Demand-Aware Plane Spanners of Bounded Degree

Esra Ceylan Klaus-Tycho Foerster Stefan Schmid Katsiaryna Zaitsava
Faculty of Computer Science, University of Vienna, Austria

***Abstract*—Plane spanners of bounded degree are efficient communication backbones for networks. However, while existing spanners provide attractive guarantees in the worst-case, they are *demand-oblivious* and may hence be suboptimal under specific traffic demands. This paper thus initiates the study of *demand-aware* plane spanners of bounded degree, geometric spanners whose topology accounts for the actual communication traffic. We show that demand-awareness can significantly reduce the distance travelled per bit, and present a spanner which exploits topological flexibilities to account for the demand, without losing desirable guarantees of demand-oblivious spanners, namely constant stretch and degree. We complement our analytical results with heuristic improvements and a simulation study exploring the benefits of demand-awareness under realistic traffic traces.**

I. Introduction

Plane and geometric spanners are fundamental building blocks of efficient communication networks. For example, such spanners enable efficient unicast, multicast, and/or broadcast operations in wireless ad-hoc and sensor networks [1]. For these applications, it is usually desirable that these spanners minimize the stretch factor: they should approximately preserve distances among the nodes in the network. Furthermore, typically the spanners should be planar and of bounded degree for scalability reasons. Conceptually similar challenges and trends also exist when provisioning fiber-optic paths in wide-area networks [2], in inter-satellite networks [3], and in mixed ground-satellite networks [4], [5].

This paper is motivated by the observation that existing plane spanners may provide suboptimal communication topologies because they are *demand-oblivious*: all communication pairs are treated equally, independently of their actual communication demand (e.g., the number of transferred bits). In practice, however, communication patterns typically feature much spatial structure, and some communication pairs may communicate more frequently than others. This may be exploited. For example, a sink in a sensor network may have to collect measurement data from all other nodes, or only communicate with nodes in their geographic proximity for event detection. Given these specific patterns, it can be beneficial to optimize spanners in a *demand-aware* manner: rather than optimizing for *all* communication pairs equally, the spanner optimizes for the communication pairs relatively to their demand. We ask the following questions:

- Given a set of nodes distributed arbitrarily on the Euclidean plane with pairwise communication demands, how to design a spanner of bounded degree and low stretch accounting for the demands?
- How much benefit do these designs yield over demand-oblivious approaches under realistic traffic traces?
- What further optimizations can we obtain by heuristic means and by dropping the planarity requirement?

Research supported by European Research Council (ERC), grant agreement No. 864228 (AdjustNet), 2020-2025

ISBN 978-3-903176-39-3

A. Motivation

We highlight the advantages of demand-aware geometric spanners in a small example. Consider the four nodes v_1, v_2, v_3, v_4 and their placement in the plane in Fig. 1.

When each node can create three connections, subject to planarity, the rhombus design on the left side of Fig. 1 is optimal in the demand-oblivious model in which the maximal stretch needs to be minimized: the shortest distance in the network divided by the direct distance, for any two nodes. As a matter of fact, the only meaningful design choice is whether to connect v_1 with v_3 or v_2 with v_4. For the first pair, the detour via v_2 or v_4 is much longer than the direct connection via e_1, whereas for the second pair of v_2 and v_4, the detour via v_1 or v_3 is small in comparison to e_2.

However, if demands are known, it can be beneficial to reconfigure and adapt the network design. For example, consider the scenario where large amounts of traffic need to be routed between v_2 and v_4, dwarfing demands between v_1 and v_3. Now, it is much more advantageous to directly connect v_2 and v_4 via e_2, as shown on the right side of Fig. 1. By scaling the demand, the absolute cost difference between the two designs can be made arbitrarily large.

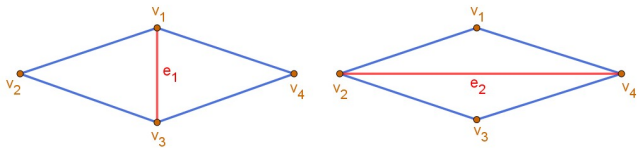


Fig. 1: Example to illustrate the benefit of demand-aware optimizations. Each of the four nodes $v_1, \ldots, v_4$ is capable of creating three (non-crossing) direct connections. The left design is optimal under demand-obliviousness, minimizing stretch, whereas the right design is optimal for traffic matrices where the demand between v_2 and v_4 dominates.

B. Contributions

This paper introduces the notion of demand-aware bounded degree plane spanners, geometric spanners whose topology accounts for the actual communication demands. We show that demand-awareness can significantly reduce the distance travelled per bit, and present a spanner which exploits topological flexibilities to account for the demand, without losing the desirable guarantees of demand-oblivious spanners, namely constant stretch and degree.

We complement our analytical results with an extensive simulation study exploring the benefits of demand-awareness under realistic traffic traces. Moreover, we show the optimal demand-aware design problem to be NP-hard and provide an integer program formulation. We also present efficient link exchange heuristics, that yield improved demand-aware designs, at the cost of increased computation time and stretch.

C. Organization

Our paper is structured as follows. We first provide a formal model in §II, followed by a detailed integer program formulation in §III, where we also prove the problem to be NP-hard. Next in §IV, we present our polynomial-time demand-aware network design algorithm with guaranteed stretch and degree bounds. Afterwards in §V, we describe our link exchange heuristics, followed by extensive simulations in §VI. Lastly, we discuss related work in §VII, concluding in §VIII.

II. Model

We are given a set of nodes embedded arbitrarily in the two-dimensional Euclidean plane, where every node can in principle connect directly to any other node. Thus, the set of nodes with all *possible* links forms a complete graph K_n, where each of the n participating nodes is represented by a vertex $v_j \in V$, $j \in \{1, \ldots, n\}$. In this graph K_n, the weight of the edge connecting two nodes $v_j, v_k \in V$ is given by the Euclidean distance $|v_j, v_k|$. The communication pattern of this network is described as an $n \times n$ matrix D with non-negative real entries. That is, an element D_{jk} of this matrix represents the communication demand between two nodes v_j and v_k, therefore we will refer to D as the *demand matrix*. In this paper we will assume the communication between these nodes to be bidirectional (undirected).

Our aim is to design a new network by extracting a (planar) subgraph $G = (V, E)$ of K_n which minimizes the total routing costs. Due to physical limitations and for scalability, the resulting graph is required to have a bounded degree. In order to compare various graphs, we define the function $pl_G(v_j, v_k)$, which returns the length of the shortest path contained in G between the nodes v_j and v_k. We want to minimize the sum of the lengths of the shortest paths between the nodes, weighted by their demand, i.e., $\min \sum_{j,k=1}^{n} pl_G(v_j, v_k) \cdot D_{jk}$.

