Genetic Algorithms for the *Tree T-Spanner* Problem

Riham Moharam, Ehab Morsy and Ismail A. Ismail

Abstract The *tree t-spanner* problem is one of the most important spanning tree optimization problems and has different applications in communication networks and distributed systems. Let $G = (V, E)$ be an undirected edge-weighted $G = (V, E)$ with vertex set V and edge set E. We consider the problem of constructing a *tree t-spanner* T in G in the sense that the distance between every pair of vertices in T is at most t times the shortest distance between the two vertices in G. The value of t, called the stretch factor, quantifies the quality of the distance approximation of the corresponding *tree t-spanner*. The problem of finding a *tree t-spanner* with the smallest possible value of t is known as the Minimum Maximum Stretch Spanning Tree (MMST) problem. It is well known that, for any $t \geq 1$, the problem of deciding whether G contains a *tree t-spanner* is NP-complete, thus, the MMST problem is NP-complete. In this paper, we present a genetic algorithm that returns a high quality solution for the MMST problem.

Keywords Tree Spanner · Stretch Factor · Minimum Maximum Stretch Spanning Tree · Genetic Algorithms · Graph Algorithms

This work is partially supported by Alexander von Humboldt foundation.

R. Moharam (✉) · E. Morsy
Department of Mathematics, Suez Canal University, Ismailia 41522, Egypt
e-mail: Riham.Sci@gmail.com

E. Morsy
e-mail: ehabmorsy@gmail.com

I.A. Ismail
Faculty of Computer and Information, Department of Computer Sciences,
6 October University, Giza, Egypt
e-mail: amr442-2@hotmail.com

T. Gaber et al. (eds.), *The 1st International Conference on Advanced Intelligent System and Informatics (AISI2015), November 28–30, 2015, Beni Suef, Egypt,*
Advances in Intelligent Systems and Computing 407, DOI 10.1007/978-3-319-26690-9_39

1 Introduction

Let $G = (V, E)$ be an undirected edge-weighted graph with vertex set V and edge set E such that $|V| = n$ and $|E| = m$. A spanning tree T in G is said to be a *tree t-spanner* if the distance between every pair of vertices in T is at most t times the shortest distance between the two vertices in G. For a given spanning tree T in G, the goodness of the distance approximation of T is estimated by the value of the stretch factor t of T [1]. The problem of finding a *tree t-spanner* with the smallest possible value of t is known as the Minimum Maximum Stretch Spanning Tree (MMST) problem [2].

The *tree t-spanner* problem is widely applied in communication networks and distributed systems. For example, it is applied to the arrow distributed directory protocol that supports the mobile object routing [3]. In particular, the MMST is used to minimize the delay of mobile object routing from the source node to every client node in case of concurrent requests through a routing tree. The worst case overhead the ratio of the protocol is proportional to the maximum stretch factor of T (see [4]). Kuhn and Wattenhofer [5] showed that the arrow protocol is a distributed ordering algorithm with low maximum stretch factor. Another application of the MMST is in the analysis of competitive concurrent distributed queuing protocols that intend to minimize the message transit in a routing tree [6].

For any $t \geq 1$, the problem of deciding whether G contains a *tree t-spanner* is NP-complete [7]. Consequently, the MMST problem, the problem of finding *a tree t-spanner* that minimizes t, is NP-complete. In this paper we present an efficient genetic algorithm to the MMST problem. Our experimental results show that the proposed algorithm returns high quality *tree t-spanner*.

The rest of this paper is organized as follows. Section 2 reviews some results on the *tree t-spanner* and related problems. Section 3 presents the proposed genetic algorithm. Section 4 evaluates our algorithm by applying it to randomly generated instances of a *tree t-spanner* problem. Section 5 makes some concluding remarks.

2 Related Work

In this section, we present results on related problems.

For an unweighted graph G, Cai and Corneil [7] produced a linear time algorithm to find a *tree t-spanner* in G for any given $t \geq 2$. Moreover, they showed that, for any $t \geq 4$, the problem of finding a *tree t-spanner* in G is NP-complete. Brandstädt et al. [8, 9] improved the hardness result in [7] by showing that a *tree t-spanner* is NP-complete even over chordal graphs which each created a cycle with length 3 whenever $t \geq 4$ and chordal bipartite graphs which each created a cycle with length 4 whenever $t \geq 5$.

