Construction of Minimum-Weight Spanners

Mikkel Sigurd

Martin Zachariasen

University of Copenhagen

Outline

- Motivation and Background
- Minimum-Weight Spanner Problem
- Greedy Spanner Algorithm
- Exact Algorithm: Column Generation Approach
- Exact Algorithm: Constrained Shortest Paths
- Implementation Details
- Experimental Results
- Conclusions and Future Work

Motivation and Background

Construction of sparse networks that (approximately) preserve distances in original (dense) networks.

Application examples:

Motivation and Background

Construction of sparse networks that (approximately) preserve distances in original (dense) networks.

Application examples:

- Metric space searching. Compact data structure for holding information about approximate distances.

Motivation and Background

Construction of sparse networks that (approximately) preserve distances in original (dense) networks.

Application examples:

- Metric space searching. Compact data structure for holding information about approximate distances.
- Message distribution in networks. Construction of a network that has both low cost and low delay.

Motivation and Background

Construction of sparse networks that (approximately) preserve distances in original (dense) networks.

Application examples:

- Metric space searching. Compact data structure for holding information about approximate distances.
- Message distribution in networks. Construction of a network that has both low cost and low delay.
- Faster approximation algorithms for geometric problems. Replacing dense graphs by sparse graphs may speed up approximation algorithms for several geometric optimization problems (e.g., TSP).

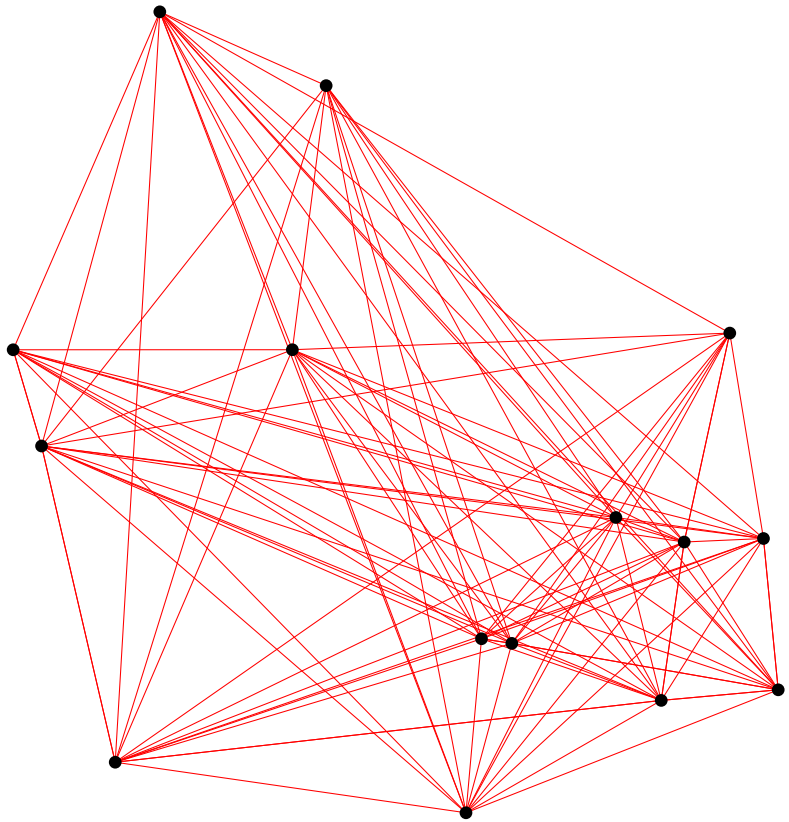
Minimum-Weight Spanner Problem (MWSP)

Given an undirected and edge-weighted graph $G = (V, E)$.

t-spanner: Subgraph $G' = (V, E')$ of G such that the shortest path between any pair of nodes is at most t times longer in G' than in G (where $t > 1$ is a fixed constant).

