Datalogisk Institut
KØBENHAVNS UNIVERSITET

Construction of Minimum-Weight Spanners

Mikkel Sigurd and Martin Zachariasen

Technical Report no. 2004/05
ISSN: 0107-8283
CR Subject Classification: G.2.1, G.2.2

DIKU
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Construction of Minimum-Weight Spanners

Mikkel Sigurd and Martin Zachariasen

Department of Computer Science, University of Copenhagen
DK-2100 Copenhagen Ø, Denmark
{sigurd,martinz}@diku.dk.

Abstract. Spanners are sparse subgraphs that preserve distances up to a given factor in the underlying graph. Recently spanners have found important practical applications in metric space searching and message distribution in networks. These applications use some variant of the so-called *greedy* algorithm for constructing the spanner — an algorithm that mimics Kruskal's minimum spanning tree algorithm. Greedy spanners have nice theoretical properties, but their practical performance with respect to total weight is unknown. In this paper we give an exact algorithm for constructing minimum-weight spanners in arbitrary graphs. By using the solutions (and lower bounds) from this algorithm, we experimentally evaluate the performance of the greedy algorithm for a set of realistic problem instances.

1 Introduction

Let $G = (V, E)$ be an undirected and edge-weighted graph. A t-spanner in G is a subgraph $G' = (V, E')$ of G such that the shortest path between any pair of nodes $u, v \in V$ is at most t times longer in G' than in G (where $t > 1$) [20]. The NP-hard minimum-weight t-spanner problem (MWSP) is to construct a t-spanner with minimum total weight [3].

Spanners — and algorithms for their construction — form important building blocks in the design of efficient algorithms for geometric problems [8, 12, 17, 21]. Also, low-weight spanners have recently found interesting practical applications in areas such as metric space searching [19] and broadcasting in communication networks [9]. A spanner can be used as a compact data structure for holding information about (approximate) distances between pairs of objects in a large metric space, say, a collection of electronic documents; by using a spanner instead of a full distance matrix, significant space reductions can be obtained when using search algorithms like AESA [19]. For message distribution in networks, spanners can simultaneously offer both low cost and low delay when compared to existing alternatives such as minimum spanning trees (MSTs) and shortest path trees. Experiments with constructing spanners for realistic communication networks show that spanners can achieve a cost that is close to the cost of a MST while significantly reducing delay (or shortest paths between pairs of nodes) [9].

The classical algorithm for constructing a low-weight t-spanner for a graph $G = (V, E)$ is the *greedy* algorithm [1]. All practical applications of spanners use

some variant of this algorithm. The greedy algorithm first constructs a graph $G' = (V, E')$ where $E' = \emptyset$. Then the set of edges E is sorted by non-decreasing weight and the edges are processed one by one. An edge $e = (u, v)$ is appended to E' if the weight of a shortest path in G' between u and v exceeds $t \cdot c_e$, where c_e is the weight of edge e. (Note that the greedy algorithm is clearly a polynomial-time algorithm; for large values of t the algorithm is identical to Kruskal's algorithm for constructing MSTs.)

The output of the greedy algorithm is a so-called *greedy spanner*. A greedy spanner has several nice theoretical properties, which are summarized in Section 2. However, it is not known how well the greedy algorithm actually performs in practice, i.e., how well the greedy spanner approximates the minimum-weight spanner for realistic problem instances.

In this paper we present the first *exact* algorithm for MWSP. We give an integer programming formulation based on column generation (Section 3). The problem decomposes into a master problem which becomes a set covering like problem and a number of subproblems which are constrained shortest path problems. The integer program is solved by branch-and-bound with lower bounds obtained by linear programming relaxations.

We have performed experiments with the new exact algorithm on a range of problem instances (Section 4). Our focus has been on comparing greedy spanners with minimum-weight spanners (or lower bounds on their weights). The results show that the greedy spanner is a surprisingly good approximation to a minimum-weight spanner — on average within 5% from optimum for the set of problem instances considered. Our conclusions and suggestions for further work are given in Section 5.

