Applied Soft Computing 9 (2009) 625–631

Contents lists available at ScienceDirect

Applied Soft Computing

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

An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem

Alok Singh *

Department of Computer and Information Sciences, University of Hyderabad, Hyderabad 500046, Andhra Pradesh, India

ARTICLE INFO

Article history:
Received 4 March 2008
Received in revised form 16 July 2008
Accepted 1 September 2008
Available online 9 September 2008

Keywords:
Artificial bee colony algorithm
Constrained optimization
Leaf-constrained minimum spanning tree
Swarm intelligence

ABSTRACT

Given an undirected, connected, weighted graph, the leaf-constrained minimum spanning tree (LCMST) problem seeks on this graph a spanning tree of minimum weight among all the spanning trees of the graph that have at least ℓ leaves. In this paper, we have proposed an artificial bee colony (ABC) algorithm for the LCMST problem. The ABC algorithm is a new metaheuristic approach inspired by intelligent foraging behavior of honeybee swarm. We have compared the performance of our ABC approach against the best approaches reported in the literature. Computational results demonstrate the superiority of the new ABC approach over all the other approaches. The new approach obtained better quality solutions in shorter time.

1. Introduction

Given an undirected, connected, weighted graph with n nodes and a positive integer $\ell(2 \leq \ell < n-1)$, the leaf-constrained minimum spanning tree (LCMST) problem seeks on this graph a spanning tree that contains at least ℓ leaves and has minimum total weight among all such spanning trees. Formally, let $G = (V, E)$ be an undirected, connected graph, where V denotes the set of nodes and E denotes the set of edges. Given a non-negative weight function $w : E \rightarrow \mathbb{R}^+$ associated with its edges and a positive integer $\ell(2 \leq \ell < n-1)$, the LCMST problem seeks a spanning tree $T \subseteq E$ that has at least ℓ leaves and that minimizes:

$$W(T) = \sum_{e \in T} w(e)$$

In general, this problem is NP-Hard [1]. If the leaf constraint ℓ is less than the number of leaves in an unconstrained minimum spanning tree (MST), then this MST is also a LCMST. Therefore, the LCMST problem can be solved in polynomial time in this case. However, if ℓ is larger than the number of leaves in any MST, then the LCMST problem is NP-Hard. Usually, the difficulty of the problem increases with increase in ℓ.

The LCMST problem has several practical applications. It finds applications in facilities location, circuit and network designs [2]. This problem can be considered as an extension of the p-median problem, where medians are also connected among themselves [3].

Several heuristics for the LCMST problem work by first computing a MST and then transforming this MST into a leaf-constrained spanning tree (LCST), i.e., a spanning tree with at least ℓ leaves. Deo and Micikevicius [1] presented one such heuristic. It iteratively tries to transform a MST into a LCST. During each iteration, it interchanges a tree edge with a non-tree edge so that the number of leaves in the tree increases with minimum increase in tree's weight. This process is repeated until either the number of leaves in the tree reaches ℓ or no edge interchange is possible that increases the number of leaves. Therefore, this heuristic some times fails to find a LCST. Its time complexity is $O(n^4)$. Julstrom [4] proposed another heuristic called ML that also begins by computing a MST. Then it iteratively transforms this MST into a LCST. During each iteration, it selects an interior node (a node that is not a leaf) from the set of current interior nodes and converts it into a leaf. The interior node is selected according to the following criterion. It tries each interior node as leaf one by one. During each trial it finds a MST on the graph induced by the remaining interior nodes and then connects all leaves including the new one to the nearest interior node. The interior node for which the resulting spanning tree is of minimum weight is chosen for conversion to leaf. This process is repeated until the number of leaves in the spanning tree reaches ℓ. If the underlying graph is complete then ML always finds a LCST. The complexity of ML is also $O(n^4)$. Julstrom [4] observed that for $\ell = 0.6n$, ML finds lower-weight

* Tel.: +91 40 23134011.
E-mail address: alokcs@uohyd.ernet.in.

1568-4946/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2008.09.001

LCSTs than the heuristic of Deo and Micikevicius [1] does and for $\ell = 0.9n$ the latter heuristic always fails to find a LCST.

Edelson and Gargano [5] developed a permutation-coded genetic algorithm for the LCMST problem that encodes a LCST using $3n - \ell - 2$ symbols. A two-level decoder converts this encoding first into Prüfer code [6] and then this Prüfer code into equivalent LCST. This scheme represents only the feasible solutions. However, chromosome length is considerably large with this scheme. Julstrom [2] presented two generational genetic algorithms. One of these two genetic algorithms uses the Blob code [7], whereas the other uses the subset coding. The subset coding represents a LCST by the set of its interior nodes only. A two-step procedure converts this coding into its equivalent LCST. The first step constructs a MST on the graph induced by the set of interior nodes, whereas the latter step connects each leaf to its nearest interior node. Like the encoding of Edelson and Gargano [5], both of these encodings also represent feasible solutions only. At the same time their chromosome lengths are much smaller. The chromosome length with Blob code is $n - 2$, whereas that with subset coding is $n - \ell$. The genetic operators for these two genetic algorithms are designed in such a way that they generate feasible solutions only. Computational results on the test instances considered in [2] showed that subset-coded genetic algorithm always finds the LCST of lowest weight followed by ML heuristic. The Blob-coded genetic algorithm performed even worse than ML heuristic. Hereafter, the subset-coded genetic algorithm will be referred to as SCGA.

