Information Sciences 180 (2010) 3182–3191

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A swarm intelligence approach to the quadratic minimum spanning tree problem

Shyam Sundar, Alok Singh *

Department of Computer and Information Sciences, University of Hyderabad, Hyderabad 500 046, India

ARTICLE INFO

Article history:
Received 2 October 2009
Received in revised form 9 March 2010
Accepted 1 May 2010

Keywords:
Artificial bee colony algorithm
Constrained optimization
Heuristic
Quadratic minimum spanning tree problem
Swarm intelligence

ABSTRACT

The quadratic minimum spanning tree problem (Q-MST) is an extension of the minimum spanning tree problem (MST). In Q-MST, in addition to edge costs, costs are also associated with ordered pairs of distinct edges and one has to find a spanning tree that minimizes the sumtotal of the costs of individual edges present in the spanning tree and the costs of the ordered pairs containing only edges present in the spanning tree. Though MST can be solved in polynomial time, Q-MST is $\mathcal{NP}$-Hard. In this paper we present an artificial bee colony (ABC) algorithm to solve Q-MST. The ABC algorithm is a new swarm intelligence approach inspired by intelligent foraging behavior of honey bees. Computational results show the effectiveness of our approach.

1. Introduction

The quadratic minimum spanning tree problem (Q-MST) is an extension of the well known minimum spanning tree problem (MST) in graphs. In Q-MST, costs are associated not only with edges of the graph but also with ordered pairs of distinct edges and the objective is to find a spanning tree that minimizes the sumtotal of the costs associated with individual edges and the costs resulting from ordered pairs consisting of those edges present in the spanning tree. Formally, let $G = (V, E)$ be a connected undirected graph, where V denotes the set of nodes and E denotes the set of edges. Given a non-negative cost function $w : E \rightarrow \mathbb{R}^+$ associated with edges of G and a non-negative cost function $c : (E \times E - \{(e, e),\ \forall e \in E\}) \rightarrow \mathbb{R}^+$ associated with ordered pairs of distinct edges, the Q-MST problem seeks a spanning tree $T \subseteq E$ that minimizes

$$\sum_{e_1 \in T} \sum_{\substack{e_2 \in T \\ e_2 \neq e_1}} c(e_1, e_2) + \sum_{e \in T} w(e)$$

Q-MST has several practical applications. It occurs when transferring oil from one pipe to another in a situation where the cost depends on the type of interface between two pipes. This quadratic cost structure also arises in the connection of overground and underground cables or in a transportation or road network with turn penalties. The presence of quadratic costs in all these cases leads to the minimum spanning tree problem with quadratic cost instead of the usual linear cost [16].

Q-MST problem was introduced and proved $\mathcal{NP}$-Hard by Asad and Xu [1,15]. Due to the $\mathcal{NP}$-Hard nature of Q-MST, any exact method is not practical even for moderately large instances and one has to look for heuristics.

Assad and Xu [1,15] proposed a branch-and-bound based exact method and two heuristic for the Q-MST and applied them on graph instances with number of nodes between 6 and 15. Even on these small instances, the solutions obtained

* Corresponding author. Tel.: +91 4023134011.
E-mail addresses: mc08pc17@uohyd.ernet.in (S. Sundar), alokcs@uohyd.ernet.in (A. Singh).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2010.05.001

by the two heuristics were far from optimal. Zhou and Gen [16] proposed a genetic algorithm based on Prüfer encoding [11] for the Q-MST problem and observed that this genetic algorithm is superior to two heuristic algorithms of [1,15] on random instances with number of nodes between 6 and 50. Soak et al. [14] proposed edge-window-decoder encoding and showed that a genetic algorithm based on this encoding performs much better than genetic algorithms based on other encodings on Q-MST and other problems. All algorithms were tested by Soak et al. on Euclidean instances with number of nodes between 50 and 100.

In this paper, we have proposed an artificial bee colony algorithm (ABC) to solve Q-MST. The ABC algorithm is a new swarm intelligence technique based on the intelligent foraging behavior of honey bees. This technique was proposed by Karaboga [2]. We have compared our ABC approach with the best approaches. Computational results demonstrate the effectiveness of our approach.

The rest of this paper is organized as follows: Section 2 provides a brief introduction to the artificial bee colony algorithm. Section 3 describes our ABC approach for Q-MST. Computational results are reported in Section 4, whereas Section 5 contains some concluding remarks.

