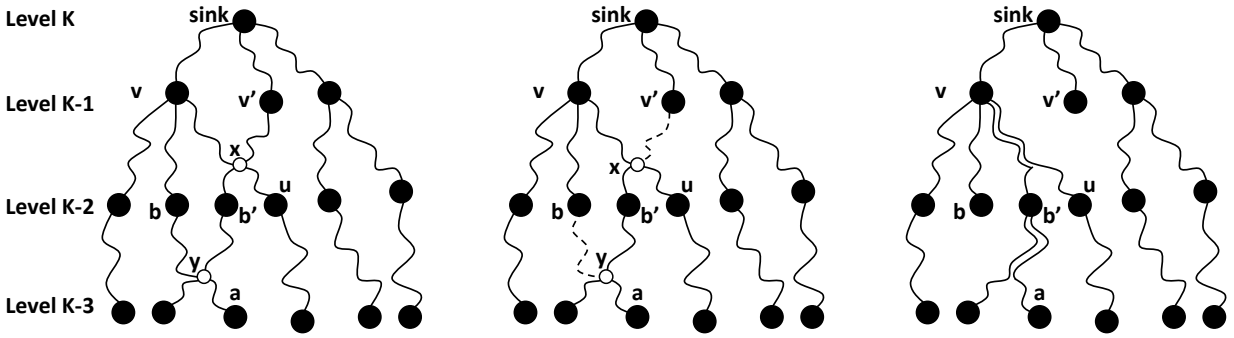
Consider that we are at some level i where $0 \leq i \leq \kappa - 1$ and suppose that there are several

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 $\overline{T}$;

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 *pseudo-leader*, v', is chosen on $p_j(v)$ using ModifyPath $(p_i(u), p_j(v))$ in Modify-Path . This pseudo-leader 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 pseudo-leaders towards the direction of s. Note that this may alter I_j to $\overline{I}_j$ by replacing the original leader by the pseudo-leader. 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 the upper bound on the length of $\overline{p}_i(u)$ is given by Lemma 2.7.1. 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



(a) Intersecting paths

(b) Pruned paths y, b and x, v'.

(c) Modified Paths $p_{a,b'}$ and $p_{b',v}$

Figure 2.1: Pruning and Tree Modification.

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 pseudo-leaders for respective types of paths.

Figure 2.1 gives an example of intersecting path and its modification to reach a pseudo-leader 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.

2.7 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 2.4. 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 2.4.1. With these values, we can immediately apply the results of section 2.4.2 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 pseudo-leaders $\overline{I}_0$ 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:

$$
\begin{aligned}
\overline{\mu}_i &= 2^{i+3} \quad \text{//upper bound on } |\overline{p}_i(u)| \\
\overline{\delta}_i &= 2^{i+3} \quad \text{//upper bound on the radius of } \overline{Z}_i^u \\
\overline{\phi}_i &= 2^i \quad \text{//lower bound on dist}(s, \overline{Z}_i^u),\ u \neq s \\
\overline{\xi}_i &= 2\overline{\delta}_i + 2\overline{\phi}_i \quad \text{//coloring radius} \\
\overline{\chi} &= 2^{17\rho} \log^2 D \quad \text{//coloring of } \overline{I}_i \text{ with radius } \xi_i
\end{aligned}
$$

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 2.7.5, 2.7.6 and 2.7.7 establishes the maximum number of pseudo-leaders in a given neighborhood.

Lemma 2.7.1. $|\overline{p}_i(u)| < \overline{\mu}_i$.

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 (pseudo-leader) on $p_j(v)$ that is close to y and in the direction of s (Lemma 2.6.1). Note that y itself cannot be the pseudo-leader for u because, if it was, then, $p_i(u)$ would not have been a pruned path. The distance from y to a pseudo-leader 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 pseudo-leader 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 2.7.2. *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 2.7.3. $N(s, 2^i - 1) \subseteq \overline{Z}_i^s$.

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 pseudo-leader 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 towards s. A pseudo-leader $\overline{\ell}_{j+1}$ 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 2.7.1. Since the pseudo-leader 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 2.7.3, we obtain the following corollary:

Corollary 2.7.4. *For any* $u \in \overline{I}_i$, $u \neq s$, $\text{dist}(s, \overline{Z}_i^u) \geq \overline{\phi}_i$.

Lemma 2.7.5 (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 2.1.3, 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 2.1.3, 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 2.7.6 (**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 2.7.1, 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 2.1.3, 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 2.1.3, 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 2.7.7. *The total number of pseudo-leaders 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 2.7.5, 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 2.7.6, 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 pseudo-leaders 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 2.7.8. *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 $\overline{I}_i$ is 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 2.7.7, 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 (2.4.2), 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 2.7.9 (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 2.7.9, we immediately obtain the following corollary when we replace the values of the parameters.

Corollary 2.7.10. *The oblivious competitive ratio of the modified tree $\overline{T}$ is $C.R.(\overline{T}) = O(2^{17\rho} \log^3 D)$.*

2.8 Lower Bound

We now present an overview of the technique used for computing the lower-bound. The lower-bound given by Imase and Waxman in [IW91] doesn't work in our case. Their technique works

for non-low-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 an 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 4 neighbors. For example, see figure 2.4.

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 forms a U^x-Path. Let u be its reference node. Let there be a node $p \in U^x$-Path such that $\text{dist}(u, p) \geq x/2 - 1$. If the

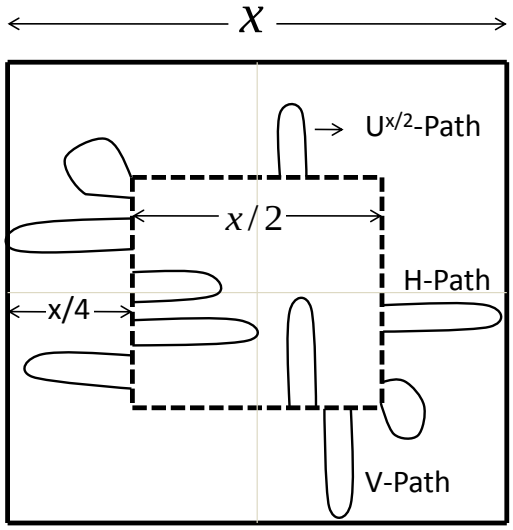


Figure 2.2: $U^{x/2}$-Paths originating from a $x/2 \times x/2$ subgrid centered in a $x \times x$ subgrid of G.

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.

Lemma 2.8.1. *In a $x \times x$ 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 that $x - 1$. □

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 a $x/2 \times x/2$ subgrid centered within an $x \times x$ subgrid of G as given in Figure 2.2. 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 a H-Path or a V-Path in it, as shown in Fig 2.3. 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.

Consider a 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 2.8.2 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 2.8.2. *The total number of GVLs in an x-class of G is $3n/128$.*

Proof. Consider a H-Type x-class decomposition of G. The 'width' of any H-Path in a subgrid

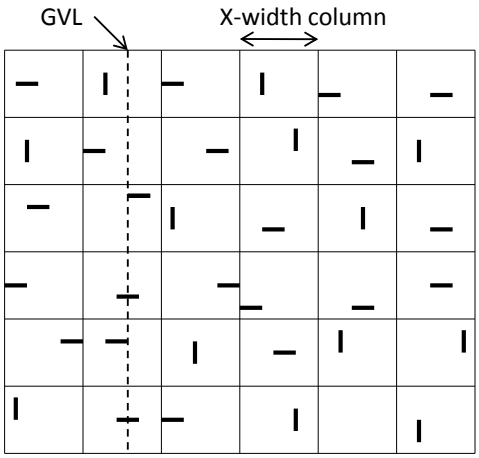


Figure 2.3: An example of a GVL in a grid where each $x \times x$ subgrid has either an H-Path or a V-Path.

is at least $x/8$. Hence, the number of vertical lines that can intersect such a 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 a 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 2.8.3. *There is a GVL (denoted by GVL*) that is common to a constant fraction of the total number of horizontal classes.*

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 2.8.4. *There exists a set S of nodes in G such that (i) S constitutes $\theta(n)$ nodes (ii) Optimal tree T^* for S has cost $O(n)$ and (iii) The induced subtree $T(S)$ has $\Omega(n \log n)$ cost.*

Proof. From Lemma 2.8.3, 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 2.4. 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 $x_i(n/x_i) + x_j(n/x_j) + \ldots = \theta(n \log n)$. 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 that 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 2.8.4, we obtain the following corollary:

Corollary 2.8.5. *In any $n \times n$ grid, any spanning tree T will have $C.R.(T) = \Omega(\log n)$.*

2.9 Simulation Results

We simulated our algorithm, denoted by Oblivious Spanning Tree (OST) and compared its performance (fusion-cost) with GRID_GIST [JNRS06] and other common trees such as MST

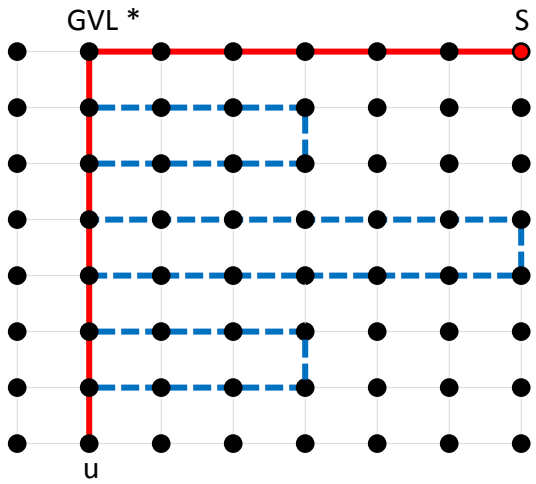


Figure 2.4: Paths in an $n \times n$ grid.

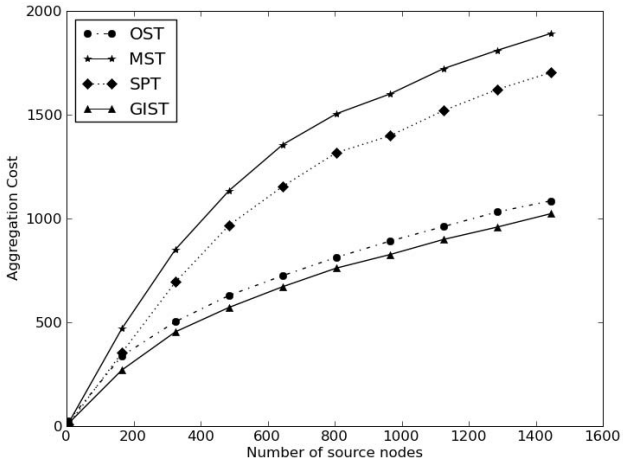


Figure 2.5: fusion-cost for varying set of source nodes in a 1600-node grid.

(Minimum Spanning Tree) and SPT (Shortest-Paths Tree). We used an $n \times n$ grid topology for our simulation using NetworkX [HSS08]. $n \times n$ grids are a special case of doubling-dimension graphs and they fall under a variation of the Steiner tree problem called "Rectilinear Steiner Problem" (RSP) where the tree structure has only vertical and horizontal lines that interconnect all points and is proved to be NP-Complete [GJ77]. 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 $n^2 = 1600$ nodes. We simulate it for random

sets of data sources, up to 1445, 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 2.5).

2.10 Conclusions

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 work 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.

Chapter 3

Planar Graphs

3.1 Overview

In the oblivious buy-at-bulk network design problem in a graph, the task is to compute a fixed set of paths for every pair of source-destination in the graph, such that any set of demands can be routed along these paths. The demands could be aggregated at intermediate edges where the fusion-cost is specified by a canonical (non-negative concave) function f. We give a novel algorithm for planar graphs which is oblivious with respect to the demands, and is also oblivious with respect to the fusion function f. The algorithm is deterministic and computes the fixed set of paths in polynomial time, and guarantees a $O(\min(\log n, \log D))$-approximation ratio for any set of demands and any canonical fusion function f, where n is the number of nodes and D the diameter of the graph. The algorithm is asymptotically optimal, since it is known that this problem cannot be approximated with better than $\Omega(\log n)$ ratio. To our knowledge, this is the first tight analysis for planar graphs, and improves the approximation ratio by a factor of $\log n$ with respect to previously known results.

3.1.1 Problem Statement

Assume that we are given a weighted graph $G = (V, E, \mathfrak{w})$, with edge weights $\mathfrak{w} : E \longrightarrow \mathbb{Z}^+$. We denote $\mathfrak{w}_e$ to be the weight of edge e. Let $d_i = (s_i, t_i)$ be a unit of demand that induces an unsplittable unit of flow from source node $s_i \in V$ to destination node $t_i \in V$. Let $A = \{d_1, d_2, \ldots, d_r\}$ be a set of demands that are routed through paths in G. It is possible that some paths may overlap. The flow of these demands forms a set of paths $P(A) = \{p(d_1), p(d_2), \ldots, p(d_r)\}$.

There is an arbitrary canonical function f at every edge where demand aggregates. This f is same for all the edges in G. Let $\varphi_e(A) = \{p(d_i) : e \in p(d_i)\}$ denote the set of paths 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 $C(A) = \sum_{e \in E} C_e(A)$. For this set A, there is an optimal set of paths $P^*(A)$ with respective cost $C^*(A)$. The approximation ratio for the paths $P(A)$ is defined as $\frac{C(A)}{C^*(A)}$. The MSBB optimization problem on input A is to find a set of paths $P(A)$ that minimizes the approximation ratio. We note that MSBB is NP-Hard as the Steiner tree problem is its special case (when $f(x) = 1$ and when there is only one destination node) [SCRS00].

An *oblivious algorithm* $\mathcal{A}_{obl}$ for the MSBB problem, computes a fixed set of paths, denoted $P(G)$ for every pair of source destination nodes in V. Given any set of demands A, the path $p(d_i)$ for each $d_i = (s_i, t_i) \in A$, is the fixed path in $P(G)$ from s_i to t_i. This gives a set of paths $P(A)$ to route the demands A. We define the *approximation ratio* of $\mathcal{A}_{obl}$, as:

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

We aim to find algorithms that minimizes the above approximation ratio for any canonical function f which is unknown to the algorithm. The best known oblivious algorithm is by Gupta *et al.* [GHR06] and provides approximation ratio $O(\log^2 n)$ for general graphs. No better result is known for planar graphs. This problem is NP-hard, since MSBB is NP-hard.

3.1.2 Contribution

We provide an oblivious algorithm FindPaths for MSBB problems in planar graphs. Our algorithm is deterministic and computes in polynomial time a fixed set of paths that guarantees $O(\min(\log n, \log D)$-approximation ratio for any canonical function f (where f is unknown to the algorithm). A lower bound of $\Omega(\log n)$ for planar graphs is provided in the context of the online Steiner tree problem by Imase and Waxman [IW91]. Thus, our bound is tight with

respect to planar graphs. It is also a $\log n$ factor improvement over the best previously known result [GHR06].