2 Greedy Spanner Algorithm

Let $n = |V|$ and $m = |E|$ for our given graph $G = (V, E)$. For general graphs it is hard to approximate MWSP within a factor of $\Omega(\log n)$ [7]. However, for special types of graphs much better approximations can be obtained. Most of these are obtained by (variants of) the greedy algorithm.

The performance of the greedy algorithm depends on the weight function on the edges of G. For general graphs, Althöfer et al. [1] showed that the greedy algorithm produces a graph with $O(n^{1+2/(t-1)})$ edges; Chandra et al. [4] showed that the weight is bounded by $n^{(2+\epsilon)/(t-1)}$ times that of a minimum spanning tree (MST) for G (for any $\epsilon > 0$). These bounds improve to $n(1+O(1/t))$ and $1+O(1/t)$, respectively, when G is planar. When the nodes of G correspond to points in some fixed dimensional space (and edge weights are equal to the Euclidean distance between the nodes), the number of edges in the greedy spanner is $O(n)$ and the weight $O(1)$ times that of a MST; the constant implicit in the O-notation depends on t and the dimension of the space [5].

Although the bounds for the greedy algorithm are promising for planar and geometric graphs, to the best of our knowledge, little is known about the tight-

ness of these bounds and on the practical performance of the greedy algorithm on realistic problem instances.

A naive implementation of the greedy algorithm runs in $O(n^3 \log n)$ time. Significant improvements have been made, both from the theoretical and practical side, on this running time. Gudmundsson et al. [11] gave a $O(n \log n)$ variant of the greedy algorithm for points in fixed dimension. Navarro and Paredes [18] gave several practical variants of the greedy algorithm for general graphs with running times down to $O(nm \log m)$. They were able to compute approximative greedy spanners for graphs with several thousand nodes in a few minutes.

Finally, we note that a greedy spanner will always contain a MST for G [20]. On the contrary, a minimum-weight spanner need not contain a MST for the graph. An example is shown in Figure 1.

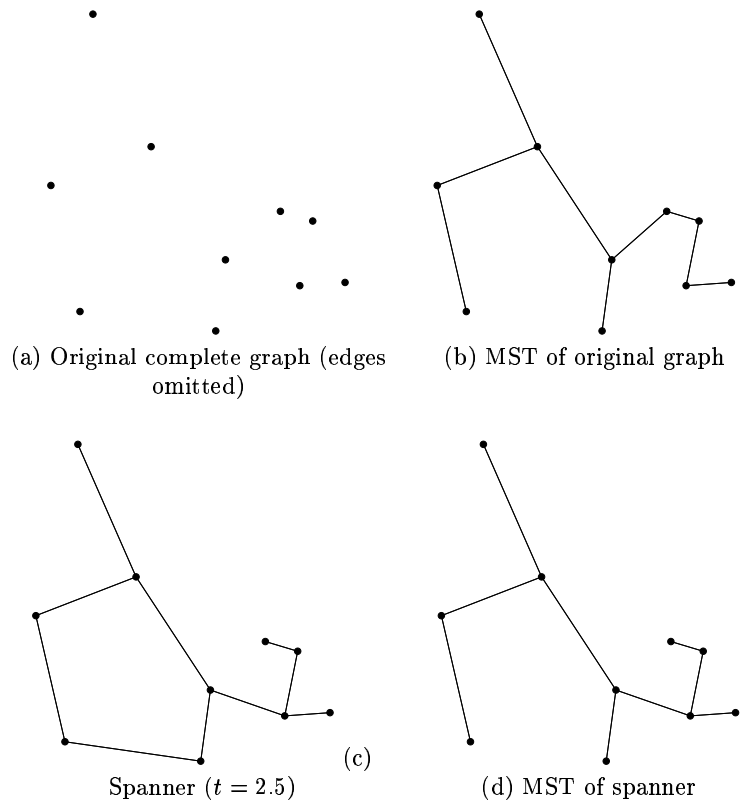

Fig. 1. An example showing that a minimum-weight spanner need not contain a MST of the original graph.