2. Artificial bee colony algorithm

The artificial bee colony (ABC) algorithm is a new swarm intelligence technique inspired by intelligent foraging behavior of honey bees. The first framework of ABC algorithm mimicking the foraging behavior of honey bee swarm in finding good solutions to combinatorial optimization problems was presented by Karaboga [2]. Further developments in the ABC Algorithm have been carried out by Karaboga and Basturk [3–7], Singh [13] and Pan et al. [9]. On the basis of their foraging behaviors, real bees are classified into three groups – employed, scouts and onlookers. All bees which are presently exploiting a food source are termed ‘‘employed”. The employed bees bring loads of nectar from the food source to the hive and may share the information about their food sources with onlooker bees. ‘‘Onlookers” are those bees that wait in the hive for employed bees to share information about their food sources. Those bees, which are presently searching for new food sources in the neighborhood of the hive, are termed ‘‘scouts”. Employed bees share information about their food sources by dancing in a common area in the hive called dance area. The nature and the duration of a dance depends on the nectar content of the food source currently being tapped by the dancing bee. Onlooker bees observe numerous dances before selecting a food source. The onlookers have a tendency to select a food source according to a probability proportional to the nectar content of that food source. Therefore, good food sources attract more bees than the bad ones. Whenever a scout or a onlooker finds a food source it becomes employed. Whenever a food source is exhausted fully, all those employed bees, which are currently exploiting it, abandon it and become scouts or onlookers. Therefore, employed bees and onlookers bees do the job of exploitation, whereas exploration is left to scouts.

Karaboga [2] modeled this intelligent foraging behavior of real honey bee swarm into artificial bee colony (ABC) algorithm to solve real world optimization problems. In ABC algorithm model, Karaboga also classified the colony of artificial bees into same three groups – employed, onlookers and scouts. First half of the colony consists of employed bees, whereas the latter half consists of onlookers. In ABC algorithm, each food source represents a candidate solution to the problem. Each employed bee is associated with a unique food source, and, therefore in ABC algorithm, the number of employed bees is equal to the number of food sources. The nectar amount of a food source is a function of the quality (fitness) of the candidate solution being represented by that food source.

ABC algorithm is an iterative algorithm. It starts by associating each employed bee with a randomly generated food source (solution). Then, during each iteration, each employed bee determines a new neighboring food source of its currently associated food source and computes the nectar amount (fitness) of this new food source. If the nectar amount of this new food source is higher than that of its currently associated food source, then this employed bee moves to this new food source abandoning the old one, otherwise it continues with the old one. When this process is completed by all employed bees, then they start sharing information about their food sources with onlookers. Each onlooker selects a food source probabilistically according to the nectar amount (fitness) of that food source. The probability p_i of selecting a food source i is computed as:

$$p_i = \frac{f_i}{\sum_{j=1}^{k} f_j}$$

where f_i is the fitness of the candidate solution associated with the food source i and k is the total number of food sources. This selection method is known by the name ‘‘roulette wheel selection method” in genetic algorithm community. In roulette wheel selection method, candidate solutions are selected from the population randomly, with their probability of selection proportional to their relative fitness in the population. Thus, fitter candidate solutions have a greater chance of survival than the weaker ones. Therefore, rich food source attracts more onlookers than the poor ones. Once all onlookers have selected their food sources in the aforementioned way, each of them determines a new neighboring food source of its selected food source and computes the nectar amount of this new food source. Among all the neighboring food sources determined by onlookers associated with a particular food source i and food source i itself, best food source will be the new location of food source i. If a candidate solution represented by a food source i does not improve for a predetermined number of iterations, then the solution represented by food source i is considered to be explored fully and hence food source i is presumed to be exhausted and its associated employed bee abandons it to become scout. This scout is transformed into an employed bee

immediately by associating it with a randomly generated new solution. When the new locations of all food sources are determined, then the next iteration of the ABC algorithm begins. This process is repeated again and again until termination criterion is satisfied.

The procedure for determining a new food source in the neighborhood of a particular food source is problem-specific. Initially, Karaboga [2] designed the ABC algorithm for real parameter optimization, and, accordingly a new neighboring food source of a particular food source is generated by changing the value of one randomly chosen solution parameter, while keeping other parameters unchanged. This is done by adding to the current value of the chosen parameter the product of a uniform variate in $[-1, 1]$ and the difference in values of this parameter for this food source and some other randomly chosen food source. But, this approach cannot be used for discrete optimization problems for which it generates at best a random effect. Singh [13] subsequently proposed a method which is appropriate for subset selection problems. In his method, to generate a neighboring solution, an object is randomly dropped from the solution and in its place another object, which is not already present in the solution is added. The object to be added is selected from another randomly chosen solution. If there are more than one candidate object for addition then ties are broken arbitrarily. This approach is based on the idea that if an object is present in one good solution then in all likelihood this object is present in many good solutions. Another advantage of this method is that it keeps in control the number of duplicate solutions in the population. If this method fails to find an object in the randomly chosen solution different from the objects in the original solution then that means that the two solutions are identical. Such a situation was termed "collision" and was resolved by making the employed bee associated with the original solution scout. This eliminates one duplicate solution.