Singh and Baghel [8] presented two metaheuristic approaches, one based on ant-colony optimization (ACO) and the other based on tabu search, for the LCMST problem and compared their approaches with the subset-coded genetic algorithm and ML heuristic [2]. Both of these methods use the subset coding [2] to represent a LCST and construct the equivalent LCST according to the two-step procedure described above. The ACO approach constructs a LCST by first identifying its $(n - \ell)$ interior nodes through ant and then constructs the complete LCST by the two-step procedure. Starting from a random initial solution, the tabu search iteratively moves from one solution to another by exchanging an interior node with a leaf node until the stopping criterion is satisfied. During each iteration, among all valid exchange moves, the move which results in a LCST of least cost is selected (even if its cost is more than the current solution). Once a move has been performed its exactly reverse move is forbidden for the duration determined by tabu tenure. This tabu search also has an in-built cycle detection mechanism. Computational results demonstrated the superiority of these approaches over subset-coded genetic algorithm and ML heuristic. The tabu search and ACO performed quite similar in terms of solution quality, but tabu search was much faster than the ACO approach. Hereafter, these tabu search and ACO approaches will be, respectively referred to as TS-LCMST and ACO-LCMST.

In this paper, we have proposed an artificial bee colony (ABC) algorithm for the LCMST problem. The artificial bee colony algorithm is a new metaheuristic approach, proposed by Karaboga [9]. It is inspired by the intelligent foraging behavior of honeybee swarm. We have compared our ABC approach against the ACO-LCMST and TS-LCMST approaches proposed in [8] and SCGA proposed in [2]. Computational results demonstrate the effectiveness of our approach in comparison to these approaches.

The rest of this paper is organized as follows: Section 2 provides an introduction to the artificial bee colony algorithm. Section 3 describes our ABC algorithm for the LCMST problem. Computational results are presented in Section 4, whereas Section 5 outlines some conclusions.

2. The artificial bee colony algorithm

The artificial bee colony algorithm is a new population-based metaheuristic approach proposed by Karaboga [9] and further developed by Karaboga and Basturk [10–13]. This approach is inspired by the intelligent foraging behavior of honeybee swarm. The foraging bees are classified into three categories—employed, onlookers and scouts. All bees that are currently exploiting a food source are classified as "employed". The employed bees bring loads of nectar from the food source to the hive and may share the information about food source with onlooker bees. "Onlookers" are those bees that are waiting in the hive for the information to be shared by the employed bees about their food sources and "scouts" are those bees which are currently searching for new food sources in the vicinity of the hive. Employed bees share information about food sources by dancing in a common area in the hive called dance area. The duration of a dance is proportional to the nectar content of the food source currently being exploited by the dancing bee. Onlooker bees which watch numerous dances before choosing a food source tend to choose a food source according to the probability proportional to the quality of that food source. Therefore, the good food sources attract more bees than the bad ones. Whenever a bee, whether it is scout or onlooker, finds a food source it becomes employed. Whenever a food source is exploited fully, all the employed bees associated with it abandon it, and may again become scouts or onlookers. Scout bees can be visualized as performing the job of exploration, whereas employed and onlooker bees can be visualized as performing the job of exploitation.

Motivated by this foraging behavior of honeybees, Karaboga [9] proposed the artificial bee colony algorithm. In the ABC algorithm, each food source represents a possible solution to the problem under consideration and the nectar amount of a food source represents the quality of the solution represented by that food source. In this algorithm also colony of artificial bees (bees for short) has same three types of bees—employed, onlookers and scouts. First half of the bee colony comprises employed bees, whereas the latter half contains the onlookers. The ABC algorithm assumes that there is only one employed bee for every food source, i.e., the number of food sources is same as the number of employed bees. The employed bee of an abandoned food source becomes a scout and as soon as it finds a new food source it again becomes employed. The ABC algorithm is an iterative algorithm. It starts by associating all employed bees with randomly generated food sources (solution). Then during each iteration, every employed bee determines a food source in the neighborhood of its currently associated food source and evaluates its nectar amount (fitness). If its nectar amount is better than that of its currently associated food source then that employed bee moves to this new food source leaving the old one, otherwise it retains its old food source. When all employed bees have finished this process, they share the nectar information of the food sources with the onlookers, each of whom selects a food source according to a probability proportional to the nectar amount of that food source. The probability p_i of selecting a food source i is determined using the following expression:

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

where f_i is the fitness of the solution represented by the food source i and m is the total number of food sources. Clearly, with this scheme good food sources will get more onlookers than the bad ones. After all onlookers have selected their food sources, each of them determines a food source in the neighborhood of his chosen food source and computes its fitness. The best food source among all the neighboring food sources determined by the onlookers associated with a particular food source i and food source i itself,

will be the new location of the food source i. If a solution represented by a particular food source does not improve for a predetermined number of iterations then that food source is abandoned by its associated employed bee and it becomes a scout, i.e., it will search for a new food source randomly. This tantamounts to assigning a randomly generated food source (solution) to this scout and changing its status again from scout to employed. After the new location of each food source is determined, another iteration of ABC algorithm begins. The whole process is repeated again and again till the termination condition is satisfied.