3.2 Definitions

Consider a *locality parameter* $\gamma > 0$. A set of clusters Z is said to γ-*satisfy* a node v in G, if there is a cluster $X \in Z$, such that the γ-neighborhood of v, $N_\gamma(v)$, (nodes within distance γ from v) is included in X, that is, $N_\gamma(v) \subseteq X$. A set of clusters Z is said to be a γ-*cover* for G, if every node of G is γ-satisfied by Z in G. The *stretch* $\sigma(Z)$ of a γ-cover Z is the smallest number such that $\mathrm{rad}(Z) = \sigma(Z) \cdot \gamma$.

We define the following coloring problem in a set of clusters Z. We first define the notion of the distance between two clusters $X_i, X_j \in Z$, $X_i \neq X_j$. We say that $\mathrm{dist}(X_i, X_j) \leq k$, if there is a pair of nodes $u \in X_i$ and $v \in X_j$ such that u is k-satisfied in X_i, v is k-satisfied in X_j, and $\mathrm{dist}(u, v) \leq k$. A valid distance-$k$ coloring of Z with a palette of χ colors $[1, \chi]$, is an assignment of an integer $\mathrm{color}(X) \in [1, \chi]$ to every $X \in Z$, such that there is no pair of clusters $X_i, X_j \in Z$, $X_i \neq X_j$, with $\mathrm{dist}(X_i, X_j) \leq k$ which receive the same color. The objective is to find the smallest χ that permits a valid distance-k coloring.

3.3 Technique Used

We build the set of paths based on sparse covers (see [Pel00] for an overview of sparse covers). A γ-cover consists of clusters such that for each node there is some cluster that contains its γ-neighborhood. We construct $O(\log D)$ levels of covers with exponentially increasing locality parameter γ. For every cluster we elect a leader. For any pair of nodes u, v we identify an appropriate common lowest-level cluster that contains both u and v, and the cluster has a respective common leader ℓ. Then the path from u to v is formed by connecting successive path segments emanating from both u and v and using intermediate leaders of lower level

Table 3.1: Our results and comparison with previous results for data-fusion schemes. n is the total number of nodes in the topology, k is the total number of source nodes and D is the diameter of graph G.

Related Work	**Algorithm Type**	**Graph Type**	**Oblivious Function** f	**Oblivious Sources**	**Approx Factor**
Lujun Jia *et al.* [JNRS06]	Deterministic	Random Deployment	×	✓	$O(\log n)$
Lujun Jia *et al.* [JLN+05]	Deterministic	Arbitrary Metric	×	✓	$O(\frac{\log^4 n}{\log\log(n)})$
	Deterministic	Doubling Metric	×	✓	$O(\log(n))$
Ashish Goel *et al.* [GE03]	Randomized	General Graph △-inequality	✓	×	$O(\log k)$
Ashish Goel *et al.* [GP09, GP10]	Randomized	General Graph	✓	×	$O(1)$
Anupam Gupta *et al.* [GHR06]	Randomized	General Graph	✓	✓	$O(\log^2 n)$
	Randomized	Low Doubling	✓	✓	$O(\log n)$
This work	Deterministic	Planar	✓	✓	$O(\min(\log n, \log D))$

clusters until the common leader ℓ is reached.

In the analysis, we introduce the notion of coloring sparse covers, where two clusters that are close receive different color. We show the existence of a sparse cover with constant coloring (based on the sparse covers in [BLT07]). This enables us to obtain optimal approximation at every level. When we combine all the levels, we get an $O(\log D)$ approximation.

In section 3.7, we provide an analysis for an $O(\log n)$-approximation by splitting the demands into ranges according to the distance between sources and destinations and mapping those demand ranges to edge-weight ranges in G. The analysis provides four sequences of interleaving ranges that has non-overlapping edge-weights. The summation of the costs for all the demands in all these sequences are then showed to be in $O(\log n)$.

3.4 Sparse Cover

A γ-cover is *sparse* if it has small degree and stretch. In [BLT07, Section 5] the authors present a polynomial time sparse cover construction algorithm Planar-Cover(G, γ) for any planar graph G and locality parameter γ, which finds a γ-cover Z with constant degree, $\beta \leq 18$, and constant stretch, $\sigma \leq 24$. Here, we show that this cover also admits a valid distance-γ coloring with a constant number of colors $\chi \leq 18$.

For any node $v \in G$, we denote by $\text{depth}_v(G)$ the shortest distance between v and an external node (in the external face) of G. We also define $\text{depth}(G) = \max_{v \in V} \text{depth}_v(G)$. The heart of sparse cover algorithm in [BLT07, Section 5] concerns the case where $\text{depth}(G) \leq \gamma$ which is handled in Algorithm Depth-Cover(G, γ). The general case, $\text{depth}(G) > \gamma$, is handled by dividing the graph into zones of depth $O(\gamma)$, as we discuss later. So, assume for now that $\text{depth}(G) \leq \gamma$.

The Algorithm Depth-Cover(G, γ), relies on forming clusters along shortest paths connecting external nodes (in the external face) of G. For every shortest path p, Algorithm Shortest-Path-Cluster$(G, p, 4\gamma)$ in [BLT07, Section 3] returns a set of clusters around the 4γ neighborhood of p with radius at most 8γ and degree 3. Then, p and all its 2γ-neighborhood is removed from G producing a smaller subgraph G' (with possibly multiple connected components). The algorithm proceeds recursively on each connected component H of G' by selecting an appropriate new shortest path p' between external nodes of H. The algorithm terminates when all the nodes have been removed. The initial shortest path that starts the algorithm consists of a single external node in G. The resulting γ-cover Z consists of the union of all the clusters from all the shortest paths. The shortest paths are chosen in such a way that a node participates in the clustering process of at most 2 paths, and this bounds the degree of the γ-cover to be at most $\beta \leq 6$, and stretch $s \leq 8$.

The analysis in [BLT07, Section 5.1.1] of Algorithm Depth-Cover relies on representing the

clustering process of G as a tree T as we outline here. Each tree node $w \in T$ represents a pair $w = (G(w), p(w))$ where $G(w)$ is a planar subgraph of G that is to be clustered, and $p(w)$ is a shortest path between two external nodes of $G(w)$. The root of the tree is $r = (G, v)$, where v is a trivial initial path with one external node $v \in G$. The children of a node $w \in T$ are all the nodes $w' = (G(w'), p(w'))$, such that $G(w')$ is a connected component that results from removing $p(w)$ and its 2γ-neighborhood from G.

Next, we extend [BLT07, Lemma 3.1] to show that we can color the clusters obtained by a shortest path clustering using a constant number of colors.

Lemma 3.4.1. *For any graph G, shortest path $p \in G$, the set of clusters returned by Algorithm* Shortest-Path-Cluster$(G, p, 4\gamma)$ *admits a valid distance-γ coloring with 3 colors.*

Proof. The algorithm divides the path p into consecutive disjoint subpaths $p_1, p_2, \ldots, p_\ell$ each of length 4γ (except for the last subpath p_ℓ which may have shorter length). The algorithm builds a cluster X_i around each subpath p_i which consists of the 4γ-neighborhood of p_i. We can show that $\text{dist}(X_i, X_{i+3}) > \gamma$. Suppose otherwise. Then, there are nodes $u \in X_i$, $v \in X_{i+3}$, which are γ-satisfied in their respective clusters and $\text{dist}(u, v) \leq \gamma$. Thus, $u \in X_{i+3}$. Then, there is a path of length at most 8γ that connects the two paths p_i and p_{i+3} which is formed through u. However, this is impossible since the paths are at distance at least $8\gamma + 1$. Therefore, we can use a palette of at most 3 colors to color the clusters, so that each cluster X_i receives color $(i \mod 3) + 1$. □

We can obtain a coloring of Z by coloring the respective levels of the tree T. Assume that the root is at level 0.

Lemma 3.4.2. *The union of clusters in any level $i \geq 0$ of the tree T, admits a valid distance-γ coloring with 3 colors.*

Proof. Consider a level $i \geq 0$ of T. From Lemma 3.4.1, the clusters produced in any node w of

level i from path $p(w)$ can be colored with 3 colors. Consider now two nodes w_1 and w_2 in level i of T. Let X be a cluster from $p(w_1)$. Any node which is γ-satisfied (with respect to G) in X, cannot have in its γ-neighborhood any node in $G(w_2)$, since $G(w_1)$ and $G(w_2)$ are disjoint. Therefore, any two nodes which are γ-satisfied in the respective clusters of w_1 and w_2 have to be at distance more than γ from each other in G. This implies that we can use same palette of 3 colors for each node in the same level i of the tree. □

Lemma 3.4.3. *Algorithm* Depth-Cover(G, γ) *returns a set of clusters Z which admits a valid distance-γ coloring with 6 colors.*

Proof. From Lemma 3.4.2, the clusters of each level of the tree T can be colored with 3 colors. From the proof in [BLT07, Lemma 5.4], any node $v \in G$ is clustered in at most 2 consecutive levels $i, i+1$ of T, and does not appear in any subsequent level. Any node u which is γ-satisfied in a cluster of level $i+2$ cannot be with distance of γ or less from v, since v doesn't appear in the level $i+2$ subgraph of G. Therefore, any node v which is γ-satisfied in a cluster of level i must be at distance more than γ than any node u which is γ-satisfied in a cluster of level $i+2$. Therefore the clusters formed at level i are at distance at least $\gamma+1$ from clusters formed at level $i+2$. Consequently, we can use color palette $[1,3]$ for odd levels and color palette $[4,6]$ for even levels, using in total 6 colors. □

We are now ready to consider the case depth$(G) > \gamma$. Algorithm Planar-Cover(G, γ) decomposes the graph into a sequence of bands, such that each band W_i has depth γ. The bands are organized into zones, such that zone S_i consists of three consecutive bands W_{i-1}, W_i, W_{i+1}. Thus, zone S_i overlaps with bands S_{i-2}, S_{i-1}, S_{i+1} and S_{i+2}. The algorithm invokes Depth-Cover$(S_i, 3\gamma-1)$ for each zone giving a γ-cover Z with degree $\beta \leq 3 \cdot 6 = 18$ and stretch $\sigma \leq 3 \cdot 8 = 24$.

We can obtain the following coloring result. Using Lemma 3.4.3, for every zone S_i we can get a valid distance-$(3\gamma-1)$ coloring with a palette of 6 colors. This implies that we can obtain

Algorithm 3: AuxiliaryPaths(G)

Input: Graph $G = (V, E, \mathfrak{w})$
Output: Set of auxiliary paths for all nodes in G

```
// Z_i is a γ_i-cover of G, 0 ≤ i ≤ κ (γ_i, κ specified in Section 3.6)
// Assume each cluster has a designated leader node
```

$\mathcal{Q} \leftarrow \emptyset$; `// set of auxiliary paths for all nodes in` G
foreach $v \in V$ **do**
  $q(v) \leftarrow \emptyset$; `// auxiliary path for node` v `from level 0 to` κ
  $x \leftarrow v$;
  for $i = 0$ **to** $\kappa - 1$ **do**
    Let $X \in Z_{i+1}$ be a cluster that γ_{i+1}-satisfies v;
    $\ell_{i+1}(v) \leftarrow$ leader of X;
    $q_i(v) \leftarrow$ shortest path from x to $\ell_{i+1}(v)$;
    $q(v) \leftarrow$ concatenate $q(v)$ and $q_i(v)$;
    $x \leftarrow \ell_{i+1}(v)$;
  end
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup q(v)$;
end
return $\mathcal{Q}$;

a valid distance-γ coloring for the zone with at most 6 colors. Zones S_i and S_{i+3} do not overlap and any two nodes satisfied in them (one from each zone) with respect to G must be more than γ distance apart. Therefore, we can color all the zones with three different palettes each consisting of 6 colors, so that zone S_i, uses the $((i \mod 3) + 1)$th palette. The coloring can be found in polynomial time. Therefore, we obtain:

Theorem 3.4.4. *Algorithm* Planar-Cover(G, γ) *produces a set of clusters Z which has degree $\beta = 18$, stretch $\sigma \leq 24$, and admits a valid distance-γ coloring with $\chi = 18$ colors.*

3.5 Algorithm

We describe how to find paths between each pair of nodes in graph $G = (V, E, \mathfrak{w})$ to route demands. To find such paths, we use Algorithm FindPaths (Algorithm 4) which relies on Algorithm AuxiliaryPaths (Algorithm 3).

Both algorithms use $\kappa + 1$ covers $Z_0, \ldots, Z_\kappa$, where in Z_0 every node in V is a cluster, and

Algorithm 4: FindPaths(G)

Input: Graph $G = (V, E, \mathfrak{w})$.
Output: Set of paths between all pair of nodes in G

// Let $Z_i, \ell_i(x)$, and $q(x)$, be as in Algorithm AuxiliaryPaths; $\ell_0(x) \leftarrow x$
// Let $\overline{q}_i(x)$ be the auxiliary path segment from x to $\ell_i(x)$; $\overline{q}_0(x) \leftarrow x$

$\mathcal{P} \leftarrow \emptyset$; // set of paths for all pairs of nodes in G
foreach *pair* $u, v \in V$ **do**
  Let γ_i be the smallest locality parameter such that $\gamma_i \geq 2 \cdot \text{dist}(u, v)$;
  Let $X \in Z_i$ be a cluster that γ_i-satisfies u (and hence $\gamma_i/2$-satisfies v);
  Let ℓ' be the leader of X (*common leader* of u, v);
  $q' \leftarrow$ concatenate shortest paths from $\ell_{i-1}(u)$ to ℓ' to $\ell_{i-1}(v)$;
  $p(u, v) \leftarrow$ concatenate $\overline{q}_{i-1}(u)$, q', and $\overline{q}_{i-1}(v)$;
  $\mathcal{P} \leftarrow \mathcal{P} \cup p(u, v)$;
end
return $\mathcal{P}$;

Z_i is a γ_i-cover of G, for $i \geq 1$, where the parameters γ_i and κ are defined in Section 3.6. We refer to the cover Z_i as the level i cover of G. We assume that each cluster in the covers has a designated leader node. There is a unique cluster, containing all nodes in G, and leader node ℓ_κ at level κ.

Algorithm AuxiliaryPaths computes an *auxiliary path* $q(v)$ from every node $v \in V$ to ℓ_κ. The auxiliary paths are built in a bottom-up fashion. An auxiliary path from any node $v \in V$ at level 0 is built recursively. In the basis of the recursion, we identify a cluster $X_1 \in Z_1$, which γ_1-satisfies node v. Let $\ell_1(v)$ denote the leader X_1. We now compute a shortest path, denoted $q_0(v)$, from v to $\ell_1(v)$. This forms the first path segment of $q(v)$. Suppose we have computed $q(v)$ up to level i, $i < \kappa$. We now want to extend this path to the next higher level $i + 1$. To compute the path segment from level i to level $i + 1$, we repeat the process of finding a cluster $X_{i+1} \in Z_{i+1}$ that γ_{i+1}-satisfies node v. Let $\ell_{i+1}(v)$ denote the leader X_1. We compute the shortest path, denoted $q_i(v)$ from $\ell_i(v)$ to $\ell_{i+1}(v)$. We then append this new path segment $q_i(v)$ to $q(v)$ to form the current extended path $q(v)$. The path building process terminates when the last leader reaches level κ.

