DOI: 10.15439/2015F249
ACSIS, Vol. 5

Genetic Algorithms for Balanced Spanning Tree Problem

Riham Moharam, Ehab Morsy
Department of Mathematics, Suez Canal
University, Ismailia 41522, Egypt.
Email: {riham.moharam, ehabmorsy}@science.suez.edu.eg

Ismail A. Ismail
Department of Computer Sciences,
6 October University, Egypt.
Email: amr442-2@hotmail.com

***Abstract*—Given an undirected weighted connected graph $G = (V, E)$ with vertex set V and edge set E and a designated vertex $r \in V$, we consider the problem of constructing a spanning tree in G that balances both the minimum spanning tree and the shortest paths tree rooted at r. Formally, for any two constants $\alpha, \beta \geq 1$, we consider the problem of computing an (α, β)-balanced spanning tree T in G, in the sense that, (i) for every vertex $v \in V$, the distance between r and v in T is at most α times the shortest distance between the two vertices in G, and (ii) the total weight of T is at most β times that of the minimum tree weight in G. It is well known that, for any $\alpha, \beta \geq 1$, the problem of deciding whether G contains an (α, β)-balanced spanning tree is NP-complete [15]. Consequently, given any $\alpha \geq 1$ (resp., $\beta \geq 1$), the problem of finding an (α, β)-balanced spanning tree that minimizes β (resp., α) is NP-complete. In this paper, we present efficient genetic algorithms for these problems. Our experimental results show that the proposed algorithm returns high quality balanced spanning trees.**

***Index Terms*—Minimum Spanning Tree, Shortest Paths Tree, Balanced Spanning Tree, Genetic Algorithms, Graph Algorithms**

I. Introduction

LET $G = (V, E)$ be an undirected edge-weighted connected graph with vertex set V and edge set E such that $|V| = n$ and $|E| = m$.

For any vertex r in G, a spanning tree T rooted at r is a shortest paths tree if, for every vertex $v \in V$, the distance between r and v in T_s equals the shortest distance between the two vertices in G. Dijkstra's algorithm is one of the well known polynomial time algorithms for computing shortest path tree in weighted graphs [22].

This work is partially supported by Alexander von Humboldt foundation

Note that, there exist weighted graphs in which the total weight of a shortest path tree may be much more than that of a minimum spanning tree, and vertices that are close to the designated root can be far away from the root in a minimum spanning tree (see [15] for an illustrative example). In this paper, we aim to find a spanning tree in weighted graphs that balances a minimum spanning tree and a shortest path tree, that is, a rooted tree of total weight at most a constant times the minimum tree weight, and the distance between the root and any vertex in the tree is at most a constant times the shortest distance between the two vertices in the graph. A formal definition of the problem can be described as follows.

Definition 1: [15] For any $\alpha, \beta \geq 1$, a rooted spanning tree T of G is called (α, β)-balanced spanning tree if it satisfies the following two conditions:

1) For every vertex v, the distance between the root and v in T is at most α times the shortest distance between the two vertices in G.
2) The total weight of T is at most β times the minimum tree weight in G.

The shortest paths tree and minimum spanning tree are widely used in network routing. In particular, the shortest paths tree minimizes the delay from the source to every destination through a routing tree, and the minimum spanning tree minimizes the total routing cost along a tree. See [7], [14], [17], [21] and the references therein. Thus, balanced spanning tree is an appropriate routing tree for networks with the above two objectives.

Given any $\alpha \geq 1$, Awerbuch et al. [4] proposed an algorithm that approximates a minimum spanning tree and a shortest paths tree in edge-weighted graphs.

Namely, they modified the algorithm described in [3], [9] to compute an $(\alpha, 1 + \frac{4}{\alpha-1})$-balanced spanning tree in $O(m + n \log n)$ time. Afterwards, Khullar et al. [15] improved the above result and presented a constructive linear time algorithm that outputs an $(\alpha, 1 + \frac{2}{\alpha-1})$-balanced spanning tree. In other words, for any $\gamma > 0$, the algorithm of Khullar et al. [15] outputs an $(1 + \sqrt{2}\gamma, 1 + \frac{\sqrt{2}}{\gamma})$-balanced spanning tree in linear time.