3 Column Generation Approach

In the new exact algorithm we consider the following generalization of MWSP. We are given a connected undirected graph $G = (V, E)$ with edge weights c_e, $e \in E$, a set of node pairs $K = \{(u_i, v_i)\}$, $i = 1, \ldots, k$, and a stretch factor $t > 1$. Let $p_{u_i v_i}$ denote a shortest path in G between u_i and v_i, $(u_i, v_i) \in K$, and let $c(p_{u_i v_i})$ denote the length of $p_{u_i v_i}$. The (generalized) minimum-weight spanner problem (MWSP) consists of finding a minimum cost subset of edges $E' \subseteq E$ such that the shortest paths $p'_{u_i v_i}$, $i = 1, \ldots, k$, in the new graph $G' = (V, E')$ are no longer than $t \cdot c(p_{u_i v_i})$. Note that this problem reduces to the normal MWSP when $K = V \times V$.

Obviously, if t is sufficiently large, an optimal solution to the MWSP will be a minimal spanning forest. On the other hand, if t is close to 1, an optimal solution will include all edges on the shortest paths $p_{u_i v_i}$ of G.

Our exact algorithm for solving MWSP is based on modeling the problem as an integer program, and solving this integer program by branch and bound using bounds from linear programming relaxations. MWSP can be modeled in several different ways, but the shortest path constraints are particularly difficult to model efficiently. Therefore, we have chosen a model in which *paths* are decision variables.

Given an instance of the MWSP, let P_{uv} denote the *set* of paths between u and v with length smaller than or equal to $t \cdot c(p_{uv})$, $\forall (u, v) \in K$, and let $P = \bigcup_{(u,v) \in K} P_{uv}$. Furthermore, let the indicator variables δ_p^e be defined as follows: $\delta_p^e = 1$, if edge $e \in E$ is on path $p \in P$ and $\delta_p^e = 0$ otherwise. Let the decision variables x_e, $e \in E$, equal 1 if edge e is part of the solution and 0 otherwise, and let decision variables y_p, $p \in P_{uv}$, equal 1 if path p connects u and v in the solution and 0 otherwise, $\forall (u, v) \in K$. Then MWSP can be formulated as follows:

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

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

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

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

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

The objective function (1.1) minimizes the total cost of the edges in the spanner. Constraints (1.2) require that for given a pair of nodes (u, v), all edges on the selected path connecting u and v must be part of the spanner. Constraints (1.3) say that every pair of nodes must be connected by at least one path.

We will solve the model with a branch and bound algorithm using the linear relaxation lower bound, which is obtained from (1) by relaxing the constraints (1.4) and (1.5). This model contains $|E| + |P|$ variables and $(|E| + 1)|K|$ constraints. Since the number of variables may be exponential in the input size, we

will not solve the relaxation of (1) directly, in fact we will not even write up the model. Instead, we will solve the *restricted master problem* (RMP):

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e & & \\
\text{subject to} \quad & \sum_{p \in P'_{uv}} y_p \delta_p^e \leq x_e & & \forall e \in E, \forall (u,v) \in K \\
& \sum_{p \in P'_{uv}} y_p \geq 1 & & \forall (u,v) \in K \\
& x_e \geq 0 & & \forall e \in E \\
& y_p \geq 0 & & \forall p \in P'
\end{aligned} \tag{2}$$

where $P'_{uv} \subseteq P_{uv}, P'_{uv} \neq \emptyset$, $\forall (u,v) \in K$ and $P' = \bigcup_{(u,v) \in K} P'_{uv}$. To begin with, it is sufficient that P'_{uv} contains only one path for every $(u,v) \in K$. Iteratively, we will add paths to P', extending the RMP. This procedure is known as *delayed column generation*. We use the Simplex method to solve the RMP in every iteration of the delayed column generation, which provides us with an optimal dual vector of the RMP. Now, consider the dual linear program of the full primal problem with dual variables π_e^{uv} and σ_{uv}, $\forall (u,v) \in K$, $\forall e \in E$:

