Soft Computing (2021) 25:11289–11305
https://doi.org/10.1007/s00500-021-05913-z

APPLICATION OF SOFT COMPUTING

# Artificial bee colony algorithm using permutation encoding for the bounded diameter minimum spanning tree problem

**Kavita Singh**[1] · **Shyam Sundar**[1]

Accepted: 25 May 2021 / Published online: 8 July 2021


**Abstract**
The bounded diameter minimum spanning tree (BD-MST) problem seeks a spanning tree ($T$) of minimum weight on a given connected, undirected and edge-weighted graph subject to the diameter of $T$ does not exceed $D \geq 2$, where $D$ is a given positive integer. The BD-MST problem is $\mathcal{NP}$-hard problem and finds many real-world applications. In this paper, we propose an artificial bee colony (ABC) algorithm for the BD-MST problem. ABC algorithm is a swarm-based metaheuristic technique based on the intelligent foraging behavior of honeybees. The proposed ABC algorithm employs permutation encoding. To exploit this encoding structure, two neighborhood strategies that help ABC algorithm in faster convergence towards finding high quality solutions are applied. On a set of Euclidean and non-Euclidean benchmark instances for various diameter bounds, the proposed approach has been compared with state-of-the-art approaches. Computational results demonstrate the effectiveness of the proposed approach to the other extant approaches in the literature.

**Keywords** Spanning tree · Bounded-diameter · Swarm intelligence · Artificial bee colony algorithm · Permutation encoding · Neighborhood strategies

## 1 Introduction

Given a connected, undirected and edge-weighted graph $G(V, E, w)$, where $V$ represents a set of vertices of $G$; $E$ represents a set of edges of $G$; and each edge $\in E$ associates with a weight $w$, the bounded diameter minimum spanning tree (BD-MST) problem consists of finding a spanning tree ($T$) of minimum weight subject to the diameter of $T$ (the maximum number of edges connecting any two vertices in $T$) does not exceed $D \geq 2$, where $D$ denotes a given positive integer.

The BD-MST problem is proven to be $\mathcal{NP}$-hard for $4 \leq D < |V|$-1 (Garey and Johnson 1979) and finds many real-world applications. For example, linear light-wave networks (LLNs) involve multi-cast calls sent by each source to multiple destinations. To minimize interference in such a network, it is preferable to use a short (in terms of diameter) spanning tree for each transmission (Deo and Abdalla 2000). Bala et al. (1993) presented an algorithm to decompose a LLN into a set of edge disjoint trees having at least one spanning tree. However, this algorithm constructs trees with smaller diameter by finding trees whose maximum node-degree has lesser value in comparison to a parameter value instead of optimizing the diameter of trees directly. Lines of the network were assumed to be same. If such lines have different bandwidths, then spanning trees should have lines of higher bandwidth as such lines are frequently used and have heavy traffic. If an algorithm that can solve BD-MST problem would be more helpful in determining an appropriate tree decomposition for such network. For that, an edge-weighted tree would be appropriate to model the LLN, where the weight of an edge would be set as 1/z. z represents the line of bandwidth. Another application of the BD-MST problem finds in the distributed mutual exclusion, where some concurrent processes try to access a shared resource with the help of executing a section of code called *critical section* (CS). Wang and Lang (1994) proposed a token-based distributed algorithm that uses logical spanning tree built on a network of processors taken from Raymond's tree-based method (Raymond 1989). In this algorithm, $k$ entries are permitted in the critical section, where $k$ is the number of processes. The algo-

✉ Shyam Sundar
ssundar.mca@nitrr.ac.in

Kavita Singh
ksingh.phd2015.mca@nitrr.ac.in

[1] Department of Computer Applications, National Institute of Technology Raipur, Raipur 492010, India

rithm needs at most $2kD$ messages for a vertex to enter the critical section, where $D$ denotes the diameter of the tree. Therefore, a small diameter plays a significant role for the efficiency of the algorithm. Also, minimizing the weight of a spanning tree leads to the reduction of the cost of the network. The BD-MST problem also finds an application in the information retrieval, where large files are compressed through using large data structures called bitmaps. Compressing these files help in occupying less memory space, while allowing reasonably fast access. Bookstein and Klein (1996) proposed a method that uses a spanning tree of a given complete edge-weighted graph to cluster and compress vectors along the paths from a selected node (say root of the tree), to all the leaves of the spanning tree, where such graph is generated on the bitmaps. In such graph, vertices represent bitmaps, and the weighted edges represent Hamming distances. Therefore, a spanning tree with a small diameter can provide high-speed retrieval. In addition, Anderson et al. (2014) used the bounded diameter minimum spanning tree as a method for selecting concatenation of pairwise registrations for finding good results on the joint image alignment problem. An MRF-based optical flow refinement technique and a method for mesh identification were demonstrated to improve the results in terms of active appearance model compactness, suitable for video-realistic synthesis.