For any $\alpha, \beta \geq 1$, the problem of deciding whether G contains an (α, β)-balanced spanning tree is NP-complete [15]. Consequently, given any $\alpha \geq 1$, the problem of finding an (α, β)-balanced spanning tree that minimizes β is NP-complete. Analogously, given any $\beta \geq 1$, the problem of finding an (α, β)-balanced spanning tree that minimizes α is also NP-complete. In this paper, we present efficient genetic algorithms for these two problems. Our experimental results show that the proposed algorithm returns high quality balanced spanning trees.

The rest of this paper is organized as follows. Section II reviews some results on related problems. Section III presents the proposed genetic algorithm. Section IV evaluates our algorithm by applying it to randomly generated instances of the balanced spanning tree problem. Section V makes some concluding remarks.

II. Related Work

In this section, we present results on related problems.

Bharath-Kumar and Jaffe [5] studied the problem of finding a rooted tree in the underlying graph such that the total distances from the root to all vertices is at most a constant times the minimum total distances from the root to all vertices.

For each graph G, the greatest distance between any two vertices in G is called the diameter of G. A tree in a graph G is called shallow-light tree if its diameter is at most a constant (greater than or equal 1) times the diameter of G and with total weight at most a constant times the minimum tree weight. Awerbuch et al. [3] proved that each graph has a shallow-light tree.

Cong et al. [9] proposed a model of timing-driven global routing for cell-based design to improve the construction of a shallow-light tree based on the idea of finding minimum spanning trees with bounded radius. They designed an algorithm to find, for any constant $\epsilon > 0$, a spanning tree with radius $(1 + \epsilon) \cdot R$ (using an analog of the classical Prim's minimum spanning tree structure), where R is the minimum possible tree radius. They find a smooth trade-off between the radius and the cost of the tree. Afterwards, they proposed a new method [10] to improve their previous algorithm based on a provably good algorithm that simultaneously minimizes both total weight and longest interconnection path length of the tree. More specifically, their algorithm produced a tree with radius at most $(1 + \epsilon) \cdot R$ and of total weight at most $(1 + \frac{2}{\epsilon})$ times the minimum tree weight.

III. Genetic Algorithm

In this section, we propose genetic algorithms to the two variants of the balanced spanning tree problem described in Section I. To avoid duplication, we present an algorithm for only one of these problems; the other variant of the problem can be described analogously. Namely, throughout this section, we focus on the problem in which we are given a constant $\alpha \geq 1$ and the objective is to compute an (α, β)- balanced spanning tree that minimizes β.

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 $d_{G'}(u, v)$. We use $w(G')$ to denote the sum $\sum_{e \in E(G')} w(e)$ of weights of all edges in G'. 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. For any edge e in G, let $Adj(e)$ denote the set of all adjacent edges to e in G, where two edges of G are called adjacent if they share a common vertex.

A. Algorithm Overview

The Genetic Algorithm (GA) is an iterative optimization approach based on the principles of genetics and natural selection [2]. The first step of genetic algorithms is to determine a suitable data structure to represent individual solutions (chromosomes), and then construct an initial population (first generation) of prescribed cardinality $pop - size$. A typical intermediate iteration of genetic algorithms can be outlined as follows. 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 applying crossover operation to the selected pair. To keep an appropriate diversity among different generations, we apply mutation operation with specific probability p_m to genes of individuals of the current generation to get more offsprings. A new generation is then selected from both the offspring and the current generation based on their fitness values (the more suitable solutions have more chances to reproduce). The algorithm terminates when it meets prescribed stopping criteria.

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

In genetic algorithms, determining representation method, population size, selection technique, crossover and mutation probabilities, and stopping criteria are crucial since they mainly affect the convergence of the algorithm (see [1], [13], [18], [19], [20]).

The rest of this section is devoted to describe steps of Algorithm 1 in details.