We are now ready to describe how Algorithm FindPaths computes the shortest paths between

all pair of nodes in G. For a pair of nodes $u, v \in V$, let y be the distance between them. Let γ_i be the smallest locality parameter such that $\gamma_i \geq 2y$. Let $X \in Z_i$ be the cluster that γ_i satisfies u, and let ℓ' be the respective leader of X. Note that by the way that we have chosen γ_i, cluster X also $\gamma_i/2$-satisfies v. Let $\overline{q}_i(u)$ denote the segment of the auxiliary path $q(u)$ from u to $\ell_i(u)$. We concatenate $\overline{q}_{i-1}(u)$, with a shortest path from $\ell_{i-1}(u)$ to ℓ', with a shortest path from ℓ' to $\ell_{i-1}(v)$, and $\overline{q}_{i-1}(v)$. This gives the path $p(u, v)$.

3.6 Analysis for $O(\log D)$-approximation

Let $G = (V, E, \mathfrak{w})$ be a planar graph with n nodes. In this section we use the following parameters:

$$
\begin{aligned}
&\kappa = 1 + \lceil \log_{4\sigma} D \rceil && \text{//highest cluster level in } G \\
&\beta = 18 && \text{//cover degree bound} \\
&\sigma = 24 && \text{//cover stretch bound} \\
&\gamma_i = (4\sigma)^{i-1} && \text{//locality parameter of level } i \geq 1 \text{ cover} \\
&\chi = 18 && \text{//coloring of each level } i
\end{aligned}
$$

Consider $\kappa + 1$ levels of covers $Z_0, \ldots, Z_{\kappa+1}$, where in Z_0 each node in V is a cluster, and each Z_i, $i \geq 1$, is a γ_i-cover of G which is obtained from Theorem 3.4.4. Thus, each Z_i, $i \geq 1$, has degree at most β, stretch at most σ, and can be given a valid distance-γ_i coloring with χ colors.

Let A denote an arbitrary set of demands. For any demand $d = (s, t) \in A$ let $p(d) = p(s, t)$ be the path given by Algorithm FindPaths. Suppose that the common leader of s and t is ℓ. The path $p(d)$ consists of two path segments: the *source path segment* $p(s)$, from s to ℓ, and the *destination path segment* $p(t)$ from ℓ to t. We denote by $p_i(s)$ the subpath between level i and level $i + 1$ (we call this the level i subpath).

Let $C^*(A)$ denote the cost of optimal paths in A. Let $C(A)$ denote the cost of the paths given by our algorithm. We will bound the competitive ratio $C(A)/C^*(A)$. For simplicity, in

the approximation analysis, we consider only the cost of the source path segments $p(s_i)$. When we consider the destination segments the approximation ratio increases by a factor of 2.

The cost $C(A)$ can be bounded as a summation of costs from the different levels as follows. For any edge e let $\varphi_{e,i}(A) = \{p_i(s) : ((s,t) \in A) \wedge (e \in p_i(v))\}$ be the set of layer-i subpaths that use edge e. 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:

$$C(A) \leq \sum_{i=0}^{\kappa-1} C_i(A). \tag{3.1}$$

For any cluster X let $X(A)$ denote the set of demands with source in X whose paths leave from the leader of X toward the leader of a higher level cluster.

Here, we provide an analysis of our algorithm that provides an $O(\log D)$-approximation for the cost of the set of paths.

Lemma 3.6.1. *For any Z_i, $2 \leq i \leq \kappa - 1$, $C^*(A) \geq R(i)/\chi$, where $R(i) = \sum_{X \in Z_i} f(|X(A)|) \cdot \gamma_i/2$.*

Proof. Let $Z_i(k)$ to be the set of clusters at level i which receive color $k \in [1, \chi]$. Consider a cluster $X \in Z_i(k)$. Consider a demand $(s,t) \in X(A)$. Since $X \in Z_i(k)$ the common leader of s and t is at a level $i+1$ or higher. From the algorithm, $\text{dist}(s,t) \geq \gamma_{i+1}/2$. Consider the subpaths from $X(A)$ of length up to $\gamma_i/2$. In the best case, these subpaths from $X(A)$ may be combined to produce a path with smallest possible total cost $f(|X(A)|) \cdot \gamma_i/2$. Any two nodes $u \in X(A)$ and $v \in Y(A)$, where $X, Y \in Z_i(k)$ and $X \neq Y$, have $\text{dist}(u,v) > \gamma_i$, since each node is γ_i-satisfied in its respective cluster and X and Y receive the same color in the distance-γ_i coloring of Z. Therefore, the subpaths of lengths up to $\gamma_i/2$ from the demands $X(A)$ and $Y(A)$ cannot combine. Consequently, $C^*(A) \geq R(i,k)$ where $R(i,k) = \sum_{X \in Z_i(k)} f(|X(A)|) \cdot \gamma_i/2$. Let

$R_{\max} = \max_{k\in[1,\chi]} R(i,k)$. We have that $C^*(A) \geq R_{\max}$. Since $R(i) = \sum_{k=1}^{\chi} R(i,k) \leq R_{\max} \cdot \chi$. We obtain $C^*(A) \geq R(i)/\chi$, as needed. □

We also get the following trivial lower bound for the special case where $0 \leq i \leq 1$, which follows directly from the observation that each demand needs to form a path with length at least 1.

Lemma 3.6.2. *For any Z_i, $0 \leq i \leq 1$, $C^*(A) \geq \sum_{X\in Z_i} f(|X(A)|)$.*

We obtain the following upper bound.

Lemma 3.6.3. *For any Z_i, $0 \leq i \leq \kappa - 1$, $C_i(A) \leq Q(i)$ where $Q(i) = \sum_{X\in Z_i} f(|X(A)|) \cdot \beta\sigma\gamma_{i+1}$.*

Proof. For any cluster $X \in Z_i$, we can partition the demands $X(A) = Y_1 \cup Y_2 \cup \ldots \cup Y_k$, where $Y_i \neq Y_j$, $i \neq j$, according to the leaders at level $i+1$ that they use, so that all demands in Y_i use the same leader in Z_{i+1}, and Y_i and Y_j use a different leader of Z_{i+1}. Next, we provide a bound on k.

Consider any two demands $d_1 = (s_1, t_1) \in X(A)$ and $d_2 = (s_2, t_2) \in X(A)$. Let ℓ_i be the leader of X. Since s_1 and s_2 are γ_i-satisfied by the cluster X of ℓ_i, they are both members of that cluster. Therefore, $\text{dist}(s_1, \ell_i) \leq \sigma\gamma_i$, and $\text{dist}(s_2, \ell_i) \leq \sigma\gamma_i$. Thus, $\text{dist}(s_1, s_2) \leq 2\sigma\gamma_i = \gamma_{i+1}/2$. Suppose that demand d_1 chooses leader ℓ_{i+1} at level $i+1$ with respective cluster X_{i+1}. Since s_1 is at least $\gamma_{i+1}/2$-satisfied in X_{i+1}, s_2 is a member of X_{i+1}. Since any node is a member of at most β clusters at level $i+1$, it has to be that the number of different level $i+1$ leaders at level $i+1$ that the demands in $X(A)$ select is bounded by β. Consequently, $k \leq \beta$.

Since f is subadditive and for any demand (s,t), $|p_i(s)| \leq \sigma\gamma_{i+1}$, $C_i(Y_j) \leq f(|Y_j|) \cdot \sigma\gamma_{i+1}$. Therefore, $C_i(X(A)) \leq \sum_{j=1}^{k} C_i(Y_j) \leq f(|X(A)|) \cdot \beta\sigma\gamma_{i+1}$. Which gives:
$C_i(A) \leq \sum_{X\in Z_i} f(|X(A)|) \cdot \beta\sigma\gamma_{i+1}$, as needed. □

Lemma 3.6.4. *For any* $0 \leq i \leq \kappa - 1$, $C_i(A) \leq C^*(A) \cdot 8\beta\sigma^2\chi$.

Proof. From Lemma 3.6.3, for any $0 \leq i \leq \kappa - 1$, $C_i(A) \leq Q(i)$. From Lemma 3.6.1, for any $2 \leq i \leq \kappa - 1$, $C^*(A) \geq R(i)/\chi$. Note that $Q(i) = R(i) \cdot 2\beta\sigma\chi\gamma_{i+1}/\gamma_i = R(i) \cdot 8\beta\sigma^2\chi$. Therefore, $C_i(A) \leq C^*(A) \cdot 8\beta\sigma^2\chi$. For $0 \leq i \leq 1$, we use the lower bound of Lemma 3.6.2, and we obtain $C_i(A) \leq C^*(A) \cdot \beta\sigma\gamma_2 = C^*(A) \cdot 4\beta\sigma^2$. □

We now give the central result of this analysis:

Theorem 3.6.5. *The oblivious approximation ratio of the algorithm is* $O(\log D)$.

Proof. Since the demand set A is arbitrary, from Lemma 3.6.4 and Equation 3.1 we obtain oblivious approximation ratio bounded by $8\kappa\beta\sigma^2\chi$. When we take into account the source path segments together with the destination path segments, the approximation ratio bound increases by a factor of 2, and it becomes $16\kappa\beta\sigma^2\chi$. Since, β, σ, χ, are constants and $\kappa = O(\log D)$, we obtain approximation ratio $O(\log D)$. □

With a more fine-tuned analysis where we separate the demands into ranges according to the distance between sources and destinations, we can obtain approximation ratio $O(\log n)$ as shown in the following analysis.

3.7 Analysis for $O(\log n)$-approximation

The following analysis provides a $O(\log n)$-approximation for the cost of paths due to our algorithm. In the lemma below, $P(G)$ is the set of paths returned by our algorithm.

Lemma 3.7.1. *For any demand* $d_k = (s_k, t_k) \in A$, *the length (number of edges) of a path* $p(d_k) \in P(G)$ *is at most* $74 \cdot \text{dist}(s_k, t_k)$.

Proof. Algorithm AuxiliaryPaths gives a set of paths between any pair of nodes in G. The auxiliary path $p(d_k)$ is built in a step-wise bottom-up fashion where there is a path $q(s_k)$ that connects s_k to ℓ' and ℓ' to t_k through a series of concatenated segments, where ℓ' is the leader of the common cluster $X \in Z_i$ that γ_i-satisfies s_k and $\frac{\gamma_i}{2}$-satisfies t_k, where level $i < \kappa$.

For every path segment $q_j(s_k)$ that connects a leader ℓ_j to ℓ_{j+1}, $0 < j < i$, there is a stretch of σ. Hence, the distance from s_k to ℓ' is $\sigma[\gamma_1 + \gamma_2 + \ldots + \gamma_i]$ and similarly, the distance from t_k to ℓ' is $\sigma[\gamma_1 + \gamma_2 + \ldots + \frac{\gamma_i}{2}]$. Hence, the total length, denoted by $|p(d_k)|$, from s_k to t_k will be:

$$\begin{aligned} |p(d_k)| &\leq 2\sigma[\gamma_1 + \gamma_2 + \ldots + \gamma_{i-1}] + \frac{3}{2} \cdot \sigma \cdot \gamma_i \qquad (3.2) \\ &= 2\sigma\left[1 + 4\sigma + (4\sigma)^2 + \ldots + (4\sigma)^{i-2}\right] + \frac{3}{2} \cdot \sigma \cdot \gamma_i \quad [\because \gamma_i = (4\sigma)^{i-1}] \\ &= 2\sigma\left[\frac{(4\sigma)^{i-1} - 1}{4\sigma - 1}\right] + \frac{3}{2} \cdot \sigma \cdot (4\sigma)^{i-1} \qquad (3.3) \end{aligned}$$

Let $\widehat{d}_k = \text{dist}(s_k, t_k)$. Since the common cluster is at level i and $\gamma_i \geq 2 \cdot \widehat{d}_k$, we have $(4\sigma)^{i-1} \geq 2 \cdot \widehat{d}_k$. Solving for $i - 1$, we get, $i - 1 \geq \lceil \log(2\widehat{d}_k - 4\sigma) \rceil = \lceil \log(2\widehat{d}_k - 96) \rceil$. Substituting the value of $i - 1$ in Eqn (3.3), we get:

$$\begin{aligned} |p(d_k)| &\leq 2\sigma\left[\frac{(4\sigma)^{\log(2\widehat{d}_k - 96)} - 1}{4\sigma - 1}\right] + \frac{3}{2} \cdot 24 \cdot (4\sigma)^{\log(2\widehat{d}_k - 96)} \\ &= 2 \cdot 24\left[\frac{(96)^{\log(2\widehat{d}_k - 96)} - 1}{96 - 1}\right] + \frac{3}{2} \cdot 24 \cdot (96)^{\log(2\widehat{d}_k - 96)} \\ &\leq \frac{1}{2}(96)^{\log(2\widehat{d}_k - 96)} - \frac{1}{2} + 36 \cdot (96)^{\log(2\widehat{d}_k - 96)} \\ &\leq 37 \cdot (96)^{\log(2\widehat{d}_k - 96)} \\ &= 37 \cdot (2^{\log 96})^{\frac{\log 2\widehat{d}_k}{\log 96}} = 37 \cdot (2)^{\log 2\widehat{d}_k} \\ &= 37 \cdot 2\widehat{d}_k \end{aligned}$$

Hence the result follows that $|p(d_k)| \leq 74 \cdot \text{dist}(s_k, t_k)$. □

The following corollary follows from Lemma 3.7.1.

Corollary 3.7.2. *The weight of any edge in $p(d_k) \in P(G)$ does not exceed $74 \cdot \widehat{d_k}$.*

Consider a demand set A_x, where $\forall\ d_k(s_k, t_k) \in A_x$, $2^x \leq \text{dist}(s_k, t_k) \leq (2^{x+\log n+1} - 1)$, where $x \in \mathbb{Z}^+$. Our algorithm on A_x will induce a set of paths $P(A_x)$ in G that uses edges of weights in three categories: $R_1(A_x)$, $R_2(A_x)$ and $R_3(A_x)$, defined as follows:

$$R_1(A_x) : e \in E \text{ such that } 1 \leq \mathfrak{w}_e \leq 2^x/n^2.$$
$$R_2(A_x) : e \in E \text{ such that } (2^x/n^2 + 1) \leq \mathfrak{w}_e \leq (74 \cdot (2^{x+\log n+1} - 1).$$
$$R_3(A_x) : e \in E \text{ such that } (74 \cdot (2^{x+\log n+1}) \leq \mathfrak{w}_e \leq D.$$

From Corollary 3.7.2, $R_3(A_x) = \emptyset$. Let $C(A_x)$ be the total cost of our algorithm. We can express $C(A_x) = C_1(A_x) + C_2(A_x) + C_3(A_x)$, where $C_i(A_x)$ is the cost incurred by using edges in $R_i(A_x)$, $1 \leq i \leq 3$. From Corollary 3.7.2, $C_3(A_x) = 0$, thus, $C(A_x) = C_1(A_x) + C_2(A_x)$.

Lemma 3.7.3. $C_1(A_x) \leq \frac{C^*(A_x)}{2}$.