Peleg and Tendler [10] proposed a polynomial time algorithm to determine a minimum value for t for the *tree t-spanner* over outerplanar graphs. In [11], Fekete and

Kremer showed that it is NP-hard to determine a minimum value for t for which a *tree t-spanner* exists even for planar unweighted graphs. They designed a polynomial time algorithm that decides if the planar unweighted graphs with bounded face length contains a *tree t-spanner* for any fixed t. Moreover, they proved that for $t = 3$, it can be decided whether the unweighted planar graph has a *tree t-spanner* in polynomial time. The problem was left open whether a *tree t-spanner* is polynomial time solvable in case of $t \geq 4$. Afterwards, this open problem is solved by Dragan et al. [12]. They proved that, for any fixed t, the *tree t-spanner* problem is linear time solvable not only for a planar graphs, but also for the class of sparse graphs which include graphs of bounded genus. Emek and Peleg [2] presented an $O(\log n)$-approximation algorithm for finding the *tree t-spanner* problem in a graph of size n. Moreover, they established that unless P = NP, the problem cannot be approximated additively by any o(n) term.

Recently, Dragan and Köhler [13] examined the *tree t-spanner* on chordal graphs, generalized chordal graphs and general graphs. For every n-vertex m-edge unweighted graph G, they proposed a new algorithm constructs a tree $(2\lfloor\log_2 n\rfloor)$-spanner in $\mathrm{O}(m\log n)$ time for chordal graphs, a tree $(2\rho\lfloor\log_2 n\rfloor)$-spanner $\mathrm{O}(m\log^2 n)$ time or a tree $(12\rho\lfloor\log_2 n\rfloor)$-spanner in $\mathrm{O}(m\log n)$ time for graphs that confess a Robertson-Seymour's tree-decomposition with bags of radius at most ρ in G and a tree $(2\lceil t/2\rceil\lfloor\log_2 n\rfloor)$-spanner in $\mathrm{O}(mn\log^2 n)$ time or a tree $(6t\lfloor\log_2 n\rfloor)$-spanner in $\mathrm{O}(m\log n)$ time for graphs that confess *a tree t-spanner*. They produced the same approximation ratio as in [2] but in a better running time.

3 Genetic Algorithm

In this section, we propose a genetic algorithm for the MMST problem, the problem of finding a *tree t-spanner* in a given edge-weighted graphs that minimizes the stretch factor t (see Sect. 1).

We first introduce some terminologies that will be used throughout this section. Let G' be a subgraph of G. The sets $V(G')$ and $E(G')$ denote the set of vertices and edges of G', respectively. The shortest distance between two vertices u and v in G' is denoted by $d_{G'}(u,v)$. For two subgraphs G_1 and G_2 of G, let $G_1 \cup G_2$, $G_1 \cap G_2$, and $G_1 - G_2$ denote the subgraph induced by $E(G_1) \cup E(G_2)$, $E(G_1) \cap E(G_2)$, and $E(G_1) - E(G_2)$, respectively.

3.1 Algorithm Overview

The Genetic Algorithm (GA) is an iterative optimization approach based on the principles of genetics and natural selection [14]. We first have to define a suitable data structure to represent individual solution (chromosomes), and then construct a set of candidate solutions as an initial population (first generation) of an appropriate cardinality $pop - size$. The following typical procedure is repeated as long as a predefined stopping criteria are met. Starting with the current generation, we use a predefined selection technique to repeatedly choose a pair of individuals (parents) in the current generation to reproduce, with probability p_c, a new set of individuals (offsprings) by exchanging some parts of between the two parents (crossover operation). To avoid local minimum, we try to keep an appropriate diversity among different generations by applying mutation operation, with specific probability p_m, to genes of individuals of the current generation. Finally, based on the values of an appropriate fitness function, we select a new generation from both the offspring and the current generation (the more suitable solutions have more chances to reproduce).