The food source in the neighborhood of a particular food source is determined by altering the value of one randomly chosen solution parameter and keeping other parameters unchanged. This is done by adding to the current value of the chosen parameter the product of an uniform variate in $[-1, 1]$ and the difference in values of this parameter for this food source and some other randomly chosen food source. Formally, suppose each solution consists of d parameters and let $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ be a solution with parameter values x_{i1}, x_{i2}, …, x_{id}. In order to determine a solution x'_i in the neighborhood of x_i, a solution parameter j and another solution $x_k = (x_{k1}, x_{k2}, \ldots, x_{kd})$ are selected randomly. Except for the value of the selected parameter j, all other parameter values of x'_i are same as x_i, i.e., $x'_i = (x_{i1}, x_{i2}, \ldots, x_{i(j-1)}, x_{ij}, x_{i(j+1)}, \ldots, x_{id})$. The value x'_i of the selected parameter j in x'_i is determined using the following formula:

$$x'_{ij} = x_{ij} + u(x_{ij} - x_{kj})$$

where u is an uniform variate in $[-1, 1]$. If the resulting value falls outside the acceptable range for parameter j, it is set to the corresponding extreme value in that range.

3. The ABC algorithm for the LCMST problem (ABC-LCMST)

Our ABC algorithm for the LCMST problem uses the subset encoding proposed in [2] to represent a LCST. The main features of our ABC algorithm are described below:

Initialization: The algorithm is initialized by assigning a randomly generated solution to every employed bee.

Probability of selecting a food source: As LCMST is a minimization problem therefore the probability p_i of selecting a food source i is determined using the following expression:

$$p_i = \frac{1/W(T_i)}{\sum_{j=1}^{m} 1/W(T_j)}$$

where $W(T_i)$ is the cost of the LCST represented by the food source i and m is the total number of food sources.

Determination of a food source in the neighborhood of a food source: LCMST is a discrete optimization problem. Moreover, we have represented the solutions by the set of their interior nodes, i.e., there is no ordering among interior nodes. Therefore, the method described in the last paragraph of previous section cannot be applied to LCMST problem. Even selecting a random node from another randomly chosen solution for altering the solution in the aforementioned way at the best produces a random effect. It is desirable to utilize somehow the information gathered about the food source by another employed bee while generating a neighboring solution. This may help in producing a good neighboring solution. In the LCMST problem, if a node is an interior node in one good solution then it is highly likely that it is an interior node in many other good solutions also. We have used this observation while generating the neighboring solutions. We randomly remove an interior node from the solution and in its place we insert an interior node selected from another randomly chosen solution. The node selected for insertion from another solution should be different from all other nodes present in the solution as well as from the node that has been removed. Ties are broken arbitrarily. If we cannot find any such node then that means that the two solutions are identical. We call such a situation a collision. If a collision occurs while generating a neighboring solution for an employed bee then original solution is abandoned and the concerned employed bee becomes scout, i.e., a new solution is generated randomly and the status of this scout bee is again changed to employed by associating it with that solution. This is done to eliminate one duplicate solution. If a collision occurs while generating neighboring solution for an onlooker bee then another solution is chosen randomly for selecting an interior node from it to insert in the original solution. This process is repeated till we find a solution that is different from the original solution. Here it is to be noted that for onlookers there is no point in generating a random solution in case of a collision because for survival this randomly generated solution has to compete with the original solution and with the solutions of other onlookers, which are also associated with the same original solution. Therefore, such randomly generated solutions seldom survive.

Cost evaluation: There is a more efficient way than the two-step procedure described in Section 1, to compute the cost of the LCST associated with a neighboring solution. This is so because the neighboring solution differs from the original solution on one node only. Suppose node v_i is removed from the set of interior nodes of the original solution and node v_j is inserted in its place to generate the neighboring solution. The total weight of the neighboring solution can be computed in a more efficient way by the following two steps. The first step constructs the MST on the new set of interior nodes in a manner exactly similar to the first step of the two-step procedure. In the second step, all those leaf nodes that were connected to an interior node other than v_i in the original solution are only checked for their edge cost with the new interior node v_j. If this cost is less than the cost of their edge with the interior node to which they are presently connected, then they are detached from the present interior node and reconnected to the tree through v_j. However, all remaining leaf nodes (leaf nodes that were connected to v_i in the original solution and v_i itself) have to be checked for least-cost connection with every interior node. A similar approach was used in [8] for the tabu search.

All randomly generated solutions have to be evaluated by the two-step procedure described in Section 1. There is no better way to evaluate their costs.

Other features: If the solution associated with an employed bee does not improve for a predetermined number of iterations then it becomes a scout. In our ABC algorithm there is a second possibility in which an employed bee can become scout. As described previously an employed bee can become scout through collision also. We have not put any limit on the number of scouts in a single iteration like other ABC algorithms. In our ABC algorithm number of scouts depends on the above two conditions. There can be many scouts in an iteration if these two conditions are satisfied many times, or there can be no scout if these two conditions remain unsatisfied.