We further introduce the *stretch factor* $t_{jk} \in \mathbb{R}_{\geq 1}$: for two distinct nodes v_j, v_k, the stretch factor is defined as the ratio between the length of the shortest path in a connected graph and the Euclidean distance [6], i.e. $t_{jk} := pl_G(v_j, v_k)/|v_j, v_k|$. The stretch factor t of a graph is the maximum stretch factor of two distinct nodes in the graph, i.e. $t := \max_{j \neq k} t_{jk}$. As a t-spanner we will denote a connected graph with stretch factor t. Geometrically, the stretch factor specifies by how much the Euclidean distance is increased compared to the length of the shortest path in K_n, i.e., the Euclidean distance.

We will assume that the given set of nodes are in general position; in particular, the slope of the line connecting two arbitrary points of G cannot be 0 or $\pm\pi/3$. In the further course this will ensure that the distance from two arbitrary nodes v_1, v_2 to v with regard to a defined metric is not equal. Otherwise ties can be broken arbitrarily by ordering points that have the same distance from v, e.g. using a counterclockwise ordering (see also [1]).

III. Optimal Demand-Aware Spanners

An optimal demand-aware spanner can be computed using an integer program formulation. In the following, we present such a program in detail. While the resulting runtime may be superpolynomial, we will subsequently show that this is unavoidable if one desires optimal solutions, as the problem is NP-hard. In the next section, we will then present a polynomial-time approximation algorithm.

A. Mixed Integer Program Formulation

Constants: Given a network topology with n nodes in the two-dimensional Euclidean plane, x_j resp. y_j represents the x- resp. y-coordinate of a node v_j and $D_{jk} \in \mathbb{R}_0^+$ is the demand between two nodes $v_j, v_k \in V$. With

$$\mathit{dist}(v_j, v_k) = |v_j, v_k| = \sqrt{(x_j - x_k)^2 + (y_j - y_k)^2}$$

we will denote the Euclidean distance between two nodes v_j, v_k. Furthermore, *max_degree* is the upper bound for the number of edges incident to a vertex.

Variables: The boolean $\mathit{exist}(v_j, v_k)$ is set to 1 if the edge between the nodes v_j and v_k exists in our graph and we set the boolean z_{jk}^{sr} to 1 if the edge between the nodes v_j and v_k is used in the path from v_s to v_r.

Objective: Our goal is to minimize the length of the shortest path for each communicating pair according to their demands.

$$\min \sum_{j,k=1}^{n} \Big(pl_G(v_j, v_k) \cdot D_{jk} \Big)$$

Constraints: Edges must exist for use in communication:

$$z_{jk}^{sr} \leq \mathit{exist}(v_j, v_k) \quad \forall j, k, s, r$$

Moreover, we are considering an undirected graph.

$$\mathit{exist}(v_j, v_k) = \mathit{exist}(v_k, v_j) \quad \forall j, k$$

Path length: The length of the path between nodes s, r is the sum of the length of every edge traversed along the path.

$$pl_G(v_s, v_r) = \sum_{j,k=1}^{n} \Big(z_{jk}^{sr} \cdot \mathit{dist}(v_j, v_k) \Big) \quad \forall s, r$$

Bounded degree: In our model the maximum degree of the graph is bounded by *max_degree*, hence the degree of every node needs to be bounded by *max_degree*.

$$\sum_{k=1}^{n} \mathit{exist}(v_j, v_k) \leq \mathit{max\_degree} \quad \forall j$$

Flow conservation: A flow that enters a node must leave it, with the exception of the start and end nodes.

$$\sum_{j=1}^{n} \left(z_{jk}^{sr} - z_{kj}^{sr} \right) = \begin{cases} -1, & \text{if } k = s \\ 1, & \text{if } k = r \\ 0, & \text{otherwise} \end{cases} \quad \forall k, s, r$$

Connectivity: To obtain a connected graph at least one edge has to be used in the path between two nodes s and r.

$$1 \leq \sum_{j,k=1}^{n} z_{jk}^{sr} \quad \forall s, r$$

Planarity: For this constraint we need to define some new constants. We can determine if two edges (v_j, v_k), (v_l, v_m) are parallel or not with the constant

$$p_{jklm} := (y_k - y_j) \cdot (x_m - x_l) - (x_k - x_j) \cdot (y_m - y_l).$$

The edges (v_j, v_k), (v_l, v_m) are parallel iff $p_{jklm} = 0$. Let $\lambda_{jklm}, \mu_{jklm} \in \mathbb{R}$ be defined as

$$\lambda_{jklm} := \begin{cases} \frac{(x_j - x_l)\cdot(y_m - y_l) - (y_j - y_l)\cdot(x_m - x_l)}{p_{jklm}}, & \text{if } p_{jklm} \neq 0 \\ 0, & \text{if } p_{jklm} = 0 \end{cases}$$

$$\mu_{jklm} := \begin{cases} \frac{(x_j - x_l)\cdot(y_k - y_j) - (y_j - y_l)\cdot(x_k - x_j)}{p_{jklm}}, & \text{if } p_{jklm} \neq 0 \\ 0, & \text{if } p_{jklm} = 0 \end{cases}$$

Two edges (v_j, v_k), (v_l, v_m) are parallel or do not intersect iff $\lambda_{jklm} \notin (0, 1)$ or $\mu_{jklm} \notin (0, 1)$. Additionally, we need a large enough constant $C > 0$. Next, we want to model the following disjunctive constraint for planarity

$$\lambda_{jklm} \leq 0 \vee \lambda_{jklm} \geq 1 \vee \mu_{jklm} \leq 0 \vee \mu_{jklm} \geq 1.$$

To achieve this we will define the boolean variables a_{jklm}^1, a_{jklm}^2 for the two cases of λ_{jklm} and b_{jklm}^1, b_{jklm}^2 for the two cases of μ.

The following constraints should hold for all j, k, l, m. For λ_{jklm} and μ_{jklm} at most one of the two cases can occur.

$$a_{jklm}^1 + a_{jklm}^2 \leq 1, \qquad b_{jklm}^1 + b_{jklm}^2 \leq 1$$

If two edges do not exist, λ_{jklm} and μ_{jklm} do not matter.

$$0 \leq \big(\textit{exist}(v_j, v_k) + \textit{exist}(v_l, v_k)\big) - \big(a_{jklm}^1 + a_{jklm}^2\big)$$
$$0 \leq \big(\textit{exist}(v_j, v_k) + \textit{exist}(v_l, v_k)\big) - \big(b_{jklm}^1 + b_{jklm}^2\big)$$

If both edges exist, one case of λ_{jklm} or μ_{jklm} has to occur.

$$\big(\textit{exist}(v_j, v_k) + \textit{exist}(v_l, v_k)\big) - \big(a_{jklm}^1 + a_{jklm}^2 + b_{jklm}^1 + b_{jklm}^2\big) \leq 1$$

Then, λ_{jklm} or μ_{jklm} should be at most 0 or at least 1.

$$\lambda_{jklm} \leq C \cdot (1 - a_{jklm}^1), \quad 1 - C \cdot (1 - a_{jklm}^2) \leq \lambda_{jklm}$$
$$\mu_{jklm} \leq C \cdot (1 - b_{jklm}^1), \quad 1 - C \cdot (1 - b_{jklm}^2) \leq \mu_{jklm}$$

B. NP-Hardness

We can show the problem of designing geometric demand-aware networks to be NP-hard, by reduction from the NP-hard Euclidean Traveling Salesman Problem in the path variant [7]. Our proof relies on minimizing the routing distance between the path's endpoints, while enforcing all-to-all connectivity. This initial construction relies on small degrees of one respectively two, which we then extend by means of additional nodes and demands to any fixed degree bound of four or larger.