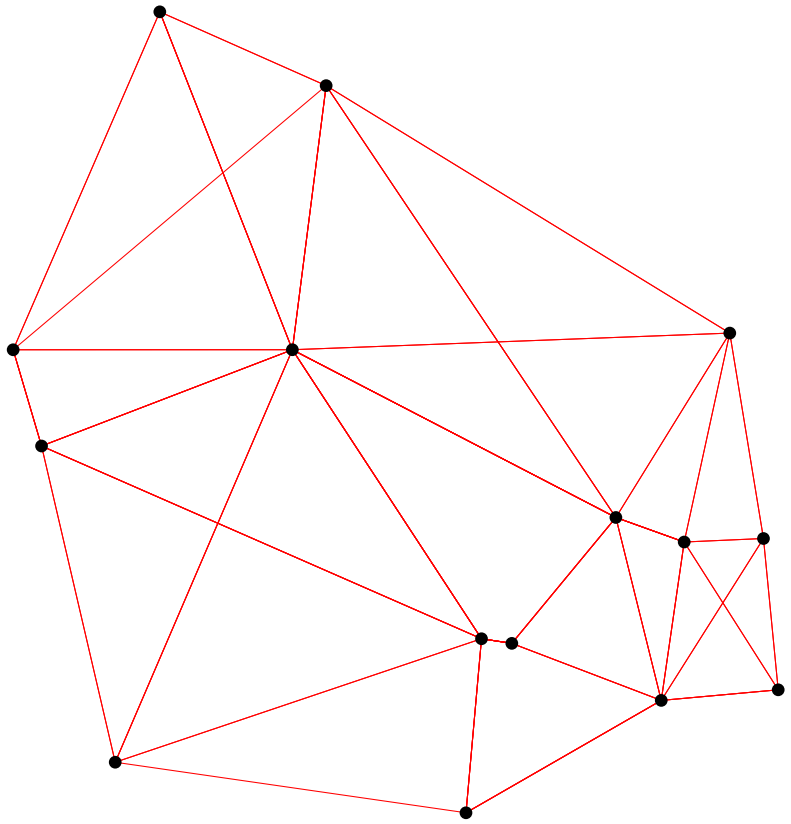
Minimum-weight t-spanner problem (MWSP): Construct a t-spanner in G with minimum total edge-weight.