4. Computational results

We have implemented ABC-LCMST in C and executed it on a Pentium 4 system with 512 MB RAM running at 3.0 GHz under Red Hat Linux 9.0. Following [13], we have used a colony of 100 bees. Half of these bees are employed and remaining half are onlookers. For a problem of size n, if the solution associated with an employed bee does not improve for $2n$ iterations then that solution is

Table 1
Results of SCGA, ACO-LCMST, TS-LCMST and ABC-LCMST on 15 Euclidean instances of size $n = 50$ and $\ell = 45$

Instance	SCGA			ACO-LCMST			TS-LCMST			ABC-LCMST		
	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.
1	10.111	10.116	0.01	10.111	10.111	0.00	10.111	10.111	0.00	10.111	10.111	0.00
2	9.652	9.652	0.00	9.652	9.652	0.00	9.652	9.652	0.00	9.652	9.652	0.00
3	9.243	9.252	0.03	9.243	9.243	0.00	9.243	9.243	0.00	9.243	9.243	0.00
4	8.510	8.514	0.02	8.510	8.510	0.00	8.510	8.510	0.00	8.510	8.510	0.00
5	9.161	9.161	0.00	9.161	9.161	0.00	9.161	9.161	0.00	9.161	9.161	0.00
6	9.032	9.032	0.00	9.032	9.032	0.00	9.032	9.032	0.00	9.032	9.032	0.00
7	8.480	8.568	0.07	8.480	8.499	0.03	8.480	8.567	0.06	8.480	8.480	0.00
8	9.128	9.132	0.00	9.128	9.131	0.00	9.128	9.132	0.00	9.128	9.128	0.00
9	8.578	8.587	0.01	8.578	8.582	0.02	8.578	8.605	0.04	8.578	8.578	0.00
10	8.554	8.570	0.01	8.554	8.566	0.01	8.554	8.554	0.00	8.554	8.554	0.00
11	9.236	9.279	0.09	9.236	9.254	0.03	9.236	9.297	0.11	9.236	9.236	0.00
12	8.807	8.807	0.00	8.807	8.807	0.00	8.807	8.807	0.00	8.807	8.807	0.00
13	9.398	9.411	0.05	9.398	9.398	0.00	9.398	9.398	0.00	9.398	9.398	0.00
14	9.577	9.591	0.02	9.577	9.598	0.01	9.577	9.586	0.01	9.577	9.577	0.00
15	8.440	8.440	0.00	8.440	8.440	0.00	8.440	8.440	0.00	8.440	8.440	0.00

replaced by some randomly generated solution. The limit of $2n$ iterations has been set empirically.

We have compared ABC-LCMST with ACO-LCMST and TS-LCMST proposed in [8] and SCGA proposed in [2]. Comparison with ACO-LCMST is especially important as both are swarm-based approaches. Therefore to allow a fair comparison with ACO-LCMST we have used a termination criterion for ABC-LCMST that is equivalent to the termination criterion used in ACO-LCMST in terms of number of fitness evaluations (number of solutions generated). ACO-LCMST is terminated when the best solution does not improve over 2000 iterations and it has been executed for at least $2000 + 20n$ iterations. In each iteration, ACO-LCMST generates $n/5$ solutions, whereas ABC-LCMST generates 100 solutions. Therefore, ABC-LCMST terminates when best solution does not improve over $4n$ iterations and it has been executed for at least $4n + n^2/25$ iterations. In terms of number of fitness evaluations this termination criterion is equivalent to that of ACO-LCMST, i.e., both ACO-LCMST and ABC-LCMST terminate when best solution does not improve over $400n$ fitness evaluations and they have been executed for at least $400n + 4n^2$ fitness evaluations. SCGA generates $20n^2$ solutions in total, whereas TS-LCMST terminates when best solution does not improve over 100 iterations. TS-LCMST generates $(n - \ell) \times \ell$ solutions in each iteration. In fact, both SCGA and TS-LCMST generates many more solutions than ACO-LCMST and ABC-LCMST.

To test ABC-LCMST, we have used the same 65 problem instances as used in [2,8]. These instances were first used in [2]. 45 of these instances are Euclidean, whereas remaining instances are randomly generated. The Euclidean instances were originally generated for the Euclidean Steiner Tree problem. These instances consist of random points in the unit square. These points can be considered as the nodes of a complete graph, whose edge-weights are the Euclidean distances between them. These instances are available from Beasley's OR-library (http://people.brunel.ac.uk/~mastjjb/jeb/info.html). The random instances were generated by Julstrom [2]. The edge-weights of these random instances are uniformly distributed in [0.01, 0.99]. There are 15 Euclidean instances for each of 50, 100 and 250 nodes, whereas there are 10 random instances for each of 100 and 300 nodes. Julstrom [2] set the leaf-constraint ℓ to 45, 90, 225 and 270 for problems of size $n = 50$, 100, 250 and 300, respectively. Like SCGA, ACO-LCMST and TS-LCMST, we have executed ABC-LCMST also 30 independent times on each instance.

