Title:	Graph Simplification for Infrastructure Network Design
Author(s):	Yaw, Sean P Middleton, Richard Stephen Hoover, Brendan Arthur
Intended for:	COCOA 2019 : Conference on Combinatorial Optimization and Applications
Issued:	2019-11-19 (Rev.1) (Draft)

Graph Simplification for Infrastructure Network Design*

Sean Yaw[1], Richard S. Middleton[2], and Brendan Hoover[2]

[1] Montana State University, Bozeman MT 59717, USA
sean.yaw@montana.edu
[2] Los Alamos National Laboratory, Los Alamos NM 87545, USA
{rsm,bhoover}@lanl.gov

Abstract. Network design problems often involve scenarios where there exist many possible routes between fixed vertex locations (e.g. building new roads between cities, deploying communications and power lines). Many possible routes in a network result in graph representations with many edges, which can lead to difficulty running computationally demanding optimization algorithms on. While producing a subgraph with a reduced number of edges would in itself be useful, it is also important to preserve the ability to interconnect vertices in a cost effective manner. Suppose there is a set of target vertices in a graph that needs to be part of any size-reduced subgraph. Given an edge-weighted, undirected graph, set of target vertices, and parameter $k \geq 1$, we introduce an algorithm that produces a subgraph with the number of edges bounded by $\min\{O(n\frac{n^{1/k}}{|T|}), O(n\,|T|)\}$ times optimal, while guaranteeing that, for any subset of the target vertices, their minimum Steiner tree in the subgraph costs at most $2k$ times the cost of their minimum Steiner tree in the original graph. We evaluate our approach against existing algorithms using data from Carbon Capture and Storage studies and find that in addition to its theoretical guarantees, our approach also performs well in practice.

Keywords: Graph Spanner · Steiner Tree · Network Design · Carbon Capture and Storage.

1 Introduction

Application-specific networks are commonly represented as generic graph structures, allowing application-agnostic algorithms to be employed for various optimization tasks. Representing a network as a graph requires having information about edges in the network, which is a challenge in certain geospatial applications (e.g. road networks, fiber routing) where one knows the vertices that need

* This research was funded by the US-China Advanced Coal Technology Consortium (under management of West Virginia University) and by the U.S. Department of Energy's (DOE) National Energy Technology Laboratory (NETL) through the Rocky Mountain CarbonSAFE project under Award No. DE-FE0029280.

to be connected, but not how to connect them. For example, if one wanted to connect a set of cities with brand new natural gas transmission pipelines, all that exists is a vertex set without explicit candidate edges from which to select for pipe installation. Edges for these networks can be generated by rasterizing the 2D weighted-cost surface into a grid, placing a vertex in the center of each cell, and determining the cost to visit each of its, up to eight, neighbors. The cost to visit each neighbor cell is based on the topography, property ownership, and physical edge construction costs of the underlying weighted-cost surface. The result of this process is the discretization of the continuous geospatial surface into a large grid graph with diagonal edges, which achieves the goal of providing a graph representation for these geospatial network design problems [18]. However, for even a moderately sized region (e.g. the gulf coast of Texas), this can result in a graph with millions of vertices and edges when the rasterized cell size is approximately a square kilometer. As regions get larger or cell size decreases, graph sizes can quickly become computationally prohibitive to use for complex optimization problems such as Mixed Integer Linear Program (MILP) formulations of optimal infrastructure deployments. Figure 1 shows an example of a rasterized 2D weighted-cost surface with three target vertices, the large grid graph with diagonal edges, and a more manageably sized subgraph.

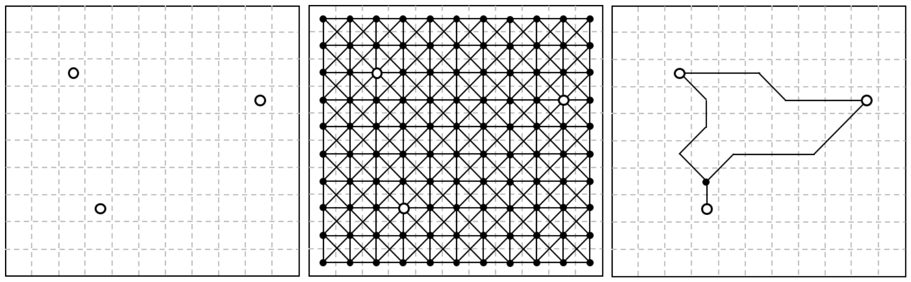

(a) Rasterized Cost Surface (b) Cost Surface Graph (c) Reduced-Edge Subgraph

Fig. 1. Example of graph construction from weighted-cost surface.

This scenario commonly arises in network design problems embedded in the real world such as road networks, physical telecommunication networks, and pipeline networks. We are motivated by the design of Carbon Capture and Storage (CCS) networks. CCS is an effort to reduce the dispersal of CO_2 into the atmosphere by capturing the CO_2 byproduct of industrial processes, and transporting it to geological features where it can be stored [19, 16]. CO_2 capture sites are not often collocated with appropriate storage sites. This requires development of a CO_2 distribution network to link capture sites with storage sites. Furthermore, like any infrastructure project, CCS networks can be costly undertakings, so care must be taken to select capture sites, storage sites, and intelligent distribution links in a cost effective manner [14].