The BD-MST problem is extensively studied. The literature has witnessed a number of exact methods, problem-specific heuristic approaches and metaheuristic techniques. Exact methods (Achuthan et al. 1994; Gouveia and Magnanti 2003; Gouveia et al. 2011; Gruber and Raidl 2008) for this problem can address only relatively small size of problem instances upto $V = 100$ due to its computational complexity. Besides these exact methods, literature has also witnessed a number of problem-specific heuristic and metaheuristic techniques for this problem. Among problem-specific heuristic approaches, Abdalla et al. (2000) proposed five heuristics (first general-iterative-refinement algorithm (IR1), second general-iterative-refinement algorithm (IR2), a composite iterative-refinement algorithm (CIR), one time tree construction (OTTC), and the special-case approach for $k$=4). For experimental purpose, authors used graphs of different orders ($50 \leq V \leq 3000$) and densities and found that among all four general-algorithms only OTTC was always successful in finding an approximate solution in complete graphs regardless of the value of the diameter bound $k$; however, it was found that the success rate of OTTC (using one source node) reduced quickly with the reduction of the the density of the graphs, particularly with small values of the diameter bound, such as $k = 5$. OTTC is a greedy-based heuristic and follows the idea of Prim's algorithm (Prim 1957) and at each iteration, in order to add an unselected vertex to the tree, OTTC selects the closest (weight) vertex whose addition into the partial tree respects the diameter constraint. The remaining approaches can be read in details in (Abdalla et al. 2000). Later, randomized greedy heuristic (RGH) (Raidl and Julstrom 2003)—a modified version of OTTC—was presented. RGH selects a single vertex as a tree center if $D$ is even, otherwise RGH selects two adjacent vertices as two tree centers ($D$ is odd). Then, at each step, instead of selecting the closest (weight) unselected vertex whose addition into the tree respects the diameter constraint, RGH selects an unselected vertex randomly, and then, it joins this vertex to the partial tree through the minimum-weight edge that respects the diameter constraint. Later, Julstrom (2009) proposed center-based tree construction (CBTC) problem-specific heuristic—an improved version of RGH. Like RGH, it is also based on the idea of tree center. In CBTC, at each iteration, in order to add an unselected vertex to the partial tree, it selects the minimum-weight edge connecting an unselected vertex to one of vertices in the partial tree, and addition of this edge into the tree respects the diameter constraint. In terms of performance, RGH is superior to CBTC on Euclidean instances, whereas CBTC is superior to RGH on non-Euclidean instances. Singh and Gupta (2007) further extended the idea of RGH (Raidl and Julstrom 2003) and CBTC (Julstrom 2009) and presented its improved versions called RGH-I and CBTC-I, respectively. According to this improvement strategy, a search is performed for each vertex $v$ (except center vertex/vertices and its immediate child vertices) whether the vertex $v$ can be connected to a vertex whose depth is less than the depth of $v$ with a smaller edge-weight. If the search is successful, the subtree rooted at vertex $v$ is disconnected from its current parent and connected to this newly selected vertex. Later, Saxena and Singh (2009) proposed another improved versions of RGH (Raidl and Julstrom 2003) and CBTC Julstrom (2009) called RGH+HT and CBTC+HT, respectively. As per this improvement greedy heuristic, it only allows a sub-tree rooted at $v$ to be joined to any vertex of the spanning tree regardless of its depth, iff, the cost of the resultant tree is further minimized subject to respecting the diameter constraint of the resultant tree. Authors improved the results of RGH-I and CBTC-I on Euclidean as well as non-Euclidean instances. Gruber and Raidl (2009) proposed a hierarchical clustering-based methods for the BD-MST problem. The clustering heuristic builds diameter constrained trees within three steps: finding a hierarchical clustering, minimizing the height of this clustering according to the diameter bound, and finally deriving a BD-MST from this height-restricted clustering. Various techniques are used within the individual phases, e.g. a GRASP-like strategy to refine cutting positions through the dendrogram, or dynamic programming to assign each cluster a good root node. Authors (Gruber and Raidl 2009) reported two variants of clustering heuristics, i.e. $Cd^A$ and $Cd^B$. Clustering heuristics have been tested on 15 Euclidean instances of V = 1000 for various diameter bounds and