A good survey on the ABC and other algorithms simulating bee swarm intelligence can be found in [8].

3. ABC algorithm for Q-MST

The main features of our ABC algorithm for Q-MST are described below:

3.1. Solution encoding

Edge-set encoding [12] has been used to represent a spanning tree. In edge-set encoding a spanning tree is represented directly by the set of its $n-1$ edges, where n is the number of nodes in the graph. The reason for this choice is that this encoding offers high locality and the problem-specific heuristics can be easily incorporated within the metaheuristic.

3.2. Initialization

The algorithm is initialized by associating a randomly generated solution to each employed bee. For the purpose of initialization phase, we have defined a special cost, called potential cost, which is associated with each edge. The potential cost of an edge e is the sum of all costs that can be attributed to e, i.e., $w(e) + \sum_{e_1 \in E,\, e_1 \neq e} (c(e, e_1) + c(e_1, e))$. Each initial random solution is generated by an iterative process that is similar to Prim's Algorithm [10]. However, instead of picking a least cost edge connecting a node in the partially constructed tree to a node not in the tree, an edge is picked at random from all the candidate edges using the roulette wheel selection, where the probability of selection of an edge is inversely proportional to the potential cost of that edge.

3.3. Probability of selecting a food source

In our ABC algorithm, for selecting a food source for an onlooker bee, we have used the binary tournament selection method in place of the usual roulette wheel selection method. In the binary tournament selection method, two different food sources are picked at random and better of the two food sources (according to cost function defined in Section 1) is selected with probability bt, otherwise worse of two food sources is selected with probability $1 - bt$. Roulette wheel selection method was also tried, but computational experiments showed that the performance of the binary tournament selection method is better than the roulette wheel selection method in terms of solution quality. Another advantage of the binary tournament selection method is that it is computationally efficient than the roulette wheel selection method.

3.4. Determination of a food source in the neighborhood of a food source

Our method follows from the ideas presented in [13]. In order to generate a solution in the neighborhood of a particular solution say solution i, first we create a copy i' of solution i. An edge e is randomly deleted from i'. This delete operation results into partitioning of the spanning tree into two components and makes solution i' infeasible. To make i' feasible again, we randomly select another solution j different from solution i'. An edge e', different from edge e is searched in j, which can connect the two components of i' and has minimum cost among all such candidate edges in solution j. Here cost of edge e' means the sum of cost of edge e' itself and the intercost with remaining edges of i'. If there is no edge e' in solution j different from edge e, which can connect the two components of solution i', then solution i' is restored by adding the deleted edge e into it. The edge e is placed in a tabu list so that it cannot be selected again in subsequent trials and another edge is deleted

randomly and the whole procedure is repeated. Note that searching for an edge connecting the two components in another solution is more efficient than searching all possible edges connecting the two components. This is based on the observation that if an edge occurs in one good solution then the same edge will occur in many good solutions. If the above procedure fails even after t_t trials then it shows a lack of diversity in the population. If such a situation arises while determining a new neighboring solution for an employed bee, then employed bee associated with this food source abandons it to become scout so that the population can be made more diversified. This scout is immediately made employed by associating it with a new randomly generated food source. However, if such a situation arises while determining a new neighboring solution for an onlooker, then we discard this solution by artificially assigning it an objective value higher than the associated employed bee solution, so that it cannot be selected in next generation. It should be noted that it is worthless to generate a food source randomly for an onlooker, because for survival this randomly generated food source has to compete with the original food source as well as with the food sources of all onlookers which are associated with the same original food source. Hence, it is highly unlikely that such a randomly generated solution survives.

Algorithm 1 gives the pseudo-code of the aforementioned procedure for determining a neighboring solution. The algorithm returns the newly determined neighboring solution if it succeeds in finding one within t_t trials, otherwise it returns the $\emptyset$.

Algorithm 1. Determination of Neighboring Food Source

```
input: A food source F_i
output; A neighboring food source F_i' or ∅ if the procedure fails to generate a neighboring food source even after t_t trials
create a copy F_i' of food source F_i
trial ← 0
clear the tabu list
while trial <t_t do
    delete an edge e not belonging to tabu list from food source F_i'
    add e to tabu list
    randomly select another food source F_j different from F_i'
    find the least cost candidate edge e' in F_j different from e
    // see Section 3.4
    if no candidate edge exists then
        restore F_i' by adding back the deleted edge e
        trial ← trial + 1
    else
        add e' to F_i'
        break
if trial = t_t then
    F_i' ← ∅
```