Note that, determining representation method, population size, selection technique, crossover and mutation probabilities, and stopping criteria in genetic algorithms are crucial since they mainly affect the convergence of the algorithm (see [15–19]).

The rest of this section is devoted to describe steps of the above algorithm in details.

3.2 Representation

Let $G = (V, E)$ be a given undirected graph such that each vertex in V is assigned a distinct label from the space $1, 2, \dots, n$, i.e., $V = \{1, 2, \dots, n\}$. Clearly, each edge $e \in E$ with end points i and j is uniquely defined by the unordered pair i, j. Moreover, every subgraph of G is uniquely defined by the set of unordered pairs of all its edges. In particular, every spanning tree T in G is induced by a set of exactly $n - 1$ unordered pairs corresponding to its edges since T is a subgraph of G that spans all vertices in V and has no cycles. Therefore, each chromosome (tree t-spanner) can be represented as a set of unordered pairs of integers each of which represent a gene (edge) in the chromosome.

3.3 Initial Population

Constructing an initial generation is the first step in typical genetic algorithms. We first have to decide the population size $pop - size$, one of the decisions that affect the convergence of the genetic algorithm. It is expected that small population size may lead to weak solutions, while, large population size increases the space and time complexity of the algorithm. Many literatures studied the influence of the population size to the performance of genetic algorithms (see [18] and the references therein). In this paper, we discuss the effect of the population size on the convergence time of the algorithm (cf. Sect. 4).

On of the most common methods is to apply random initialization to get an initial population. Namely, we compute each chromosome in the initial population by repeatedly applying the following simple procedure as long as the cardinality of the set of visited vertices is less than n (or as long as the cardinality of the set of traversed edges is less than $n - 1$). Let T denote the tree constructed so far by the procedure (initially, T consists of a random vertex from $V(G)$). We first select a random vertex $v \notin V(T)$ from the set of the neighbors of all vertices in T, and then add the edge $e = (u, v)$ to T, where u is the neighbor of v in T. It is easy to verify that the above procedure returns a tree after exactly $n - 1$ iterations. The generated tree T is added to the initial population.

The above algorithm is repeated as long as the number of constructed population is less than $pop - size$.

3.4 Fitness Function

Fitness function is a function used to validate each chromosome. Here, the objective function of the MMST is to minimizes the maximum ratio between all pairs of vertices in the underlying graph G. Formally, the objective function of the MMST is to minimizes $\max_{u,v \in V} \frac{d_T(u,v)}{d_G(u,v)}$, where $d_T(u, v)$ and $d_G(u, v)$ are the distances between u and v in T and G, respectively.

3.5 Selection Process

In this paper, we present three common selection techniques: roulette wheel selection, stochastic universal sampling selection, and tournament selection. All these techniques are called fitness-proportionate selection techniques since they are based on a predefined fitness function used to evaluate the quality of individual chromosomes. Throughout the execution of the proposed algorithm, the reverse of this ratio is used as the fitness function of the corresponding chromosome. We assume that the same selection technique is used throughout the whole algorithm. The rest of this section is devoted to briefly describe these selection techniques.

Roulette Wheel Selection (RWS): [14, 20] Here, the probability of selecting a chromosome is based on its fitness value. More precisely, each chromosome is selected with the probability that equals to its normalized fitness value, i.e., the ratio of its fitness value to the total fitness values of all chromosomes in the set from which it will be selected.

Stochastic Universal Sampling Selection (SUS): [20, 21] Instead of a single selection pointer used in roulette wheel approach, SUS uses h equally spaced pointers, where h is the number of chromosomes to be selected from the underlying population. All chromosomes are represented in number line randomly and a single pointer $ptr \in (0, \frac{1}{h}]$ is generated to indicate the first chromosome to be selected. The remaining $h - 1$ individuals whose fitness spans the positions of the pointers $ptr + i/h, i = 1, 2, \dots, h - 1$ are then chosen.

Tournament Selection (TRWS): [14, 21] This is a two stages selection technique. We first select a set of $k < pop - size$ chromosomes randomly from the current population. From the selected set, we choose the more fit chromosome by applying the roulette wheel selection approach. Tournament selection is performed according to the required number of chromosomes.