We defer the proof details due to space constraints.

Theorem 3.1: Designing a geometric demand-aware network with minimum route length is NP-hard.

IV. A Polynomial-Time Demand-Aware Spanner

This section presents and analyzes our proposed polynomial-time approximation algorithm to design a constant-degree spanner which achieves a provably low stretch and allows to account for communication demands. Our algorithm builds upon the traditional (demand-oblivious) construction by Bonichon et al. [1], but we observe and exploit flexibilities in this construction, using alternative links which can optimize the topology in a demand-aware manner.

To this end, we first closely follow Bonichon et al.'s construction in §IV-A, §IV-B and then present how to exploit demand-awareness in §IV-C, showing that it can significantly improve the total cost.

In a nutshell, our algorithm proceeds as follows. Starting with n points $v_1, \ldots, v_n$ in the two-dimensional Euclidean plane and a matrix $D \in \mathbb{R}_{\geq 0}^{n \times n}$, the goal is to obtain a connected graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ of bounded degree. E is the set of edges of our graph G. An edge $e = (v_j, v_k)$, $j, k \in \{1, \ldots, n\}$, is weighted with its length, the Euclidean distance $|v_j, v_k|$. The entries D_{jk} of the $n \times n$ matrix D are the nonnegative demands between two points v_j and v_k.

The algorithm begins with the complete Euclidean graph K_n and step by step selects specific edges to reduce the degree of the graph. To this end, for each node, we first in §IV-A subdivide the plane into six conic regions, alternatingly denoted as positive and negative cones. Then in §IV-B, each node selects for each positive cone just one edge to remain. However, in the negative cones, a node could still be connected to every other node in the worst case. Hence, in §IV-C we reduce the number of edges in negative cones to at most three. Moreover, we also guarantee that the resulting maximum degree is bounded by 9 and that our construction results in a 10-spanner of K_n. Afterwards in §IV-D, we investigate the demand-awareness of our method and give an example where we improve over the work of Bonichon et al. [1] by a factor of three. In addition, we show that our method is efficient, incurring at most quadratic run time.

A. Defining the Building Blocks: The Cones of a Node

Our algorithm revolves around the cones of a point v.

Definition 4.1: A *cone* is the area between two non-parallel rays in the plane with the intersection point as its apex.

To obtain the cones required for this algorithm, we translate the positive x-axis to our point v and then rotate this half-line counterclockwise by angles of $k\pi/3$ with $k = 0, 1, \ldots, 5$ (see Fig. 2). Each pair of successive rays defines a cone with apex v. In this way we obtain six cones C^v for a node v, which are all congruent to each other.

We will distinguish between two types of cones, positive and negative cones. Opposite of a positive cone C_i^v is the corresponding negative cone $\overline{C_i^v}$. Moreover, the clockwise and counterclockwise neighboring cones cannot be of the same type. Starting at the upper half, the cone in the middle is named C_1^v and the negative cone on the opposite is labeled $\overline{C_1^v}$ (see Fig. 2). The labeling continues counterclockwise and follows the rules above, which means that the next positive

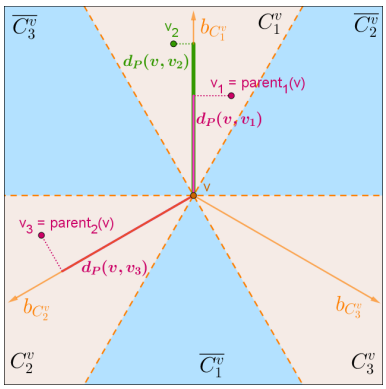


Fig. 2: Projective distance and parent of a cone

cone after C_1^v is labeled C_2^v and the last one C_3^v. Hence, we get the cone-sequence $\left(C_1^v, \overline{C_3^v}, C_2^v, \overline{C_1^v}, C_3^v, \overline{C_2^v}\right)$. We need a cyclic numbering of the cones so that $i \pm 1$, $i = 1, 2, 3$, is well-defined. As one can see, the counterclockwise neighboring cone of a positive cone C_i^v is the negative cone $\overline{C_{i-1}^v}$ and the clockwise neighbor is $\overline{C_{i+1}^v}$. The same applies to a negative cone $\overline{C_i^v}$. We will use the variable i only for the numbering of cones thus omit specifying $i = 1, 2, 3$. Moreover, this numbering has the property that $u \in C_i^v \Leftrightarrow v \in \overline{C_i^u}$.

Let $b_{C_i^v}$ resp. $b_{\overline{C_i^v}}$ be the bisector ray of the cone C_i^v resp. $\overline{C_i^v}$. To select specific nodes in each cone, we have to define the function $d_P(v, v_1)$ for two points v, v_1.

Definition 4.2: The *projective distance* $d_P(v, v_1)$ between a node v and $v_1 \in C_i^v$ resp. $v_1 \in \overline{C_i^v}$ returns the Euclidean distance between v and the projection of v_1 onto the bisector $b_{C_i^v}$ resp. $b_{\overline{C_i^v}}$.

We observe that the projective distance is symmetric, i.e. $d_P(v, v_1) = d_P(v_1, v)$ for all nodes $v, v_1 \in V$.

Definition 4.3: Given two points $v_1, v_2 \in C_i^v$, we say v_1 is *closer* to v than v_2 if $d_P(v, v_1) < d_P(v, v_2)$.

From our assumption, that the nodes in V are in general position, we can conclude that the closest point of a cone is uniquely defined which leads us to the following definition.

Definition 4.4: The closest point to v in a positive cone C_i^v is called $parent_i(v)$.

See Fig. 2 for an example of the definitions above.

B. First Step: Connecting via Cones

Regarding these definitions, the algorithm starts with selecting edges from the complete Euclidean graph K_n as follows:

- *In each positive and non-empty cone C_i^v of every node $v \in V$ select the edge $(v, parent_i(v)) \in K_n$.*

The resulting subgraph after this step will be called $S_1 = (V, E_1)$. Although we are constructing an undirected graph, we will refer to the edges selected in this step as directed. An edge $e = (v_j, v_k) \in E_1$ is outgoing from v_j and incoming at v_k, if $v_k = parent_i(v_j)$, i.e. the node v_k is selected by the positive cone $C_i^{v_j}$ of v_j.