Minimum-Weight Spanner Example



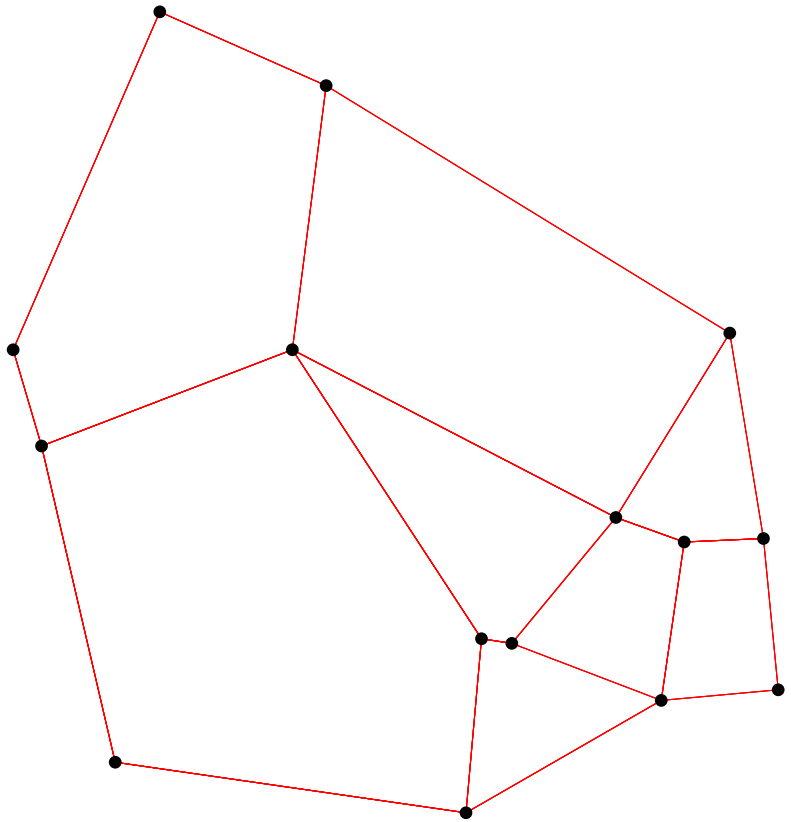
Complete Euclidean graph with 15 nodes

Minimum-Weight Spanner Example



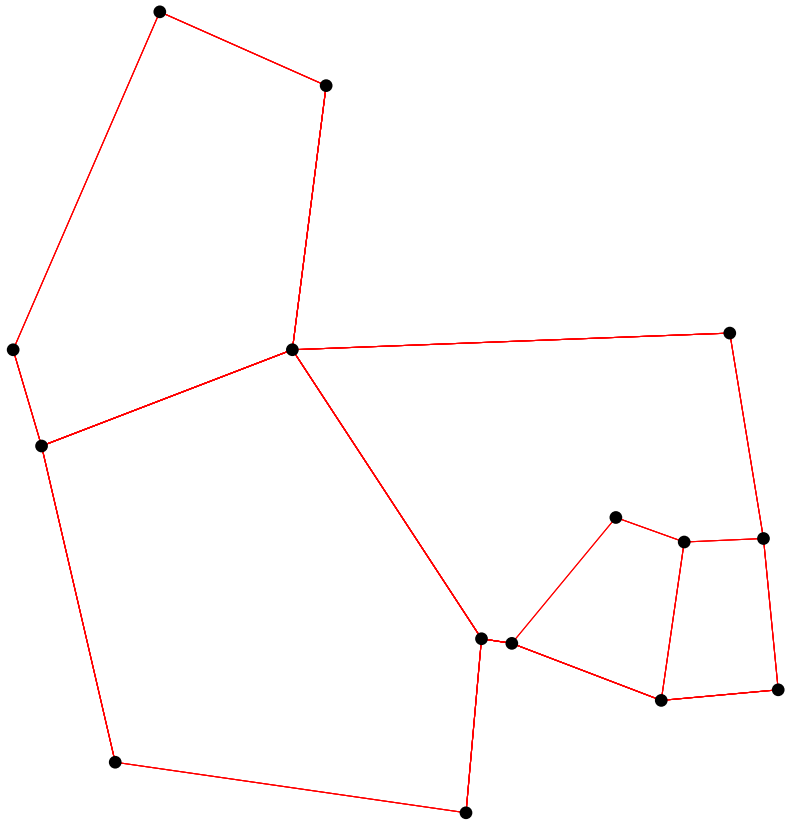
Minimum weight t-spanner for $t = 1.2$

Minimum-Weight Spanner Example



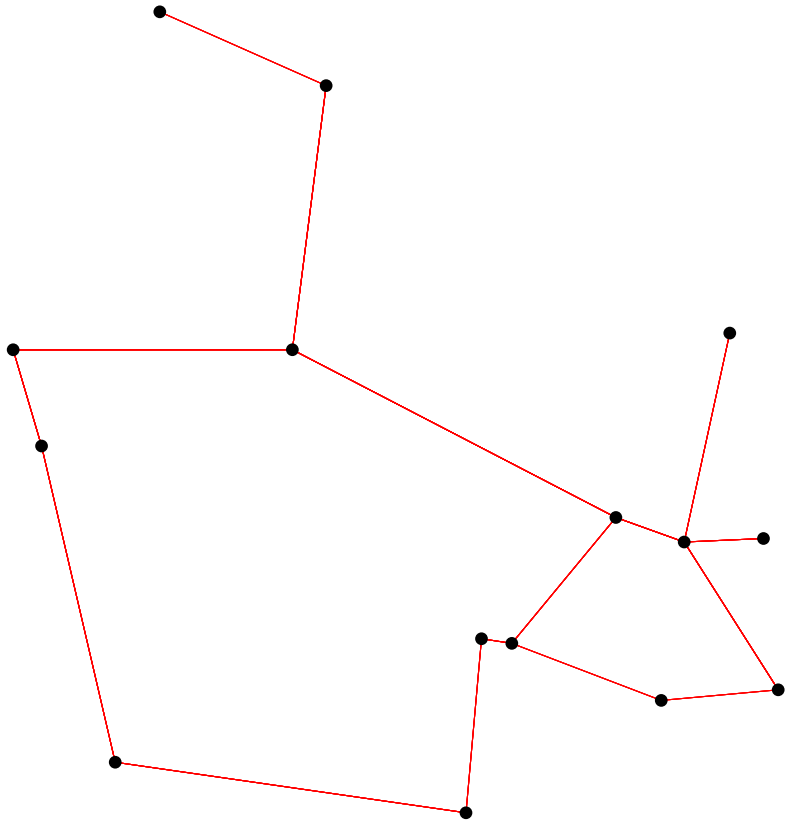
Minimum weight t-spanner for $t = 1.4$

Minimum-Weight Spanner Example



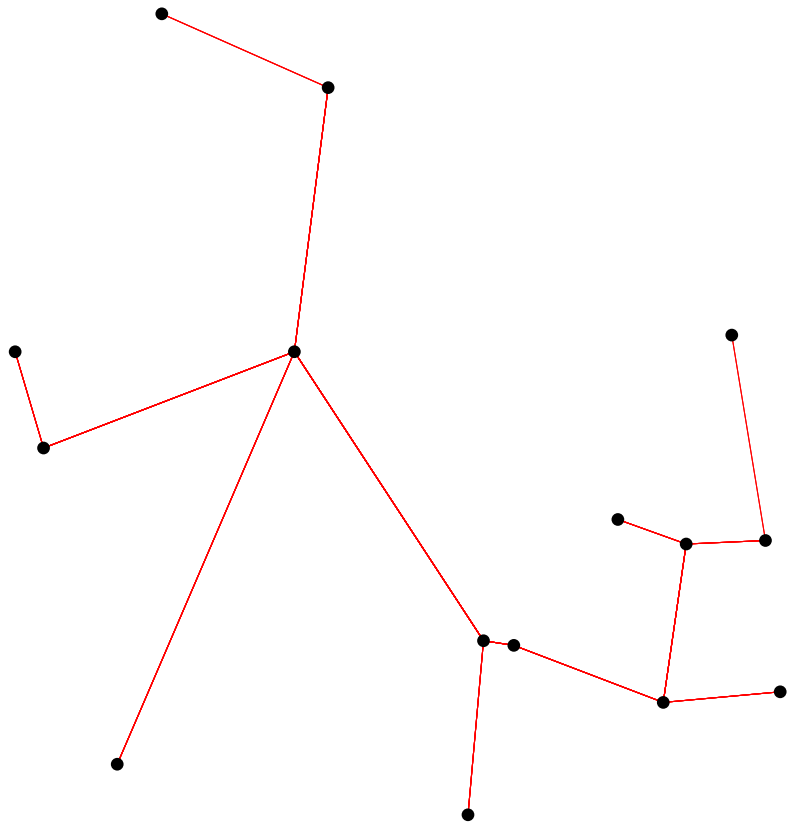
Minimum weight t-spanner for $t = 1.7$

Minimum-Weight Spanner Example



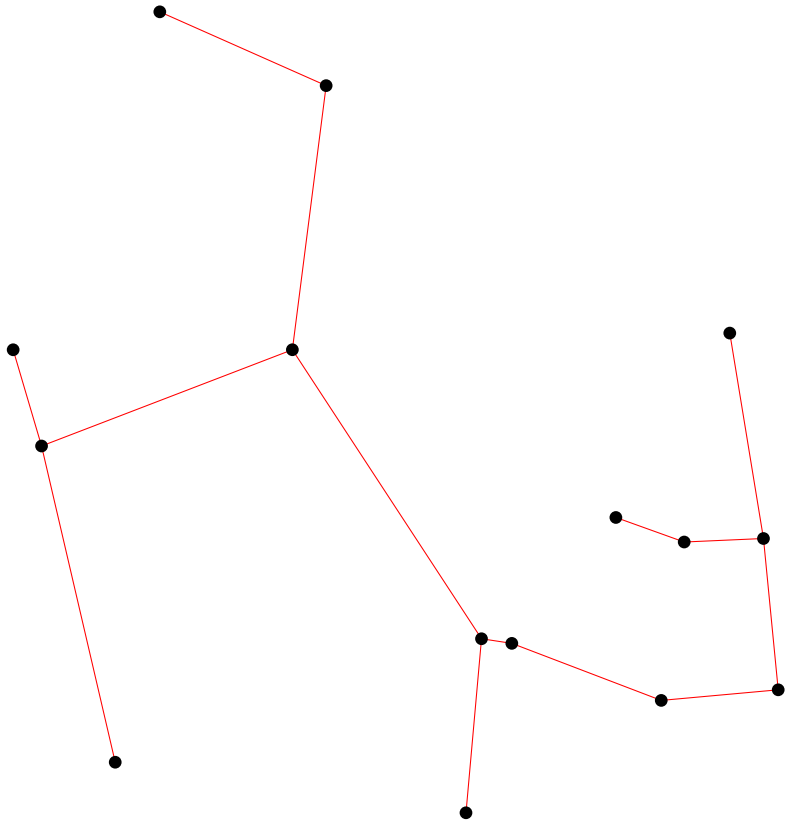
Minimum weight t-spanner for $t = 2$

Minimum-Weight Spanner Example



Minimum weight t-spanner for $t = 3$

Minimum-Weight Spanner Example



Minimum spanning tree (or minimum weight t-spanner for $t = \infty$)

Minimum-Weight Spanner Problem (MWSP)

Known to be an NP-hard problem [Cai, 1994].

Minimum-Weight Spanner Problem (MWSP)

Known to be an NP-hard problem [Cai, 1994].

Several approximation algorithms exist, in particular for the geometric versions of the problem; the greedy algorithm is particularly well studied.

All practical applications use some variant of the greedy algorithm.