Algorithm 1 Genetic Algorithm for the Balanced Spanning Tree 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 $p_m$, a shortest paths tree $T_s$,
  a minimum spanning tree $T_m$, and a real number $\alpha \geq 1$.
Output: An $(\alpha, \beta)$-balanced spanning tree that minimizes $\beta$.
1.  Compute an initial population $I_0$ (cf. Section III-C).
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 $I_{gen-1}$ (cf. Section III-D).
6.        Apply crossover operator with probability $p_c$ to the selected pair of chromosomes to get
          two offsprings (cf. Section III-E).
7.     Endfor
8.     For each chromosome in $I_{gen-1}$, apply mutation operator with probability $p_m$ to get an
       offspring (cf. Section III-F).
9.     Extend $I_{gen-1}$ with valid offsprings output from lines 6 and 8.
10.    Find the minimum total weight chromosome $T_{gen-1}$ in $I_{gen-1}$.
11.    If $gen \geq 2$ and $w(T_{gen-2}) = w(T_{gen-1}) = w(T_{gen})$, then break.
12.    Select $pop-size$ chromosomes from $I_{gen-1}$ to form $I_{gen}$ (cf. Section III-D).
13.    $gen \leftarrow gen + 1$.
14. Endwhile
15. Output $T_{gen}$.
```

B. Representation

Let $G = (V, E)$ be a given undirected graph such that $V = \{1, 2, \dots, n\}$. Note that, each edge $e \in E$ with end points i and j can be defined by an unordered pair $\{i, j\}$, and consequently, any subgraph G' of G is uniquely defined by the set of unordered pairs of all its edges. In particular, any spanning tree T in G is induced by a set of exactly $n-1$ 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 (spanning tree) can be represented as a sequence of ordered pairs of integers each of which represent a gene (edge) in the chromosome.

C. Initial Population

Before initializing the first generation of the genetic algorithm, we have to first decide its size $pop-size$. As mentioned before, this decision on population size affect the convergence of the algorithm. In particular, 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 [19] and the references therein). In this paper, we discuss the effect of the population size on the convergence time of the algorithm (cf. Section IV).

We apply random initialization to generate an initial population. Namely, we compute each chromosome in the initial population by repeatedly applying the following simple procedure as long as the length of the chromosome (number of 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 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 only if it is valid, where a tree T is said to be a valid chromosome if, for every vertex v in T, it holds that $d_T(r,v) \leq \alpha \cdot d_G(r,v)$.

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

D. Selection Process

In this paper, we apply four common selection techniques: random selection, roulette wheel selection, stochastic universal sampling selection, and tournament selection. All these techniques, except the random selection, are called fitness-proportionate selection techniques since they are based on a predefined fitness function used to evaluate the quality of individual chromosomes. Here, the objective function of the underlying balanced spanning tree problem (i.e., the ratio of the the minimum weight tree to the total weight of the chromosome) is used as the fitness function of each chromosome. We assume that the same selection technique is used throughout the execution of the algorithm. The rest of this section is devoted to briefly describe these selection techniques.