Design of CCS networks, and other geospatial networks, is not as simple as determining edge locations to connect a pre-determined set of capture and storage vertices. Instead, a graph representation of the network needs to be generated that supports deploying an unknown subset of the available capture and storage vertices. This is because the graph feeds a variety of risk analysis studies that determine the best capture and storage locations to open based on cost, capture, and storage objectives, thereby driving final deployment decisions. Contingency plans are also considered in case a storage location's predicted geologic parameters (e.g. capacity, supported CO_2 injection rate) do not match reality [17]. Therefore, the graph needs to be able to support various possibilities, while remaining small enough to enable efficient computation and maintaining some guarantee that the solution cost is within some bound of optimal.

The goal of this research is to simplify a graph by reducing the number of edges and vertices in the graph, while maintaining some guarantee that vertices in a target set can all be connected with a Steiner tree of bounded cost. In the case of the CCS problem, the initial graph is the large grid graph output of the rasterizing process shown in Figure 1b and the target set of vertices is the set of possible capture and storage locations. Formally, we look for a subgraph of a given graph that spans some target vertices, while minimizing the number of edges in the subgraph and ensuring the subgraph contains Steiner trees for any subset of the target vertices with cost bounded by the cost of the optimal Steiner tree in the original graph. To this end, we introduce an algorithm that produces a subgraph with at most a $\min\{O(n\frac{n^{1/k}}{|T|}), O(n\,|T|)\}$ factor of the optimal number of edges, where n is the number of vertices, T is the set of target vertices, and k is an algorithm parameter controlling the quality of solution. This subgraph is guaranteed to have Steiner trees that cost at most $2k$ times the cost of optimal Steiner trees in the original graph, for any subset of the target vertices.

2 Related Work

The problem in the literature that is most closely related to ours is the k-spanner problem [5]. A *spanner* (or k-spanner) is a subgraph that results from trimming edges from a graph, while preserving some degree of pairwise distance between vertices. Formally, given an edge weighted graph, a k-spanner is a subgraph whose distance between each pair of vertices is at most k times the distance between that pair in the original graph. Spanners with a minimal number of edges have been extensively studied [4]. Most well known, it has been shown that for $k \geq 1$, every graph with n vertices has a $(2k-1)$-spanner with at most $O(n^{1+1/k})$ edges [1]. The problem we consider in this paper differs from finding k-spanners in two important ways:

1. Spanners traditionally aim to connect all vertices in the original graph, whereas we look to connect only a designated subset of those vertices. Forcing a solution to include excess vertices unnecessarily enlarges it.

2. Spanners bound the shortest path between each pair of vertices, whereas we aim to bound the minimum Steiner tree cost of any subset of the designated subset of vertices.

Prior work has been done on *subset spanners* (or pairwise spanners), which are subgraphs that approximately preserve distances between pairs of a subset of the vertices. However, this prior work differs from ours by considering a spanner variant with additive error instead of multiplicative [7, 10]. In other words, distances between each vertex pair is at most k plus, not times, the distance between that pair in the original graph. Work has also been done that seeks spanners that exactly preserve pairwise distance instead of approximately preserving it [6, 3, 2]. This could be useful for our application area, but does not enable relaxing the distance requirement in favor of reducing the graph size, which is our primary goal, given the intractability of optimal network design optimizations on large graphs. Furthermore, none of these efforts explicitly consider Steiner trees in the subgraph, necessitating a new approach for finding spanners.

Delaunay triangulation has been shown to produce paths between any two points that are most 2.418 times their Euclidean distance [11]. However, this assumes that the graph is embedded in a metric space, which cannot be assumed for general geospatial network where edge weights need not abide by the triangle inequality. Therefore, for non-metric instances, Delaunay triangulation's distance preserving guarantee does not hold. Nonetheless, it is reasonable to assume that edge weights in physical networks are likely distance dependent, so we do evaluate our solution against Delaunay triangulations in Section 5.

3 Problem Formulation

We consider a graph G consisting of a set of vertices V, undirected edges E, and a value $k \geq 1$. A subset of vertices is designated as target vertices $T \subseteq V$. A non-negative cost $c(e)$ is associated with each edge e in E, reflecting the construction cost for that edge. For any subset of target vertices $T' \subseteq T$, the cost of a minimum Steiner tree on G is denoted $S_{G,T'}$. Our goal is to find a subgraph of G that includes all vertices in T, bounds the cost of the minimum Steiner tree for any subset of T, and has a minimal number of edges. The problem is formally defined below.