Minimum-Weight Spanner Problem (MWSP)

Known to be an NP-hard problem [Cai, 1994].

Several approximation algorithms exist, in particular for the geometric versions of the problem; the greedy algorithm is particularly well studied.

All practical applications use some variant of the greedy algorithm.

No (non-trivial) exact algorithm is known.

Greedy Spanner Algorithm

1. Set $G' = (V, E')$ where $E' = \emptyset$.
2. Sort edges in $G = (V, E)$ by non-decreasing weight.
3. Process edges $e \in E$ in sorted order:
   - Let c_e be the weight of edge $e = (u, v)$.
   - If the shortest path between u and v in G' exceeds $t \cdot c_e$, then append $e = (u, v)$ to E'.

Greedy Spanner Algorithm

1. Set $G' = (V, E')$ where $E' = \emptyset$.
2. Sort edges in $G = (V, E)$ by non-decreasing weight.
3. Process edges $e \in E$ in sorted order:
   - Let c_e be the weight of edge $e = (u, v)$.
   - If the shortest path between u and v in G' exceeds $t \cdot c_e$, then append $e = (u, v)$ to E'.

Resulting t-spanner denoted greedy spanner; known to have weight $O(1)$ times that of a MST for points in fixed dimensional space.

[Das, Narasimhan & Salowe, 1995; Das & Narasimhan, 1997]

New Exact Algorithm for MWSP

Model a generalization of MWSP as an integer program.

New Exact Algorithm for MWSP

Model a generalization of MWSP as an integer program.

- $K = \{(u_i, v_i) | i = 1, \ldots, k\}$: Node pairs with length constraints.
- P_{uv}: Set of paths between u and v satisfying length constraint.
- $P = \bigcup_{(u,v) \in K} P_{uv}$: Set of all paths satisfying length constraint.
- $\delta_p^e = 1$ if edge $e \in E$ is on path $p \in P$.
- $x_e = 1$ if edge e is part of the solution.
- $y_p = 1$ if path p is part of the solution.

Integer Programming (IP) Formulation

$$\text{minimize} \quad \sum_{e \in E} c_e x_e \tag{1.1}$$

$$\text{subject to} \quad \sum_{p \in P_{uv}} y_p \delta_p^e \leq x_e \qquad \forall e \in E, \forall (u,v) \in K \tag{1.2}$$

$$\sum_{p \in P_{uv}} y_p \geq 1 \qquad \forall (u,v) \in K \tag{1.3}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{1.4}$$

$$y_p \in \{0,1\} \qquad \forall p \in P \tag{1.5}$$

Properties of the IP Model

- Proposed model contains an exponential number of variables (no useful polynomial sized model is known).
- Model provides a good LP-relaxation lower bound.
- LP-relaxation can be solved by delayed column generation.
- Integer solutions obtained by embedding lower bound computations in branch and bound.

IP Column Generation

- Work on a restricted linear program with fewer variables.
- Add variables iteratively according to Dantzig's pricing rule:
  Add variable with minimum reduced cost $c_p^{\pi,\sigma}$.
- Given dual variables π_e^{uv} and σ_{uv}, solve the pricing problem

$$\min_{p \in P} c_p^{\pi,\sigma} = \min_{p \in P_{uv}\ \forall (u,v) \in K} \sum_{e \in E} \delta_p^e \pi_e^{uv} - \sigma_{uv}$$

- The pricing problem is a constrained shortest path problem between u and v on the graph with edge costs π_e^{uv} and weights c_e.

Constrained Shortest Path Problem (CSPP)

- Find a shortest path from u to v with respect to the cost of edges such that the total weight is below a given threshold.
- NP-hard problem, but has a FPTAS [Warburton, 1987].

Constrained Shortest Path Problem (CSPP)

- Find a shortest path from u to v with respect to the cost of edges such that the total weight is below a given threshold.
- NP-hard problem, but has a FPTAS [Warburton, 1987].
- We solve CSPP by a labeling algorithm:
  - Enumerate all paths from u to v discarding dominated and infeasible paths.
  - Domination and infeasibility can be checked quickly by precomputing shortest path trees from v w.r.t. edge costs and weights, respectively.

Implementation Details

- Used ABACUS branch and price framework [Thienel, 1995].
- LP models solved by CPLEX 7.0.
- Computer: Pentium IV 3.0 GHz, 2GB memory
- Master LP contains $O(n^4)$ constraints. We solve master problem with fewer constraints, generating violated constraints iteratively.
- Branching on binary edge variables x_e.
- Each instance allowed 30 minutes of CPU time.