3.6 Crossover Process

In each iteration of the algorithm we repeatedly select a pair of chromosomes (parents) from the current generation and then apply crossover operator with probability p_c to the selected chromosomes to get new chromosomes (offsprings). Simulations and experimental results of the literatures show that a typical crossover probability lies between 0.75 and 0.95. There are two common crossover techniques: single-point crossover and multi-point crossover. Many researchers studied the influence of crossover approach and crossover probability to the efficiency of the whole genetic algorithm, see for example [17, 19] and the references therein. In this paper, we use a multi-point crossover approach by exchanging a randomly selected set of edges between the two parents.

In particular, for each selected pair of chromosomes T_1 and T_2, we generate a random number $s \in (0, 1]$. If $s < p_c$ holds, we apply crossover operator to T_1 and T_2 as follows.

Define the two sets $E_1 = E(T_1) - E(T_2)$ and $E_2 = E(T_2) - E(T_1)$ ($|E_1| = |E_2|$ holds). Let $t = |E_1| = |E_2|$, and generate a random number k from $[1, t]$. We first choose a random subset E_1' of cardinality k from E_1, and then add E_1' to T_2 to get a subgraph T' (i.e., $T' = T_2 \cup E_1'$). Clearly, T' contains k cycles each of which contains a distinct edge from E_1'. For every edge $e = (u, v)$ in E_1', we apply the following procedure to fix a cycle containing e. Let $\widetilde{T}$ be the current subgraph (initially, $\widetilde{T} = T'$). We first find a path $P_{\widetilde{T}}(u, v)$ between u and v in $\widetilde{T} - \{e\}$. We then choose an edge $\widetilde{e}$ in $P_{\widetilde{T}}(u, v)$ randomly and delete it from subgraph $\widetilde{T}$.

Similarly, we apply the above crossover technique by interchanging the roles of T_1 and T_2 one more offspring. Finally, we add each of the resulting spanning trees to the set of generated offsprings.

3.7 Mutation Process

To maintain the diversity among different generations of the population (and hence avoid local minimum), we apply a genetic (mutation) operator to chromosomes of the current generation with predefined (usually small) probability p_m. Namely, for each chromosome T, we generate a random number $s \in (0, 1]$, and then mutate T if $s < p_m$ holds by replacing a random edge (gene) in T with a random edge from $E(G) - E(T)$. Many results analyzed the role of mutation operator in genetic algorithms [15–17].

Formally, a chromosome T is mutated as follows. We first select a random edge $e = (u, v)$ in the graph G but not in the chromosome T, i.e., e is randomly chosen from the set $E(G) - E(T)$ of edges, It is easy to see that the subgraph $T \cup \{e\}$ contains exactly one cycle including e. We then select a random edge e' in the path $P_T(u, v)$ between u and v in T. Let T' denote the offspring obtained from T by exchanging the two edges e and e', i.e., $T' = (T - \{e\}) \cup \{e'\}$. It is easy to see that T' is a spanning tree in G.

A formal description of the proposed genetic algorithm is described in Algorithm 1.

Algorithm 1 Genetic Algorithm for the *Tree t-Spanner* Problem

```
Input: An edge-weighted graph $G$, a population size $pop - size$, a maximum
  number of generations $maxgen$, a crossover probability $p_c$, a mutation probability $p_m$.
Output: A tree t-spanner that minimizes $t$.
1. Compute an initial population $I_0$ (cf. Sect. 3.3).
2. $gen \leftarrow 1$.
3. While ($gen \leq maxgen$) do
4.   For $i = 1$ to $pop - size$ do
5.     Select a pair of chromosomes from $I_{gen-1}$ (Sect. 3.5).
6.     Apply crossover operator with probability $p_c$ to the selected pair of
       chromosomes to get two offsprings (Sect. 3.6).
7.   Endfor
8.   For each chromosome in $I_{gen-1}$, apply mutation operator with
     probability $p_m$ to get an offspring (Sect. 3.7).
9.   Extend $I_{gen-1}$ with valid offsprings output from lines 6 and 8.
10.  Find the chromosome $T_{gen-1}$ with the best fitness value in $I_{gen-1}$.
11.  If $gen \geq 2$ and the fitness values of $T_{gen-2}$, $T_{gen-1}$, and $T_{gen}$ are identical, then break.
12.  Select $pop - size$ chromosomes from $I_{gen-1}$ to form $I_{gen}$ (Sect. 3.5).
13.  $gen \leftarrow gen + 1$.
14. Endwhile
15. Output $T_{gen}$.
```