See Fig. 3 for an example of the graph S_1 with eight nodes. The corresponding positive resp. negative cones of the chosen edges are colored red resp. blue.

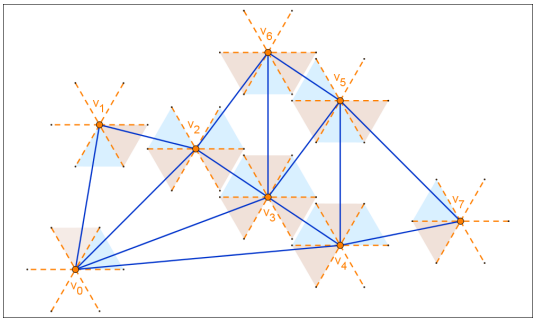


Fig. 3: Example - resulting graph S_1

Theorem 4.1: The subgraph S_1 of K_n fulfills that 1) every node in S_1 has at most one outgoing edge in each positive cone, 2) S_1 is a 2-spanner of K_n, 3), S_1 is a plane graph, and all faces in S_1 (except the outer face) are triangles.

We refer to Bonichon et al. [1] for the correctness of Theorem 4.1. Even if the number of edges in all positive cones of a node v is bounded by three in this graph, we do not have a bound for the number of incoming edges in the negative cones, i.e. the degree of S_1 is still not bounded.

C. Second Step: Introduce Flexibility with Guarantees

The goal of this step is to take the graph S_1 from Section IV-B and extract a subgraph S_2 with bounded degree, while at the same time introducing flexibilities for demand-awareness and retaining stretch guarantees. For this we more or less reverse the first step and will now select specific edges in each negative cone $\overline{C_i^v}$ of a point $v \in V$.

Definition 4.5: Let $children_i(v)$ be the set of all points $v_j \in \overline{C_i^v}$ with an incoming edge $e = (v_j, v) \in S_1$ at v, i.e. $v = parent_i(v_j)$. Now we can define some special points in $children_i(v)$.

- $closest_i(v)$ is the closest node to v.
- $first_i(v)$ is the first node of $children_i(v)$ in counterclockwise order.
- $second_i(v)$ is the second node of $children_i(v)$ in counterclockwise order.
- $penultimate_i(v)$ is the penultimate node of $children_i(v)$ in counterclockwise order.
- $last_i(v)$ is the last node of $children_i(v)$ in counterclockwise order.

These five points do not have to be defined in every negative cone, e.g. if $\overline{C_i^v} = children_i(v) = \emptyset$. If $children_i(v) \neq \emptyset$, we always start labeling these special nodes with $closest_i(v)$. After that, we continue labeling with the ordered nodes $first_i(v)$, $second_i(v)$, $penultimate_i(v)$ resp. $last_i(v)$. In case $closest_i(u) = first_i(v)$, we call the node the $closest_i(u)$ and say that $first_i(v)$ is not defined. The same applies to the ordered points $second_i(v)$, $penultimate_i(v)$ resp. $last_i(v)$.

Although we defined some special points for the negative cones, not all of them are always relevant for us and they can therefore be ignored. The edge $(closest_i(v), v)$, if $closest_i(v)$ is defined, will always be picked. But to see which of the ordered nodes are possible candidates to be selected, we need to check whether they are *i-relevant*.

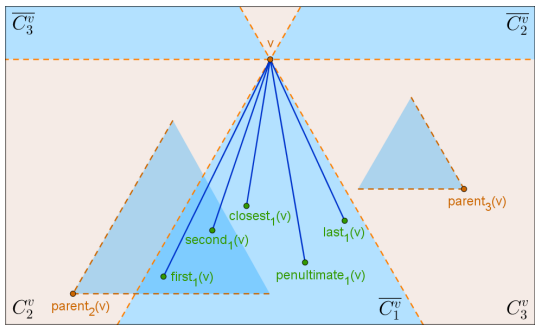


Fig. 4: A node is (not) i-relevant

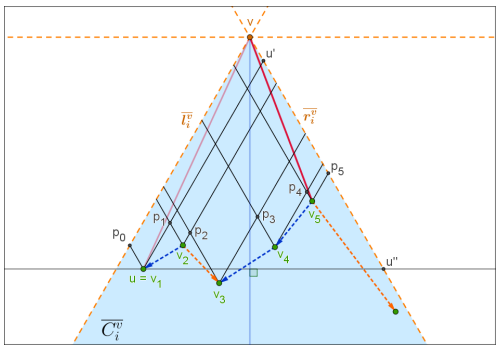


Fig. 5: Estimate of the length of a path between v and u

Definition 4.6: A point $v_1 \in \overline{C_{i\pm1}^v}$ is *i-relevant with respect to v* if all of the following conditions are met.

1) $v_1 \neq closest_{i\pm1}(v)$
2) v_1 fulfills one of the following:
   a. $v_1 = first_{i-1}(v)$
   b. $v_1 = second_{i-1}(v)$ is closer to v than $first_{i-1}(v)$
   c. $v_1 = penultimate_{i+1}(v)$ is closer to v than $last_{i+1}(v)$
   d. $v_1 = last_{i+1}(v)$
3) $v_1 \in \overline{C_i^{parent_i(v)}}$

Fig. 4 shows an example of points which are resp. are not relevant with respect to v. The nodes $first_1(v)$ and $second_1(v)$ fulfill all the conditions from Definition 4.6 and hence are 2-relevant. $penultimate_1(v)$ is not closer to v than $last_1(v)$ and is therefore not 3-relevant. $last_1(v)$ is also not 3-relevant because the last requirement is not fulfilled, i.e. $last_1(v) \notin \overline{C_3^{parent_3(v)}}$.

When a point is defined as $special\_point_{i\pm1}(v)$, we will just say this point is i-relevant and omit *with respect to v*.

To obtain a subgraph $S_2 = (V, E_2)$ with bounded degree of the graph S_1 from the first step we select edges as follows:

In each negative cone $\overline{C_i^v}$ of every node $v \in V$ choose the following edges $e \in S_1$ if the corresponding nodes exist, i.e.:

1) $(closest_i(v), v)$,
2) either $(first_i(v), v)$ if $first_i(v)$ is (i+1)-relevant or $(second_i(v), v)$ if $second_i(v)$ is (i+1)-relevant,
3) either $(last_i(v), v)$ if $last_i(v)$ is (i-1)-relevant or $(penultimate_i(v), v)$ if $penultimate_i(v)$ is (i-1)-relevant.

In this step we remove some edges of S_1 to obtain S_2. Therefore, it is not assured that the resulting graph S_2 is still connected. We now prove that this property still holds. We only give a short sketch here due to space constraints.

Theorem 4.2: The subgraph S_2 of S_1 is planar and connected.