Proof. Since for any demand $d_k \in A_x$, $2^x \leq \text{dist}(s_k, t_k)$, the optimal cost will be $C^*(A_x) \geq f(|A_x|) \cdot 2^x$, when all the demands in A_x merge and use the edge of the lowest cost in the range.

In the worst case, let all the demands use all the edges of maximum weight in $R_1(A_x)$, $2^x/n^2$. Since the number of edges in G is at most $n^2/2$, the resulting cost is given by $C_1(A_x) \leq f(|A_x|) \cdot \frac{2^x}{n^2} \cdot \frac{n^2}{2} \leq \frac{f(|A_x|) \cdot 2^x}{2}$. Since $C^*(A_x) \geq f(|A_x|) \cdot 2^x$, we get the relation $C_1(A_x) \leq \frac{C^*(A_x)}{2}$. □

Lemma 3.7.4. $C_2(A_x) = O(C^*(A_x) \log n)$.

Proof. When demands in A_x use edges in $R_2(A_x)$, it implies that the length of an edge in any of those demand paths ranges between $2^x/n^2 + 1$ and $74 \cdot (2^{x+\log n+1} - 1)$. Hence, we

consider those levels of clusters, used by our algorithm, whose diameter is in this range. All the lower level clusters are ignored. Let level i denote the lowest level cluster whose diameter is at least $2^x/n^2 + 1$ and let level j denote the highest level cluster whose diameter is at most $74 \cdot (2^{x+\log n+1} - 1)$.

For level i, we have $\sigma\gamma_i \geq \frac{2^x}{n^2} + 1$. Solving for $i-1$, we get, $i - 1 \geq \frac{\log(\frac{2^x+n^2}{24n^2})}{\log 96}$. Likewise, for level j, we have $j - 1 \leq \frac{\log(\frac{2^x \cdot n}{3})}{\log 96}$. The number of levels $j - i$ that have the edge weights in $R_2(A_x)$ is $\frac{\log(\frac{8n^3 2^x}{2^x+n^2})}{\log 96} = O(\log n)$.

Since there are $O(\log n)$ levels from i to j and since Lemma 3.6.4 shows that the cost of our algorithm at every level is bounded by $C^*(A_x) \cdot 8\beta\sigma^2\chi$, we have $C_2(A_x) \leq O(C^*(A_x) \log n)$. □

Lemma 3.7.5. $C_1(A_x) \leq C_2(A_x)$.

Proof. We know that $C(A_x) = C_1(A_x) + C_2(A_x)$. We also know that $C(A_x) \geq C^*(A_x)$. From Lemma 3.7.3, $C_1(A_x) \leq \frac{C^*(A_x)}{2}$. Therefore, to satisfy $C(A_x) \geq C^*(A_x)$, $C_2(A_x)$ must be at least $\frac{C^*(A_x)}{2}$. Hence, it follows that $C_1(A_x) \leq C_2(A_x)$. □

Consider an arbitrary set of demands A in G with diameter D, where the distances of the demand pairs will range from 1 to D. We group the demand sets consecutively as $A = \{A_{x_1}, A_{x_2}, A_{x_3}, \ldots, A_{x_m}\}$, where $x_i = i \cdot \log n$, $0 \leq i \leq \log D$. This set of demands forms a *sequence*. We will have four such sequences such that $A = \{A^1 \cup A^2 \cup A^3 \cup A^4\}$, where:

$$A^1 = \{A_{x_1}, A_{x_5}, A_{x_9}, \ldots\}.$$
$$A^2 = \{A_{x_2}, A_{x_6}, A_{x_{10}}, \ldots\}.$$
$$A^3 = \{A_{x_3}, A_{x_7}, A_{x_{11}}, \ldots\}.$$
$$A^4 = \{A_{x_4}, A_{x_8}, A_{x_{12}}, \ldots\}.$$

The edge-weight ranges that map from each of the above demand sequences are interleaved. Note that for any A_{x_i}, $A_{x_j} \in A^k$, $R(A_{x_i})$ is disjoint from $R(A_{x_j})$.

Theorem 3.7.6. *The oblivious approximation ratio of the algorithm is* $O(\log D)$.

Proof. Consider the demand sequence $A^1 \in A$. The following holds:

$$C(A^1) = C_1(A^1) + C_2(A^1) \tag{3.4}$$

which is the sum of the cost incurred from the edge-weights in $R_1(A^1)$ and $R_2(A^1)$.

For costs incurred from $R_2(A^1)$, the induced edge-weights can be summed as they are disjoint:

$$C_2(A^1) = C_2(A_{x_1}) + C_2(A_{x_5}) + C_2(A_{x_9}) + \dots$$

And for costs incurred from $R_1(A^1)$, the edges could merge and hence

$$\begin{aligned} C_1(A^1) &\leq C_1(A_{x_1}) + C_1(A_{x_5}) + C_2(A_{x_9}) + \dots \\ &\leq C_2(A_{x_1}) + C_2(A_{x_5}) + C_2(A_{x_9}) + \dots \qquad \text{(by Lemma 3.7.5)} \end{aligned} \tag{3.5}$$

Therefore, from Eqns (3.4) and (3.5), we get:

$$C(A^1) \leq 2 \cdot [C_2(A_{x_1}) + C_2(A_{x_5}) + C_2(A_{x_9}) + \dots] \tag{3.6}$$

Since the edge-weights in $R_2(A_x)$ are disjoint, the following expression holds:

$$C^*(A^1) \geq C_2^*(A^1) \geq C_2^*(A_{x_1}) + C_2^*(A_{x_5}) + C_2^*(A_{x_9}) + \dots \tag{3.7}$$

Simplifying Eqn (3.6), we get:

$$
\begin{aligned}
C(A^1) &\leq 2 \cdot [C_2(A_{x_1}) + C_2(A_{x_5}) + C_2(A_{x_9}) + \ldots] \\
&\leq 2 \cdot \mathcal{C} \cdot [C_2^*(A_{x_1}) + C_2^*(A_{x_5}) + C_2^*(A_{x_9}) + \ldots] \cdot \log n && \text{(by Lemma 3.7.4)} \\
&\leq 2 \cdot \mathcal{C} \cdot C_2^*(A^1) \cdot \log n && \text{(by Eqn (3.7))} \\
&\leq 2 \cdot \mathcal{C} \cdot C^*(A^1) \cdot \log n && (\because\ C^*(A^1) \geq C_2^*(A^1)) \qquad (3.8)
\end{aligned}
$$

where $\mathcal{C}$ is a constant in the above inequalities. The analysis for A^2, A^3 and A^4 is similar to that of A^1. Considering all the four sequences, the total cost for demand set A can be summed-up as follows:

$$
\begin{aligned}
C(A) &= C(A^1) + C(A^2) + C(A^3) + C(A^4) \\
&\leq 2 \cdot \mathcal{C} \cdot [C^*(A^1) + C^*(A^2) + C^*(A^3) + C^*(A^4] \cdot \log n && \text{(By Eqn (3.8))} \\
&\leq 2 \cdot \mathcal{C} \cdot 4 \cdot C^*(A) \cdot \log n \\
&\leq 8\mathcal{C} \cdot C^*(A) \cdot \log n
\end{aligned}
$$

Hence, it follows that $C(A) = O(C^*(A) \log n)$. □

We now give the central result of the paper. From Theorem 3.6.5 and Theorem 3.7.6, the following corollary is obtained:

Corollary 3.7.7. *The oblivious approximation ratio of the algorithm is* $O(\min(\log D, \log n))$.

3.8 Conclusions

We provide a set of paths for the multi-sink buy-at-bulk network design problem in planar graphs. Contrary to many related work where the source-destination pairs were already given, or when the source-set was given, we assumed the obliviousness of the set of source-destination pairs. Moreover, we considered an unknown fusion cost function at every edge of the graph.

We presented nontrivial upper and lower bounds for the cost of the set of paths. We have demonstrated that a simple, deterministic, polynomial-time algorithm based on sparse covers can provide a set of paths between all pairs of nodes in G that can accommodate any set of demands. We have shown that this algorithm guarantees $O(\min(\log n, \log D))$-approximation.

Chapter 4

Minor-Free Graphs

4.1 Overview

Minor-free graphs are those graphs that do not have K_5 or $K_{3,3}$ in them. A more detailed definition is given in 4.2.

In the oblivious buy-at-bulk network design problem for minor-free graphs, the task is to compute a fixed set of paths for every source node in the graph to the sink, such that any set of demands can be routed along their respective paths to the sink. The demands could be aggregated at intermediate edges where the fusion-cost is specified by a canonical (non-negative concave) function f. We give a novel algorithm for minor-free graphs which is oblivious with respect to the demands, and is also oblivious with respect to the fusion function f.

4.1.1 Problem Statement

Assume that we are given a weighted graph $G = \{V, E, w\}$, with edge weights $w : E \longrightarrow \mathbb{Z}^+$. We denote w_e to be the weight of edge e. Let $A = \{s_1, s_2, \ldots, s_r\}$ be a set of demand (source) nodes in G and $A \subseteq V$. Let $d_i = (s_i, s)$ be a unit of demand that induces an unplittable unit of flow along a path $p(d_i)$ from source node s_i to the sink node $s \in V$. It is possible that some paths may overlap. The flow of these demands forms a tree $T'_A \subseteq T$ where T is a spanning tree on G.

There is an arbitrary canonical function f at every edge where demand aggregates. This f is same for all the edges in G. Let $\varphi_e(A) = \{p(d_i) : e \in p(d_i)\}$ denote the set of paths 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 $C(A) = \sum_{e \in E} C_e(A)$.

For this set of demands A, the cost of the tree T'_A is $C(T'_A) = \sum_{e \in E} C_e(A)$. For the same demand set A, there is an optimal set of paths that induces a Steiner tree T_A with respective cost $C^*(T_A)$. The approximation ratio for the cost of these trees is defined as $\frac{C(T'_A)}{C^*(T_A)}$.

The SSBB optimization problem on input A is to find a tree $T'(A)$ that minimizes the approximation ratio. We note that SSBB is NP-Hard as the Steiner tree problem is its special case (when $f(x) = 1$ and when there is only one destination node) [SCRS00].

An *oblivious algorithm* $\mathcal{A}_{obl}$ for the SSBB problem, computes a fixed spanning tree, denoted T for all the sources in G to the sink s. Given any set of demands A, the path $p(d_i)$ for each $d_i = (s_i, s) \in A$, is a fixed path in T from s_i to s. This gives a set of paths T'_A to route the demands in A. We define the *approximation ratio* of $\mathcal{A}_{obl}$, as:

$$A.R.(\mathcal{A}_{obl}) = \max_A \frac{C(T'_A)}{C^*(T_A)}.$$

We aim to find algorithms that minimizes the above approximation ratio for any canonical function f which is unknown to the algorithm.

4.1.2 Contribution

We provide an oblivious algorithm for SSBB problems in minor-free graphs. Our algorithm is deterministic and computes in polynomial time a spanning tree that guarantees $O(2^{\sqrt{\log D}} \cdot \log^{3\sqrt{\log D}+4} n)$-approximation over the optimal cost for minor-free graphs G, where D is the diameter of the graph, n the total number of nodes, σ and χ are the stretch and chromatic number of G.

4.2 Definitions

4.2.1 H-Minor Free Graphs

The *contraction* of an edge $e = (u, v)$ in a graph G is the replacement of nodes u, v with a new vertex whose incident edges are the edges other than e that were incident to u or v. A graph H is a minor of G if H is a subgraph of a graph obtained by a series of edge contractions of G. A graph is *H-minor-free* if H is not a minor of G.

For example, it is well known that planar graphs are exactly all the graphs whose set of minors exclude $K_{3,3}$ and K_5. In other words, every minor-free family of graphs is contained in a $K_{r,r}$-free family. For instance, planar graphs are $K_{3,3}$-free. Furthermore, the classical Kuratowski-Wagner Theorem [Kur30, Wag37] states that a graph is planar if and only if it has no K_5 or $K_{3,3}$ minors. (For three different proofs of the theorem, see [Tho81]).

4.2.2 Partition

For a graph $G = (V, E)$, define a *cluster* $C(G)$ as a subset of vertices $C \subseteq V$. When the context is clear, we will use C to refer $C(G)$. Define a *connected cluster* C if there exists a path between all pairs of vertices of C.

A *strong diameter cluster* is one where the shortest path between any pair of nodes in C constitutes nodes that belong to C and the $\text{dist}(u, v) \leq d$, where $u, v \in C$ and d is the diameter of C. A *weak diameter cluster* is one where the shortest path between any pair of nodes in C contains nodes that do not belong to C and $\text{dist}(u, v) > d$.

A *partition* of a graph G produces clusters that are disjoint. A *strong partition* is one where all the clusters of that partition are strong diameter clusters. Similarly, a *weak partition* is one where all the clusters of the partition are weak diameter clusters.

Define a γ-partition with respect to a locality parameter γ as a partition of G that produces clusters whose diameter is γ. Define an (γ, k)-partition to be a partition of G whose clusters have diameter γ and that which can be colored in at least k colors.

4.2.3 Coloring

Define $N(C, r)$ as the set of clusters $\mathcal{C} = \{C^1, C^2, \ldots, C^\ell\}$ that are at most r-distance apart from any node $u \in C$. We shall call them r-neighborhood clusters of C. In other words, $N(C, r) = \{\mathcal{C} : \text{dist}(u, v) \leq r, \text{ where } u \in C, v \in C^i \in \mathcal{C}\}, 1 \leq i \leq \ell, C^i \neq C$.

The clusters formed by an r-partition can be colored with different colors. Those clusters that are less than r-distance apart ($N(C, r)$) will have colors different from the color of C. A *minimal* coloring of clusters w.r.t r is the process of coloring a set of clusters with minimal number of colors. This minimal number of colors needed for proper coloring of clusters is denoted by the chromatic number notation χ.

Lemma 4.2.1. *For any path p where $\text{len}(p) = y$, the number of clusters that p traverses through is at most $y/r \cdot x$, where $x = N(C, r)$ and $y > r$.*

Proof. There can be at most x differently colored clusters in every r-segment of any path p. Since there are y/r such segments, the total number of clusters is at most $x \cdot y/r$. □

4.2.4 Laminar Family

A family $\mathcal{L}$ of sets is laminar if for every $A, B \in \mathcal{L}$, either $A \cap B = \emptyset$, or $A \subseteq B$ or $B \subseteq A$. A laminar family can be represented by a tree structure with the leaves as the nodes of the graph. If $G_i = (V_i, E_i)$, $G_j = (V_j, E_j)$ and $G_k = (V_k, E_k)$ are any three clusters of G, and if $V_j \subset V_i$, and, $V_k \cap V_i = \emptyset$, then, component G_j resides completely within G_i and G_k is disjoint from G_i.

4.2.5 Hierarchical Partitioning

Hierarchical partitioning of $G = (V, E)$ is the process of recursively dividing V into disjoint clusters $V = \{C^1, C^2, \ldots, C^n\}$ such that $V = \bigcup C^i$, $1 \leq i \leq n$ and each $C^i \in V$ is a set of smaller clusters. Informally, by 'smaller' clusters, we mean clusters of smaller diameter w.r.t the next higher level cluster.