4 Experimental Results

In this section, we evaluate the proposed genetic algorithm by applying it to several random edge-weighted graphs. In particular, we generate a random graph G of n nodes by applying Erdos and Renyi [22] approach in which an edge is independently included between each pair of nodes of G with a given probability p. Here, we generate random graphs with sizes 6, 10, 15, and 20, and a randomly chosen probability p. Moreover, all edge weights of the generated graphs are set to random integers from the interval $[1, 1000]$.

For each of the generated graphs, we apply the proposed algorithm with different selection techniques. We set the population size $pop - size = 30$, the maximum number of iterations the genetic algorithm executes $maxgen = 300$, the crossover probability $p_c = 0.9$, and the mutation probability $p_m = 0.2$. All previous parameters are summarized in Table 1. The algorithm terminates if either the number of iterations exceeds $maxgen$ or the solution does not change for three consecutive iterations. All obtained solutions are compared with the corresponding optimal solutions obtained by considering all possibilities of all spanning trees in the underlying graphs.

All results presented in this section were performed in MATLAB R2014b on a computer powered by a core i7 processor and 16 GB RAM.

The results of applying our genetic algorithm to random graphs with sizes $n = 6$, $n = 10$, $n = 15$, and $n = 20$, are shown in Table 2. In particular, Table 2 compare the values of t returned by the algorithm with the corresponding optimal stretch factor. It is seen that the proposed algorithm outputs optimal solution to MMST all the instances the algorithm applies to.

Table 1 Values of algorithm parameters

Parameter	Value
n	6, 10, 15, 20
$pop - size$	30
$maxgen$	300
p_c	0.9
p_m	0.2

Table 2 Values of t corresponding to a random graphs with size n

t	t-Optimal	t-RWS	t-SUS	t-TRWS
n				
6	1.0833	1.0833	1.0833	1.0833
10	1.0964	1.0964	1.0964	1.0964
15	1.1579	1.1579	1.1739	1.1579
20	1.0132	1.0484	1.0132	1.0132

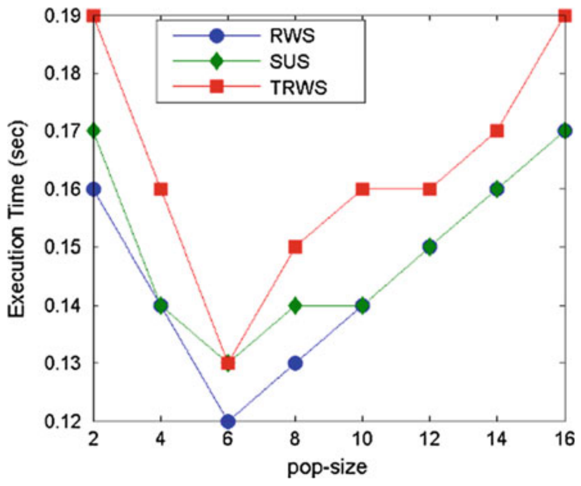

Fig. 1 The influence of *pop* − *size* on the running time of the algorithm (n = 6)