Random Selection (RS): [2] It is the simplest selection operator, where each chromosome has the same probability to be selected. That is, from a population of size q, each chromosome has the chance to be chosen with probability $1/q$.

Roulette Wheel Selection (RWS): [2], [8] In the roulette wheel technique, 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): [6], [8] It is a single phase sampling; 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 $ptr + i/h$, $i = 1, 2, \ldots, h-1$ are then chosen.

Tournament Selection (TRWS): [2], [6] In this approach, we first select a set of $k < pop - size$ chromosomes randomly from the current population. From the selected set, we choose the required number of chromosomes by applying the roulette wheel selection approach.

E. 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], [20] 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 $P_{\widetilde{T}}(u, v)$ by applying the selection technique used in the algorithm to the set of all edges in $P_{\widetilde{T}}(u, v)$, assuming that the weight of each edge is its fitness value. Finally, we delete $\widetilde{e}$ from subgraph $\widetilde{T}$. Note that, edges of large weights in $P_{\widetilde{T}}(u, v)$ have more chances to be deleted, and hence it is more likely that the current subgraph $\widetilde{T}$ attains total weight less than that of $T' - \{e\}$. In general, it is more likely that offsprings output from the crossover operation attains total weights less than that of its parents.

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 if it is valid.

F. Mutation Process

Mutation is a genetic operator that intends to maintain the diversity among different generations of the population by altering some random genes (edges) in a chromosome. This may allow the algorithm to get better solutions by avoiding local minimum. Many results analyzed the role of mutation operator in genetic algorithms [1], [13], [17].

For a chromosome T in the current population, we apply mutation operator such that each edge in T is mutated with probability p_m. The standard range of p_m lies between $1/\ell$ and 0.5, where ℓ is the length (number of edges) of the chromosome. With $p_m = 1/\ell$, at least one gene (edge) on average should mutate. On the other hand, with $p_m = 0.5$, half of the edges should mutate on average, and consequently a random offspring is generated. In particular, for each edge e in T, we generate a random number $s \in (0, 1]$, and then mutate e if $s < p_m$ holds by replacing e with a random edge from $Adj(e) - E(T)$. Let T' denote the subgraph obtained after applying mutation operator to T. If T' is disconnected, then we discard it. Otherwise, T' is a spanning tree.

Finally, we add the resulting spanning tree to the set of generated offsprings only if it is valid.