Fig. 1 explains the procedure for determining the neighboring food source with the help of an example. Suppose we have a complete graph with 6 nodes. Fig. 1(a) and (b) respectively shows the intercost matrix and the graph itself. The diagonal elements of this intercost matrix represent the costs of individual edges, whereas off diagonal elements represent the intercosts among edges. Consider the spanning tree shown in Fig. 1(c). It has a cost of 374. In order to generate a solution in the neighborhood of this solution, we first delete an edge say $e15$ (as shown in Fig. 1(d) by dashed line). This creates two components. The one component contains nodes 1, 2, 6, whereas the other contains nodes 3, 4, 5. Now, another solution is chosen randomly say the solution shown in Fig. 1(e). This solution contains 3 candidate edges $e6$, $e8$ and $e12$, which can be inserted in place of $e15$. The total costs of $e6$, $e8$, $e12$ are 172 (88 + 16 + 12 + 9+6 + 8 + 19 + 9+5), 68 (2 + 13 + 16 + 13 + 12 + 2 + 1 + 7+2), 117 (28 + 8 + 17 + 11 + 9+11 + 17 + 4+12) respectively. Among these three candidate edges, edge $e8$ has the least cost. Therefore, edge $e8$ is selected for insertion in place of $e15$. This leads to a tree shown in Fig. 1(f), which has a total cost of 333.

3.5. Other features

If a food source (solution) does not improve for a predetermined number of iterations*limit*, then employed bee associated with that food source abandons it and becomes a scout. The parameter *limit* is an important control parameter of the ABC algorithm as it is responsible for maintaining the delicate balance between exploration and exploitation. A small value of *limit* parameter favors exploration over exploitation whereas reverse is true for large value. There is also a second possibility in which an employed bee can become a scout as described in Section 3.4. This second possibility is introduced as an autocorrection measure as it forces the ABC algorithm to move towards exploration whenever there is a lack of diversity in the population. Unlike the traditional ABC algorithm, in our ABC algorithm there is no upper limit on the number of scouts in a single iteration. In fact, number of scout in an iteration depends on the aforementioned two condition.

	e1	e2	e3	e4	e5	e 6	e7	e8	e9	e10	e11	e12	e13	e14	e15
e1	84	7	18	16	14	16	7	13	10	2	3	8	11	20	4
e2	7	41	7	13	17	12	9	8	10	3	11	3	4	8	16
e3	10	3	23	19	10	8	14	17	12	3	10	14	2	20	5
e4	18	19	5	16	11	14	7	12	1	17	14	3	11	17	2
e5	6	6	5	8	37	6	7	10	14	18	5	16	3	6	15
e6	8	15	5	4	11	88	9	17	19	9	5	4	12	15	20
e7	13	1	17	9	20	13	27	7	17	20	16	11	15	19	8
e8	2	18	3	18	13	13	17	2	1	7	2	6	10	5	20
e9	1	10	12	18	18	12	2	16	10	8	8	17	18	14	7
e10	6	7	4	20	5	9	12	13	10	4	20	11	9	9	16
e11	1	10	17	4	19	6	7	12	2	16	80	9	5	9	2
e12	11	14	1	15	5	5	15	8	17	4	12	28	6	20	17
e13	13	12	18	9	16	8	15	2	19	16	10	18	36	14	19
e14	9	4	12	9	10	17	5	4	4	14	19	7	1	5	19
e15	9	9	10	18	18	17	5	4	11	4	11	20	13	6	45