Proof Sketch: Planarity of S_2 follows from the planarity of S_1 (Theorem 4.1) and $S_2 \subset S_1$, as we only removed edges from S_1. To prove the connectivity of S_2 we need to show that for every removed edge (u, v) in S_1 but not in S_2 there exists a path from u to v. Hence, for every node v the set $children_i(v)$ is connected. The idea is to consider two neighboring nodes in the counterclockwise ordering of $children_i(v)$. If these two nodes do not have a direct connection, we can iteratively construct a path in S_2 connecting them. As we have n nodes, we will obtain a path after a finite number of steps. ■

A consequence of this theorem is that for two arbitrary nodes $u, v \in V$ there exists a path in S_2. Let $pl_{S_2}(u, v)$ be the length of the shortest path in S_2 between the two nodes $u, v \in V$. We obtain the following property.

Theorem 4.3: S_2 is a 10-spanner of K_n.

Proof: In combination with Theorem 4.1, we only need to prove that S_2 is a 5-spanner of S_1. To obtain an upper bound for the stretch factor of S_2 in S_1, we need an estimate for the length of a path in S_2 between two nodes v, u. First, we estimate the path length $pl_{S_2}(v_j, v_{j+1})$ between two neighboring nodes $v_j, v_{j+1} \in children_i(v)$ and then between a node v and $u \in children_i(v)$. For this we will consider the rays defining the cones of v_j, v_{j+1}, the cones of the nodes in $children_i(v)$ and v and the intersection of these (see Fig. 5). Moreover, we will look at an equilateral triangle in $\overline{C_i^v}$ with v as a vertex. By using properties of our construction, e.g. that the angle between two consecutive rays of a node is $\pi/3$, we obtain $pl_{S_2}(v_j, v_{j+1}) \leq 2|v_j, v_{j+1}|$ for two neighboring nodes in the canonical path and $pl_{S_2}(v, u) \leq 5|v, u|$ for the path length between v and a children u of v. ■

We constructed this graph S_2 by selecting at most one edge in every positive cone and at most three edges in every negative cone of a node. Hence, the maximal degree of S_2 is bounded by 12. Theorem 4.4 will provide an even better upper bound for the degree of S_2.

Theorem 4.4: The degree of S_2 is bounded by 9.

Proof: By definition the edge from a node v to $closest_i(v)$ is always contained in S_2, if $closest_i(v)$ exists. Therefore, we will prove by case differentiation that for every i not all of the following three edges can be contained in S_2:

- $(parent_i(v), v)$,
- $(first_{i-1}(v), v)$ or $(second_{i-1}(v), v)$,
- $(penultimate_{i+1}(v), v)$ or $(last_{i+1}(v), v)$.

Let $u = parent_i(v)$. The first case is $(u, v) \notin S_2$, which implies that the edge to $parent_i(v)$ is not in S_2. Otherwise we will consider $(u, v) \in S_2$ and v being a special node. For each case of a special node we can find an empty region whereby some of the special nodes of v cannot be relevant and therefore cannot be selected. ■

As mentioned at the beginning of this section, our approach is based on the algorithm described in [1]. The first selection

process, explained in § IV-B, was mainly adopted from their paper. However, an important difference to the work of Bonichon et al. is in step 2, since this step turns our algorithm into a demand-aware one. In [1], besides $closest_i(v)$, only $first_i(v)$ and $last_i(v)$ are considered, whereas in our construction there is the possibility to choose between nodes, namely between $first_i(v)$ and $second_i(v)$ resp. $penultimate_i(v)$ and $last_i(v)$. Hence, we can take the results from [1] as a base for our method, but have to adjust them for demand-awareness.

D. Demand-Aware Spanner and Runtime

Given the concepts above, we can now summarize our algorithm, see Algorithm 1, and analyze its runtime. Our algorithm consists of two parts (the two outer `for` loops), each implementing a corresponding step described in Section IV-B and IV-C. For the first step, we have n nodes and three positive cones per node, hence the loop in line $2-9$ will run $3n$ times. Additionally, the projective distance $d_P(v,u)$ of two nodes v and $u \in C_i^v$ can be computed in constant time and finding a minimum in linear time. Hence, we obtain a time complexity of $O(n^2)$ for the implementation of the first step. As noted in the proof of Theorem 4.1 the resulting subgraph S_1 after the first step is closely related to the TD-Delaunay graph which can be computed in $O(n \log n)$ time [8]. Therefore, this intuitive approach with a time complexity of $O(n^2)$ can be improved to at least $O(n \log n)$.

Let us consider the second outer `for` loop. Like the first, it runs at most $3n$ times. Finding $closest_i(v)$ in each non-empty negative cone and adding the corresponding edge to E_2 has a time complexity of $O(n)$. Defining the special nodes can also be done in linear time. Moreover, checking the `if` conditions for the special nodes takes constant time. Therefore, we obtain a quadratic time complexity for the second nested `for` loop which implies $O(n^2)$ for the whole algorithm.

Theorem 4.5: Computing a demand-aware spanner with Algorithm 1 finishes in a run time of $O(n^2)$.

To account for the demand, we observe that in the second step of the algorithm, we have the possibility to choose in a negative cone $\overline{C_i^v}$ between the edges

- $(first_i(v), v)$ and $(second_i(v), v)$ resp.
- $(penultimate_i(v), v)$ and $(last_i(v), v)$

if the corresponding nodes are $i \pm 1$ relevant. We exploit this flexibility and choose based on the demand.

To give an example and illustrate the potential gain, Fig. 6 shows an example with six nodes where accounting for the demand can reduce the total routing costs by a factor of 3. In this example $closest_i(v)$, $penultimate_i(v)$ and $last_i(v)$ lie almost on the left resp. right boundary of the cone $\overline{C_i^v}$ with $penultimate_i(v)$ and $last_i(v)$ still being $(i-1)$-relevant. Moreover, we assume $|v, last_i(v)| \gg |v, penultimate_i(v)|$.

The blue edges are the edges to the closest nodes according to negative cones, which means these edges are selected independently of the demand, but the negative cone $\overline{C_i^v}$ has the option to select between the edge to $penultimate_i(v)$ resp. $last_i(v)$ (corresponding edges are colored red). By selecting the edge $(last_i(v), v)$ we obtain additional routing costs arbitrarily close to $2\,|v, penultimate_i(v)| \cdot Demand(v, penultimate_i(v))$, but selecting $(penultimate_i(v), v)$ would cause arbitrarily small costs. Assuming the demand of v to $penultimate_i(v)$ is much higher than to $last_i(v)$, the total routing costs to $penultimate_i(v)$ and $last_i(v)$ are arbitrarily close to three times higher when selecting the edge $(last_i(v), v)$.

Theorem 4.6: Computing a demand-aware spanner with Algorithm 1 can improve the path length objective by a factor of at least arbitrarily close to 3 over the construction by Bonichon et al. [1].