IV. Experimental Results

In this section, we evaluate the proposed genetic algorithms by applying it to several random instances of both variants of the balanced spanning tree problem. In particular, we generate a random graph G of n nodes by applying Erdos and Renyi [11] approach in which an edge is independently included between each pair of nodes of G with 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 range $[1, 100]$.

For each of the generated graphs, we apply the proposed algorithm with different selection techniques and different values of α (resp., β). 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 $p_m = 0.01$. 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 valid spanning trees in the underlying graphs. There are many algorithms for finding all spanning trees in undirected graphs, see for example [12].

We discuss the effect of the values of α (resp., β) on the convergence of the algorithm. It is seen that the running time of the algorithm decreases as the the value of α (resp., β) increases, see Figures 1-2 (resp. Figures 5-6).

We also study the effect of the population size $pop-size$ to the convergence of the algorithm. From the experimental results, we observe that a constant fraction of the number of nodes n is an appropriate value for the population size, see Figures 3-4 and Figures 7-8.

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

A. Minimizing β

In this section, we present our experimental results for the problem in which α is given and the objective is to minimize β. We apply our algorithm with different values of α from the range $[1, 2]$. 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 I, Table II, Table III, and Table IV, respectively. In particular, Tables I-IV compare the values of β returned by the algorithm with the corresponding optimal value of β. It is seen that the proposed algorithm outputs optimal balanced spanning tree for all the instances the algorithm applies to.

TABLE I
Values of β corresponding to a random graph with $n = 6$.

β / α	**β-Optimal**	**β-RS**	**β-RWS**	**β-SUS**	**β-TRWS**
1.1	1.142	1.142	1.142	1.142	1.142
1.2	1.047	1.142	1.142	1.047	1.142
1.3	1.047	1.047	1.047	1.047	1.047
1.4	1	1	1	1	1

TABLE II
VALUES OF β CORRESPONDING TO A RANDOM GRAPH WITH $n = 10$.

α \ β	β-Optimal	β-RS	β-RWS	β-SUS	β-TRWS
1.1	1.021	1.021	1.021	1.021	1.021
1.2	1.010	1.021	1.010	1.010	1.010
1.3	1	1	1	1	1

TABLE III
VALUES OF β CORRESPONDING TO A RANDOM GRAPH WITH $n = 15$.

α \ β	β-Optimal	β-RS	β-RWS	β-SUS	β-TRWS
1.1	1.025	1.074	1.074	1.025	1.025
1.2	1.025	1.074	1.062	1.025	1.025
1.3	1.025	1.074	1.062	1.025	1.025
1.4	1.012	1.049	1.049	1.012	1.012
1.5	1	1.049	1.049	1	1

TABLE IV
VALUES OF β CORRESPONDING TO A RANDOM GRAPH WITH $n = 20$.

α \ β	β-Optimal	β-RS	β-RWS	β-SUS	β-TRWS
1.1	1.161	1.161	1.161	1.161	1.161
1.2	1.104	1.106	1.104	1.104	1.104
1.3	1.104	1.104	1.104	1.104	1.104
1.4	1.104	1.104	1.104	1.104	1.104
1.5	1.078	1.078	1.078	1.078	1.078
1.6	1.078	1.078	1.078	1.078	1.078
1.7	1.054	1.063	1.054	1.054	1.054
1.8	1.054	1.063	1.054	1.054	1.054
1.9	1.054	1.063	1.054	1.054	1.054
2	1.009	1.054	1.009	1.009	1.009