Tables 1–3 show the results of ABC-LCMST, ACO-LCMST, TS-LCMST and SCGA on Euclidean instances, whereas Tables 4 and 5 do the same on random instances. For each instance and for each approach, these tables report the best solution found over 30 trials and mean and standard deviation of these solution values. Data for

Table 2
Results of SCGA, ACO-LCMST, TS-LCMST and ABC-LCMST on 15 Euclidean instances of size $n = 100$ and $\ell = 90$

Instance	SCGA			ACO-LCMST			TS-LCMST			ABC-LCMST		
	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.
1	12.590	12.626	0.04	12.590	12.614	0.02	12.590	12.590	0.00	12.590	12.590	0.00
2	12.374	12.398	0.02	12.374	12.401	0.02	12.374	12.374	0.00	12.374	12.374	0.00
3	12.762	12.774	0.02	12.762	12.772	0.04	12.762	12.762	0.00	12.762	12.762	0.00
4	12.899	12.911	0.01	12.899	12.912	0.01	12.899	12.899	0.00	12.899	12.899	0.00
5	13.299	13.326	0.06	13.299	13.314	0.04	13.299	12.338	0.05	13.299	13.299	0.00
6	12.753	12.772	0.03	12.752	12.761	0.02	12.752	12.767	0.04	12.752	12.752	0.00
7	13.305	13.345	0.04	13.305	13.383	0.03	13.305	13.332	0.03	13.305	13.306	0.00
8	12.964	13.039	0.10	12.964	13.000	0.05	12.964	12.974	0.05	12.964	12.964	0.00
9	13.202	13.238	0.03	13.202	13.237	0.04	13.202	13.202	0.00	13.202	13.202	0.00
10	13.123	13.136	0.02	13.123	13.142	0.02	13.123	13.123	0.00	13.123	13.123	0.00
11	13.578	13.583	0.01	13.578	13.583	0.01	13.578	13.578	0.00	13.578	13.578	0.00
12	13.174	13.213	0.03	13.174	13.196	0.02	13.174	13.204	0.03	13.174	13.174	0.00
13	12.752	12.796	0.06	12.752	12.799	0.07	12.752	12.822	0.08	12.752	12.752	0.00
14	13.280	13.317	0.03	13.279	13.295	0.02	13.279	13.284	0.01	13.279	13.279	0.00
15	12.216	12.226	0.01	12.216	12.221	0.01	12.216	12.220	0.01	12.216	12.216	0.00

Table 3
Results of SCGA, ACO-LCMST, TS-LCMST and ABC-LCMST on 15 Euclidean instances of size $n = 250$ and $\ell = 225$

Instance	SCGA			ACO-LCMST			TS-LCMST			ABC-LCMST		
	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.
1	19.921	19.994	0.08	19.921	19.959	0.04	19.921	19.976	0.09	19.921	19.921	0.00
2	20.279	20.342	0.06	20.279	20.316	0.03	20.279	20.302	0.03	20.279	20.281	0.00
3	19.841	19.897	0.05	19.826	19.876	0.04	19.826	19.859	0.03	19.826	19.840	0.01
4	20.455	20.540	0.09	20.455	20.469	0.02	20.454	20.472	0.02	20.454	20.456	0.00
5	19.957	20.021	0.08	19.959	19.995	0.04	19.957	19.983	0.04	19.957	19.958	0.00
6	20.341	20.376	0.03	20.341	20.364	0.02	20.341	20.368	0.03	20.341	20.342	0.00
7	20.025	20.059	0.03	20.025	20.049	0.07	20.025	20.044	0.02	20.025	20.025	0.00
8	19.594	19.640	0.03	19.594	19.636	0.04	19.594	19.630	0.03	19.594	19.594	0.00
9	20.461	20.510	0.05	20.447	20.470	0.01	20.447	20.514	0.05	20.447	20.449	0.00
10	19.888	19.994	0.04	19.885	19.980	0.05	19.885	19.971	0.05	19.885	19.900	0.02
11	19.044	19.086	0.03	19.044	19.072	0.02	19.044	19.082	0.03	19.044	19.052	0.00
12	19.844	19.890	0.04	19.844	19.880	0.03	19.844	19.880	0.04	19.844	19.844	0.00
13	19.777	19.854	0.05	19.773	19.825	0.04	19.773	19.829	0.04	19.773	19.778	0.01
14	19.938	20.036	0.07	19.938	19.968	0.04	19.938	20.001	0.07	19.938	19.938	0.00
15	19.494	19.532	0.03	19.494	19.502	0.01	19.494	19.510	0.01	19.494	19.494	0.00

Table 4
Results of SCGA, ACO-LCMST, TS-LCMST and ABC-LCMST on 10 random instances of size $n = 100$ and $\ell = 90$

Instance	SCGA			ACO-LCMST			TS-LCMST			ABC-LCMST		
	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.
1	5.594	5.843	0.14	5.594	5.681	0.08	5.594	5.724	0.12	5.594	5.598	0.02
2	5.574	5.696	0.09	5.574	5.632	0.04	5.574	5.574	0.00	5.574	5.575	0.01
3	5.923	5.990	0.08	5.923	5.951	0.01	5.923	5.946	0.04	5.923	5.931	0.01
4	5.389	5.575	0.17	5.389	5.574	0.14	5.389	5.569	0.14	5.389	5.417	0.06
5	5.904	5.958	0.09	5.904	5.904	0.00	5.904	5.946	0.16	5.904	5.904	0.00
6	6.292	6.414	0.09	6.293	6.352	0.02	6.292	6.355	0.05	6.292	6.296	0.01
7	5.558	5.761	0.11	5.558	5.735	0.07	5.558	5.670	0.14	5.558	5.583	0.05
8	5.423	5.509	0.13	5.423	5.458	0.01	5.423	5.529	0.16	5.423	5.435	0.02
9	5.138	5.208	0.06	5.138	5.169	0.03	5.138	5.201	0.09	5.138	5.138	0.00
10	5.709	5.949	0.12	5.709	5.812	0.12	5.709	5.915	0.12	5.709	5.719	0.04