Definition 1. *Given $G = (V, E, T, c, k)$, a connected, edge weighted, and undirected graph, where $T \subseteq V$ is a set of target vertices, $c\colon E \to \mathbb{R}_{\geq 0}$ is an edge cost function, and $k \geq 1$, the* ***Minimal Steiner-Preserving Subset Spanner*** *problem seeks a subgraph of G, $G' = (V', E', T, c, k)$ such that $V' \subseteq V$, $E' \subseteq E$ induced by the set V', $T \subseteq V'$, $|E'|$ is minimized, and for any $T' \subseteq T$, $S_{G',T'} \leq k * S_{G,T'}$.*

3.1 Computational Complexity

The minimal Steiner-preserving subset spanner problem is related to the MINIMUM EDGE k-SPANNER problem: Given $G = (V, E, k)$, a connected graph with $k \geq 1$, find a spanning subgraph $G' = (V, E', k)$ of G with the minimum number of edges such that, for any pair of vertices, the length of the shortest path (i.e. number of edge) between the pair in G' is at most k times the shortest distance between the pair in G.

Theorem 1. *The MINIMUM EDGE k-SPANNER problem is reducible to the minimal Steiner-preserving subset spanner problem via an S-reduction (cost-preserving reduction).*

Proof. Deferred to Appendix A.

Due to Theorem 1, the following complexity results for the MINIMUM EDGE k-SPANNER problem hold for the minimal Steiner-preserving subset spanner problem as well:

– When $k = 2$, the MINIMUM EDGE k-SPANNER problem is NP-Hard to approximate within a bound of $\frac{\alpha \log|V|}{k}$ for some $\alpha > 0$ [12].
– When $k \geq 3$, the MINIMUM EDGE k-SPANNER problem is NP-Hard to approximate within a bound of $2^{(log^{1-\epsilon} n)/k}$, for all $\epsilon > 0$, if $NP \nsubseteq BPTIME(2^{polylog(n)})$ [8].
– When $k \geq 3$, the MINIMUM EDGE k-SPANNER problem is NP-Hard to approximate within a bound of $n^{1/(\log \log n)^c}$ if the exponential time hypothesis holds, due to the spanner problem's relationship to the Label Cover and the Densest k-Subgraph problems [13].

Because of these complexity results, we pursue suboptimal approaches with performance guarantees for finding solutions to minimal Steiner-preserving subset spanner problem instances in Section 4.

4 Algorithm

In this section, we detail an approach for finding approximate minimal Steiner-preserving subset spanners within graphs. An initial idea could be to leverage the relationship to the MINIMUM EDGE k-SPANNER problem detailed in the proof to Theorem 1. Specifically, we use a variant called the edge-weighted MINIMUM EDGE k-SPANNER where the edges in the input graph have weights: For an input graph $G = (V, E, T, c, k)$, run a solution to the MINIMUM EDGE k-SPANNER problem on $Q = (V, E, c, k)$. This would result in a solution $Q' = (V, E', c, k)$ embedded with Steiner trees for any subset of T costing at most k times their optimal costs in G, since $T \subseteq V$. However, this requires including all vertices in V in the solution, which can dramatically increase the solution size when the size of V is on the order of millions of vertices and the size of T is on the order of tens of vertices. Instead we pursue an approach that attempts to more rigorously control the size of the solution.

The classic *greedy spanner* (GS) algorithm to the edge-weighted MINIMUM EDGE k-SPANNER problem works by iteratively adding individual edges to the

Algorithm 1 Greedy Subset Spanner, on input $G = (V, E, T, c, k)$

```
Step 1  Find paths for pairs of vertices in $T$.
          Let $c' = c$.
          forall vertex pairs in $T$, in order of increasing path cost in $G$
              Find cheapest path in $F = (V, E, T, c', k)$.
              forall Edge $e$ in path
                  Mark $e$ and let $c'(e) = \frac{c(e)}{k}$.
              endforall
          endforall
Step 2  Build $G_{GSS}$.
          Let $E_{GSS} = \emptyset$ and $V_{GSS} = \emptyset$.
          for Edge $e$ in $E$
              if $e$ is marked
                  Add $e$ to $E_{GSS}$ and endpoints of $e$ to $V_{GSS}$.
              endif
          endfor
          Let $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$.
```

solution graph $G_{GS} = (V, E_{GS}, c, k)$ [1]. Given an input of $G = (V, E, c, k)$, GS first sets $E_{GS} = \emptyset$ and sorts E by non-decreasing weight. Then, for each edge $e = (v_1, v_2)$, GS computes the cheapest path from v_1 to v_2 in G_{GS} and adds e to E_{GS} if the cost of that path is larger than k times the cost of e. The algorithm we introduce in this section, GSS for greedy subset spanner, could be seen as a generalization of the GS algorithm. Instead of greedily adding one edge at a time, we greedily add all edges from a path between one pair of vertices in T at a time.