$$\begin{aligned}
\text{maximize} \quad & \sum_{(u,v) \in K} \sigma_{uv} & & \\
\text{subject to} \quad & \sum_{(u,v) \in K} \pi_e^{uv} \leq c_e & & \forall e \in E \\
& -\sum_{e \in E} \delta_p^e \pi_e^{uv} + \sigma_{uv} \leq 0 & & \forall p \in P_{uv}, \forall (u,v) \in K \\
& \pi_e^{uv} \geq 0 & & \forall e \in E, \forall (u,v) \in K \\
& \sigma_{uv} \geq 0 & & \forall (u,v) \in K
\end{aligned} \tag{3}$$

If the dual values obtained by solving the RMP constitute a feasible solution to (3), the weak duality theorem tells us that dual solution is an optimal solution to (3), which implies that the optimal solution to the RMP is also an optimal solution to the full problem. Otherwise the dual solution to the RMP violates one of the constraints $-\sum_{e \in E} \delta_p^e \pi_e^{uv} + \sigma_{uv} \leq 0$ of (3) for some $p \in P$. The amount of violation is called the *reduced cost* of $p \in P$ with respect to π and σ, denoted $c_p^{\pi,\sigma}$:

$$c_p^{\pi,\sigma} = \sum_{e \in E} \delta_p^e \pi_e^{uv} - \sigma_{uv}, \quad \forall p \in P_{uv}, \ \forall (u,v) \in K. \tag{4}$$

If $c_p^{\pi,\sigma} \geq 0$, $\forall p \in P$ the dual vector constitutes a feasible solution to (3), which in turn means that the current primal solution to the RMP is an optimal solution for the full master problem. In this case we may stop the delayed column

generation procedure. Otherwise we add a path p with $c_p^{\pi,\sigma} < 0$ to the RMP, which corresponds to adding a violated dual constraint to the dual of the RMP. Hence, in every iteration of the delayed column generation procedure we solve problem $\min_{p \in P} c_p^{\pi,\sigma} = \min_{p \in P} \sum_{e \in E} \delta_p^e \pi_e^{uv} - \sigma_{uv}$, which is called the *pricing problem*. For a given $(u, v) \in K$, σ_{uv} is a constant and hence the pricing problem is the problem of finding a shortest path $p \in P_{uv}$ between u and v with edge weights π_e^{uv}. This problem is recognized as the *constrained shortest path problem* (CSPP), since we require that $p \in P_{uv}$, i.e., that the length of p is no longer than t times the length of the shortest path between u and v with respect to the original edge costs.

3.1 Constrained Shortest Path Problem

The (resource) constrained shortest path problem (CSPP) is a generalization of the classic shortest path problem, where we impose a number of additional resource constraints. Even in the case of a single resource constraint the problem is weakly NP-complete [10]. The problem is formally stated as follows: Given a graph $G = (V, E)$ with a weight $w_e \geq 0$ and a cost $c_e \geq 0$ on every edge $e \in E$, a source $s \in V$ and a target $t \in V$ and a resource limit B, find a shortest path from s to t with respect to the *cost* of the edges, satisfying that the sum of the *weights* on the path is at most B.

Several algorithms have been proposed for its solution (see [26] for a survey). Algorithms based on dynamic programming were presented in [2, 15]. A *labelling* algorithm that exploits the dominance relation between paths (i.e. path p dominates path q if it has cost $c_p \leq c_q$ and weight $w_p \leq w_q$) was proposed in [6]. Lagrangean relaxation has been used successfully to move the difficult resource constraint into the objective function in a two phase method, first solving the Lagrangean relaxation problem and then closing the duality gap [13, 26]. The CSP can be approximated to any degree by polynomial algorithms by the *fully polynomial time approximation scheme* (FPTAS) presented in [24].

We use a labelling algorithm similar to [6] with the addition of a method of early termination of infeasible paths. Our algorithm grows paths starting from the source s until we reach the target t. In the algorithm we represent a path p_i by a label (p_i, n_i, w_i, c_i), stating that the path p_i ending in node n_i has total weight w_i and total cost c_i. If two paths p_1, p_2 end in the same node, p_1 dominates p_2 if $w_1 \leq w_2$ and $c_1 \leq c_2$. We may discard all dominated labels since they cannot be extended to optimal paths.