Table 5
Results of SCGA, ACO-LCMST, TS-LCMST and ABC-LCMST on 10 random instances of size $n = 300$ and $\ell = 270$

Instance	SCGA			ACO-LCMST			TS-LCMST			ABC-LCMST		
	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.	Best	Mean	S.D.
1	7.835	7.938	0.06	7.835	7.882	0.05	7.835	7.928	0.07	7.828	7.842	0.02
2	8.091	8.223	0.07	8.030	8.149	0.04	8.098	8.229	0.08	8.030	8.080	0.02
3	7.893	8.047	0.08	7.903	7.979	0.05	7.857	8.046	0.12	7.857	7.891	0.02
4	8.061	8.185	0.06	8.052	8.119	0.04	8.052	8.167	0.12	8.052	8.054	0.00
5	7.818	7.992	0.10	7.818	7.896	0.05	7.794	7.952	0.11	7.794	7.838	0.03
6	8.092	8.186	0.06	8.068	8.155	0.04	8.068	8.165	0.08	8.041	8.089	0.02
7	7.974	8.111	0.09	7.946	8.023	0.04	7.918	8.088	0.11	7.918	7.956	0.03
8	7.972	8.122	0.07	7.985	8.067	0.04	7.986	8.130	0.09	7.972	8.001	0.02
9	7.752	7.883	0.09	7.714	7.789	0.04	7.714	7.908	0.08	7.714	7.728	0.02
10	7.684	7.854	0.06	7.695	7.798	0.05	7.684	7.812	0.07	7.684	7.706	0.02

ACO-LCMST and TS-LCMST are taken from [8], whereas data for SCGA are taken from [2]. Table 6 compares the performance of ABC-LCMST with ACO-LCMST, TS-LCMST and SCGA in terms of best and average solution quality. Table 6 clearly establishes the superiority of ABC-LCMST over all the other approaches in terms of solution quality. The best and average solutions obtained by ABC-LCMST are always as good as or better than those obtained with all the other approaches except in one case, where average solution obtained by TS-LCMST is better. Moreover, Tables 1–5 show that standard deviations of solution values obtained by ABC-LCMST are also the smallest among all the approaches. This shows its robustness.

Table 7 reports average time to reach the best solution and average execution time for ABC-LCMST, ACO-LCMST and TS-LCMST on Euclidean instances, whereas Table 8 does the same for random instances. These tables also report the maximum and average number of fitness evaluations required by ABC-LCMST and ACO-LCMST to reach the best solution. Data for ACO-LCMST and TS-LCMST are taken from [8]. No such timing or fitness evaluation information is provided in [2] for SCGA. ABC-LCMST and TS-LCMST are much faster than ACO-LCMST. This is due to the fact that all the new solutions except initial solution in TS-LCMST and almost all the new solutions (except randomly generated solutions) in ABC-LCMST differ at just one

Table 6
Comparison of ABC-LCMST with SCGA, ACO-LCMST and TS-LCMST in terms of best and mean solution quality on 45 Euclidean instances and 20 random instances.

	ABC-LCMST																	
	Euclidean instances (45)						Random instances (20)						Overall (65)					
	Best cost			Mean cost			Best cost			Mean cost			Best cost			Mean cost		
	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>
SCGA	0	38	7	0	5	40	0	12	8	0	0	20	0	50	15	0	5	60
ACO-LCMST	0	43	2	0	9	36	0	12	8	0	0	20	0	55	10	0	9	56
TS-LCMST	0	45	0	0	17	28	0	16	4	1	0	19	0	61	4	1	17	47

interior node from an already existing solution. Therefore, the cost of constructing and evaluating such a new solution is much less than those of a new solution in ACO-LCMST, where every new solution has to be constructed from scratch and evaluated using the two-step procedure. On Euclidean instances ACO-LCMST requires more fitness evaluations than ABC-LCMST, whereas this situation is reversed in case of random instances. Since ABC-LCMST consistently performs better, therefore, this can be attributed to premature convergence of ACO-LCMST on random instances.

Table 7
Average time to reach the best solution and average execution time of ACO-LCMST, TS-LCMST and ABC-LCMST on 45 Euclidean instances. Table also shows maximum and average number of fitness evaluations required to reach the best solution for the ACO-LCMST and ABC-LCMST.