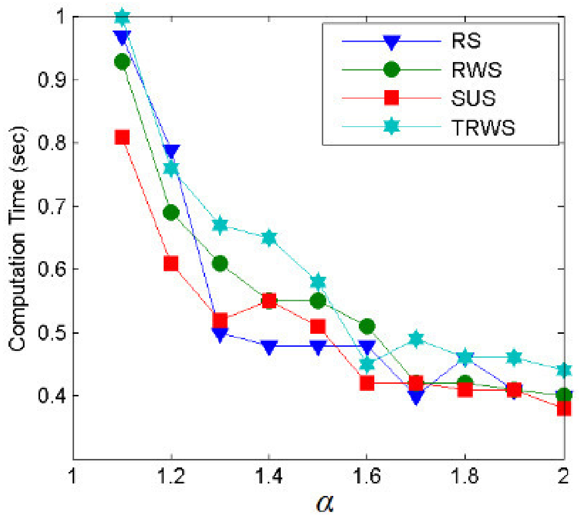

Fig. 1. The influence of α on the running time of the algorithm ($n = 15$)

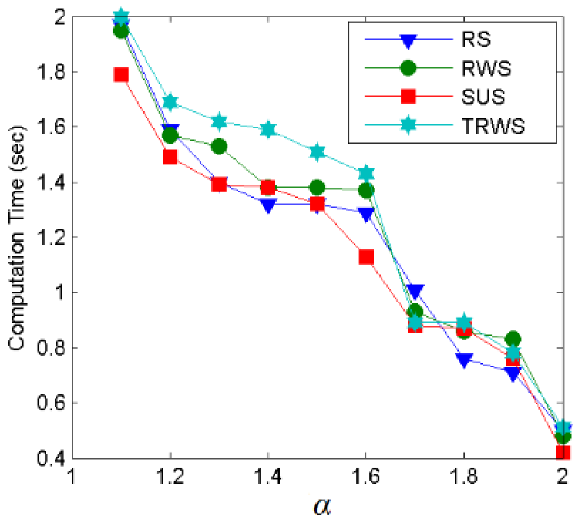

Fig. 2. The influence of α on the running time of the algorithm ($n = 20$)

Figures 1-2 show the influence of the value of α and the used selection technique on the execution time of the algorithm for graphs of $n = 15$ and $n = 20$, respectively. It is expected that the number of valid offsprings obtained in each iteration increases by relaxing the value of α, and consequently it is more likely that the execution time of the algorithm decreases as the value of α increases.

We also evaluate the influence of the population size on the convergence of the proposed genetic algorithm. For a graph of n nodes, we apply the algorithm with population sizes $n/3$, $2n/3$, n, $4n/3$, and $2n$. Figures 3-4 illustrate the running time of the algorithm applied to graphs of sizes $n = 15$, and $n = 20$, respectively, for fixed value $\alpha = 1.3$. We observe that the execution time of the algorithm increases as the population size increases, and the algorithm attains the least running time when the population size is set to a constant fraction of the graph size n.

B. Minimizing α

In this section, we present our experimental results for the problem in which β is given and the objective is to

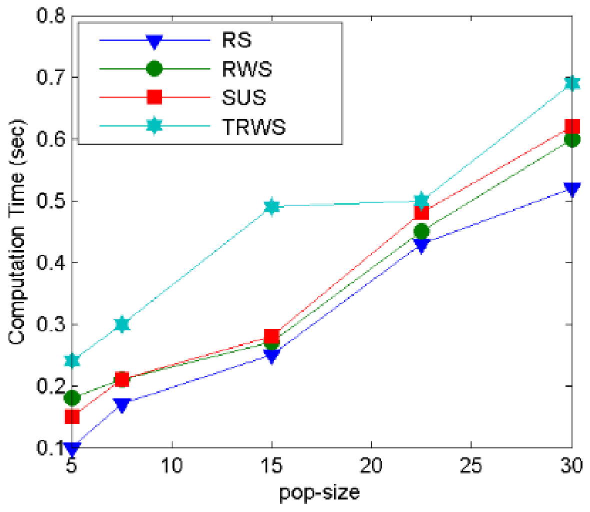

Fig. 3. The influence of $pop - size$ on the running time of the algorithm ($n = 15$)

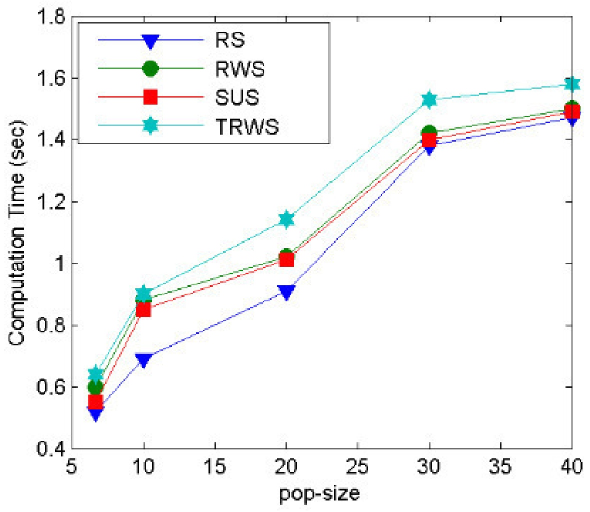

Fig. 4. The influence of $pop - size$ on the running time of the algorithm ($n = 20$)

TABLE V
VALUES OF α CORRESPONDING TO A RANDOM GRAPH WITH $n = 6$.

β \ α	**α-Optimal**	**α-RS**	**α-RWS**	**α-SUS**	**α-TRWS**
1.1	1.3571	1.3571	1.3571	1.3571	1.3571
1.2	1.285	1.285	1.285	1.285	1.285
1.3	1.285	1.285	1.285	1.285	1.285
1.4	1	1	1	1	1

TABLE VI
VALUES OF α CORRESPONDING TO A RANDOM GRAPH WITH $n = 10$.

β \ α	**α-Optimal**	**α-RS**	**α-RWS**	**α-SUS**	**α-TRWS**
1.1	1.238	1.238	1.238	1.238	1.238
1.2	1.096	1.096	1.096	1.096	1.096
1.3	1	1	1	1	1

TABLE VII
VALUES OF α CORRESPONDING TO A RANDOM GRAPH WITH $n = 15$.

β \ α	**α-Optimal**	**α-RS**	**α-RWS**	**α-SUS**	**α-TRWS**
1.1	1.173	1.208	1.173	1.173	1.173
1.2	1.157	1.157	1.157	1.157	1.157
1.3	1	1	1	1	1

TABLE VIII
VALUES OF α CORRESPONDING TO A RANDOM GRAPH WITH $n = 20$.

β \ α	**α-Optimal**	**α-RS**	**α-RWS**	**α-SUS**	**α-TRWS**
1.1	1.444	1.444	1.444	1.444	1.444
1.2	1.112	1.444	1.112	1.112	1.112
1.3	1.112	1.444	1.112	1.112	1.112
1.4	1.048	1.112	1.048	1.048	1.048
1.5	1.048	1.048	1.048	1.048	1.048
1.6	1.048	1.048	1.048	1.048	1.048
1.7	1	1	1	1	1

minimize α. We apply our algorithm with different values of β. 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 V, Table VI, Table VII, and Table VIII, respectively. Also, for this problem, the proposed algorithm outputs optimal balanced spanning tree for all instances the algorithm applies to.

Figures 5-6 show the influence of the value of β and the used selection technique to the execution time of the algorithm for graphs of $n = 15$ and $n = 20$, respectively. Most probably, the number of valid offsprings obtained in each iteration increases by relaxing the value of β, and consequently it is more likely that the execution time of the algorithm decreases as the value of β increases.