A *weak hierarchical partition* is a type of hierarchical partitioning of G where the resulting clusters formed during the partitioning are all weak diameter clusters. Building a tree with this type of partition will result in an overlay tree.

A *strong hierarchical partition* is a type of hierarchical partitioning of G where the resulting clusters formed during the partitioning are all strong diameter clusters. A strong hierarchical partition of G results in a laminar family of sets. Constructing a tree with this type of partition will result in a spanning tree.

The hierarchical partitioning of $G = (V, E)$ induces a laminar family $\mathcal{L}$. If T is the rooted construction tree whose nodes are sets in $\mathcal{L}$, and $C^i \in \mathcal{L}$ completely contains $C^j \in \mathcal{L}$ iff C^j is a cluster formed by the partition of cluster C^i. Observe that the tree T obtained by our hierarchical partition has the property that every $C' \subseteq \mathcal{L}$ corresponds to a subtree T' of T.

Given a diameter γ, the partitioning of G is a (σ, χ, r)-partition when the individual partitions of G has a stretch of σ, accepts a coloring of χ where χ is an assignment of colors to all the partitions and where each partition gets a unique color and the minimum distance between any two partitions is r. Similarly, a (σ, r)-partition when the individual partitions of G has a stretch of σ and the minimum distance between any two partitions is r.

Define *adjacent* clusters C^a and C^b as those where there is at least one edge connecting a node $u \in C^a$ to a node $v \in C^b$. Define *adjacent* leaders ℓ^a and ℓ^b if their respective clusters (C^a, C^b) are connected by at least one common edge.

4.3 Technique Used

We decompose the given minor-free graph G using path-separators and clustering based on a locality parameter γ. The recursive decomposition of G results in a set of non-laminar clusters for all $(\log D)$ levels of the hierarchical decomposition with exponentially increasing γ. For every cluster, a leader is chosen. We ensure that the clusters formed do not interfere with previously formed clusters. If so, the newer cluster will be merged with the interfering clusters to form an augmented cluster. This gives a set of non-laminar clusters $\mathbb{C}$. The algorithms and detailed description are explained in section 4.4.

We then construct a set of laminar clusters $\mathbb{C}^L$ from $\mathbb{C}$ for each level in the hierarchy. The construction of laminar clusters is done using the same algorithm used for our original decomposition of G. The adjacent clusters are connected with their respective leaders to form a graph H. The clusters at any particular level i are contracted and represented by their respective leaders. Edges are formed between the adjacent leaders to form a graph H' and where the edges between the leaders are made to be of unit weight. Based on the properties of H', we decompose H with specific locality parameter. This decomposition of H will result in laminar clusters for that particular level. Note that the resulting laminar clusters will have an augmented stretch. A more detailed description is given in section 4.5.

Once laminar clusters at all $O(\log D)$ levels are constructed, we construct a spanning tree T by the following method. We connect the adjacent clusters by an edge which is the shortest path between the leaders. This forms a graph H. We compute H', a tree of leaders by running a Breadth-First-Search on H that would result in a spanning tree T. This tree T naturally defines a sequence of paths from all the leaf nodes to the root. A thorough treatment of this process is given in subsection 4.6.1.

The computation of paths is done on the spanning tree T. For each leaf node, we choose the shortest path between successive leaders until the root is reached. Since, the paths in T

could have cycles, the choice of the shortest path is carefully done to avoid cycles by choosing a next higher-level leader that is in the direction of the root. A more detailed explanation is given in subsection 4.6.1.

4.4 Strong Partitioning in Minor-Free Graphs

In this section, we describe a technique to partition a given minor-free graph G using path-separators and clustering. We focus on decomposing a given component of a particular level.

We begin by introducing some terms and notations. Given a distance parameter γ and τ, any component H at a particular level will be decomposed in several stages as given in Decompose-Component (8) and Recursive-Decomposition (9). Each stage of decomposition will result in clusters of diameter $\gamma - 2j\tau$, where j denotes the stage and $0 \leq j \leq \log n - 1$. The clustering algorithm is detailed in 6 and 7. Repeated decomposition of components at a particular level i will result in a set of clusters $\mathbb{C}_i$ with their respective leaders $\mathbb{L}$. We run this decomposition process for all $\log n$ levels, which results in a set of clusters $\mathbb{C}$ as a complete partition of G. This set of clusters $\mathbb{C}$ is later used to construct a spanning tree T.

We start with an informal description of the decomposition algorithm:Decompose-Component which when given a component H of stage j and a distance parameter γ, decomposes H into a set of several strongly connected components H' that belong to the next stage $j+1$. The partitioning is done by removing path-separator $S = \{P_1, P_2, \ldots, P_\ell\}$ from H. For all P_i, each path-separator $q \in P_i$ is used to partition H using Decompose-Component and cluster using Partition-Algorithm with successively decreasing diameter γ.

Now, we describe the Partition-Algorithm. Given a set of clusterable paths Q and a locality parameter γ, the algorithm results in a set of strongly connected clusters C. Each path $\sigma \in Q$ is divided into subpaths $\sigma_j, \sigma_{j+1}, \ldots, \sigma_{j+k}$ where $\text{len}(\sigma_n) = \gamma$, $1 \leq n \leq (j+k-1)$ and $\text{len}(\sigma_{j+k}) \leq \gamma$. A set of nodes γ-distance around σ, is identified in G; $Z = \text{zone}(\gamma, \sigma)$.

For each subpath $\sigma_i \in \sigma$, the first node is chosen as a leader ℓ_i. A set of unclustered nodes $N_{uc} \in Z$ is identified to form clusters from. For the first leader ℓ_1, we choose those nodes that are at most γ-distance from ℓ_1 to form cluster C_1. When a node v chooses a leader, all the nodes in the path to the leader are also implicitly chosen to be in the same cluster. Also, every time a cluster is formed, N_{uc} is adjusted accordingly; $N_{uc} \leftarrow N_{uc} \setminus \bigcup_{1 \leq k \leq i} C_k$.

Assuming that leaders up to ℓ_i have been clustered, we try to form a cluster for ℓ_{i+1}. A set of nodes S from the adjusted N_{uc} are chosen such that $\forall v \in S$, $\text{dist}(v, \ell_{i+1}) \leq \gamma$ to form a cluster C_{i+1}. Note that a node $v \in N_{uc}$ will *prefer* to be associated with a leader ℓ_j where the index j is the least among those that satisfy $\text{dist}(v, \ell_j) \leq \gamma$. Such a preference rule will prevent the paths from nodes of different clusters to intersect. The process of clustering continues until all subpaths of σ are processed resulting in a set of clusters C.

During the clustering process, the coloring of the clusters are also taken care of. For each stage of the partition, we are given a palette of colors $\chi = \{\chi_1, \chi_2, \ldots, \chi_x\}$. For the i^{th}-cluster formed in a given stage, the color $\chi[i \bmod x]$ is assigned to it. During the formation of clusters, we observe that a currently formed cluster could possibly *interfere* with a previously formed cluster. We say that a new cluster C_a *interferes* with an old cluster C_b if at least one shortest path from a node $u \in C_b$ to its leader $\ell_b \in C_b$ goes through C_a, and the rest of the shortest paths (if any) would go through some of the newly formed clusters. We say that the shortest path is 'broken' by C_a. Note that if there is at least one shortest path in C_b whose nodes are not a member of C_a, then, we consider that the shortest path is not broken and that C_b is not interfered by C_a.

Based on the possibility of a previously formed cluster being interfered by a currently formed cluster, we now describe algorithm Augment-Clusters (5). This algorithm takes a list of currently formed clusters $\mathbb{C}_{curr}$ and a list of previously formed clusters $\mathbb{C}_{old}$ and outputs a list of newly formed clusters $\mathbb{C}_{new}$ by 'augmenting' any currently formed cluster $\overline{C} \in \mathbb{C}_{curr}$ that interferes with a previously formed cluster $C \in \mathbb{C}_{old}$. Define a shortest path p that starts from $u \in C$

and ends at its leader $\ell \in C$. For every currently formed cluster $\overline{C} \in \mathbb{C}_{curr}$, we compute the set of all nodes $A \in \mathbb{C}_{old}$ whose shortest path to their leader is broken by $\overline{C}$. For each node $u \in A$, if all shortest paths from u to its leader ℓ goes through some clusters in $\mathbb{C}_{curr}$ and at least one shortest path goes through $\overline{C}$, we augment $\overline{C}$ by including u to it to produce $\mathbb{C}_{new}$. This augmentation continues for all such $u \in A$ to result in $\mathbb{C}_{new}$. Note that if the computation of A results in an empty set, then, $\mathbb{C}_{curr}$ becomes $\mathbb{C}_{new}$.

We now describe the Path-Segment-Algorithm as follows. Given a component H, a path separator q, locality parameter γ and $\tau > 0$, the algorithm outputs a set of clusters $\mathbb{C}$ with their leaders $\mathbb{L}$ by identifying a set of clusterable segments $Q \in \sigma$ and calling Partition-Algorithm on Q.

We assume that the algorithm knows all the path-separators and their clusters it has seen so far. Let there be a path-separator q traversing from a to b across the given component H at some stage in the algorithm. For all $x \in H$, define a distance parameter d to be the shortest distance to a path-separator (except q) seen up to this point in the algorithm. We categorize the nodes in H into three types. Nodes of Type 1 satisfy the condition $d > \gamma + 2\tau$. Such nodes are unclustered nodes. Nodes of Type 2 satisfy $\gamma + \tau + 1 \leq d \leq \gamma + 2\tau$. And, finally, nodes of Type 3 satisfy $0 \leq d \leq \gamma + \tau$.

We traverse along q from a to b and identify segments that can be clustered, called 'clusterable segments'. The clusterable segments are essentially those segments that are a distance of $\tau+\gamma+1$ away from the nearest path separator. A detailed method of identifying such segments is given in Algorithm 7.

We now provide an overview of the decomposition of the given graph G. We describe two algorithms - Decompose-Component 8 and Recursive-Decomposition 9. The Decompose-Component algorithm partitions a given component H of graph G into a set of strongly-connected components H'. Given the parameters γ, τ, the set of old clusters $\mathbb{C}_{old}$ and a set of color palettes $X = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\log n}\}$, this algorithm uses a k-path separator $S = \{P_1 \cup P_2 \cup \ldots \cup P_\ell\}$ for

partitioning. The partitioning is done in a sequential order which is determined by the sequence of the path separators in S. At any given stage, a partition is made by removing the set of nodes that form a path separator P_i giving rise to a new connected-component. Furthermore, for each path $q \in P_i$, we identify the clusterable segments and cluster them with with diameter γ and a predefined color palette $X[j]$. All the individual clusters formed will have unique colors from this palette. The algorithm returns a set of connected components H', a set of clusters with their leaders and a modified γ parameter for the next stage of decomposition.

The algorithm Recursive-Decomposition, which works in a top-down fashion, splits a given component H into multiple sub-components. This is done by a call to Decompose-Component, which removes the given path-separator $S = \{P_1 \cup P_2 \cup \cdots \cup P_l\}$ of H and the corresponding clusters around it. This removal results in a set of components of the next stage, denoted by $H' = \{H_1^1, H_1^2, \ldots, H_1^x\}$, for an arbitrary $x > 0$ and a set of clusters $\mathbb{C}'$.

At the beginning, let $G = G \setminus s$, have the single component $H \in \mathbb{H}$. G is decomposed into several smaller components by Decompose-Component, the set of which is denoted by $G_1 = \{H_1^1, H_1^2, \ldots, H_1^x\}$. We continue decomposing the components of H'. Suppose, during the recursive decomposition process, we have decomposed G upto stage j. The components of $(j+1)^{th}$ stage are created by the following mechanism. We decompose each component $H_j \in G_j$ by calling Decompose-Component on each of them. This call generates $G_{j+1} = \{H_{j+1}^1, H_{j+1}^2, \ldots, H_{j+1}^x\}$. Note that it is assumed that a path-separator S is given for every component of every stage. This recursive decomposition process continues by acting on G_{j+1} [1]. This process is carried out recursively on the subsequent set of components H' until no new components are created, i.e., when the H' is a set of individual nodes. A formal description of this recursive process is given in Algorithm 9 (Recursive-Decomposition).

The result of this process provides a set of clusters and their leaders of all stages $0 \leq j \leq \log n - 1$. Algorithm 8 and 9 describes the decomposition in detail.

[1] In the Recursive-Decomposition algorithm, we do not show the subscript indicating the stages to avoid notational clutter.

Algorithm 5: Augment-Clusters($\mathbb{C}_{curr}, \mathbb{C}_{old}$)

Input: The current cluster set $\mathbb{C}_{curr}$ and the old cluster set $\mathbb{C}_{old}$
Output: Augmented cluster set $\mathbb{C}_{new}$

$\mathbb{C}_{new} \leftarrow \emptyset$;
foreach $\overline{C} \in \mathbb{C}_{curr}$ **do**
    `// All nodes` $\in \mathbb{C}_{old}$ `whose shortest path to their leader is broken by` $\overline{C}$
    $A = \{u \mid u \notin \overline{C} \wedge u \in C \in \mathbb{C}_{old} \wedge \exists v \in \overline{C} \wedge v \in p(u, \ell \in C)\}$
    if $A \neq \emptyset$ **then**
        foreach $u \in A$ **do**
            if *All shortest paths from* u *to its leader* $\ell \in C$ *goes through some clusters in* $\mathbb{C}_{curr}$ *and at least one shortest path goes through* $\overline{C}$ **then**
                $\overline{C} \leftarrow \overline{C} \cup u$; `// Augmentation step`
            end
        end
    end
    $\mathbb{C}_{new} \leftarrow \mathbb{C}_{new} \cup \overline{C}$;
end
return $\mathbb{C}_{new}$;

Lemma 4.4.1. *There are* $\log n$ *stages in the one-level recursive decomposition of* G.

Proof. A path-separator $p \in P_i$ will divide any component G into two connected components H_1 and H_2 whose sizes are at most $n/2$ each. Since the decomposition of G recursively progresses until it comes to a stage where the component size shrinks to 1, it is clear that the number of stages it takes for G to shrink from size n to 1 is $\log n$. □

Lemma 4.4.2. *The maximum diameter of a cluster is* $O(\gamma \log n)$.

Proof. Let $\Delta(C)$ denote the diameter of a new cluster and Δ_{old} be the diameter of an immediately old cluster. The expression for diameter of a cluster is given by $\Delta(C) = \gamma + \Delta_{old}$. A new cluster C will have a diameter of γ initially. If C interferes with any old cluster, due to augmentation process, the diameter of C will increase by up to $\Delta_{old}(= \gamma + 2\tau)$. Likewise, Δ_{old} could have been an augmented cluster, whose diameter would have been increased by $\gamma + 4\tau$. Since there are $\log n$ stages at a particular level, the maximum diameter that a cluster could have would be $\sum_{j=0}^{\log n - 1} \gamma + (2\tau)j = \gamma + (\gamma + 2\tau) + (\gamma + 4\tau) + (\gamma + 6\tau) + \ldots + (\gamma + 2(\log n - 1)\tau) \leq \gamma \log n$. □

Algorithm 6: Partition-Algorithm($G, Q, \gamma, \mathbb{C}_{old}, \mathcal{P}$)

Input: G, set of segments Q, parameter $\gamma > 0$, set of old clusters $\mathbb{C}_{old}$ and a color palette $\mathcal{P} = \{color_0, color_1, \ldots, color_{m-1}\}$.