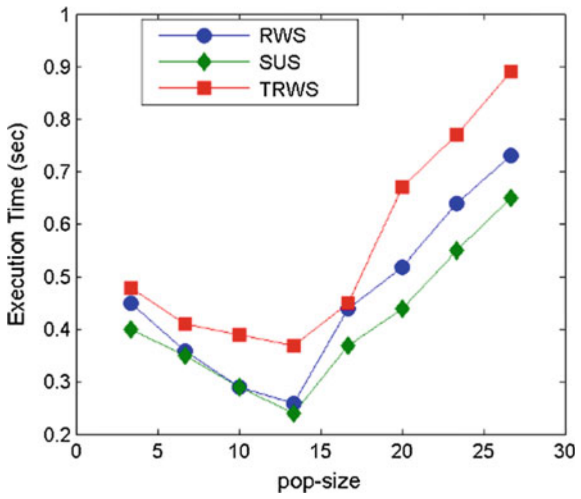

Fig. 2 The influence of *pop* − *size* on the running time of the algorithm (n = 10)

We discuss the effect of the population size *pop* − *size* on the convergence of the algorithm. Given a random graph with size n, we apply the algorithm with population sizes $n/3$, $2n/3$, n, $4n/3$, $5n/3$, $2n$, $7n/3$ and $8n/3$. Figures 1, 2, 3 and 4 illustrate the running time of the algorithm applied to graphs of sizes $n = 6$, $n = 10$, $n = 15$ and $n = 20$, respectively. The algorithm attains the least running time when the population size is set to a constant fraction of the graph size n.

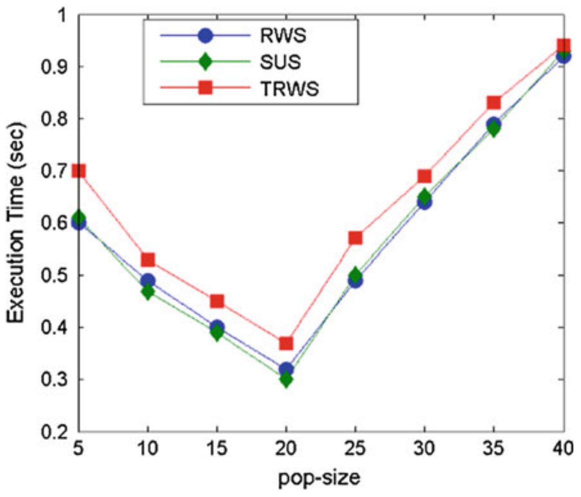

Fig. 3 The influence of *pop* − *size* on the running time of the algorithm (n = 15)

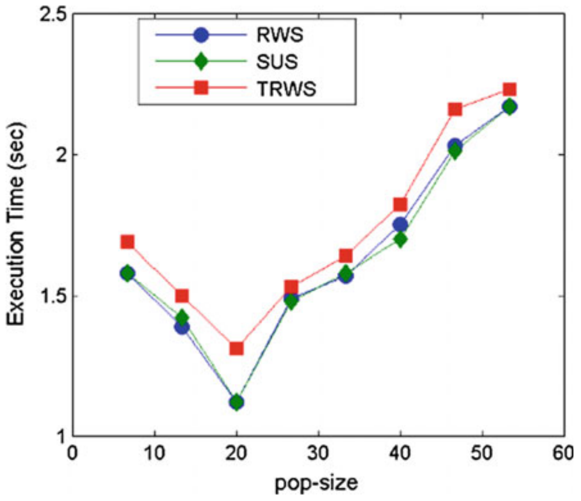

Fig. 4 The influence of *pop* − *size* on the running time of the algorithm (n = 20)

5 Conclusion

In this paper, we have studied the problem of finding the Minimum Maximum Stretch Spanning Tree (MMST) that aims to find a spanning tree T in a given graph G such that the maximum ratio of the distances between every pair of vertices in T to the shortest distance between the two vertices in G is minimized. We have designed a genetic algorithm for the MMST problem which have been evaluated by applying it to random instances of the problem. Experimental results have shown that the proposed algorithm outputs a high quality solutions to the MMST problem. It will be interesting to adapt our algorithm to be applied to the *subgraph t-spanner* problem: the problem of finding a minimum weight subgraph in G such that the distance between any two vertices in this subgraph is at most a given $t \geq 1$ times the shortest distance between the two vertices in G. See [23–26] and the references therein.