We evaluate the influence of the population size on the convergence of the proposed genetic algorithm. For a graph of n nodes, we apply the algorithm with population sizes $n/3$, $2n/3$, n, $4n/3$, and $2n$. Figures 7-8 illustrate the running time of the algorithm applied

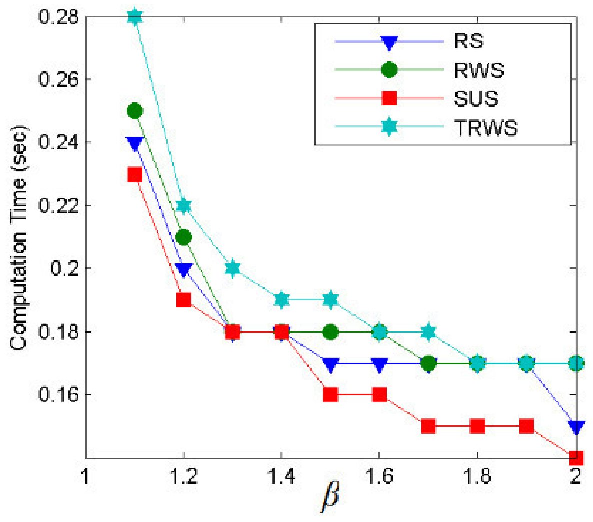

Fig. 5. The influence of β on the running time of the algorithm ($n = 15$)

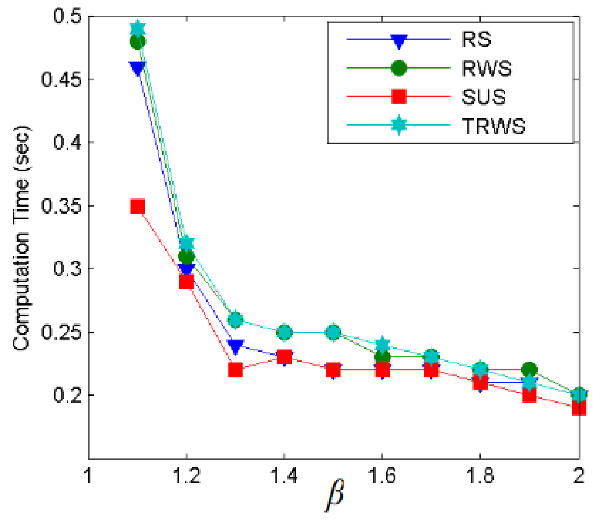

Fig. 6. The influence of β on the running time of the algorithm ($n = 20$)

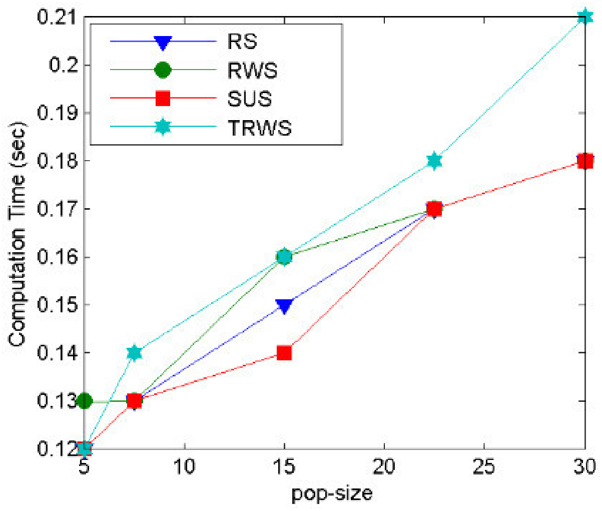

Fig. 7. The influence of $pop - size$ on the running time of the algorithm ($n = 15$)

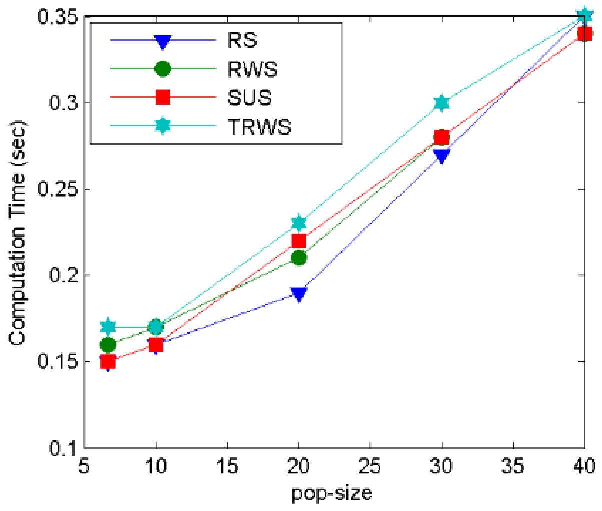

Fig. 8. The influence of $pop - size$ on the running time of the algorithm ($n = 20$)

to graphs of sizes $n = 15$, and $n = 20$, respectively, for fixed value $\beta = 1.3$. We observe that the algorithm attains the least running time when the population size is set to a constant fraction of the graph size n.

V. Conclusion

In this paper, we have studied the problem of finding a tree that balances a shortest path tree and a minimum spanning tree in undirected edge-weighted graphs. In particular, we have focused on two NP-complete variants of the problem in which we are given a bound on how far the required tree is from the shortest path tree (resp., minimum spanning tree), and the objective is to find the closest tree to the minimum spanning tree (resp., shortest path tree) under this bound. We have designed genetic algorithms for these problems. We have evaluated our algorithm by applying it to random graph instances. The algorithm outputs optimal balanced spanning tree for all instances it has been applied to. It will be interesting to relax our model to balanced subgraphs instead of balanced trees, that is, the problem of finding a minimum weight subgraph such that the distance between any two vertices u and v in the subgraph is at most a given constant times the shortest distance between the two vertices in the underlying graph (this problem is known as t-spanner problem in the literatures). See [23], [24], [25], [26] and the references therein.

References

[1] O. Abdoun, J. Abouchabaka, and C. Tajani, Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem, CoRR abs/1203.3099, 2012.

[2] Andris P. Engelbrecht, Computational Intelligence: an introduction, John Wiley & Sons, 2007.

[3] B. Awerbuch, A. Baratz, and D. Peleg, Cost-sensetive analysis of communication protocols, Proc. on Principles of Distributed Computing, pp. 177-187, 1990.

[4] B. Awerbuch, A. Baratz, and D. Peleg, Efficient broadcast and light-weight spanners, Manuscript, 1991.

[5] K. Bharath-Kumar and J. M. Jaffe, Routing to multiple destinations in computer networks, IEEE Transactions on Communications 31 (3), pp. 343-351, 1983.

[6] T. Blickle and L. Thiele, A comparison of Selection Schemes Used in Genetic Algorithms (Technical Report No. 11), Swiss Federal Institute of Technology (ETH) Zurich, Computer Engineering and Communications Networks Lab (TIK), 1995.