In our algorithm, we store the labels (paths) in a heap H ordered by the cost of the path. Iteratively, we pick the cheapest path from the heap and extend it along all edges incident to its endpoint. If the extended path does not violate the maximum weight constraint, we create a new label for this path. We remove all dominated labels, both labels generated earlier that are dominated by the new labels and new labels which are dominated by labels generated earlier. This is done efficiently by maintaining a sorted list for every node n consisting of all labels for node n. When creating a new label, we only need to run through this list to remove dominated labels. When we reach the target node t, we know we

have created the cheapest path satisfying the maximum weight constraint since we pick the cheapest label in every iteration.

The algorithm above is the standard labelling algorithm. Before we start the labelling algorithm we do two shortest path computations, creating a shortest path tree T^c from t with respect to edge costs c_e and a shortest path tree T^w from t with respect to edge weights w_e. Whenever we extend a path to a node n, we check if the new label (p, n, w, c) satisfies $w + T^w(n) \leq B$ and $c + T^c(n) \leq M$, where M is the maximum cost constraint imposed on the problem (note that we are only interested in finding paths with reduced cost $c_p^{\pi,\sigma} \leq 0$). If one of the inequalities is not satisfied we can discard this path, since it will not be able to reach t within the maximum weight and cost limits. The algorithm is sketched below.

CSP(G, w, c, B, M, s, t)

1. Compute shortest path trees T^w and T^c.
2. $H \leftarrow (\{\}, s, 0, 0)$
3. Pick a cheapest label (p, n, w, c) in H. If $n = t$ then we are done, p is the cheapest path satisfying the maximum weight constraint.
4. For all nodes n_i adjacent to n by edge e_i, extend the path and create new labels $(p \cup \{e_i\}, n_i, w + w_{e_i}, c + c_{e_i})$. Discard new labels where $w + w_{e_i} + T^w(n_i) > B$ or $c + c_{e_i} + T^c(n_i) > M$. Discard dominated labels. Goto 3.

In every iteration of the column generation procedure, we run the CSP algorithm for every pair of nodes (u, v) with edge costs π_e^{uv} and edges weights c_e, maximum cost limit $M = \sigma^{uv}$ and maximum weight limit $B = t \cdot c(p_{uv})$, where $c(p_{uv})$ is the cost of a shortest path from u to v with respect to edge costs c_e. Note that it is not possible to run an "all constrained shortest path" algorithm, since the edge costs π_e^{uv} differ for every pair of nodes.

3.2 Implementation Details

Delayed column generation allows us to work on a LP which only contains a small number of variables. However, our restricted master problem (2) still contains a large number of constraints (for complete graphs there will be $O(n^4)$ constraints). The constraints are all needed to ensure primal feasibility of the model, but in practice only a small number of the constraints will be active. We will remove the constraints of the form $\sum_{p \in P_{uv}} y_p \delta_p^e \leq x_e$ from the RMP to reduce the size of the master LP. This will allow infeasible solutions, but we will check if any violated constraints exist and add these constraints to the model.

Checking whether a constraint is violated is straightforward: for all node pairs $(u, v) \in K$ and all edges $e \in E$, we compute the sum $\sum_{p \in P_{uv}} y_p \delta_p^e$ and check whether this quantity is greater than x_e. If so, we add the constraint $\sum_{p \in P_{uv}} y_p \delta_p^e \leq x_e$ to the model. Computational experiments show that we only need to add a small fraction of the constraints to the model. This greatly speeds up the simplex iterations in the algorithm.

Branching is done by selecting an edge variable x_e with maximum value in the fractional solution and demanding $x_e = 1$ on one branch and $x_e = 0$ on the other. On the $x_e = 0$ branch the pricing problem changes a little bit, since edge e is not allowed in any of the paths generated. This can be handled very efficiently by deleting the edge from the graph on which the pricing algorithm runs. The $x_e = 1$ branch does not change the pricing problem. We evaluate the branch and bound nodes in depth-first order.