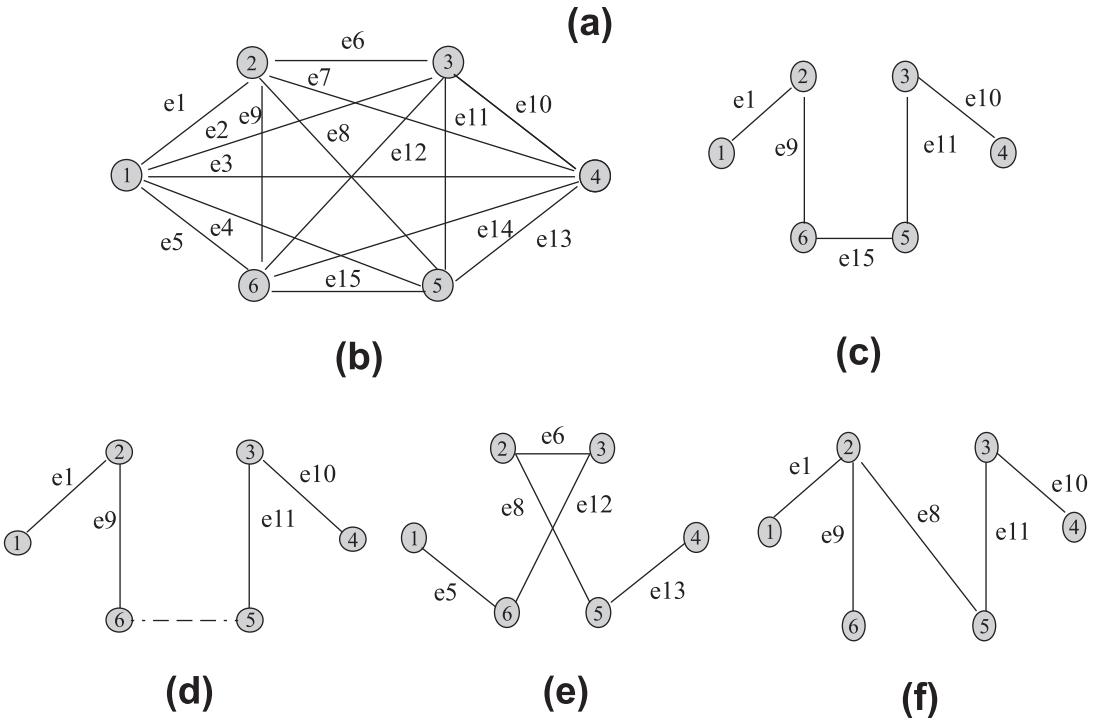

Fig. 1. An example illustrating the determination of a neighboring food source.

3.6. Local search

After completion of ABC algorithm, an attempt is made to further ameliorate the quality of the best solution obtained through ABC algorithm by using a local search. Our local search follows an iterative process. During each iteration the local search considers each edge of the solution one-by-one. It deletes the edge in consideration from the solution and evaluates each graph edge connecting the two resulting components for inclusion. The edge that yields a solution of least cost will be

selected for inclusion. The local search is applied again and again till a complete iteration fails to ameliorate the solution. We have also experimented with this local search by placing it inside the ABC algorithm but the resulting approach was found to be too slow to be of any practical use.

Algorithm 2 gives the pseudo-code of our ABC approach for the Q-MST.

Algorithm 2. ABC algorithm for Q-MST

```
input: A graph G = (V, E)
output: bests, the best spanning tree found by the algorithm
bests ← ∅
artificially assign the worst possible fitness to bests
foreach employed bee i do
    generate a random food source F_i
    // see Section 3.2
    associate F_i with i
    if F_i is better than bests then
        bests ← F_i
repeat
    foreach employed bee i do
        generate a new food source F'_i in the neighborhood of F_i
        // see Algorithm 1
        if F'_i = ∅ then
            replace F_i with a randomly generated food source
        else
            if F'_i is better than F_i then
                replace F_i with F'_i
            else
                if F_i does not change for limit iterations then
                    replace F_i with a randomly generated food source
        if F_i is better than bests then
            bests ← F_i
    foreach onlooker bee j do
        select a food source F_j for onlooker j// see Section 3.3
        generate a new food source F'_j in the neighborhood of F_j
        // see Algorithm 1
        if F'_j = ∅ then
            artificially assign a fitness value worse than F_j to F'_j
    foreach onlooker bee j do
        if F'_j is better than F_j then
            replace F_j with F'_j
        if F_j is better than bests then
            bests ← F_j
until stopping condition satisfied
improve bests using local search // see Section 3.6
```

4. Computational results

Our ABC algorithm for the Q-MST problem has been implemented in C and executed on a Linux based 3.0 GHz core 2 duo system with 2 GB RAM. In all our computational experiments with our ABC algorithm we have used a population of 400 bees, 200 of these bees are employed and the remaining 200 are onlookers. We have set *limit* = 150, t_t = 5 and *bt* = 0.8. All these parameter values are set empirically after a large number of trials. These parameter values provide good results though they may not be optimal for all instances. In subsequent subsections, we compare the performance of our ABC approach with genetic algorithms proposed in [16,14], which are the best heuristics known for the Q-MST Problem.

4.1. Comparison of our ABC approach with genetic algorithm of Zhou and Gen

First, we will compare our ABC approach with genetic algorithm of Zhou and Gen [16]. As the test instances used in [16] were not available, therefore, to compare our ABC approach with the genetic algorithm proposed in [16], we have reimple-

mented this genetic algorithm. This also facilitated comparison on larger instances as Zhou and Gen originally tested their genetic algorithm on smaller instances with number of nodes between 6 and 50. Hereafter, the genetic algorithm of Zhou and Gen will be referred to as ZG-GA. All parameter settings for ZG-GA are same as used in [16] except the termination condition.