For an input graph $G = (V, E, T, c, k)$, the basic approach we take is to construct an edge set for G_{GSS} by iteratively adding edges from paths between pairs of vertices in T. Each pair of vertices from T is first sorted by non-decreasing cost of their cheapest path in G. A cheapest path for each pair of vertices from T is generated, one at a time in sorted order. Once a path is generated, its edges are added to G_{GSS} and the cost of each edge is reduced to be $\frac{1}{k}$ times its original cost. Reducing the cost of selected edges encourages their use in the cheapest paths for the remaining pairs in T. Reducing the edge cost by $\frac{1}{k}$ ensures that cheapest paths selected will be within k of their actual cheapest paths and is used in the analysis of the algorithm's performance. Note that the cost does not keep lowering if an edge is selected multiple times. Edges costs are either their original costs, or $\frac{1}{k}$ times their original costs. This process builds an edge and vertex set (vertices included in the selected edges) for G_{GSS}. Details of the basic GSS algorithm are presented in Algorithm 1.

Step 1 of GSS runs in $O(|V|^4)$ time since it executes a shortest path search ($O(|V|^2)$) for each pair of points in T ($O(|V|^2)$). Step 2 goes through each edge, and thus runs in $O(|V|^2)$ time. Thus the running time of GSS is $O(|V|^4)$.

Since GSS runs in polynomial time and Theorem 1 showed the problem to be NP Hard, we cannot hope solutions produced by GSS to be optimal. In the following theorems, we establish a guarantee on the cost of Steiner trees for any subset of T and a provide an approximation ratio for the number of edges in solutions output by GSS.

Theorem 2. *Given a graph $G = (V, E, T, c, k)$ and a solution provided by GSS, $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$, for any subset T' of T, the cost of the minimum Steiner tree for T' in G_{GSS} is at most $2k$ times the cost of the minimum Steiner tree for T' in G.*

Proof. Suppose that $G = (V, E, T, c, k)$ and $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$ are the input and output respectively of GSS. The *metric closure* of a set of vertices S in a graph G is the complete (over S) subgraph in which each edge is weighted by the cheapest path distance between vertices in G. It is well known that the cost of a Steiner tree constructed as a minimum spanning tree on a metric closure is at most twice the cost of a minimum Steiner tree in the original graph [20]. If $\mathrm{MST}_{MC_T,G}$ denotes the minimum spanning tree on the metric closure of T in G and $\mathrm{ST}_{T,G}$ denotes the minimum Steiner tree of T on G,

$$\mathrm{cost}(\mathrm{MST}_{MC_T,G}) \leq 2\ \mathrm{cost}(\mathrm{ST}_{T,G}) \tag{1}$$

If $k = 1$, G_{GSS} contains the metric closure of T since G_{GSS} is composed of the cheapest paths in G between each element of T. If $k > 1$, G_{GSS} no longer contains the metric closure of T in G, but contains a path between each pair of vertices from T that costs at most k times the cost of that path in the metric closure in G. Thus, a minimum spanning tree on T in G_{GSS}, $\mathrm{MST}_{T,G_{GSS}}$, would be at most k times the cost of the minimum spanning tree of the metric closure of T in G:

$$\mathrm{cost}(\mathrm{MST}_{T,G_{GSS}}) \leq k\ \mathrm{cost}(\mathrm{MST}_{MC_T,G}), \forall k \geq 1 \tag{2}$$

Combining Equations 1 and 2 shows that G_{GSS} contains a Steiner tree for T of cost at most $2k$ times the optimal cost:

$$\mathrm{cost}(\mathrm{MST}_{T,G_{GSS}}) \leq k\ \mathrm{cost}(\mathrm{MST}_{MC_T,G}) \leq 2k\ \mathrm{cost}(\mathrm{ST}_{T,G}), \forall k \geq 1 \tag{3}$$

Equation 3 holds for all subsets of T, since the metric closures of subsets of T are embedded in the metric closure for T itself, and for all $k \geq 1$, G_{GSS} contains, within k, cheapest paths between all pairs of vertices from any subset of T, since it contains them for all pairs of vertices from T itself. $\square$

Theorem 3. *Given a graph $G = (V, E, T, c, k)$ and a solution provided by GSS, $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$, $|E_{GSS}| \in O(n\,|T|^2)$, where n is the number of vertices in V.*

Proof. For each pair of points from T, GSS provides a path that is within k of its least cost path. Since edge costs are non-negative, each path can be made loopless, thereby being less than or equal to $n-1$ edges. Since there are $O(|T|^2)$ pairs of vertices, $|E_{GSS}| \in O(n\,|T|^2)$. □

The next theorem bounds the size of E_{GSS} in relation to E_{GS}. To avoid pathological cases discussed in the proof, the input graph needs to be pre-processed by running GSS on it with $T = V$. This does not affect the conclusions from Theorem 2 or 3, since the pre-processed graph still contains a path between each pair for vertices that costs at most k times the original cost. Since GSS is run twice (first on $T = V$ and then on the actual subset $T \subseteq V$), the running time of this modification is still $O(|V|^4)$.

Theorem 4. *Given a graph $G = (V, E, T, c, k)$ and a solution provided by GSS, $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$, $|E_{GSS}| \in O(n^{1+1/k})$ when $k \geq 3$, where n is the number of vertices in V.*

Proof. Deferred to Appendix B.