Output: A set of strongly connected clusters $\mathbb{C}_{new}$ and respective leaders $\mathbb{L}_{new}$.

$\mathbb{C}_{curr} \leftarrow \emptyset$; // Set of strong partitions

$\mathbb{L} \leftarrow \emptyset$; // Set of leaders of the partitions/clusters

$j = 1$; // Beginning index of subpaths

foreach $\sigma \in Q$ **do**

  Partition σ into subpaths $\sigma_j, \sigma_{j+1}, \ldots, \sigma_{j+k}$ where $\text{len}(\sigma_n) = \gamma$, $1 \leq n \leq (j + k - 1)$ and $\text{len}(\sigma_{j+k}) \leq \gamma$;

  $Z \leftarrow \text{zone}(\gamma, \sigma)$;

  for $i = j$ **to** $j + k$ **do**

    Let $color[i]$ denote i^{th} color in the palette $\mathcal{P}$;

    $\ell_i \leftarrow$ first node in σ_i ; // Choose a leader for each segment

    $\mathbb{L} \leftarrow \mathbb{L} \cup \ell_i$; // Preferentially ordered list of leaders

    $N_{uc} = \{v \mid v \in Z \setminus \mathbb{C}_{curr}\}$; // Set of remaining unclustered nodes in Z

    while $N_{uc} \neq \emptyset$ **do**

      $C = \{v \mid v \in N_{uc} \ \wedge (\ell_k \in \mathbb{L}$ is the leader with smallest index k such that it has the shortest distance to $v\}$;

      $\mathbb{C}_{curr} \leftarrow \mathbb{C}_{curr} \cup C$; // ℓ_k is the leader of C

      $color(C) \leftarrow color[i \ mod \ |\mathcal{P}|]$;

      $N_{uc} \leftarrow N_{uc} \setminus C$;

    end

  end

  $j \leftarrow j + k + 1$;

end

$\mathbb{C}_{new} \leftarrow$ Augment-Clusters($\mathbb{C}_{curr}, \mathbb{C}_{old}$);

return $\mathbb{C}_{new}, \mathbb{L}$; // $\mathbb{C} = \{C_1, C_2, \ldots, C_x\}$, $L = \{\ell_1, \ell_2, \ldots, \ell_x\}$

Algorithm 7: Path-Segment-Algorithm($G, H, q, \gamma, \tau, \mathbb{C}_{old}$)

Input: H, shortest path $q \in G$, parameter $\gamma > 0$ and $\tau > 0$.

Output: A set of clusters with their leaders for a given path-separator.

```
1  𝕃 ← ∅ ;                                                    // Set of leaders
2  ℂ ← ∅ ;                                                    // Set of clusters
3  Q ← ∅ ;                                          // set of clusterable segments of q
   // For all x ∈ H, let d be the shortest distance to a path-separator
      (except q) seen up to this point in the algorithm.
   // Type 1:  If d > γ + 2τ, x is an unclustered node.
   // Type 2:  If γ + τ + 1 ≤ d ≤ γ + 2τ, x is a clustered node of Type 2.
   // Type 3:  If 0 ≤ d ≤ γ + τ, x is a clustered node of Type 3.
   // Let q span from point a to point b.  We traverse one node at a time
      along q from a till end point b is reached.
4  Let x be a point on q during the traversal;
5  while b is not reached do
6      if a is of Type 1 then
7          while x is Type 1 or 2 do
8              Continue traversing q;
9          end
10         if x is of Type 3 then
11             Let x' be the node immediately preceding x;
12             Let σ be the segment from a to x';
13             Q ← Q ∪ σ;
14             a ← x;
15         end
16     end
17     if a is of Type 2 then
18         if x is of Type 1 then
19             a ← x;
20         end
21         while x is Type 2 do
22             Continue traversing q;
23         end
24     end
25     if a is of Type 3 then
26         while x is of Type 3 or 2 do
27             Continue traversing q;
28         end
29         if x is of Type 1 then
               // Ignore the segment from a to x
30             a ← x;
31         end
32     end
33 end
34 ℂ, 𝕃 ← Partition-Algorithm(G, Q, γ, ℂ_old);
35 return 𝕃, ℂ;
```

Algorithm 8: Decompose-Component$(G, H, \gamma, \tau, \mathbb{C}_{old}, X)$

Input: G, component H that is k-path separable and parameter γ, set of old clusters $\mathbb{C}_{old}$ and set of color palettes $X = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\log n}\}$.

Output: A set of strongly connected sub-components H', clusters $\mathbb{C}$ and modified γ.

```
// This algorithm works on only one component
// base case
```
if *H consists of a single vertex v* **then**
  $H' \leftarrow \{v\}$; **return** H';
end
```
// main case
```
Let $S = P_1 \cup P_2 \cup \cdots \cup P_\ell$ be a k-path separator of H;
$j = 0$;
foreach $P_i \in S$ **do**
  $H \leftarrow H - \bigcup_{1 \le i \le i-1} P_i$;
  foreach $q \in P_i$ **do**
    $L, \mathbb{C} \leftarrow$ `Path-Segment-Algorithm`$(G, H, q, \gamma, \tau, \mathbb{C}_{old}, X[j])$;
    $\gamma \leftarrow \gamma - 2\tau$;
    $j \leftarrow j + 1$ `// Move on to next color palette`
  end
end
$H' \leftarrow H - S$; `// H' is a set of strongly-connected residual components`
return $H', \mathbb{C}, \gamma$

Algorithm 9: Recursive-Decomposition$(G, \mathbb{H}, \gamma, \tau, \mathbb{C} = \emptyset, X)$

Input: G, a set of strongly-connected components $\mathbb{H}$ that excludes sink s, a set of palettes X.

Output: A set of clusters $\mathbb{C}$ of varying sizes of different stages of a level.

```
// This algorithm works on only one level, with successively decreasing
   cluster size for successive stages.
```
foreach *component $H \in \mathbb{H}$* **do**
  $H', \mathbb{C}', \gamma \leftarrow$ `Decompose-Component`$(G, H, \gamma, \tau, \mathbb{C}, X)$;
  $\mathbb{C} \leftarrow \mathbb{C} \cup \mathbb{C}'$;
  if *elements of H' are not individual nodes* **then**
    `// Next stage`
    $\mathbb{C} \leftarrow$ `Recursive-Decomposition`$(G, H', \gamma, \tau, \mathbb{C}, X)$;
  end
end
return $\mathbb{C}$;

Lemma 4.4.3. *The maximum number of colors in a color palette $\mathcal{P}$ is $2\log n + 1$.*

Proof. Assume that a path q is partitioned and clustered by the Partition-Algorithm with a locality parameter γ. The initial diameter of a partitioned cluster will be 2γ. From Lemma 4.4.2, the maximum diameter that this cluster can have after augmentation would be $O(\gamma \log n)$. If two clusters C_a and C_b on q has to have the same color, then, they have to be at least γ-distance apart. Considering both of them to have the worst-case diameter of $O(\gamma \log n)$, the distance between them must be at least $2\gamma \log n + \gamma$ to share the same color. Since this distance is on the γ-partitioned path q, the number of unique colors in a palette would be $\frac{2\gamma \log n + \gamma}{\gamma} = 2\log n + 1$. □

Lemma 4.4.4. *The total number of color palettes needed to color the entire graph G is $O(\log n)$.*

Proof. Let us consider a component H at a particular level and stage of decomposition. For a given stage, the decomposition will require a constant number of colors and since there are $O(\log n)$ stages of decomposition in a level (from Lemma 4.4.1), we would require a palette of $O(\log n)$ colors. Since the decomposition of G requires $\log n$ levels, it follows that we would require $O(\log n)$ palettes. □

Lemma 4.4.5. *The total number of colors is $\chi = O(\log^2 n)$.*

Proof. From Lemma 4.4.4, a total of $\log n$ color palettes are needed to color the partitions of a given graph G. Since each palette has $O(\log n)$ colors (from Lemma 4.4.3), it follows that a total of $O(\log^2 n)$ colors are required to color the partitions of G. □

Lemma 4.4.6. *The minimum distance between two clusters at any given level is 2τ, where $\tau = \Omega(\gamma/\log n)$.*

Proof. At any level i, the algorithm Decompose-Component decomposes a given component H into several sub-components. During decomposition, clusters are formed around each path

$q \in P_i$ such that their diameters are $\gamma - 2\tau$ and also that the clusters do not have any nodes that is within a distance of τ from P_i. Each cluster that is formed is at least τ-distance away from a path separator P_i. Hence, it follows that two clusters at the same level i will be at least 2τ distance away from each other. □

Lemma 4.4.7. *Given G and the distance parameter γ, the* Recursive-Decomposition *algorithm results in a $(O(\log n), O(\log^2 n), \Omega(1/\log n))$-partition of G.*

Proof. From Lemma 4.4.2, Lemma 4.4.5 and Lemma 4.4.6, the value of maximum diameter of a partitioned cluster is $(O(\log n)$, the number of colors assigned for each partition of G is $O(\log^2 n)$ and the minimum distance between two clusters is a factor of $\Omega(1/\log n))$. Hence, the partitioning of G using Recursive-Decomposition algorithm for all levels results in a $(O(\log n), O(\log^2 n), \Omega(1/\log n))$-partition. □

4.5 Construction of Laminar Clusters

To build a spanning tree, we need a hierarchical distribution of the clusters formed by the Recursive-Decomposition algorithm. Given a set of non-laminar clusters at level $(i-k)$, we construct a set of laminar clusters $\mathbb{C}_i^L$. The construction of laminar clusters is described as follows.

Define *iteration* as one step in the process of forming a laminar cluster. The laminar clusters are formed in a sequence of iterations, $1 \leq j \leq \frac{\log D}{k}$. Each iteration involves clusters of k levels. In other words, to form an i^{th}-level laminar cluster C_i^a, the Decompose-Component algorithm is used with a clustering parameter of $(\chi+1)\cdot 2^k$. Since each step of the iteration jumps k levels, the total number of iterations needed to form the laminar clustering for G with diameter $\log D$, is $\frac{\log D}{k}$. The locality parameter used for successive levels increases by a factor of 2. Hence, for every k levels, $\frac{\gamma_i}{\gamma_{i-k}} = 2^k$.

Assume that the set of m non-laminar level-$(i-k)$ clusters are $\{C_{i-k}^1, C_{i-k}^2, \ldots, C_{i-k}^m\}$ which

are formed by the Decompose-Component algorithm. Each cluster C^j_{i-k}, $1 \leq j \leq m$ is contracted and represented by a single leader node. This contraction process results in a new set of leaders $\mathcal{L}_{i-k} = \{\ell^1_{i-k}, \ell^2_{i-k}, \ldots, \ell^m_{i-k}\}$, where $\ell^j_{i-k} \in \mathcal{L}_{i-k}$ is the leader of C^j_{i-k}. Recall that $(\ell^i_{i-k}, \ell^j_{i-k}) \in \mathcal{L}_{i-k}$ are *adjacent* leaders if their respective clusters (C^i_{i-k}, C^j_{i-k}) are connected by at least one common edge. In $\mathcal{L}_{i-k}$, those leaders that are *adjacent* are connected to each other by a shortest path in G and where the path is contained in the respective clusters. The set of connections between adjacent leaders in $\mathcal{L}_{i-k}$ produces a connected graph. Denote this resulting graph as H. Note that the contraction process will not change its "minor-free" property and the resulting contracted graph H will still be a minor-free graph.

Lemma 4.5.1. *If a shortest path in H has at least $\chi + 1$ leaders, then, its length is $\Omega(\frac{\gamma_{i-k}}{\log n})$.*

Proof. A color palette that has χ colors can uniquely color χ leaders in the shortest path. From pigeonhole principle, there will be at least two leaders in the path that will be assigned the same color if the path has more than χ leaders. In that case, every $(\chi+1)^{th}$ leader will share the same color. From Lemma 4.4.6, the minimum distance between any two clusters of same color is $\Omega(\frac{\gamma_{i-k}}{\log n})$, which suggests that there will be at least $\chi + 1$ leaders between the clusters. Hence, the length of the shortest path will be $\Omega(\frac{\gamma_{i-k}}{\log n})$. □

Given $\frac{\gamma_i}{\gamma_{i-k}} = 2^k$, the following corollary is provided.

Corollary 4.5.2. *If a shortest path has at least $(\chi + 1) \cdot 2^k$ leaders, then, its length is at least $\frac{\gamma_i}{\log n}$.*

We now construct another graph H' from H by assigning the weights of all the edges of H to unit distance. Algorithm Decompose-Component with a clustering parameter of $\gamma' = (\chi+1) \cdot 2^k$ is applied on H' to create laminar clusters. We analyze H' and prove the following properties. The same properties (with appropriately modified parameter values) are later proved in H.

Lemma 4.5.3. *The diameter of a cluster $C^j_{i-k} \in H'$ is $O((\chi + 1) \cdot 2^k \cdot \sigma)$.*

Proof. When H' is decomposed with a clustering parameter of $\gamma' = (\chi+1)\cdot 2^k$, it would result in clusters of diameter (from Lemma 4.4.2) at most $O(\gamma' \cdot \log n) = O((\chi+1)\cdot 2^k \cdot \log n)$, where $\sigma = \log n$. □

Lemma 4.5.4. *The minimum distance between any two leaders of same color is $2\tau'$ where $\tau' = \Omega(\frac{\gamma'}{\log n})$.*

Proof. When H' is decomposed with a clustering parameter of $\gamma' = (\chi+1)\cdot 2^k$, from Lemma 4.4.6, the minimum distance between two leaders of the same color is $\Omega(\frac{(\chi+1)\cdot 2^k}{\log n})$. □

Lemma 4.5.5. *Given H' and the distance parameter γ', the* Decompose-Component *algorithm results in a $(O(\log n)), \Omega(1/\log n))$-partition of G.*

Proof. From Lemma 4.5.3 and Lemma 4.5.4, the value of maximum diameter of a laminar cluster in H' is $(O(\log n)$ and the minimum distance between two clusters that share the same color is a factor of $\Omega(1/\log n))$. Hence, the partitioning of H' using Decompose-Component algorithm results in a $(O(\log n), \Omega(1/\log n))$-partition. □

We now cluster H using Decompose-Component algorithm with a clustering parameter of $(\chi+1)\cdot 2^k$. Define $\widehat{\sigma}_i$ to be the accumulated stretch of an arbitrary cluster $C_i^a \in H$. At every iteration, a new stretch of $(\chi+1)\cdot 2^k \cdot \sigma$ is introduced by the clustering algorithm.