References

1. Peleg, D., Ulman, J.D.: An optimal sychronizer for the hypercube. SIAM J. Comput. 740–747 (1989)
2. Emek, Y., Peleg, D.: Approximating minimum max-stretch spanning trees on unweighted graphs. SIAM J. Comput. 1761–1781 (2008)
3. Demmer, M.J., Herlihy, M.P.: The arrow distributed directory protocol. In: Proceeding of the 12th International Symposium on Distributed Computing (DISC), pp. 119–133. Springer (1998)
4. Peleg, D., Reshef, E.: Low complexity variants of the arrow distributed directory. J. Comput.. Syst. Sci, pp. 474–485 (2001)
5. Kuhn, F., Wattenhofer, R.: Dynamic analysis of the arrow distributed protocol. Theory Comput. Syst. pp. 875–901 (2006)
6. Herlihy, M., Tirthapura, S., Wattenhofer, R.: Competitive concurrent distributed queuing. In: Proceedings of the 20th Annual ACM Symposium on Princibles of Distributed Computing, pp. 127–133 (2001)
7. Cai. L., Corneil, D., Tree Spanners. SIAM J. Discret. Math. 359–387 (1995)
8. Brandstädt, A., Dragan, F.F., Le, H.-O., Le, V.B.: Tree spanners on chordal graphs: complexity and algorithms. Theor. Comput. Sci. 329–354 (2004)
9. Brandstädt, A., Dragan, F.F., Le, H.-O., Le, V.B., Uehara, R.: Tree spanners for bipartite graphs and probe interval graphs. Algorithmica 27–51 (2007)
10. Peleg, D., Tendler, D.: Low stretch spanning trees for planar graphs, Technical Report. MCS01-14, Weizmann Science Press of Israel (2001)
11. Fekete, S.P., Kremer, J.: Tree spanners in planar graphs, Discret. Appl. Math, pp. 85–103 (2001)
12. Dragan, F.F., Fomin, F.V., Golovach, P.A.: Spanners in sparse graphs. J. Comput. Syst. Sci. pp. 1108–1119 (2010)
13. Dragan, F.F., Köhler, E.: An approximation algorithm for the tree t-spanner problem on unweighted graphs via generlized chordal graphs. Algorithmica 884–905 (2014)
14. Engelbrecht, A.P.: Computational Intelligence: An Introduction. Wiley, New York (2007)
15. Abdoun, O., Abouchabaka, J., Tajani, C.: Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem, CoRR abs/1203.3099 (2012)
16. Hesser, J., Manner, R.: Towards an optimal mutation probability for genetic algorithms. In: Proceedings of 1st Workshop in Parallel Problem Solving From Nature, pp. 2332 (1991)
17. LIN, W-Y., LEE, W-Y., Hong, T-P.: Adapting crossover and mutation rates in genetic algorithms. In: The Sixth Conference on Artificial Intelligence and Applications, Kaohsiung, Taiwan (2001)
18. Roeva, O., Fidanova, S., Paprzycki, M.: Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In: Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 371–376 (2013)
19. Vekaria, K., Clack, C.: Selective crossover in genetic algorithms: an empirical study.Lecture Notes in Computer Science, vol. 1498, pp. 438–447 (1998)
20. Chipperfield, A., Fleming, P., Pohlheim, H., Fonseca, C.: The Matlab Genetic Algorithm User's Guide, UK SERC (1994)
21. Blickle, T., Thiele, L.: A Comparison of Selection Schemes used in Genetic Algorithms, Zurich (1995)

22. Erdos, P., Renyi, A.: On random graphs. Publ. Math. **290** (1959)
23. Sigurd, M., Zachariasen, M.: Construction of Minimum-Weight Spanners, pp. 797–808. Springer, Berlin (2004)
24. Farley, A.M., Zappala, D., Proskurowski, A., Windisch, K.: Spanners and message distribution in networks. Dicret. Appl. Math. 159–171 (2004)
25. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast greedy algorithms for constructing sparse geometric spanners. SIAM J. Comput. pp. 1479–1500 (2002)
26. Navarro, G., Paredes, R., Chavez, E.: t-Spanners as a data structure for metric space searching. In: International Symposium on String Processing and Information Retrieval, SPIRE, LNCS 2476, pp. 298–309 (2002)