We have used the general branch-and-cut framework ABACUS [23] with CPLEX [14] in the implementation of our algorithm.

4 Experimental Results

One of the main contributions of this paper is to evaluate the quality of the greedy spanner algorithm on a set of relevant problem instances. We have created two types of graphs, denoted *Euclidean* graphs and *realistic* graphs, respectively.

The *Euclidean* graphs are generated as follows. The vertices correspond to points that are randomly and uniformly distributed in a k-dimensional hypercube. Then a complete Euclidean graph is constructed from the set of points. Graphs classes for all combinations of dimensions $k = 5, 10, 20, 25$ and $n = 20, 30, 40, 50$ have been generated. For every class of graphs, we have created 50 instances, which we have solved by both the greedy spanner algorithm and our exact method with stretch factor $t = 1.2, 1.4, 1.6, 1.8$.

The *realistic* graphs originate from the application in broadcasting and multicasting in communications networks, where spanners may be used to reduce the cost of sending data while preserving low delay. We have created a set of instances following the design proposed in [9, 25]. For $n = 16, 32, 64$ we place n nodes uniformly random in a square. We add edges according to one of two different schemes:

1. Place edges completely at random where the probability of adding an edge is the same for all pairs of nodes.
2. Favour local edges so that the probability of adding an edge between u and v is $\alpha e^{-d/\beta L}$, where d is the distance between u and v and L is the diameter of the square.

By varying the probability in method 1, and α and β in method 2, we generate graphs with average node degree $D = 4$ and $D = 8$, respectively. To model realistic networks, we keep $\beta = 0.14$ fixed and set α to get the desired average node degree for $n = 16, 32, 64$ [9].

We assign a cost to every edge in the graph according to one of three methods:

1. Unit cost. All edges have cost 1.
2. Euclidean distance. All edges have cost equal to the Euclidean distance between the endpoints.
3. Random cost. Edge cost is assigned at random from the set $\{1, 2, 4, 8, 16\}$ to all edges.

Table 1. Average excess from optimum of greedy spanner (in percent) for Euclidean graphs

# Nodes	20				30				40				50			
Dimension	5	10	20	25	5	10	20	25	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	2.50	0.01	0.00	0.00	3.10	0.01	0.00	0.00
$t = 1.4$	7.61	0.83	0.00	0.00	10.88	1.20	0.01	0.00	13.00	1.43	0.00	0.00	*13.59*	1.75	0.01	0.00
$t = 1.6$	16.22	9.55	0.62	0.31	*20.11*	13.36	1.00	0.29	-	*15.39*	1.38	0.40	-	*18.44*	1.50	0.41
$t = 1.8$	21.99	33.03	18.45	13.68	-	*47.93*	*32.97*	*17.97*	-	-	-	-	-	-	-	-

Table 2. Average excess from optimum of greedy spanner (in percent) for realistic graphs

Edge cost	Unit						Euclidean						Random					
Avg. degree	4			8			4			8			4			8		
# Nodes	16	32	64	16	32	64	16	32	64	16	32	64	16	32	64	16	32	64
$t = 1.1$																		
÷ locality	0.00	0.00	0.00	0.00	0.00	0.00	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	0.00	0.00	0.00	0.00	0.00	0.00	0.41	0.12	0.09	1.43	1.40	1.00	0.00	0.00	0.00	0.00	0.00	0.00
$t = 2.0$																		
÷ locality	7.63	*5.35*	-	-	-	-	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	16.84	*15.30*	-	-	-	-	3.84	*2.73*	2.32	4.85	-	-	1.74	0.90	1.67	5.74	4.34	*3.71*
$t = 4.0$																		
÷ locality	*13.45*	-	-	-	-	-	1.31	*6.29*	-	2.10	-	-	0.64	*4.72*	-	2.64	-	-
+ locality	*9.66*	-	-	-	-	-	1.43	-	-	-	-	-	0.00	-	-	3.95	-	-

For every class of graphs, we have created 50 instances, which we have solved by both the greedy spanner algorithm and our exact method with stretch factor $t = 1.1, 2.0, 4.0$. The tests have been carried out on a 933 MHz Intel Pentium III computer, allowing each instance 30 minutes of CPU time.