We have compared our ABC approach with ZG-GA on a set of 18 instances that were generated exactly in the manner as in [16]. All instances represent complete graphs with integer edge-costs uniformly distributed in [1,100]. The intercosts between edges are also integers and uniformly distributed in [1,20]. For each value of $n \in 25, 50, 100, 150, 200, 250$, there are 3 instances leading to a total of 18 instances. We have experimented only up to instances with $n = 250$ due to the prohibitive size of intercost matrix at higher values of n.

We have allowed our ABC approach to execute till the best solution fails to improve over $max(10n, 1000)$ iterations. To allow a fair comparison, ZG-GA terminates when the best solution does not improve over an equivalent number of solution generated. The ZG-GA uses a population of 300, therefore, it terminates when the best solution does not improve over $max(13.33n, 1333)$ iterations. Each approach is executed 20 times on each instance. The local search described in Section 3.6 can be used to improve the best solution of ZG-GA also. Therefore, to compare the ABC algorithm with local search, we have incorporated local search with ZG-GA also.

Tables 1 and 2 compare the performance of ZG-GA with ABC. Table 1 reports the performance of two approaches without using the local search, whereas Table 2 reports the same when the local search described in Section 3.6 is used to improve

Table 1
Results of ABC algorithm without local search and ZG-GA without local search on 18 random instances.

Instance	ZG-GA				ABC			
	Min	Avg	SD	TET	Min	Avg	SD	TET
25.1	5301	5460.05	81.37	1.14	5085	5085.85	3.71	0.91
25.2	5293	5431.10	73.57	1.00	5081	5101.50	6.81	1.02
25.3	5273	5389.50	63.58	1.16	4962	4962.00	0.00	1.05
50.1	22,737	23042.70	197.05	3.89	21,126	21160.60	35.05	8.68
50.2	22,520	22987.10	188.91	3.53	21,123	21187.55	38.91	8.83
50.3	22,522	22924.10	187.30	3.92	21,059	21095.10	60.29	9.51
100.1	94,436	95269.90	460.75	28.17	89,098	89430.30	175.02	116.52
100.2	93,901	95134.55	486.44	20.51	89,202	89550.00	224.87	115.80
100.3	94,582	95264.10	382.27	35.58	89,007	89272.55	186.83	98.71
150.1	216,274	217621.65	632.34	106.62	205,638	206470.30	549.31	444.13
150.2	216,499	217807.55	758.47	106.70	205,874	206311.65	241.59	373.85
150.3	216,440	217633.30	569.42	110.11	205,634	206168.55	316.51	432.44
200.1	390,370	391159.85	833.92	392.66	371,797	372558.45	389.85	1139.51
200.2	388,929	390574.50	905.05	400.40	371,951	372339.90	317.48	1153.35
200.3	388,664	390690.50	1103.59	361.57	372,156	373029.90	1425.28	1274.43
250.1	612,520	614215.20	1118.04	1915.19	587,928	588809.60	589.13	2557.50
250.2	611,079	614537.65	1402.57	1767.05	588,068	588781.70	386.55	2834.89
250.3	611,826	614209.00	1219.67	1819.58	587,927	588584.10	483.73	2322.16

Table 2
Results of ABC algorithm with local search and ZG-GA with local search on 18 random instances.

Instance	ZG-GA				ABC			
	Min	Avg	SD	TET	Min	Avg	SD	TET
25.1	5085	5219.40	76.57	1.14	5085	5085.85	3.71	0.91
25.2	5121	5217.80	59.72	1.00	5081	5101.20	6.64	1.02
25.3	4962	5078.80	83.35	1.16	4962	4962.00	0.00	1.05
50.1	21,363	21699.30	165.83	3.90	21,126	21157.25	34.40	8.68
50.2	21,311	21721.85	136.93	3.54	21,123	21179.85	32.47	8.84
50.3	21,393	21681.90	177.81	3.92	21,059	21091.95	59.67	9.51
100.1	90,051	90530.60	273.26	28.93	89,098	89404.60	167.87	116.66
100.2	90,248	90820.95	332.93	21.17	89,202	89520.45	190.05	115.95
100.3	90,132	90645.70	260.72	36.36	89,007	89242.60	138.59	98.88
150.1	208,174	208713.05	436.44	111.50	205,619	206404.30	405.97	444.87
150.2	207,907	208623.70	414.48	112.32	205,874	206300.55	243.36	374.33
150.3	207,622	208601.55	421.41	114.54	205,634	206160.10	316.07	432.93
200.1	375,468	376233.55	650.49	409.82	371,797	372527.60	381.44	1141.42
200.2	375,119	376205.70	486.18	420.46	371,864	372306.60	311.74	1155.60
200.3	375,181	376347.15	577.42	381.04	372,156	372842.90	735.21	1276.71
250.1	592,265	593658.70	929.13	1970.74	587,924	588785.10	578.65	2563.41
250.2	592,084	593539.45	837.63	1822.70	588,068	588731.45	368.08	2840.91
250.3	591,841	593443.30	704.45	1870.19	587,883	588534.95	463.20	2328.29

Table 3
Results of ABC algorithm, EWD + CGPX with dK-TCR, EWD + ANX with dK-TCR and ZG-GA on 6 Euclidean instances.