Algorithm 1 Geometric Demand-Aware Spanner (GDA)

Input: Set V of n nodes, $n \times n$ demand-matrix D

1: Initialize empty sets E_1, E_2
2: **for** every positive & non-empty C_i^v of each $v \in V$ **do**
3: | **for** $u \in C_i^v$ **do**
4: | | Define $d_{v,u} := d_P(v,u)$ (projective distance)
5: | **end for**
6: | Find $u_p := parent_i(v)$ with smallest proj. dist. to v
7: | Add (v, u_p) to E_1
8: | Add v to $children_i(u_p)$
9: **end for**
10: **for** every $\overline{C_i^v}$ of each $v \in V$ with $children_i(v) \neq \emptyset$ **do**
11: | Find $closest_i(v)$ with smallest proj. distance to v
12: | Add $(closest_i(v), u)$ to E_2
13: | **if** $|children_i(v)| \geq 2$ **then**
14: | | Define $first_i(v)$, $second_i(v)$, $penultimate_i(v)$, $last_i(v)$ in counterclockwise order in $children_i(v)$
15: | | **if** $first_i(v)$ & $second_i(v)$ are $(i+1)$-rel. **then**
16: | | | Choose $(v, first_i(v))$ or $(v, second_i(v))$ according to higher demands and add it to E_2
17: | | **else if** $first_i(v)$ or $second_i(v)$ are $(i+1)$-rel. **then**
18: | | | Add edge from v to $(i+1)$-rel. node to E_2
19: | | **end if**
20: | | **if** $penultimate_i(v)$ & $last_i(v)$ are $(i-1)$-rel. **then**
21: | | | Choose $(v, penultimate_i(v))$ or $(v, last_i(v))$ according to higher demands and add it to E_2
22: | | **else if** $penultimate_i(v)$ or $last_i(v)$ are $(i-1)$-rel. | **then** add edge from v to (i-1)-rel. node to E_2
23: | | **end if**
24: | **end if**
25: **end for**

Output: Connected and planar graph $S_2 = (V, E_2)$ with stretch factor $t \leq 10$ and maximum degree 9

V. Improving Demand-Aware Spanners Further

In the previous section, we pointed out the topological flexibilities offered by our spanner construction, and how they can be exploited to render the network more demand-aware, while preserving approximation guarantees.

In the following, we will build upon this algorithm and propose two heuristics to improve the demand-awareness further. We present a *Link Exchange* algorithm to improve the objective value for a given graph. In this algorithm, we

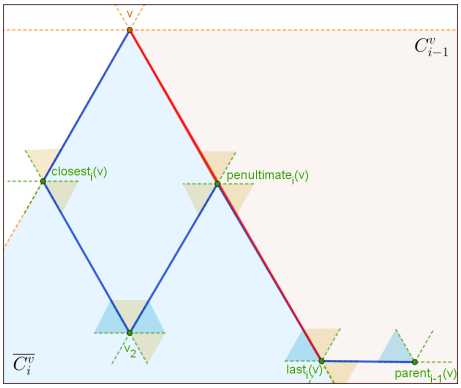


Fig. 6: Example - cost effectiveness of including the demand

consider all the edges which are not contained in the given graph and sort them in descending order according to the demand between their endpoints. Our approach is to attempt to insert one of these edges at a time and check if this improves the objective, while maintaining degree bounds. To maintain planarity, the algorithm first removes the intersecting links in our graph with the considered edge. Next, the objective value is computed. If it is infinite, we cannot serve all demands and can therefore investigate the next edge in the list, else we continue with the degree constraint. If the degree of both vertices of the edge is still at most nine, we can continue. Otherwise, we have to remove one or two edges at the endpoints, where the algorithm computes all possible objective values, chooses the smallest one and compares this with the initial value. Once we checked all edges, we start with the next iteration, containing all edges not present in the current network design. If there are no changes after a complete iteration, the algorithm terminates.

To study the costs incurred by considering only planar graphs, we can also specify a *non-planar* link exchange algorithm, where the check for intersecting edges is skipped.

VI. Evaluation

To evaluate the efficiency of our algorithms in realistic scenarios, we conducted several experiments using real topologies and considering both real and synthetic traffic matrices. Next, we discuss our methodology and the obtained results.

A. Methodology

We implemented and compared four different algorithms. We start with the state-of-the art spanner algorithm from Bonichon et al. [1] which does not take the demands into account, henceforth referred to as *Oblivious*. We compare Oblivious to our geometric demand-aware spanner Algorithm 1, denoted as *GDA*, which improves the local selection of links according to traffic sizes. On top of GDA, we then perform our link exchange approach, evaluating both planar (*LE+P*) and non-planar (*LE+NP*) variants, where we apply the same upper degree bound of nine as for GDA (Thm. 4.4). To bound the run time for LE+P/NP, we observed that the improvement after the first iteration through all links was almost negligible and we hence bound the number of iterations to two. Moreover and in the same fashion, we also cap the run time of all algorithms to 10^3, returning the then best-so-far obtained result. Along the same lines, we omit the MIP results due to their excessive time complexity, as our link exchange algorithms already perform very close to a lower cost bound provided by K_n.

B. Experimental Setup

The performed experiments can be divided into two parts based on the origin of the input data: 1) real topologies and real demand matrices, and 2) real topologies and synthetically generated demand matrices. We choose the Geant topology from the SNDlib library [9] as an example of real topology in the first part and consider 100 different real demand matrices. In the second part, 150 topologies from Topology Zoo [10] are considered with the number of vertices ranging from 20 to 90. Per topology, 10 demand matrices were generated using a standard gravity model [11]; for this purpose, an exponential distribution was utilized with setting the scale to 1,000. Herein, the average of the results of all experiments with the equal number of nodes was calculated.

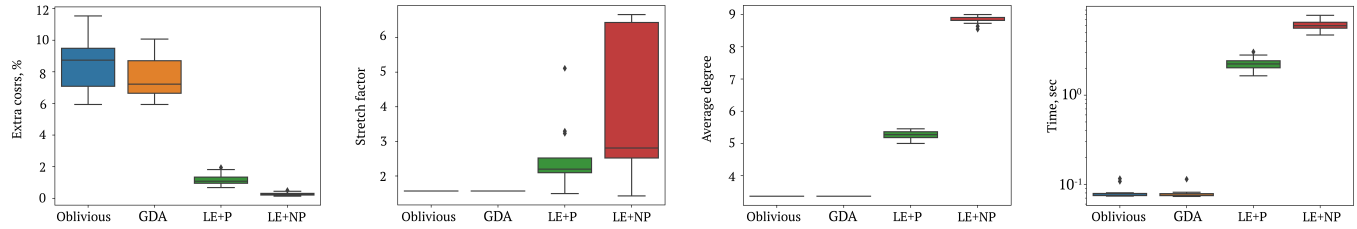
Four aspects of the specified algorithms are compared: total costs, stretch factor, average degree of the design, and run time performance. To show the results, we present boxplots (for Geant) and line charts (for Topology Zoo). On the total costs plot, we compare with an (infeasible) complete design K_n as a lower bound, and display by how many percent the costs increase in comparison.