n	Instances	ACO-LCMST				TS-LCMST		ABC-LCMST			
		Fitness evaluations		Time (s)		Time (s)		Fitness evaluations		Time (s)	
		Maximum	Average	Best	Total	Best	Total	Maximum	Average	Best	Total
50	1	12,282	6,950	0.05	0.26	0.00	0.04	8,422	3,361	0.01	0.08
50	2	14,928	8,696	0.06	0.26	0.00	0.04	6,245	3,109	0.01	0.08
50	3	18,917	9,658	0.06	0.25	0.00	0.04	6,358	2,909	0.01	0.08
50	4	11,140	6,769	0.04	0.26	0.00	0.04	8,792	3,104	0.01	0.08
50	5	13,213	8,040	0.05	0.23	0.00	0.04	6,437	2,975	0.01	0.08
50	6	16,309	9,641	0.06	0.25	0.00	0.04	5,795	2,937	0.01	0.08
50	7	15,623	8,196	0.05	0.25	0.00	0.04	11,432	4,439	0.01	0.08
50	8	21,644	8,971	0.06	0.26	0.00	0.04	17,260	3,812	0.01	0.08
50	9	13,247	8,453	0.06	0.25	0.00	0.04	5,462	2,758	0.01	0.08
50	10	30,768	10,118	0.07	0.26	0.00	0.04	15,560	3,943	0.01	0.08
50	11	19,479	8,554	0.06	0.25	0.00	0.04	8,530	3,973	0.01	0.08
50	12	21,855	10,245	0.07	0.28	0.01	0.04	5,611	2,980	0.01	0.08
50	13	20,935	11,014	0.07	0.24	0.00	0.04	8,526	3,863	0.01	0.08
50	14	14,388	7,363	0.05	0.24	0.00	0.04	14,887	4,373	0.01	0.08
50	15	18,298	10,068	0.07	0.26	0.00	0.04	7,131	2,852	0.01	0.08
100	1	64,660	43,572	1.26	3.07	0.21	0.63	62,960	23,655	0.16	0.57
100	2	73,891	58,509	1.77	3.42	0.05	0.47	42,561	19,021	0.13	0.56
100	3	79,193	55,596	1.63	3.32	0.06	0.48	42,374	17,299	0.12	0.56
100	4	70,010	52,215	1.57	3.37	0.11	0.54	55,681	27,704	0.20	0.58
100	5	98,572	50,353	1.54	3.24	0.06	0.48	62,422	24,672	0.17	0.58
100	6	78,174	56,890	1.70	3.39	0.09	0.52	54,362	22,294	0.15	0.57
100	7	61,949	45,881	1.37	3.16	0.06	0.48	88,573	28,657	0.20	0.59
100	8	100,077	58,983	1.85	3.56	0.11	0.53	44,728	21,338	0.15	0.56
100	9	67,122	45,691	1.32	3.04	0.15	0.58	55,372	26,342	0.18	0.57
100	10	63,985	43,252	1.22	2.90	0.10	0.52	56,405	23,567	0.17	0.57
100	11	75,190	50,252	1.53	3.30	0.06	0.48	57,394	25,168	0.18	0.58
100	12	83,513	52,994	1.61	3.33	0.06	0.48	40,130	21,262	0.15	0.56
100	13	70,050	49,523	1.45	3.10	0.05	0.48	61,798	21,007	0.15	0.57
100	14	70,550	50,570	1.58	3.36	0.11	0.53	58,784	22,506	0.16	0.57
100	15	70,251	50,398	1.52	3.22	0.07	0.49	52,936	21,268	0.15	0.57
250	1	638,837	514,037	109.41	139.95	5.81	22.16	316,788	164,652	5.43	11.70
250	2	635,512	457,588	102.39	133.10	7.96	24.62	322,469	193,942	6.47	11.94
250	3	763,628	515,024	108.09	138.16	11.08	27.54	340,877	232,031	7.73	12.25
250	4	552,125	443,405	97.08	127.59	8.60	24.99	484,439	242,741	7.94	12.64
250	5	524,918	425,117	90.01	119.81	7.50	24.27	337,062	181,361	5.96	11.80
250	6	649,309	485,753	106.24	136.13	9.03	25.62	421,918	266,922	8.73	12.87
250	7	610,854	490,428	103.27	133.10	5.96	22.37	316,679	201,150	6.66	11.86
250	8	682,867	528,962	110.05	139.90	6.38	22.60	290,282	190,380	6.25	11.62
250	9	692,341	462,919	100.73	130.49	6.37	22.79	389,490	216,266	7.47	12.64
250	10	497,090	431,217	92.05	121.87	10.49	26.92	441,768	266,188	9.01	13.32
250	11	699,693	560,621	115.16	144.76	6.77	23.20	431,431	220,247	7.51	12.58
250	12	562,824	430,443	91.43	121.30	5.68	22.01	350,276	185,244	6.19	12.01
250	13	741,087	515,675	111.54	141.82	9.54	25.78	367,011	206,413	6.85	12.18
250	14	621,732	501,645	106.17	135.55	7.97	24.31	345,245	205,215	6.73	11.80
250	15	701,567	543,268	114.71	144.27	7.21	23.54	363,700	171,574	5.67	11.69

Table 8
Average time to reach the best solution and average execution time of ACO-LCMST, TS-LCMST and ABC-LCMST on 20 random instances. Table also shows maximum and average number of fitness evaluations required to reach the best solution for the ACO-LCMST and ABC-LCMST.