Two Types of Test Instances

Two Types of Test Instances

Euclidean: Complete Euclidean graphs on n vertices randomly drawn from a k-dimensional hypercube.

$n = 20, 30, 40, 50$ / $k = 5, 10, 20, 25$ / $t = 1.2, 1.4, 1.6, 1.8$

Two Types of Test Instances

Euclidean: Complete Euclidean graphs on n vertices randomly drawn from a k-dimensional hypercube.

$n = 20, 30, 40, 50$ / $k = 5, 10, 20, 25$ / $t = 1.2, 1.4, 1.6, 1.8$

Realistic: Similar to networks appearing in communication network applications: n nodes randomly distributed in a square, edges added according to/not according to locality, and edge costs random or Euclidean. Average degree D fixed.

$n = 16, 32, 64$ / $D = 4, 8$ / $t = 1.1, 2.0, 4.0$

Two Types of Test Instances

Euclidean: Complete Euclidean graphs on n vertices randomly drawn from a k-dimensional hypercube.

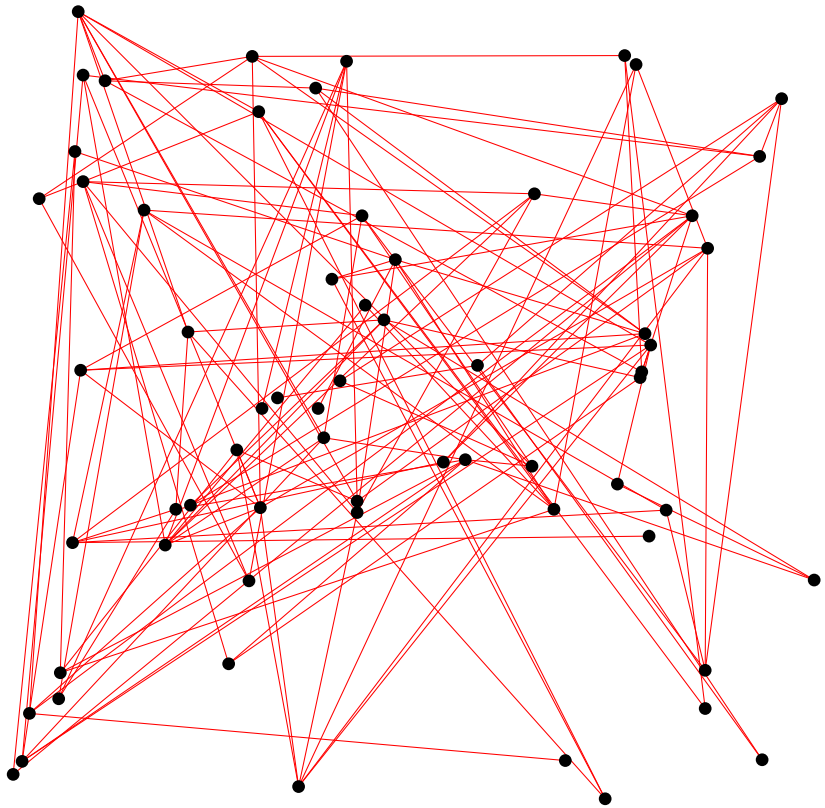
$n = 20, 30, 40, 50$ / $k = 5, 10, 20, 25$ / $t = 1.2, 1.4, 1.6, 1.8$

Realistic: Similar to networks appearing in communication network applications: n nodes randomly distributed in a square, edges added according to/not according to locality, and edge costs random or Euclidean. Average degree D fixed.