Our simulations were executed on an HP DL380 G9 with 2x IntelXeons E5-2697V3 SR1XF with 2.6 GHz, 14 cores each, and a total of 128 GB DDR4RAM. The host machine was running Ubuntu 18.04.4 LTS. The algorithms are implemented in Python (3.7) using the NetworkX library (2.4), where each of the four algorithms only runs on a single core.

C. Large-Scale Results

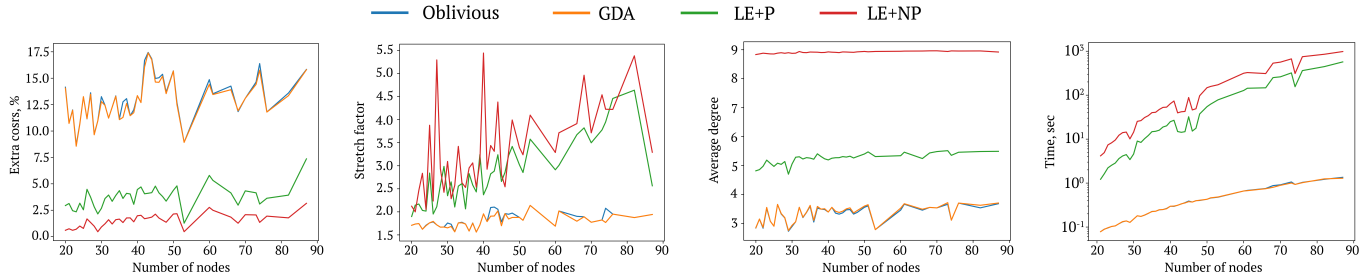
The overall trend is that w.r.t. total costs, LE+NP performs best, followed by LE+P and GDA, with Oblivious being last. On the other hand, the trend for the remaining three metrics is reversed, with Oblivious and GDA performing essentially along the same lines, followed by LE+P and lastly LE+NP. Each of the considered metrics is discussed separately next.

1) Total costs: In general, the total costs for GDA are better than the ones for the Oblivious algorithm. In the case of Geant, we observe an average improvement of 1%, with Topology Zoo showing similar results. However, there are some places where the improvement reaches a value of 1-1.5%. Moreover, after applying the planar link exchange module (LE+P), a noticeable decrease in total costs is being observed. Comparing to GDA, the improvement in Geant is around 7% on average, with results in Topology Zoo being up to 10-12% better. Already here, we are quite close to the lower bound provided by the complete graph K_n. The non-planar exchange of links (LE+NP) gives an even stronger effect: in both cases the obtained values are only 1-2 extra percent away from the theoretical (unreachable) lower bound, though all in all, both LE+P and LE+NP remain relatively close



(a) Total costs (b) Stretch factor (c) Average degree (d) Computation time

Fig. 7: Boxplots for the simulations in the Geant network



(a) Total costs (b) Stretch factor (c) Average degree (d) Computation time

Fig. 8: Line charts for the simulations in Topology Zoo networks

2) Stretch factor: Prior work [1] established that Oblivious has a stretch factor of at most 6, with our demand-aware spanner GDA not having a stretch factor beyond 10 (Theorem 4.3). In our experiments we see that the stretch factor never gets close to these bounds: for both Geant and Topology Zoo, the stretch factor is always lower then 2. On the other hand, the link exchange modules provide no guarantees on the stretch. While the stretch factor value exceeds 6 in the worst case for LE+NP in Geant, and 5 for LE+P, it is mostly below 3 for LE+P. For Topology Zoo it also rises compared to Oblivious and GDA, but is not larger than 5, except for LE+NP for a few outliers. In general, the stretch factor for LE+NP is usually bigger on average then for LE+P. Upon closer investigation, we saw that the extreme stretch factors were created by nodes very close to each other, with small demand, where the exchange heuristics might remove their direct connection for total gain.

3) Average degree: We measured the average degree of vertices in graph obtained after completion of each of the algorithms. The average degree for both Oblivious and GDA differs only slightly, in both Geant and Topology Zoo cases it does not exceed 4. Both versions of the Link Exchange module show an increase in the average degree, which is explained by the addition of a greater number of links to the graph. The result for LE+NP has the most links out of all the algorithms and shows that the majority of the vertices regardless of topology have degree 9, resulting in an average degree approximately equal to 9. LE+P on the other hand mostly hovers between 5 and 6, which can be explained with planar graphs always having an average degree smaller than 6, and LE+P behaving quite aggressive in its optimization.

4) Computation Time: The running time of each of the considered algorithms increases for larger topology size. As the total costs must be obtained at least once per iteration while exchanging links, the processing time is greater for any of the link exchange modifications comparing to Oblivious and GDA. The Oblivious and GDA algorithms are both very fast: it takes less then 1 sec to run these algorithms for topologies with less then 50 nodes. LE+P is generally faster then LE+NP: this can be explained by the fact that we need to calculate the total costs more often when the degree exceeds 9 during link insertion, and this happens frequently in the case when planarity is not required. For the maximum of 50 considered nodes the performance does not reach the chosen upper bound of 1000 sec. However, link exchange in bigger topologies may halt its optimization due to specified run time bounds.

D. Summary and Discussion

Our demand-aware GDA slightly improves w.r.t. the primary objective of total costs over Oblivious, but at the same time retains essentially identical stretch (even a bit better for some Topology Zoo networks), average degree, and computation time. We hence see GDA as a good first step to introduce demand-aware geometric spanners with provable performance guarantees. At the same time, we see our link-exchange algorithms utilizing demand-awareness in a stronger fashion. Even for larger networks, they stay close to a complete graph K_n, with the planar version maintaining low average degree. On the other hand, the link-exchange algorithms lose provable stretch guarantees, but as we see in the plots, the planar version still maintains a low stretch throughout. Herein, if a certain stretch must be guaranteed, it would moreover be easy to introduce an invariant that prohibits exchanges violating a maximum stretch bound. As thus, we believe planar link-exchange algorithms to introduce interesting flexibilities for demand-aware geometric applications, and we hope that

future work can benefit from our studies. For computation time, the link exchange algorithms could easily be optimized by parallelization. For example, when checking which edges to remove to maintain degree bounds, all computations are independent, and already on our machine we would expect this insight to yield a speed-up of $\approx 10\times$.