Corollary 1. *GSS is a $\min\{O(n\frac{n^{1/k}}{|T|}), O(n\,|T|)\}$-approximation algorithm when $k \geq 3$ and a $O(n\,|T|)$-approximation algorithm when $k < 3$.*

Proof. Consider a graph $G = (V, E, T, c, k)$, a solution provided by GSS $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$, and an optimal solution $G_{OPT} = (V_{OPT}, E_{OPT}, T, c, k)$. $|E_{OPT}| \geq |T| - 1$, since the optimal edge set must, at minimum, span all vertices in T. By Theorem 4, $|E_{GSS}| \in O(n^{1+1/k}) = O(n\frac{n^{1/k}}{|T|}\,|T|)$ when $k \geq 3$. Thus, $|E_{GSS}| \leq O(n\frac{n^{1/k}}{|T|})\,|E_{OPT}|$. By Theorem 3, $|E_{GSS}| \in O(n\,|T|^2)$. Thus, $|E_{GSS}| \leq O(n\,|T|)\,|E_{OPT}|$. Therefore, $|E_{GSS}| \leq \min\{O(n\frac{n^{1/k}}{|T|}), O(n\,|T|)\}\,|E_{OPT}|$ when $k \geq 3$ and $|E_{GSS}| \leq O(n\,|T|)\,|E_{OPT}|$ otherwise. □

5 Evaluation

In this section, we evaluate the performance of the algorithm presented in Section 4 using realistic data and scenarios. We label the algorithm comprised of Algorithm 1 and the heuristic improvement detailed in Section **??** the Greedy Subset Spanner (GSS) algorithm. In order to characterize GSS's performance, we implemented two additional solutions:

1. Greedy Spanner (GS) is the simple solution to the edge-weighted MINIMUM EDGE k-SPANNER problem introduced in [1], with its implementation outlined in the beginning of Section 4.
2. Delaunay triangulation (DT) is a simple spanning strategy of calculating a Delaunay triangulation and replacing each edge in the triangulation with the cheapest path in the original graph between those points. Theoretical limitations of this approach are detailed in Section 2.

We seek to compare these algorithms based on the quality of solution, as quantified by the size of the output graphs (i.e. number of edges) and cost of minimum Steiner trees in the output graphs, for various subsets of vertices. The dataset we used to evaluate these algorithms encompasses southern Indiana, in the Illinois Basin [9, 15]. Figure 2 shows the geographical area of the dataset as well as the possible capture and storage locations and the edges selected by GSS when $k = 1.5$. The base graph generated by this dataset was constructed as the rasterization of the 2D cost surface as described in Section 1. The number of edges in the base graph is 118563 and the number of vertices is 29879, 43 of which are possible capture and storage locations.

When considering the number of edges in the output subgraph, we collapse degree 2 vertices that are not in T by making a single edge whose cost is the sum of the costs of the two replaced edges. This is done because those vertices dictate the routing of the edge, but not the overall connectivity or cost of the graph. For each edge in the graph, we keep track of its collapsed vertices for determining actual edge routes. To calculate the minimum Steiner trees, we use the popular minimum spanning tree based approximation algorithm [20].

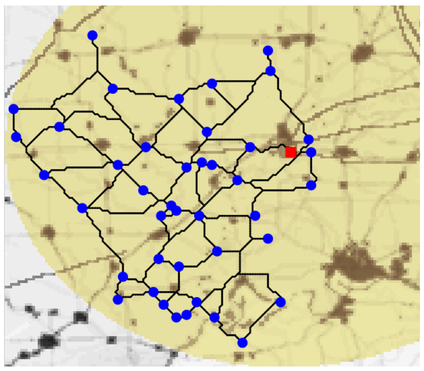
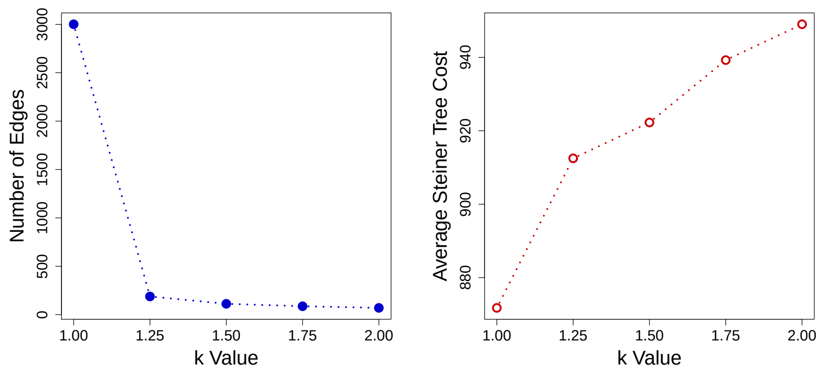


Fig. 2. Dataset map indicating capture (square) and storage (circle) locations as well as the subset of edges selected by the GSS algorithm when $k = 1.5$.