Instance	ZG-GA				EWD + CGPX (dK-TCR)[a]				EWD + ANX (dK-TCR) [a]				ABC			
	Min	Avg	SD	TET	Min	Avg	SD	TET	Min	Avg	SD	TET	Min	Avg	SD	TET
50	27,042	27734.85	438.57	3.49	25,339	25488.50	78.55	302.95	25,339	25455.56	70.94	343.00	25,200	25201.30	2.37	7.89
60	38,150	38640.70	435.31	4.80	36,014	36237.29	147.66	448.40	36,086	36262.50	126.05	495.70	35,544	35617.40	33.57	10.24
70	51,660	52854.85	530.68	6.72	48,601	48838.94	174.74	699.70	48,538	48877.77	155.32	716.60	48,125	48189.25	39.51	18.69
80	68,064	68853.80	437.71	8.67	63,831	64162.39	216.06	1044.45	63,546	63933.63	235.12	1086.70	63,066	63186.70	50.32	16.73
90	84,840	86072.30	743.66	10.75	80,033	80407.34	199.07	1323.75	79,627	80168.46	222.96	1337.20	78,879	79093.95	117.02	26.38
100	103,949	105769.35	709.83	13.26	98,774	99325.42	304.86	1773.45	98,342	98932.05	226.19	1828.90	96,783	97100.50	153.14	30.89

[a] Executed on a 1.6 GHz Pentium 4 System.

the best solution obtained by the two approaches. For each instance these tables report the best and average solution found by each of the two methods, standard deviation of solution values and average execution time in seconds. These tables clearly show the superiority of ABC over ZG-GA in terms of solution quality. Best and average solutions found by ABC are always better than ZG-GA. Moreover, standard deviations of solution values are also less for ABC. This shows its robustness. Except for small instances, ZG-GA is faster than the ABC approach. This is due to the premature convergence of ZG-GA at suboptimal solutions in many cases.

4.2. Comparison of our ABC approach with two best genetic algorithms of Soak et al.

We have used the same set of Euclidean instances as used in [14]. This set contains 6 instances each representing a complete graph with number of nodes between 50 and 100. In all these instances, nodes are distributed uniformly at random on a 500×500 grid. The edge costs are the integer Euclidean distance between these points. The intercost between edges are uniformly distributed between [1,20]. Soak et al. presented a number of genetic algorithms and we have taken the two best performing genetic algorithms (EWD + CGPX with dK-TCR and EWD + ANX with dK-TCR) for comparison with our ABC approach. We have also included the ZG-GA in this comparison. To allow a fair comparison we have not used the local search of Section 3.6 with our ABC approach or with ZG-GA. The genetic algorithms in Soak et al. generates a total of 10,00,000 solutions, therefore, we have also allowed our ABC approach and ZG-GA to generate only 10,00,000 solutions. Like EWD + CGPX with dK-TCR and EWD + ANX with dK-TCR, we have also executed our ABC approach and ZG-GA 20 times on each instance.

Table 3 reports the performance of ZG-GA, EWD + CGPX with dK-TCR, EWD + ANX with dK-TCR along with our ABC approach. The results reported for EWD + CGPX with dK-TCR, EWD + ANX with dK-TCR are obtained from the results presented in Table IX of [14]. As table IX of [14] presents the results in terms of gap, we have converted the results presented there to our format before reporting. As shown by this table, ABC approach has performed best in terms of solution quality followed by EWD + ANX with dK-TCR and EWD + CGPX with dK-TCR. ZG-GA has performed the worst. Best and average solution quality of our approach are always better than other approaches. Moreover, its standard deviation of solution values is also the smallest among all the four approaches. EWD + CGPX with dK-TCR and EWD + ANX with dK-TCR were executed on Pentium 4, 1.6 GHz, therefore, it is not possible to compare their execution times with our ABC approach and ZG-GA. However, we can safely say that even after compensating for processing speed, ABC approach and ZG-GA are faster than the two genetic algorithms. Though ZG-GA is fastest, it has performed the worst.

We have also executed our ABC approach and ZG-GA on these 6 Euclidean instances with the same termination condition as used in Section 4.1. Table 4 reports the results obtained without using the local search, whereas Table 5 reports the results with local search. Best and average solutions obtained by our ABC approach improved slightly in comparison to Table 3 at the expense of increased computation times. Except for one instance where the best solution found by ZG-GA with local search is better than the ABC with local search, solution quality of our ABC approach is always better. However, on the same instance ABC without local search found the better best solution than ZG-GA without local search. Therefore, even on this instance the superiority of ZG-GA with local search over ABC with local search is purely due to the local search.