Lemma 4.5.6. *The maximum diameter of a cluster $C_i^j \in H$ is $O((\chi+1)\cdot\sigma\cdot\widehat{\sigma}_{i-k}\cdot\gamma_i)$.*

Proof. Since the clustering parameter is $\gamma = (\chi+1)\cdot 2^k$, the diameter of each resulting cluster, from Lemma 4.4.2, would be $\gamma\cdot\sigma$. Since, each cluster $C_i^j \in H$ has a prior diameter of $\widehat{\sigma}_{i-k}\cdot\gamma_{i-k}$ and $\frac{\gamma_i}{\gamma_{i-k}} = 2^k$, the new, extended diameter of C_i^j would be $(\chi+1)\cdot 2^k\cdot\sigma\cdot\widehat{\sigma}_{i-k}\cdot\gamma_{i-k} = O((\chi+1)\cdot\sigma\cdot\widehat{\sigma}_{i-k}\cdot\gamma_i)$. □

Lemma 4.5.7. *The minimum distance between any two clusters at level i that share the same color in H is $\Omega(\frac{(\chi+1)\cdot 2^k\cdot\gamma_{i-k}}{\log n})$.*

Proof. From Lemma 4.4.6 and since $\gamma' = (\chi + 1) \cdot 2^k$ it follows that the minimum distance is $\Omega(\frac{(\chi+1)\cdot 2^k \cdot \gamma_{i-k}}{\log n})$, where γ_{i-k} is the prior diameter of the clusters. □

Lemma 4.5.8. *The accumulated stretch of a laminar cluster up to iteration j is $\widehat{\sigma}_j$, $1 \leq j \leq \frac{\log D}{k}$.*

Proof. For each iteration, the stretch of a cluster increases by a factor of $(\chi+1) \cdot 2^k \cdot \sigma$. Let the accumulated stretch for a cluster at the end of iteration $(j-1)$ be $\widehat{\sigma}_{j-1}$. Since the diameter of any cluster before the iteration begins is 1, the stretch is 1. For the trivial case, after iteration $j = 1$, the diameter of a cluster (From Lemma 4.5.6) would be $[(\chi+1) \cdot 2^k \cdot \sigma] \cdot \widehat{\sigma}_0 \cdot \gamma_0 = [(\chi+1) \cdot \sigma] \cdot \gamma_1$, where $\widehat{\sigma}_0 = 1$ and $\gamma_0 \cdot 2^k = \gamma_1$. Hence, the accumulated stretch of a cluster just after the first iteration is $\widehat{\sigma}_1 = [(\chi + 1) \cdot \sigma]$. After the second iteration, the diameter of a cluster would be $[(\chi + 1) \cdot 2^k \cdot \sigma] \cdot \widehat{\sigma}_1 \cdot \gamma_1 = [(\chi + 1) \cdot 2^k \cdot \sigma)] \cdot [(\chi + 1) \cdot \sigma)] \cdot \gamma_1 = [(\chi + 1) \cdot \sigma]^2 \cdot \gamma_2$. Therefore, the accumulated stretch of a cluster just after iteration $j = 2$ is $\widehat{\sigma}_2 = [(\chi + 1) \cdot \sigma]^2$.

At the end of any iteration j, $0 \leq j \leq \frac{\log D}{k}$, since the diameter is $O((\chi + 1) \cdot \sigma \cdot \widehat{\sigma}_j \cdot \gamma_i)$, the accumulated stretch can be deduced to be $(\chi + 1) \cdot \sigma \cdot \widehat{\sigma}_{j-1} = \widehat{\sigma}_j = [(\chi + 1) \cdot \sigma]^j$. □

4.6 Spanning Tree Construction

We describe the construction of a spanning tree T from a laminar set of clusters $\mathbb{C}$. The Spanning-Tree algorithm considers a set of laminar clusters at any level i to give a spanning tree T_i. The construction is as follows. For the level $i = 0$, the trivial case, all the clusters are individual nodes. Hence, the tree will be a node.

Assuming that we have built the spanning tree up to a level $i-k$ and we would like to build the tree for level i. For every C_{i-k} in the given C_i, we recursively build spanning trees T_{i-k} to get a set $\mathbb{T}_{i-k}$ of such spanning trees. Recall that *adjacent* clusters as those where there is at least one edge connecting a node from one cluster to a node in the other. Based on this, we develop a graph H by connecting the leaders of $\mathbb{C}_{i-k}$ with edges if the corresponding clusters

are adjacent. The weights of these edges would be the shortest path distances between the leaders. Once H is developed, we compute H', a tree of leaders of level $i-k$, by running a Breadth-First-Search on H. This tree H' provides a 'zoomed-out' view of the real-spanning tree T.

From H', we now identify those real edges E that connect different clusters. Once identified, the union of $\mathbb{T}_{i-k}$ and E provides the spanning tree T_i. We now explain the method to identify the set of edges E in H'. Let us consider any pair of adjacent leaders ℓ_a and ℓ_b that belongs to clusters C^a_{i-k} and C^b_{i-k} respectively. Let p be the shortest path in $(C^a_{i-k} \cup C^b_{i-k})$ that connects ℓ_a to ℓ_b. We include this path p in E. We repeat this process of identifying those edges for all pairs of adjacent clusters in H' to get E. Once E is computed, the union of $\mathbb{T}_{i-k}$ with E will provide a single large spanning tree T_i.

Algorithm 10: Spanning-Tree($\mathbb{C}^L_i$, k)

Input: A laminar set of clusters $\mathbb{C}^L_i$, $k > 0$.
Output: A spanning tree T_i, where T_i is the set of edges in G.

Let $\mathbb{C}_{i-k}$ be the set of laminar clusters at level $i-k$.
if $i > k$ **then**
    $\mathbb{T}_{i-k} \leftarrow \emptyset$; `// $\mathbb{T}_{i-k}$ is the set of spanning trees`
    foreach $C_{i-k} \in \mathbb{C}^L_i$ **do**
        $T_{i-k} \leftarrow$ `Spanning-Tree`(C_{i-k});
        $\mathbb{T}_{i-k} \leftarrow \mathbb{T}_{i-k} \cup T_{i-k}$
    end
    $H \leftarrow$ Graph obtained by connecting the leaders of $\mathbb{C}_{i-k}$ with edges if the corresponding clusters are adjacent. Weights of the edges are the shortest path distances between the leaders;
    $H' \leftarrow$ Compute Breadth-First-Search tree on H ; `// $H'$ is a tree of leaders`
    foreach *pair of adjacent leaders* $(\ell_a \in C^a_{i-k}, \ell_b \in C^b_{i-k}) \in H'$, $\forall a, b \in |\mathbb{C}_{i-k}|$ **do**
        Let p be a shortest path in $(C^a_{i-k} \cup C^b_{i-k})$ that connects ℓ_a and ℓ_b;
        $E \leftarrow E \cup p$;
    end
    $T_i \leftarrow \mathbb{T}_{i-k} \cup E$;
    return T_i;
else if $i \leq k$ **then**
    `// All clusters are individual nodes`
    return $v \in \mathbb{C}^L_i$;

For any cluster X, let $X(A)$ denote the set of demands with source in X whose paths leave from the leader of X toward the leaders of a higher level cluster.

We give the following trivial upper and lower bound for the special case where $0 \leq i \leq 1$, which follows directly from the observation that each demand needs to form a path of length at least 1 unit.

Lemma 4.6.1. *The number of clusters along the shortest path p from any leader ℓ_{i-k} to ℓ_i is $J_k = 2^k \cdot \widehat{\sigma}_i \cdot \chi = 2^{k+\log(\widehat{\sigma}_i \chi)}$.*

Proof. The shortest path p from a leader ℓ_{i-k} to ℓ_i will be of length $\widehat{\sigma}_i \gamma_i$. Consider a segment of length γ_{i-k} in p. This segment will have χ clusters. Since there are $\widehat{\sigma}_i \cdot \gamma_i / \gamma_{i-k}$ segments in p, the total number of clusters along the path p will be $\widehat{\sigma}_i \cdot \frac{\gamma_i}{\gamma_{i-k}} \cdot \chi = 2^k \widehat{\sigma}_i \chi$. Hence, the total number of clusters that needs to be passed through is given by $2^k \widehat{\sigma}_i \chi = 2^{k+\log(\widehat{\sigma}_i \chi)}$. □

Lemma 4.6.2. *For any level $0 \leq i \leq \log n - 1$, $C_i(A) \leq n \cdot f(2^i)$, where n is the number of nodes and $f(2^i)$ is the diameter of C_{i+1}.*

Proof. The diameter of T_i, is given by $\Delta(T_i) = J_k \Delta(T_{i-k}) + \widehat{\sigma}_i \gamma_i$, where J_k is the number of clusters at level $(i-k)$. If we consider demands to traverse a maximum of k levels to reach its leader, the expression for diameter can be rewritten as: $\Delta(T_i) = J_k \Delta(T_{i-k})$.

From Lemma 4.6.1, we observe that

$$\begin{aligned} \Delta(T_i) &= J_k \Delta(T_{i-k}) \\ &= J_k^{i/k} \Delta(T_1) \\ &= 2^{(k+\log \widehat{\sigma}_i \chi)(i/k)} \cdot 1 \end{aligned}$$

□

4.6.1 Computation of Paths

To compute the set of paths, we consider the spanning tree T. All the leaf nodes of T have a unique path to the root r by following a sequence of leaders all the way to r. Consider one such path from a leaf node u to r, denoted by $p(u, r)$. Consider any two successive leaders ℓ_1 and ℓ_2 in $p(u, r) \in T$. A shortest path $p(\ell_1, \ell_2)$ between ℓ_1 and ℓ_2 is chosen from the spanning tree T. Note that this new path segment $p(\ell_1, \ell_2)$ could possibly be shorter and contained in the shortest path of the spanning tree T. This is because, in T, there could be redundant paths that originates from a node u to its leader ℓ_1 and the path from ℓ_1 to its leader ℓ_2 could go through u, forming a loop. The concatenation of shortest paths $p(\ell_i, \ell_{i+1})$, from when ℓ_i is a leaf node until $\ell_{i+1} = r$, gives $p'(u, r)$. In a similar fashion, we choose the shortest paths from all the leaf nodes of T to r, which gives the required set of paths $P = \bigcup_{u \in L} p'(u, r)$, where L is the set of leaf nodes of T.

For analysis sake, we do not consider the strict version of the paths P and instead we use the paths in the spanning tree T that could possibly be longer with redundant paths. Hence, the result of our analysis on the upper bound will be pessimistic, i.e., worse than what the analysis on P would provide.

4.6.2 Competitive Ratio

Let A denote an arbitrary set of demands. Let $C^*(A)$ denote the cost of optimal paths in A to the root. Let $C(A)$ denote the cost of the paths given by our algorithm. 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 levels. For any cluster X, let $X(A)$ denote the set of demands with source in X whose paths leave from the leader of X toward the leader of a higher level cluster.

Lemma 4.6.3. *The cost of aggregation from level $i = 0$ to $i = 1$ is $C_1(A) \leq Q(1)$ where*

$Q(1) = n \cdot f(2^1)$.

Proof. In this base case, demands from all the n sources has to traverse at most a distance of 2^1 to reach their leaders. Hence, it follows that the cost would be at most $n \cdot f(2^1)$. □

Lemma 4.6.4. *The cost of aggregation from level $i = 0$ to $i = 1$ is $C_1^*(A) \geq (n-1)$.*

Proof. The unit demands from each of the n nodes would have to traverse at least a distance of 1 unit to reach a neighbor node. The nodes at level $i = 0$ form a spanning tree rooted at r with $n-1$ edges. Hence, the total cost of sending the demands would be at least $(n-1) \cdot 1$. □

Lemma 4.6.5. *The optimal cost of aggregation at any level $1 \leq i \leq \log n - 1$ is $C_i^*(A) \geq \frac{n}{\chi} \cdot 2^{i-k}$.*

Proof. Consider a cluster $C_{i-k}^a \in \mathbb{C}_{i-k}$ with leader ℓ_{i-k}^a. Consider a demand $(s, r) \in C_{i-k}^a$ where $s \in C_{i-k}$ and root r is at a level i or higher. The least cost can be achieved when the demand at s traverses to the next leader at a distance of at least 2^{i-k}. If we consider such aggregation for each of the n/χ clusters, the optimal cost would be at least $\frac{n}{\chi} \cdot 2^{i-k}$. □

Lemma 4.6.6. *The approximation ratio for the cost of aggregation between level $i-k$ and i is $2^{\log \chi} \cdot 2^{k + i\frac{\log(\widehat{\sigma}_j \chi)}{k}}$, where $i = jk$.*

Proof. Let iteration j result in an accumulated stretch of $\widehat{\sigma}_j$ at level i.

$$\begin{aligned}
\frac{C(i)}{C^*(i)} &\leq \frac{n \cdot \Delta(T_i)}{\frac{n}{\chi} \cdot 2^{i-k}} \\
&= \frac{\chi \cdot \Delta(T_i)}{2^{i-k}} \\
&= \chi \cdot 2^{(k + \log \widehat{\sigma}_j \chi)(i/k) - i + k} \\
&= \chi \cdot 2^{i(\frac{\log \widehat{\sigma}_j \chi}{k}) + k} && \text{(From Lemma 4.6.2)} \\
&= 2^{\log \chi} \cdot 2^{i(\frac{\log \widehat{\sigma}_j \chi}{k}) + k} \\
&= 2^{\log \chi + (\log \widehat{\sigma}_j \chi) i/k + k}
\end{aligned}$$

Substituting $k = \sqrt{\log D}$ above, the approximation ratio is $2^{\log \chi} \cdot 2^{\sqrt{\log D} + i\frac{\log(\widehat{\sigma}_j \chi)}{\sqrt{\log D}}}$. □

Lemma 4.6.7. *At iteration j, $C_j(A) \leq f(|X(A)|) \cdot \widehat{\sigma}_j \cdot 2^{jk}$, where $|X(A)|$ is the number of sources and jk is the number of levels demand traverses to reach the next leader.*

Proof. Consider a cluster X at level $i = jk$. If $X(A)$ is the demand generated from the sources in X, the demands have to traverse to the next higher level cluster leader. The demands have to traverse at least a distance of $\widehat{\sigma}_j \cdot 2^{jk}$ in the spanning tree T to reach the next level leader, where 2^{jk} is the diameter at level i and $\widehat{\sigma}_j$ is the accumulated stretch resulting from the laminar construction.