In the Tables 1 and 2 we show the greedy spanner's average excess from optimum (in percent). Note the slightly different layout of these two tables, in particular with respect to the number of nodes. For some instances we have compared the greedy spanner with a lower bound on the minimum weight spanner, since we did not obtain the optimal value within the CPU time limit. This means that the true average excess from optimum may be a bit lower than stated in the table; the results for these test classes are written in italics. Some of the problem classes have not been solved since the exact method was too time consuming.

The greedy algorithm performs significantly worse on Euclidean graphs than on realistic graphs. Also, the performance of the greedy algorithm decreases as the stretch factor increases. Interestingly, the greedy spanner does not necessarily become worse for larger problem instances; for almost all test classes considered,

the average excess from optimum decreases as the problem instance increases. When local edges are favored in the realistic problem instances, the quality of the greedy spanner decreases for unit and Euclidean costs.

The running times are dependent on the size of the problem, the edge cost function and the stretch factor. Naturally, the spanner problem becomes harder to solve for larger problem instances. It turns out that spanner problems with unit edge costs are considerably harder to solve by our exact method than spanner problems with Euclidean costs or random costs. The difficulties for problems with unit cost are caused by the large number of paths of equal cost, which makes it hard for the branch and bound algorithm to choose the best path. Thus, we need to evaluate more branch and bound nodes to solve unit cost problems.

We experienced that our solution times increased when the stretch factor increased. This is because the set of feasible paths between any two nodes increases, potentially increasing the number of columns in our master problem. On 75% of the test instances the lower bound provided in the root node was optimal. On the remaining 25% of the test instances, the lower bound in the root node was 6.4% from the optimum on average. This shows the quality of the proposed integer programming formulation.

5 Conclusions

In this paper we presented a first exact algorithm for the MWSP. Experiments on a set of realistic problem instances from applications in message distribution in networks were reported. Problem instances with up to 64 nodes have been solved to optimality. The results show that the total weight of a spanner constructed by the greedy spanner algorithm is typically within a few percent from optimum for the set of realistic problem instances considered. Thus the greedy spanner algorithm appears to be an excellent choice for constructing approximate minimum-weight spanners in practice.

5.1 Generalizations of the MWSP

The minimum-weight spanner problem may be seen as a special case of a larger class of network problems with shortest path constraints. This class of problems is normally hard to model due to the difficult shortest path constraints.

The exact method we present in this paper can easily be modified to solve this larger class of network problems with shortest path constraints. Here we describe how our method can be applied to several generalizations of the MWSP.

MWSP with variable stretch factor. Consider the MWSP as defined above, but with variable stretch factor for every pair of nodes. This is a generalization of the MWSP. Our method can easily be modified to solve this problem, since we already generate different variables for every pair of nodes satisfying the maximum cost constraint. Thus, it is possible to allow different cost constraints for every pair of nodes.

MWSP with selected shortest path constraints. Consider the MWSP where we do not have maximum length constraints on all pairs of nodes. The solution to this problem may be a disconnected graph consisting of several smaller spanners. This generalization can also be used to model problems where there are maximum cost constraints between a root node and all other nodes [16]. Our method can be modified to solve this problem by removing the covering constraints (1.3) for pairs of nodes (u, v) which have no shortest path constraints imposed.

5.2 Future Work

The main focus in this paper has been to evaluate the quality of the greedy algorithm, and to present a framework for a promising exact algorithm. It would interesting to evaluate our algorithm on a wider set of problem instances, and to examine the practical performance of other heuristic spanner algorithms, including variants of the greedy algorithm (e.g., in which simple local improvements are made in the greedy spanner), Θ-graphs and well-separated pair decompositions.