First, we explore the tradeoff between the parameter k and the quality of output for the GSS algorithm by running it with k values between 1.0 and 2.0. The number of edges in the resulting graph is shown in Figure 3a. As expected, the size of the output graph decreases as the parameter k increases, since a larger k allows for more expensive Steiner trees and thus more sharing of edges. The number of edges was reduced by 94% going from $k = 1.0$ to $k = 1.25$, 40% going from $k = 1.25$ to $k = 1.5$, and about 20% from $k = 1.5$ to $k = 1.75$ and $k = 1.75$ to $k = 2.0$. The average cost of Steiner trees for 30 random subsets of capture and storage vertices ranging in size from 10 to 30 vertices is shown in Figure 3b. The increase in Steiner tree cost is much less dramatic than the decrease in graph size for increased k values. The average cost of Steiner trees increased by 4.5% going from $k = 1.0$ to $k = 1.25$ and about 1% for all other jumps in k. This suggests that GSS is an efficient way to reduce graph size while preserving inexpensive Steiner trees.



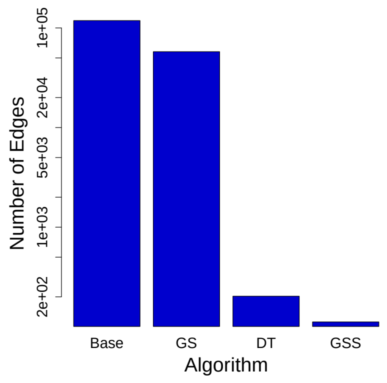
(a) Number of Edges (b) Steiner Tree Cost

Fig. 3. GSS performance metrics for various values of k.

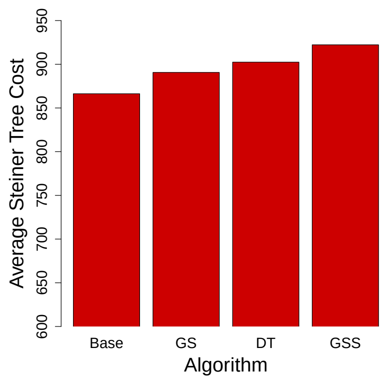
Finally, we seek to compare the GSS algorithm with GS and DT. We also include the base graph, Base, to provide baseline comparison. We parameterized GSS and GS with $k = 1.5$. The number of edges in the output graph for each algorithm is shown in Figure 4a with a log scale, since the differences between algorithms is quite drastic. The number of edges in the base graph is 118563 whereas the output from GS has 58895 edges, DT has 203 edges, and GSS has 112 edges. This means that GGS reduced the number of edges in the base graph by over 99.9%. The average cost of Steiner trees for 30 random subsets of capture and storage vertices ranging in size from 10 to 30 vertices is shown in Figure 4b. As expected, the cost of Steiner trees increases as the size of the graph size decreases, however the increase is markedly small. The average cost of Steiner trees using GS was 3% more than the cost of Steiner trees on the base network, while GSS Steiner trees cost on average 6% more and DT Steiner trees cost on average 4% more than Steiner trees on the base network.

6 Conclusions

Geospatial infrastructure network design problems using optimal approaches can quickly become intractable due to the size of the graph representing many possible edge locations. In this paper, we formalized the problem of generating a reduced edge subgraph that preserves the cost of Steiner trees over any subset of a target set of vertices and proposed an algorithm for it, GSS. GSS provides



(a) Number of Edges - Log Scale



(b) Steiner Tree Cost

Fig. 4. Algorithm performance metrics comparison.

theoretical approximation ratios for both the cost of Steiner trees as well as the number of edges in the subgraph it generates. Using cost surface and capture and storage locations data from CCS studies, we evaluated *GSS* against other possible approaches. *GSS* proved effective at significantly reducing the number of edges while displaying a very small increase in Steiner tree cost in these real world scenarios. Of further interest was the performance of using Delaunay triangulation to find subgraphs. Despite the lack of a theoretical guarantee, and the cost surface not being a metric space, Delaunay triangulation proved quite effective and was very quick in practice. Important future work includes finding a tighter approximation ratio for the number of edges and removing the approximation ratio on Steiner tree cost. An additional avenue of interest is to change the objective from minimizing the total number of edges to minimizing vertices with degree larger than two. A single source-to-sink path can consist of many edges (a route through a cost surface). With the edge minimization objective, that path contributes many edges, whereas with the degree larger than two minimization objective, that path only counts as a single edge. If further optimization processes that use the subgraph only consider paths between vertices, and not underlying routes, it is more accurate to have that path count as a single edge instead of many.