n	Instances	ACO-LCMST				TS-LCMST		ABC-LCMST			
		Fitness evaluations		Time (s)		Time (s)		Fitness evaluations		Time (s)	
		Maximum	Average	Best	Total	Best	Total	Maximum	Average	Best	Total
100	1	44,761	24,105	0.76	3.27	0.08	0.51	109,172	30,070	0.22	0.62
100	2	35,548	16,721	0.47	3.18	0.22	0.65	63,497	28,122	0.21	0.61
100	3	42,704	18,391	0.51	3.07	0.10	0.53	66,598	22,631	0.16	0.60
100	4	38,455	21,061	0.61	3.16	0.12	0.55	61,524	16,015	0.12	0.59
100	5	25,212	15,376	0.41	3.21	0.08	0.51	21,407	10,502	0.08	0.59
100	6	37,874	19,618	0.55	3.11	0.06	0.49	104,385	37,063	0.27	0.65
100	7	41,410	24,026	0.72	3.12	0.07	0.50	75,027	25,238	0.18	0.60
100	8	33,324	15,529	0.42	3.09	0.07	0.50	67,384	28,676	0.21	0.62
100	9	15,460	12,549	0.33	3.25	0.06	0.49	58,597	12,920	0.09	0.59
100	10	42,967	22,489	0.66	3.15	0.07	0.50	72,813	26,617	0.19	0.61
300	1	315,241	182,170	50.66	178.07	18.51	53.91	454,594	273,354	13.32	23.94
300	2	431,270	283,834	88.52	179.50	17.93	53.40	676,480	414,069	20.41	27.50
300	3	576,632	274,553	84.55	178.05	18.99	54.45	644,331	376,949	18.60	26.26
300	4	328,868	214,275	65.91	182.44	18.80	53.85	457,398	296,571	14.78	25.25
300	5	413,370	249,156	78.39	181.40	18.39	53.41	639,208	358,762	17.38	25.49
300	6	462,654	270,781	82.57	177.45	15.98	51.57	701,823	332,493	16.30	25.14
300	7	486,168	248,448	76.84	180.92	17.04	52.42	530,404	341,212	17.18	25.56
300	8	417,268	249,170	76.94	181.29	15.00	50.37	707,478	385,721	18.79	26.13
300	9	480,267	227,015	67.63	178.67	15.23	50.43	481,555	242,397	11.86	23.93
300	10	446,709	287,399	91.28	178.87	16.47	51.74	665,520	381,278	19.03	26.78

5. Conclusions

In this paper we have designed and implemented an artificial bee colony algorithm ABC-LCMST for the LCMST problem. We have compared our approach against three other metaheuristic approaches—a genetic algorithm, an ant-colony optimization approach and a tabu search approach. Our approach outperformed all the other approaches. Best as well as average solution qualities of ABC-LCMST are always as good as or better than those obtained with other approaches except in one case, where average solution quality of tabu search is better. Computational times for ABC-LCMST are also quite small. In particular, ABC-LCMST completely outperforms ACO-LCMST both in terms of solution quality as well as running time. This is especially important as both approaches are based on swarm intelligence.

To our knowledge this is the first application of artificial bee colony algorithm for a discrete optimization problem. This paper demonstrates the capability of artificial bee colony algorithm in solving a discrete optimization problem. Ideas presented in this paper can be applied to many other problems also. The concept of collisions introduced here keeps a check on the number of duplicate solutions in the population. Like other population-based approaches, the presence of duplicate solutions can impair the performance of ABC algorithm also. The procedure that we have developed for determining the neighboring solutions can be adapted to other subset selection problems such as 0/1 knapsack problem, quadratic knapsack problem, minimum k-cardinality tree problem, minimum vertex cover problem, etc.

Acknowledgements

We are thankful to Prof. Bryant A. Julstrom for making available his random data set for the LCMST problem. We are also grateful to two anonymous reviewers for their valuable comments and suggestions, which helped in improving the presentation of this paper.

References

[1] N. Deo, P. Micikevicius, A heuristic for the leaf-constrained minimum spanning tree problem, Congressus Numerantium 141 (1999) 61–72.
[2] B.A. Julstrom, Codings and operators in two genetic algorithms for the leaf-constrained minimum spanning tree problem, International Journal of Applied Mathematics and Computer Science 14 (2004) 385–396.
[3] C.J. Hoelting, D.A. Schoenefeld, R.L. Wainwright, Approximation techniques for variations of the p-median problem, in: Proceedings of the 1995 ACM Symposium on Applied Computing, pp. 293–299, ACM Press, 1995.
[4] B.A. Julstrom, Better greedy heuristics for the leaf-constrained minimum spanning tree problem in complete graphs, Communicated to Discrete Applied Mathematics (2004).
[5] W. Edelson, M.L. Gargano, Leaf-constrained minimal spanning trees solved by a genetic algorithm, Congressus Numerantium 157 (2002) 41–48.
[6] R. Even, Algorithmic Combinatorics, Macmillan, New York, USA, 1973.
[7] S. Picciotto. How to encode a tree, Ph.D. Thesis, University of California, San Diego, USA, 1999.
[8] A. Singh, A.S. Baghel, New metaheuristic approaches for the leaf-constrained minimum spanning tree problem, Asia-Pacific Journal of Operational Research 25 (2008) 575–589.
[9] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[10] B. Basturk, D. Karaboga, An artificial bee colony (ABC) algorithm for numeric function optimization, in: Proceedings of the IEEE Swarm Intelligence Symposium, Indianapolis, IN, USA, May 12–14, 2006.
[11] 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.
[12] D. Karaboga, B. Basturk, Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems, Lecture Notes in Artificial Intelligence 4529 (2007) 789–798, Springer-Verlag, Berlin.
[13] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.