Since $C(A) = \sum_{e \in E} C_j(A)$, the total cost incurred for all the $\log D/k$ iterations will be $C(A) \leq \sum_{j=1}^{\frac{\log D}{k}} C_j(A)$. □

Lemma 4.6.8. *For any cluster X at level i, $2 \leq i \leq \log D - 1$, $C^*(A) \geq R(i)/\chi$, where $R(i) = \sum_{X \in Z_i} f(|X(A)|) \cdot (\gamma_{i-k}/2 \log n)$, where Z_i is the set of clusters at level i.*

Proof. Let $Z_i(c)$ be the set of clusters at level i which receive color $c \in [1, \chi]$. Consider a cluster $X \in Z_i(c)$. Consider a demand $(s, t) \in \chi(A)$. From the cluster construction algorithm, the minimum distance between two clusters is $2\tau = 2\Omega(\gamma_{i-k}/\log n)$. Consider the subpaths from $X(A)$ of length up to $\gamma_{i-k}/\log n$. In the best case, these subpaths from $X(A)$ may be combined to produce a path with the smallest possible total cost $\Omega(f(|X(A)|) \cdot \gamma_{i-k}/\log n)$. For any two nodes $u \in X(A)$ and $v \in Y(A)$, where $X, Y \in Z_i(c)$ and $X \neq Y$, $\text{dist}(u, v) > 2\gamma_{i-k}/\log n$. The subpaths of lengths up to $2\gamma_{i-k}/\log n$ cannot combine. Consequently, $C^*(A) \geq R(i, c)$ where $R(i, c) = \sum_{X \in Z_i(c)} f(|X(A)|) \cdot 2\gamma_{i-k}/\log n$. Let $R_{max} = max_{c \in [1,\chi} R(i, c)$. We have that $C^*(A) \geq R_{max}$. Since $R(j) = \sum_{c=1}^{\chi} R(i, c) \leq R_{max} \cdot \chi$. We obtain $C^*(A) \geq R(i)/\chi$, as needed. □

We also get the following trivial lower bound for the special case where $0 \leq i \leq 1$, which follows directly from the observation that each demand needs to form a path with length at

least 1.

Lemma 4.6.9. *For any Z_i, $0 \leq i \leq 1$, $C^*(A) \geq \sum_{X \in Z_i} f(|X(A)|)$.*

Lemma 4.6.10. $\frac{C(A)}{C^*(A)} \leq O(2^{\sqrt{\log D}} \cdot \log^{2\sqrt{\log D}+3} n)$.

Proof.

$$\begin{aligned}
\frac{C(A)}{C^*(A)} &\leq \sum_{j=1}^{\frac{\log D}{k}} \frac{f(|X(A)|) \cdot \widehat{\sigma}_j \cdot 2^{jk}}{\frac{f(|X(A)|) \cdot \frac{2\gamma_{jk-k}}{\log n}}{\chi}} && \text{(By Lemmas 4.6.7, 4.6.8 and 4.5.8)} \\
&\leq \sum_{j=1}^{\frac{\log D}{k}} \frac{\widehat{\sigma}_j \cdot 2^{jk} \cdot \chi}{\frac{2\gamma_{jk-k}}{\log n}} \\
&\leq \sum_{j=1}^{\frac{\log D}{k}} [(\chi+1) \cdot \sigma]^j \cdot 2^{k-1} \cdot \chi \cdot \log n && (\because\ \widehat{\sigma}_j = [(\chi+1) \cdot \sigma]^j) \\
&\leq \sum_{j=1}^{\frac{\log D}{k}} \chi^{j+1} \cdot \log^{j+2} n \cdot 2^k \\
&\leq \sum_{j=1}^{\frac{\log D}{k}} 2^k \cdot \log^{3j+4} n && (\because\ \chi = O(\log^2 n))
\end{aligned}$$

When $k = \sqrt{\log D}$, the competitive ratio is $\frac{C(A)}{C^*(A)} \leq O(2^{\sqrt{\log D}} \cdot \log^{3\sqrt{\log D}+4} n)$. □

The following corollaries follows from 4.6.10.

Corollary 4.6.11. *If $D << n$, the approximation ratio of our algorithm is $O(\log^{3\sqrt{\log D}+4} n)$.*

Corollary 4.6.12. *If $D >> n$, the approximation ratio of our algorithm is $O(2^{\sqrt{\log D}})$.*

4.7 Conclusions

We provide a set of paths for the single-sink buy-at-bulk network design problem in minor-free graphs. The spanning tree and the resulting set of paths was computed with the assumption

that the source-destination pairs and fusion-cost functions at every edge were unknown. We presented nontrivial upper bound for the cost of the set of paths. We have demonstrated that a simple, deterministic, polynomial-time algorithm based on sparse covers can provide a set of paths between all nodes in G to root r. We have shown that this algorithm guarantees $O(2^{\sqrt{\log D}} \cdot \log^{3\sqrt{\log D}+4} n)$-approximation.

Chapter 5

Conclusions and Outlook

In this dissertation, we proposed algorithms for constructing efficient transportation structures for a variety of problems including data communication, VLSI circuitry, transportation and logistics, publish/subscribe systems, distributed paging and for oil/gas pipelines. In such distributed networks, constructing efficient, near-optimal communication structure that is oblivious to the number and location of the sources and to the fusion cost function is very important. Our proposed algorithms are centralized algorithms that are simple, deterministic and provide a polynomial-time approximation guarantees for a variety of scenarios such as Single-Sink Buy-at-Bulk (SSBB) on Doubling-Dimension Graphs where we provide a $(\log^3 D)$-approximation, Multi-Sink Buy-at-Bulk (MSBB) for Planar Graphs where we guarantee $O(\min(\log n, \log D))$-approximation and MSBB for Minor-Free graphs where we provide a $O(2^{\sqrt{\log D}} \cdot \log^{3\sqrt{\log D}+4} n)$-approximation over the optimal cost.

Network design problems have been studied in the past using a variety of approaches. Some of them include greedy algorithms, primal-dual approaches, iterative routing (a polyhedral approach), randomization, metric embedding techniques, matching-based augmentation and more.

In the recent years, metric embedding techniques have been used widely for network design problems. Y. Bartal's work and subsequent improvements on embedding a general metric space into distributions over tree metrics have seen widespread use. The central idea is that given any metric space, a tree metric can be randomly generated such that distances in the original metric space are closely approximated by the expected distances in the computed random tree. Many optimization problems on general metrics can be reduced to tree metrics using this technique and are often much simpler to solve. Network Design algorithms have also been widely studied

in Operations Research under the name of "Discrete Network Optimization". Many network flow problems have been optimized using improvements over linear programming and related models for minimization. This research work takes a step further by developing deterministic algorithms as against randomized schemes and still provide close approximations.

5.1 Outlook

There are various real-world applications of network design. An emerging class of applications that has gained focus in Cloud Computing. More specifically, the datacenter networks that are at the core of cloud computing. Massive datacenters have been installed around the globe by Google, Microsoft and Amazon (EC2) and have already rolled out a variety of offerings using their datacenters and related middleware.

The datacenter networks that interconnect various geographically distributed datacenters are designed with multiple overlays in such a fashion that they are highly-available and fault-tolerant. Furthermore, distributed systems such as these are also designed for extreme low-latency. Such requirements need efficient design of networks and related overlays. With the computing and storage increasingly moving from PC-like clients to large Internet services, most applications and services are being offered by Web applications. This shift toward server-side computing has provided plenty of advantages to vendors. To provide efficient and cost-effective services (IaaS - Infrastructure as a Service, Saas - Software as a Service etc), the vendors aim at optimizing all aspects of their offerings - network design, fault-tolerance, high-availability, low-latency, consistency, network partition tolerance etc. Among these, network design is an important component that companies such as Google and Amazon focus heavily and rely on robust hardware that are well-tested with various configurations.

Datacenters are geographically distributed (possibly among several continents). There have been many studies on how to cluster the servers, route data efficiently among the datacenters and within the datacenters. Such problems are central to the operational efficiency of the sys-

tems. Oblivious network design is central to addressing such problems in a variety of ways. For example, sparse-cover and related partitioning techniques can be applied to efficient clustering of servers within a datacenter (for private clouds) and among datacenters. Since all these can be readily mapped to planar graphs, our solution to find a set of paths for any given set of demand pairs can be used.

Distributed version of our algorithms can be more useful in the datacenter scenario. It would be nice to have localized algorithms running on separate datacenters to organize themselves automatically and configure themselves for optimal performance for any kind of traffic demands. This could be a very promising area of future study. Likewise, in datacenters, one has to address the storage hierarchy that needs to quantify say latency, bandwidth and capacity characteristics of a large-scale distributed storage system.

Another emerging area that has its core in efficient network design is in big-data analytics. Today, we see an explosive growth in data and one needs to mine it properly and analyze it to make sense of it and predict different parameters. Such tasks calls for large-scale distributed systems that can accept chunks of data and process them in parallel to provide very fast, near-real-time analytics. To distribute the streaming incoming data (as chunks) to several thousand nodes is a non-trivial task. The process of splitting a data set into smaller fragments (shards) and distributing them across a large number machines is hard from both theoretical and engineering perspective. The problems such as how large or small should the shards be, which machines to load, where the machines are, how quickly can they loaded etc are crucial to the generation of revenue by the vendors. The sharding policy can vary depending on space constraints and performance considerations. Moreover, one must know in advance which nodes to load, which nodes are lightly loaded and how to ensure fault-tolerance. Principles from oblivious network design, facility-location and k-Median problems come in handy in solving such issues.

To summarize, highly scalable architectures are maturing and have reached a stage where

more challenges abound - real-time coordination among various geographically distributed datacenters, power, performance (throughput, latency, resiliency), privacy etc. This involves rigorous theoretical studies that further extend and improve the current optimizations provided by randomized algorithms and heuristics. The merger of strong theoretical analysis with smart engineering skills have always helped companies outlast their competition. A promising research agenda for the future calls for studies that not only involve pure theory that are highlighted here and in related literature, but also, system design and practical engineering skills that are related to real-world applications with real, pressing needs.

Bibliography

[AA97] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *FOCS '97*, page 542, Washington, DC, USA, 1997. IEEE Computer Society.

[AGGM06] Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. In *ICDCS '06*, page 75, Washington, DC, USA, 2006.

[Bar94] Y. Bartal. Competitive analysis of distributed online problems - distributed paging. In *Ph.D. Dissertation*, 1994.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *STOC '98*, pages 161–168, New York, NY, USA, 1998. ACM.

[BLT07] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Improved sparse covers for graphs excluding a fixed minor. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 61–70, New York, NY, USA, 2007. ACM.

[Bor26] O. Boruvka. O jistm problému minimélním. In *Práca Moravské Přírodověcké Společnosti* 3 , *(in Czech)*, pages 37–58, 1926.

[CHKS06] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design. In *FOCS '06*, pages 677–686, Washington, DC, USA, 2006.

[FGNW06] Stefan Funke, Leonidas Guibas, An Nguyen, and Yusu Wang. Distance-sensitive information brokerage in sensor networks. In *DCOSS 2006*, volume 4026 of *LNCS*, pages 234–251, San Francisco, USA, 2006. IEEE, Springer.

[FLL06] Pierre Fraigniaud, Emmanuelle Lebhar, and Zvi Lotker. A doubling dimension threshold $\theta(\log \log n)$ for augmented graph navigability. In *ESA, LNCS 4168*, pages 376–386. Springer, 2006.

[Fra07] Pierre Fraigniaud. The inframetric model for the internet. Technical report, 2007.

[GE03] Ashish Goel and Deborah Estrin. Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk. In *SODA '03*, pages 499–505, Philadelphia, PA, USA, 2003. SIAM.

[GGMZ09] Jie Gao, Leonidas J. Guibas, Nikola Milosavljevic, and Dengpan Zhou. Distributed resource management and matching in sensor networks. In *Proc. of (IPSN'09)*, pages 97–108, April 2009.

[GHR06] Anupam Gupta, Mohammad T. Hajiaghayi, and Harald Räcke. Oblivious network design. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 970–979, New York, NY, USA, 2006. ACM.

[GJ77] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[GK10] Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, In Press, Corrected Proof:–, 2010.

[GKL03] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS '03*, page 534, Washington, DC, USA, 2003. IEEE Computer Society.

[GMM01] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single sink edge installation problems. In *STOC '01*, pages 383–388, New York, NY, USA, 2001. ACM.

[GP09] Ashish Goel and Ian Post. An oblivious o(1)-approximation for single source buy-at-bulk. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:442–450, 2009.

[GP10] Ashish Goel and Ian Post. One tree suffices: A simultaneous o(1)-approximation for single-sink buy-at-bulk. *Foundations of Computer Science, Annual IEEE Symposium on*, 2010.

[GR10] Fabrizio Grandoni and Thomas Rothvoss. Network Design via Core Detouring for Problems Without a Core. In *ICALP*, pages 490–502, 2010.

[HSS08] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[JLN+05] Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 386–395, New York, NY, USA, 2005. ACM.

[JNRS06] Lujun Jia, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Gist: Group-independent spanning tree for data aggregation in dense sensor networks. In *DCOSS*, pages 282–304, 2006.

[KMW05] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. On the locality of bounded growth. In *PODC '05*, pages 60–68, New York, NY, USA, 2005. ACM.

[KRX08] Goran Konjevod, Andréa W. Richa, and Donglin Xia. Dynamic routing and location services in metrics of low doubling dimension. In *DISC '08*, pages 379–393. Springer-Verlag, 2008.

[KSW09] Jon Kleinberg, Aleksandrs Slivkins, and Tom Wexler. Triangulation and embedding using small sets of beacons. *J. ACM*, 56(6):1–37, 2009.

[Kur30] C. Kuratowski. Sur le probleme des courbes gauches en topologie. In Fundamenta Mathematicae, pages 15: 271–283, 1930.

[MP94] Y. Mansour and D. Peleg. An approximation algorithm for minimum-cost network design. Technical report, Jerusalem, Israel, Israel, 1994.

[Nie06] T. Nieberg. *Independent and Dominating Sets in Wireless Communication Graphs*. PhD thesis, University of Twente, Zwolle, April 2006.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, Philadelphia, PA, USA, 2000.

[PP06] Sriram V. Pemmaraju and Imran A. Pirwani. Energy conservation via domatic partitions. In *MobiHoc 2006*, pages 143–154, New York, NY, USA, 2006. ACM.

[SCRS00] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM J. on Optimization*, 11(3):595–610, 2000.

[Tal02] Kunal Talwar. The single-sink buy-at-bulk lp has constant integrality gap. In *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 475–486, London, UK, 2002. Springer-Verlag.

[Tho81] C. Thomassen. Kuratowski's theorem. In Journal of Graph Theory, pages 5:225–241, 1981.

[Vo6] Stefan Voß. Steiner tree problems in telecommunications. In Mauricio G. C. Resende and Panos M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 459–492. Springer US, 2006.

[Wag37] K. Wagner. Über eine eigenschaft der ebenen komplexe. In Mathematische Annalen, pages 114(1): 570–590, 1937.

Vita

Srivathsan Srinivasagopalan was born in Madras (now, Chennai), India, in 1974. He obtained his bachelor's degree in computer science and engineering in 1997 from SRM Engineering College, Madras. He received his master's degree in computer science from University of Texas, Dallas in 1999. He has more than seven years of work experience in various telecommunication startup companies taking on roles as software design engineer, test engineer and team-lead (all for wireless, optical, ethernet - networking startup companies). During his doctoral studies at Louisiana State University, he has co-authored ten refereed conference papers, two journal publications (one more in review) and several invited talks.

His research interests are in network algorithms, communication graphs and approximation schemes. He gravitates naturally to well-motivated problems where theoretical challenges abound and those that have immediate practical applications.