$n = 16, 32, 64$ / $D = 4, 8$ / $t = 1.1, 2.0, 4.0$

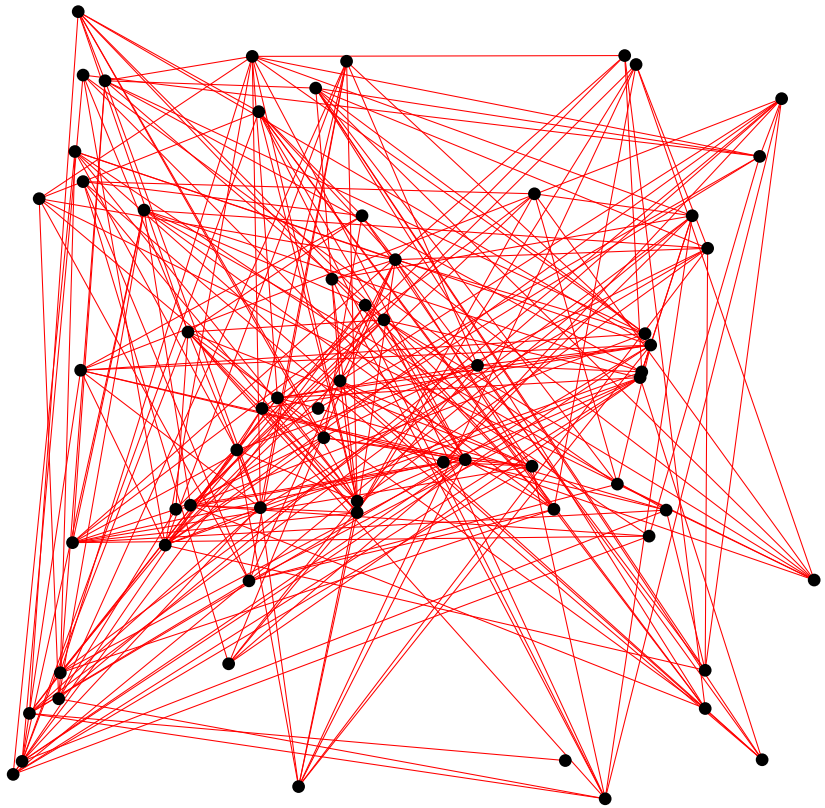
50 instances for every class/size of graphs.