[7] R Campos and M Ricardo, A fast algorithm for computing minimum routing cost spanning trees, Computer Networks 52 (17), pp. 3229-3247, 2008.

[8] A. Chipperfield, P. Fleming, H. Pohlheim, and C. Fonseca, The Matlab Genetic Algorithm User's Guide, UK SERC, 1994.

[9] J. Cong, A. Kahng, G. Robins, M.Sarrafzadeh, and C. K. Wong, Performance-driven global routing for cell based IC's, Proc. IEEE Intl. Conference on Computer Design, pp. 170-173, 1991.

[10] J. Cong, A. Kahng, G. Robins, M.Sarrafzadeh, and C. K. Wong, Provably good performance-driven global routing, IEEE Transaction on CAD, pp. 739-752, 1992.

[11] P. Erdos and A. Renyi, On Random Graphs, Publ. Math, 290, 1959.

[12] H. N. Gabow and E. W. Myers, Finding all spanning trees of directed and undirected graphs, SIAM Journal on Computing, 7, pp. 280-287, 1978.

[13] J. Hesser and R. Männer, Towards an Optimal Mutation Probability for Genetic Algorithms, Proceedings of 1st workshop in Parallel problem solving from nature, pp. 23-32, 1991.

[14] G. Huang, X. Li, and J. He, Dynamic Minimal Spanning Tree Routing Protocol for Large Wireless Sensor Networks. In Proceedings of the 1st IEEE Conference on Industrial Electronics and Applications, Singapore, pp. 1-5, 2006.

[15] S. Khullar, B. Raghavachari, and N. Young, Balancing minimum spanning trees and shortest-path trees, Algorithmica 14, pp. 305-322, 1995.

[16] E. Kreyszig, Advanced Engineering Mathematics, John Wiley & Sons, 2011.

[17] C. Li, H. Zhang, B. Hao, and J. Li, A Survey on Routing Protocols for Large-Scale Wireless Sensor Networks. Sensors 11, pp. 3498-3526, 2011.

[18] W-Y. LIN, W-Y. LEE, and T-P. Hong, Adapting Crossover and Mutation Rates in Genetic Algorithms, the Sixth Conference on Artificial Intelligence and Applications, Kaohsiung, Taiwan, 2001.

[19] O. Roeva, S. Fidanova, and M. Paprzycki, Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling. In the Proceedings of the Federated Conference on Computer Science and Information Systems pp. 371-376, 2013.

[20] K. Vekaria and C. Clack, Selective Crossover in Genetic Algorithms: An Empirical Study, volume 1498 of Lecture Notes in Computer Science, pp. 438-447, 1998.

[21] B. Xiao, Q. ZhuGe, and E. H.-M. Sha, Minimum Dynamic Update for Shortest Path Tree Construction, Global Telecommunications Conference, San Antonio, TX, pp. 126-130, 2001.

[22] B. Ye We and K. Chao, Spanning Trees and Optimization Problems, Chapman & Hall, 2004.

[23] M. Sigurd and M. Zachariasen, Construction of Minimum-Weight Spanners, Springer, Verlag Berlin Heidelberg, pp. 797-808, 2004.

[24] A. M. Farley, D. Zappala A. Proskurowski and K. Windisch, Spanners and Message Distribution in Networks, Dicrete Applied Mathematics, pp. 159-171, 2004.

[25] J. Gudmundsson, C. Levcopoulos and G. Narasimhan, Fast Greedy Algorithms for Constructing Sparse Geometric Spanners, SIAM Journal on Computing, pp. 1479-1500, 2002.

[26] G. Navarro, R. Paredes, and E. Chavez, t-Spanners as a Data Structure for Metric Space Searching, International Symposium on String Processing and Information Retrieval, SPIRE, LNCS 2476, pp. 298-309, 2002.