References

1. Althöfer, I., Das, G., Dobkin, D., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete & Computational Geometry **9**(1), 81–100 (1993)
2. Bodwin, G.: Linear size distance preservers. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 600–615. SODA '17, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2017)

3. Bodwin, G., Williams, V.V.: Better distance preservers and additive spanners. In: Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 855–872. SODA '16, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2016)
4. Bose, P., Smid, M.: On plane geometric spanners: A survey and open problems. Computational Geometry **46**(7), 818 – 830 (2013), euroCG 2009
5. Chew, P.: There is a planar graph almost as good as the complete graph. In: Proceedings of the Second Annual Symposium on Computational Geometry. pp. 169–177. SCG '86, ACM, New York, NY, USA (1986)
6. Coppersmith, D., Elkin, M.: Sparse sourcewise and pairwise distance preservers. SIAM Journal on Discrete Mathematics **20**(2), 463–501 (2006)
7. Cygan, M., Grandoni, F., Kavitha, T.: On pairwise spanners. In: 30th International Symposium on Theoretical Aspects of Computer Science. p. 209 (2013)
8. Dinitz, M., Kortsarz, G., Raz, R.: Label cover instances with large girth and the hardness of approximating basic k-spanner. ACM Trans. Algorithms **12**(2), 25:1–25:16 (Dec 2015). https://doi.org/10.1145/2818375, http://doi.acm.org/10.1145/2818375
9. Ellett, K., Rupp, J., Medina, C., Barnes, D., Stauffer, P., Middleton, R.S., Harris, D., Bing, B., Wei, N., Li, J., et al.: A hierarchical approach for evaluating carbon storage resource estimates in deep saline formations. In: AAPG — SEG International Conference and Exhibition (2015)
10. Kavitha, T., Varma, N.M.: Small stretch pairwise spanners. In: International Colloquium on Automata, Languages, and Programming. pp. 601–612. Springer (2013)
11. Keil, J.M., Gutwin, C.A.: Classes of graphs which approximate the complete euclidean graph. Discrete & Computational Geometry **7**(1), 13–28 (Jan 1992)
12. Kortsarz, G.: On the hardness of approximating spanners. Algorithmica **30**(3), 432–450 (Jan 2001)
13. Manurangsi, P.: Almost-polynomial ratio eth-hardness of approximating densest k-subgraph. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 954–961. STOC 2017, ACM, New York, NY, USA (2017)
14. Middleton, R.S., Bielicki, J.M.: A scalable infrastructure model for carbon capture and storage: SimCCS. Energy Policy **37**(3), 1052 – 1060 (2009)
15. Middleton, R.S., Ellett, K., Stauffer, P., Rupp, J.: Making carbon capture, utilization and storage a reality: Integrating science and engineering into a business plan framework. In: AAPG — SEG International Conference and Exhibition (2015)
16. Middleton, R.S., Keating, G.N., Stauffer, P.H., Jordan, A.B., Viswanathan, H.S., Kang, Q.J., Carey, J.W., Mulkey, M.L., Sullivan, E.J., Chu, S.P., Esposito, R., Meckel, T.A.: The cross-scale science of CO2 capture and storage: from pore scale to regional scale. Energy Environ. Sci. **5**, 7328–7345 (2012)
17. Middleton, R.S., Keating, G.N., Viswanathan, H.S., Stauffer, P.H., Pawar, R.J.: Effects of geologic reservoir uncertainty on co2 transport and storage infrastructure. International Journal of Greenhouse Gas Control **8**, 132 – 142 (2012)
18. Middleton, R.S., Kuby, M.J., Bielicki, J.M.: Generating candidate networks for optimization: The CO2 capture and storage optimization problem. Computers, Environment and Urban Systems **36**(1), 18 – 29 (2012)
19. Stauffer, P.H., Keating, G.N., Middleton, R.S., Viswanathan, H.S., Berchtold, K.A., Singh, R.P., Pawar, R.J., Mancino, A.: Greening coal: Breakthroughs and challenges in carbon capture and storage. Environmental Science & Technology **45**(20), 8597–8604 (2011)
20. Vazirani, V.: Approximation Algorithms, chap. 3, pp. 27–29. Springer (2001)

A Proof of Theorem 1

The MINIMUM EDGE k-SPANNER problem is a special case of the minimal Steiner-preserving subset spanner problem. Given $Q = (V, E, k)$ with $k \geq 1$ as an input to the MINIMUM EDGE k-SPANNER problem, construct an instance to the minimal Steiner-preserving subset spanner problem $G = (V, E, T, c, k)$ by using the same V, E, and k, and letting $T = V$ and $c(e) = 1$ for all e in E. We wish to show that a minimal Steiner-preserving subset spanner problem solution to G is a solution to the MINIMUM EDGE k-SPANNER problem if and only if a MINIMUM EDGE k-SPANNER problem solution to Q is a solution to the minimal Steiner-preserving subset spanner problem.

Suppose $G' = (V', E', T, c, k)$ is a solution to the minimal Steiner-preserving subset spanner problem. This means that every subset T' of T has a Steiner tree in G' whose cost is at most k times the cost of the minimum Steiner tree of T' in G. Since $T = V$, $V' = V$ and every pair of points in V is a subset of T. Thus, because a Steiner tree on two points is merely the shortest path between them, the shortest paths in G' between any pair of vertices in V is at most k times the distance between the pair in G. Therefore, G' is a k-spanner of G, and thus Q.