Realistic Graph Examples



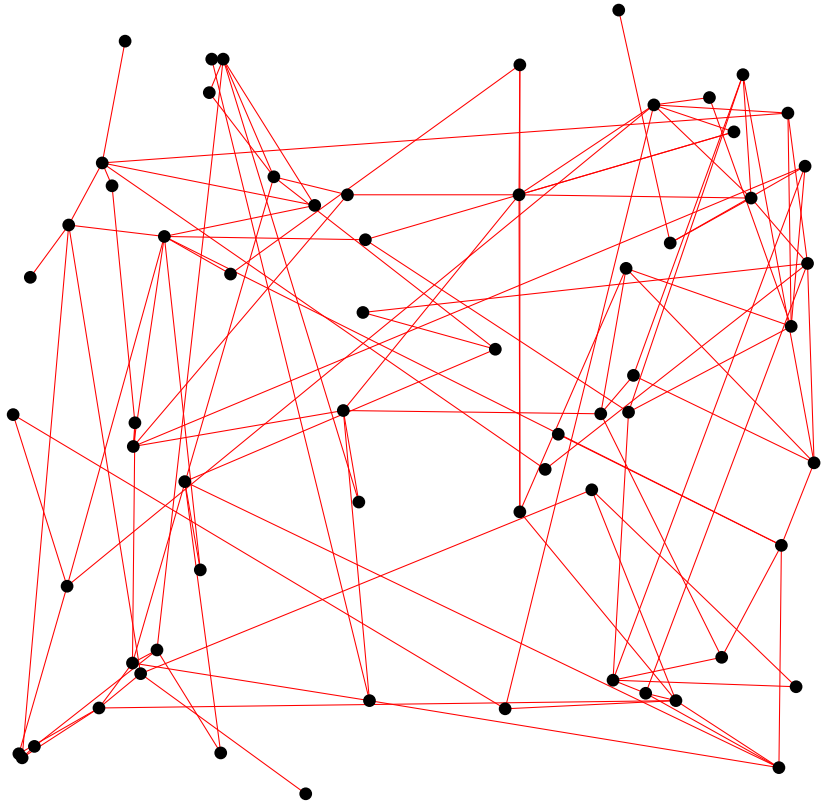
$n = 64$, $D = 4$, no locality of edges

Realistic Graph Examples



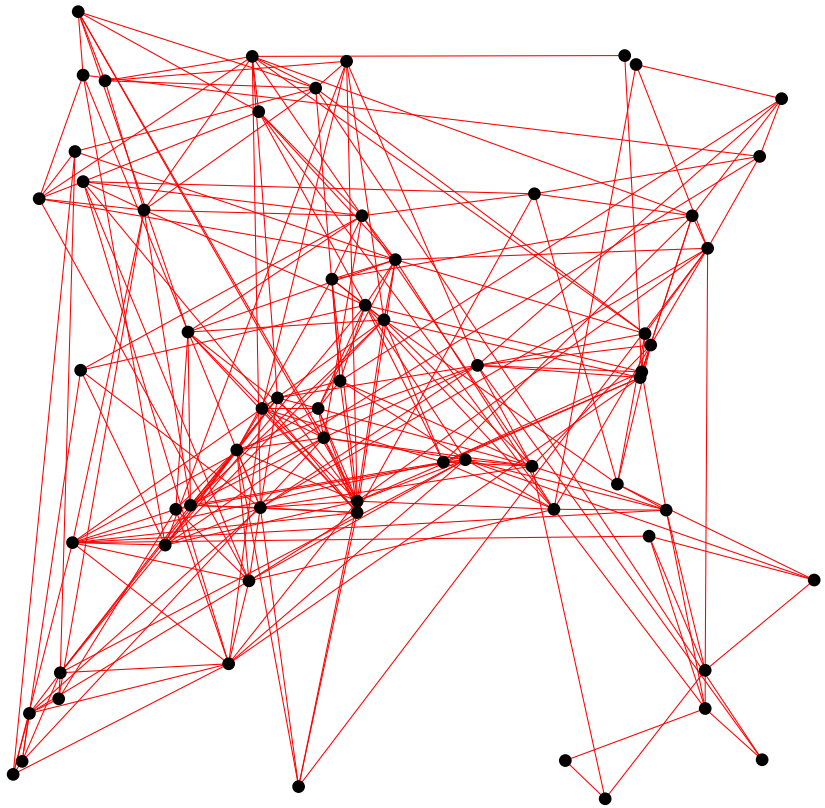
$n = 64$, $D = 8$, no locality of edges

Realistic Graph Examples



$n = 64$, $D = 4$, with locality of edges

Realistic Graph Examples



$n = 64$, $D = 8$, with locality of edges

Computational Results: Euclidean Graphs

# Nodes	20				30			
Dimension	5	10	20	25	5	10	20	25
$t = 1.2$	0.90	0.00	0.00	0.00	1.63	0.00	0.00	0.00
$t = 1.4$	7.61	0.83	0.00	0.00	10.88	1.20	0.01	0.00
$t = 1.6$	16.22	9.55	0.62	0.31	*20.11*	13.36	1.00	0.29
$t = 1.8$	21.99	33.03	18.45	13.68	-	*47.93*	*32.97*	*17.97*

# Nodes	40				50			
Dimension	5	10	20	25	5	10	20	25
$t = 1.2$	2.50	0.01	0.00	0.00	3.10	0.01	0.00	0.00
$t = 1.4$	13.00	1.43	0.00	0.00	*13.59*	1.75	0.01	0.00
$t = 1.6$	-	*15.39*	1.38	0.40	-	*18.44*	1.50	0.41
$t = 1.8$	-	-	-	-	-	-	-	-

Average excess from optimum of greedy spanner (in percent).

Computational Results: Realistic Graphs

Edge cost	Euclidean						Random					
Avg. deg.	4			8			4			8		
# Nodes	16	32	64	16	32	64	16	32	64	16	32	64
$t = 1.1$												
$\div$ locality	0.05	0.02	0.02	0.57	0.23	0.14	0.00	0.00	0.00	0.00	0.00	0.00
$+$ locality	0.41	0.12	0.09	1.43	1.40	1.00	0.00	0.00	0.00	0.00	0.00	0.00
$t = 2.0$												
$\div$ locality	3.25	2.22	1.29	4.71	8.48	4.35	2.76	1.17	2.46	4.22	3.29	3.33
$+$ locality	3.84	*2.73*	2.32	4.85	-	-	1.74	0.90	1.67	5.74	4.34	*3.71*
$t = 4.0$												
$\div$ locality	1.31	*6.29*	-	2.10	-	-	0.64	*4.72*	-	2.64	-	-
$+$ locality	1.43	-	-	-	-	-	0.00	-	-	3.95	-	-

Average excess from optimum of greedy spanner (in percent).

Generalizations handled by Exact Algorithm

MWSP with variable stretch factor. Allow different stretch factors for every pair of nodes. Easily handled by changing the definition of the feasible set of paths P_{uv} for a pair of nodes u and v.

MWSP with selected shortest path constraints. Only a subset of all pairs have maximum length constraints. As an example, only paths involving one particular node (the root) have maximum length constraints. Has applications in, e.g., VLSI design, where trees with both low cost and low delay are wanted.

Conclusions and Future Work

- Total weight of a greedy spanner is within a few percent from optimum for the set of realistic instances; for Euclidean graphs and large stretch factors the quality of the greedy spanner deteriorates significantly.
- It would be interesting to test the exact algorithm and the greedy spanner algorithm on a larger set of problem instances. Also, improvements to the greedy spanner algorithm would be worthwhile to investigate.