Table 4
Results of ABC algorithm without local search and ZG-GA without local search on 6 Euclidean instances using the termination criterion of Section 4.1.

Instance	ZG-GA				ABC			
	Min	Avg	SD	TET	Min	Avg	SD	TET
50	27,229	27821.15	391.84	2.87	25,200	25201.30	2.37	4.35
60	37,921	38698.10	456.02	4.41	35,487	35615.35	42.91	8.44
70	51,760	52556.15	515.70	7.04	48,125	48173.10	31.28	16.84
80	67,732	68751.25	496.28	10.53	63,057	63150.85	46.24	20.83
90	84,154	85886.20	797.53	16.02	78,879	79090.15	129.69	37.49
100	103,976	105257.60	645.33	19.37	96,750	97038.70	154.05	76.92

Table 5
Results of ABC algorithm with local search and ZG-GA with local search on 6 Euclidean instances using the termination criterion of Section 4.1.

Instance	ZG-GA				ABC			
	Min	Avg	SD	TET	Min	Avg	SD	TET
50	25,262	25414.30	99.48	2.87	25,200	25201.30	2.37	4.35
60	35,447	35860.55	129.93	4.43	35,466	35603.10	48.31	8.45
70	48,365	48609.95	195.97	7.12	48,125	48166.20	27.71	16.86
80	63,248	63664.60	215.92	10.72	63,022	63130.30	47.55	20.89
90	79,361	79828.60	282.68	16.40	78,879	79076.65	122.44	37.59
100	97,414	98021.35	397.91	20.07	96,750	97018.75	139.48	77.12

5. Conclusions

In this paper we have presented an artificial bee colony algorithm for the Q-MST problem. We have compared our approach against the genetic algorithms proposed in [16,14]. Our approach obtained better quality solutions than other approaches. Though, the genetic algorithm of [16] is faster than our ABC approach, it performed worst in terms of solution quality among all the approaches considered in this paper.

As a future work, we intend to extend our approach to other $\mathcal{NP}$-Hard constrained spanning tree problems such as diameter-constrained minimum spanning tree (DCMST) problem, bounded-diameter minimum spanning tree (BDMST) problem, etc.

Acknowledgments

We are thankful to Dr. Sang-Moon Soak for providing the Q-MST instances used in [14]. We also thank three anonymous reviewers for their valuable comments and suggestions, which helped in improving the presentation of this paper.

References

[1] A. Assad, W. Xu, The quadratic minimum spanning tree problem, Naval Research Logistics 39 (1992) 399–417.
[2] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[3] D. Karaboga, B. Basturk, An artificial bee colony (ABC) algorithm for numeric function optimization, in: Proceedings of the IEEE swarm intelligence symposium, Indianapolis, USA, May 12–14, 2006.
[4] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (2007) 459–471.
[5] D. Karaboga, B. Basturk, Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems, Lecture Notes in Artificial Intelligence, vol. 4529, Springer-Verlag, Berlin, 2007. pp. 789–798.
[6] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.
[7] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Applied Mathematics and Computation 214 (2009) 108–132.
[8] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artificial Intelligence Review 31 (2009) 61–85.
[9] Q.-K. Pan, M.F. Tasgetiren, P.N. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Information Sciences (2010), doi:10.1016/j.ins.2009.12.025.
[10] R.C. Prim, Shortest connection networks and some generalizations, Bell Systems Technical Journal 36 (1957) 1389–1401.
[11] H. Prüfer, Neuer beweis eines satzes über permutationen, Archiv für Mathematik and Physik 27 (1918) 742–744.
[12] G.R. Raidl, B.A. Julstrom, Edge-sets: an effective evolutionary coding of spanning trees, IEEE Transactions on Evolutionary Computation 7 (2003) 225–239.
[13] A. Singh, An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem, Applied Soft Computing 9 (2009) 625–631.
[14] S.-M. Soak, D.W. Corne, B.-H. Ahn, The edge-window-decoder representation for tree-based problems, IEEE Transactions on Evolutionary Computation 10 (2006) 124–144.
[15] W. Xu, On the quadratic minimum spanning tree problem. in: M. Gen, W.Xu. Ashikaga (Eds.), Proceedings of 1995 Japan–China International Workshops on Information Systems, 1995, pp. 141–148.
[16] G. Zhou, M. Gen, An effective genetic algorithm approach to the quadratic minimum spanning tree problem, Computers and Operations Research 25 (1998) 229–237.