Clearly, there is still room for improvements on the exact algorithm. Providing our exact method with a good upper bound computed by, e.g., the greedy algorithm will probably help to prune the branch and bound tree. By adding more variables in each column generation iteration the procedure may be accelerated. Finally, column generation stabilization will most likely have an accelerating effect [22].

References

1. I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On Sparse Spanners of Weighted Graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.
2. Y. P. Aneja, V. Aggarwal, and K. P. K. Nair. Shortest Chain Subject to Side Constraints. *Networks*, 13:295–302, 1983.
3. L. Cai. NP-Completeness of Minimum Spanner Problem. *Discrete Applied Mathematics*, 48:187–194, 1994.
4. B. Chandra, G. Das, G. Narasimhan, and J. Soares. New Sparseness Results on Graph Spanners. *International Journal of Computational Geometry and Applications*, 5:125–144, 1995.
5. G. Das and G. Narasimhan. A Fast Algorithm for Constructing Sparse Euclidean Spanners. *Int. J. Comput. Geometry Appl.*, 7(4):297–315, 1997.
6. M. Desrochers and F. Soumis. A Generalized Permanent Labelling Algorithm for the Shortest Path Problem with Time Windows. *INFOR*, 26:191–211, 1988.
7. Y. Dodis and S. Khanna. Design Networks with Bounded Pairwise Distance. In *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*, pages 750–759, 1999.
8. D. Eppstein. Spanning Trees and Spanners. In *Handbook of Computational Geometry*, pages 425–461, 1999.
9. A. M. Farley, D. Zappala A. Proskurowski, and K. Windisch. Spanners and Message Distribution in Networks. *Discrete Applied Mathematics*, 137:159–171, 2004.

10. M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H.Freeman and Co., San Francisco, 1979.
11. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast Greedy Algorithms for Constructing Sparse Geometric Spanners. *SIAM Journal on Computing*, 31(5):1479–1500, 2002.
12. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate Distance Oracles for Geometric Graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 828–837, 2002.
13. G. Y. Handler and I. Zang. A Dual Algorithm for the Constrained Shortest Path Problem. *Networks*, 10:293–310, 1980.
14. *ILOG CPLEX 7.0, Reference Manual.* ILOG, S.A., France, 2000.
15. H.C. Joksch. The Shortest Route Problem with Constraints. *J. Math. Anal. Appl.*, 14:191–197, 1966.
16. S. Khuller, B. Raghavachari, and N. Young. Balancing Minimum Spanning and Shortest-Path Trees. *Algorithmica*, 14(4):305–321, 1995.
17. G. Narasimhan and M. Zachariasen. Geometric Minimum Spanning Trees via Well-Separated Pair Decompositions. *ACM Journal of Experimental Algorithmics*, 6, 2001.
18. G. Navarro and R. Paredes. Practical Construction of Metric t-Spanners. In *Proc. 5th Workshop on Algorithm Engineering and Experiments (ALENEX'03)*, 2003.
19. G. Navarro, R. Paredes, and E. Chavez. t-Spanners as a Data Structure for Metric Space Searching. In *International Symposium on String Processing and Information Retrieval, SPIRE, LNCS 2476*, pages 298–309, 2002.
20. D. Peleg and A. Schaffer. Graph Spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.
21. S. B. Rao and W. D. Smith. Improved Approximation Schemes for Geometrical Graphs via "Spanners" and "Banyans". In *Proc. Thirtieth Annual ACM Symposium on Theory of Computing*, pages 540–550, 1998.
22. M. Sigurd and D. Ryan. Stabilized Column Generation for Set Partitioning Optimization. In preparation.
23. S. Thienel. *ABACUS — A Branch-And-CUt System.* PhD thesis, Universität zu Köln, Germany, 1995.
24. A. Warburton. Approximation of Pareto Optima in Multiple–Objective, Shortest Path Problems. *Operations Research*, 35:70–79, 1987.
25. B. M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–22, 1988.
26. M. Ziegelmann. *Constrained Shortest Paths and Related Problems.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2001.