VII. Related Work

That traffic patterns often feature much structure is well-document in the literature and also applies to other networks, e.g., [12]. In particular, we are not the first to explore the design of demand-aware networks, and we refer the reader to our recent overview articles on the topic [13]–[15]. In contrast to existing literature, however, we consider a geometric setting where nodes are embedded in the plane, as often assumed in the literature, e.g., in sensor networks. This makes our problem technically fairly different, and rather, needs to be seen from the perspective of geometric spanners. There already exist many different methods for constructing t-spanners for a set of points in the Euclidean plane with various properties like small size, degree or weight, see [6]. Plane geometric graphs have also been studied well, e.g. in [8], [16]–[19]. These algorithms focus on minimizing the stretch factor.

However, besides a low stretch factor network topologies often require a bounded degree as well. This raises the question about the smallest maximum degree and stretch factor which is achievable for a plane spanner of a two-dimensional Euclidean graph. Several researchers tried to solve this problem. One method, used in [20]–[24], starts with the classical L_2-Delaunay triangulation of a set of points in the Euclidean plane and then extracts a lower degree spanning subgraph. For example, using the L_1-Delaunay triangulation Bonichon et al. [25] were able to construct a plane spanner of maximum degree 4 with stretch factor of around 156. Another approach taken in [1], [26] is based on the Triangular-Distance-Delaunay (short TD-Delaunay) triangulation defined by Chew [8]. It selects edges from the TD-Delaunay graph to get a spanning subgraph and then modifies the graph to decrease the maximum degree.

While existing work aims at reducing the maximum degree and the stretch factor of a graph, we can still build upon this prior work. In particular, our work extends the approach by Bonichon et al. [1], in that we also start with a TD-Delaunay triangulation, but additionally exploit the flexibility to choose between edges to render the network demand-aware.

VIII. Conclusion

We introduced the natural notion of spanners which are demand-aware, and presented a first construction which leverages topological flexibilities to account for the demand.

We understand our work as a first step and believe that our work opens several interesting avenues for future research. For example, it will be interesting to explore demand-dependent metrics which capture to which extent demand-awareness can improve performance, but also settings where some links are fixed [27], [28]. Today, we still do not fully understand the exact approximation factors which can be achieved by geometric spanners of given degrees. Moreover, we also plan to investigate three-dimensional demand-aware network design, e.g., with satellites [3]–[5].

References

[1] N. Bonichon *et al.*, "Plane Spanners of Maximum Degree Six," in *ICALP*. Springer, 2010, pp. 19–30.

[2] R. Durairajan *et al.*, "Greyfiber: A system for providing flexible access to wide-area connectivity," *CoRR*, vol. abs/1807.05242, 2018.

[3] D. Bhattacherjee and A. Singla, "Network topology design at 27, 000 km/hour," in *CoNEXT*. ACM, 2019.

[4] M. Handley, "Using ground relays for low-latency wide-area routing in megaconstellations," in *HotNets*. ACM, 2019.

[5] Y. Hauri *et al.*, ""internet from space" without inter-satellite links," in *HotNets*. ACM, 2020.

[6] G. Narasimhan and M. Smid, *Geometric Spanner Networks*. Cambridge University Press, 2007.

[7] C. H. Papadimitriou, "The euclidean traveling salesman problem is np-complete," *Theor. Comput. Sci.*, vol. 4, no. 3, pp. 237–244, 1977.

[8] L. P. Chew, "There Are Planar Graphs Almost as Good as the Complete Graph," *J. Comp. Sys. Sciences*, vol. 39, no. 2, pp. 205 – 219, 1989.

[9] S. Orlowski *et al.*, "Sndlib 1.0—survivable network design library," *Networks: An International Journal*, vol. 55, no. 3, pp. 276–286, 2010.

[10] S. Knight *et al.*, "The internet topology zoo," *IEEE JSAC*, vol. 29, no. 9, pp. 1765–1775, 2011.

[11] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *ACM SIGCOMM CCR*, vol. 35, no. 5, pp. 93–96, 2005.

[12] C. Avin *et al.*, "On the complexity of traffic traces and implications," in *Proc. ACM SIGMETRICS*, 2020.

[13] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," in *ACM SIGCOMM CCR*, 2018.

[14] K.-T. Foerster and S. Schmid, "Survey of reconfigurable data center networks: Enablers, algorithms, complexity," *SIGACT News*, vol. 50, no. 2, pp. 62–79, 2019.

[15] M. Nance Hall *et al.*, "A survey of reconfigurable optical networks," *Optical Switching and Networking*, vol. 41, p. 100621, 2021.

[16] P. Bose and M. Smid, "On plane geometric spanners: A survey and open problems," *Computational Geometry*, vol. 46, no. 7, pp. 818–830, 2013.

[17] L. P. Chew, "There Is a Planar Graph Almost as Good as the Complete Graph," in *SOCG*. ACM, 1986.

[18] J. M. Keil and C. A. Gutwin, "Classes of Graphs Which Approximate the Complete Euclidean Graph," *Discrete & Computational Geometry*, vol. 7, no. 1, pp. 13–28, 1992.

[19] G. Xia, "The Stretch Factor of the Delaunay Triangulation Is Less than 1.998," *SIAM Journal on Computing*, vol. 42, no. 4, p. 1620–1659, 2013.

[20] P. Bose, P. Carmi, and L. Chaitman-Yerushalmi, "On bounded degree plane strong geometric spanners," *Journal of Discrete Algorithms*, vol. 15, pp. 16–31, 2012.

[21] P. Bose, J. Gudmundsson, and M. Smid, "Constructing Plane Spanners of Bounded Degree and Low Weight," *Algorithmica*, vol. 42, no. 3, pp. 249–264, Jul 2005.

[22] P. Bose, M. Smid, and D. Xu, "Delaunay and Diamond Triangulations Contain Spanners of Bounded Degree," *International Journal of Computational Geometry & Applications*, vol. 19, pp. 119–140, 2009.

[23] I. A. Kanj and L. Perkovic, "On Geometric Spanners of Euclidean and Unit Disk Graphs," in *STACS*, 2008.

[24] X.-Y. Li and Y. Wang, "Efficient Construction of Low Weight Bounded Degree Planar Spanner," in *Computing and Combinatorics*, T. Warnow and B. Zhu, Eds. Springer, 2003, pp. 374–384.

[25] N. Bonichon *et al.*, "There are Plane Spanners of Degree 4 and Moderate Stretch Factor," *Discrete & Computational Geometry*, vol. 53, no. 3, pp. 514–546, Apr 2015.

[26] I. Kanj, L. Perković, and D. Türkoglu, "Degree Four Plane Spanners: Simpler and Better," *Journal of Computational Geometry*, vol. 8, no. 2, pp. 3–31, 2017.

[27] K.-T. Foerster, M. Pacut, and S. Schmid, "On the complexity of non-segregated routing in reconfigurable data center architectures," *Comput. Commun. Rev.*, vol. 49, no. 2, pp. 2–8, 2019.

[28] T. Fenz *et al.*, "Efficient non-segregated routing for reconfigurable demand-aware networks," *Comput. Commun.*, vol. 164, pp. 138–147, 2020.