Suppose $Q' = (V, E', k)$ is a solution to the MINIMUM EDGE k-SPANNER problem. This means that every pair of points in V have a path whose cost is at most k times the cost of the cheapest path between those points in Q. Given a subset T' of T and a minimal Steiner tree of T' in Q, adjacent vertices in the Steiner tree can be connected by the shortest path in Q' between those vertices. Since those paths in Q' are at most k times their cost in Q, this yields a Steiner tree whose cost is at most k times the cost of the minimal Steiner tree in Q. Therefore, Q' is Steiner-preserving subset spanner of Q, and thus G.

Since any solution to Q suffices as a solution to G and conversely, the minimum solutions must coincide. □

B Proof of Theorem 4

Suppose that $G_{GSS} = (V_{GSS}, E_{GSS}, T, c, k)$ is the solution provided by GSS on input $G = (V, E, T, c, k)$. Further, suppose that $G_{GS} = (V, E_{GS}, c, k)$ is the solution provided by the GS algorithm described at the beginning of this section. It has been shown that the GS algorithm produces a solution with at most $O(n^{1+1/k})$ edges when $k \geq 3$ [1]. We will show that $|E_{GSS}| \leq |E_{GS}|$ for any set T and $k \geq 1$.

Without loss of generality, assume that edge costs are unique. This can be achieved by small perturbations of the edge costs. We begin by showing that for any $k \geq 1$, $E_{GSS} = E_{GS}$ when $T = V$. This result is not dependent on the pre-processing modification to GSS described above.

Let $e = (v_1, v_2)$ be the least cost edge that meets one of the following conditions:

1. $e \in E_{GSS}$ and $e \notin E_{GS}$
2. $e \in E_{GS}$ and $e \notin E_{GSS}$

Case 1. Suppose that e meets condition 1 (e is in E_{GSS}, but not in E_{GS}). We will show the contradiction that e cannot be in E_{GSS}. Since e is not in E_{GS}, there is a path p from v_1 to v_2 in E_{GS} such that:

1. For each edge $e' \in p$, $\text{cost}(e') < \text{cost}(e)$, since GS schedules edges in order of cost.
2. $\text{cost}(p) \leq k \text{ cost}(e)$, otherwise e would have been scheduled.

All edges from p must be in E_{GSS}, since there does not exist a lower cost edge that is in E_{GS} but not in E_{GSS}. When GSS considers e, all edges in p must have already been added to E_{GSS}, since GSS schedules edges in order of increasing cost. Therefore, when GSS considers e, the reduced cost of each edge e' in p is $\frac{\text{cost}(e')}{k}$. This means that the reduced cost of p is,

$$\sum_{e' \in p} \frac{\text{cost}(e')}{k} = \frac{1}{k} \text{ cost}(p) \leq \frac{1}{k} k \text{ cost}(e) = \text{cost}(e)$$

Therefore, the cost of p found by GSS is at most the cost of e. Increasing k by a small amount in GSS yields a strict inequality, which means that GSS would select path p over edge e, thereby excluding e from E_{GSS}.

Case 2. Suppose that e meets condition 2 (e is in E_{GS}, but not in E_{GSS}). Since e is not in E_{GSS}, there is a path p from v_1 to v_2 in E_{GSS} such that:

1. For each edge $e' \in p$, $\text{cost}(e') < \text{cost}(e)$, since GSS schedules edges in order of cost.
2. $\frac{1}{k} \text{ cost}(p) < \text{cost}(e)$, otherwise e would have been scheduled.

All edges from p must be in E_{GS}, since there does not exist a lower cost edge that is in E_{GSS} but not in E_{GS}. When GS considers e, all edges in p must have already been added to E_{GS}, since GS schedules edges in order of increasing cost. Therefore, p would have been scheduled instead of e, since $\text{cost}(p) < k \text{ cost}(e)$.

Thus, there cannot be an edge e such that either e is in E_{GSS} but not in E_{GS}, or e is in E_{GS} but not in E_{GSS}. Therefore, when $T = V$, every edge in E_{GSS} is in E_{GS}, and $E_{GSS} = E_{GS}$.

If T is a strict subset of V, there is no guarantee that the size of E_{GSS} decreases compared to E_{GSS} when $T = V$. In certain pathological cases, the size of E_{GSS} can in fact increase with a smaller set T. One such scenario occurs when the graph contains a complete component of m vertices paired with a vertex v connected to the other m vertices. With careful edges cost assignments and selection of k, GSS can be made to select the m edges incident to v, when $T = V$, but select the $O(m^2)$ edges in the complete component when T equals that complete component. As such, we leverage the pre-processing modification to GSS discussed before this theorem, where the input graph is pruned to only include edges from GSS when $T = V$. Using the pruned graph with the input T, GSS cannot select edges that are not in the pruned graph, so E_{GSS} cannot grow larger than the pruned graph. Thus, if E_{GSS-V} denotes the edges scheduled by GSS when $T = V$ and E_{GSS-S} denotes the edges scheduled by GSS for some strict subset $S \subset V$, $E_{GSS-S} \subseteq E_{GSS-V}$.

Therefore, for $T \subseteq V$, $|E_{GSS}| \leq |E_{GS}|$ and $|E_{GSS}| \in O(n^{1+1/k})$. □