obtains in general high solution quality in few seconds which is better than CBTC and RGH and is even competitive with ACO approach (Gruber et al. 2006). Steitz (2015) proposed two new problem-specific heuristic approaches (called node selection tree construction (NSTC) and savings tree construction (STC)). Patvardhan et al. (2014) presented parallel versions of CBTC (Julstrom 2009), RGH (Raidl and Julstrom 2003) and Center-based Least Sum-of-costs (CBLSoC) (Patvardhan and Prakash 2009) problem-specific heuristic approaches. Patvardhan et al. (2015) presented some fast and effective problem-specific heuristic approaches for the BD-MST problem and tested their approach on Euclidean instances upto $V = 10000$. Authors of (Steitz 2015) and (Patvardhan et al. 2015) compared their problem-specific heuristics with the existing problem-specific heuristics on the Euclidean instances only.

In addition, the literature has also witnessed a number of metaheuristic approaches for the BD-MST problem. Raidl and Julstrom (2003) proposed an evolutionary algorithm based on edge-set encoding (RJ-ESEA). RJ-ESEA uses a crossover based on RGH heuristic and four different mutations. Later, Julstrom (2003) presented an evolutionary algorithm using permutation encoding and used RGH heuristic for decoding the encoded solution into a spanning tree. Authors (Julstrom 2003) showed that their proposed evolutionary algorithm based on permutation encoding performs better than RJ-ESEA (Raidl and Julstrom 2003) in term of solution-quality, but several times slower. Julstrom (2004) also proposed two generational evolutionary algorithms based on the permutation-encoding and random-key techniques. These two proposed approaches are comparable with an evolutionary algorithm using permutation encoding (Julstrom 2003). Singh and Gupta (2007) proposed a permutation-coded evolutionary algorithm (PEA-I). In PEA-I, RGH-I heuristic is used to decode the encoded solution (the order of vertices) into a bounded-diameter spanning tree. PEA-I, in comparison to approaches proposed in (Raidl and Julstrom 2003; Julstrom 2003, 2004), takes a much shorter time to obtain a better solution quality on each considered problem instance. Binh et al. (2008) proposed a hybrid genetic algorithm (HGA) that uses the idea of multiple populations. For multiple populations generation, authors used different well-known heuristics. The individuals from each population are then migrated to a final population called *GA_final* for merging and competing to breed. HGA is better than RJ-ESEA (Raidl and Julstrom 2003) and PEA-I (Singh and Gupta 2007) on Euclidean instances overall, but is marginally better than non-Euclidean instances in terms of solution quality. Authors (Binh et al. 2008) used the same termination criterion as used for RJ-ESEA and PEA-I, but did not discuss or report the computational time for obtaining solution quality of each Euclidean and non-Euclidean instances. Since *GA_final* in HGA uses the same individual representation and operators as that of RJ-ESEA (Raidl and Julstrom 2003), therefore, one can infer that HGA obtains overall better results than that of PEA-I (Singh and Gupta 2007) at the cost of much higher computational time. Gruber and Raidl (2005) proposed a variable neighborhood search (VNS) approach with four neighborhood strategies (i.e. node-swap neighborhood, level-change neighborhood, arc-exchange neighborhood and center change-level neighborhood) for the BD-MST problem. These neighborhood strategies have been used as local search methods in VNS approach. Later, Gruber et al. (2006) proposed an evolutionary algorithm (EA) with a new solution encoding scheme called level encoding and an ant colony optimization (ACO) algorithm. They also used node-swap neighborhood and arc-exchange neighborhood as local search methods in EA. In ACO, they used node-swap neighborhood and arc-exchange neighborhood as local search methods. In comparison to VNS approach, EA and ACO find better solution quality on almost all instances. Lucena et al. (2010) presented a hybrid semi-greedy heuristic (LRS) which combines the ideas of greedy randomized adaptive search procedures (GRASP) and iterated local search (ILS). Later, Torkestani (2012) presented a learning automata-based distributed algorithm (LABD) for this problem. LABD consists of several stages. At each stage, the constructed bounded-diameter spanning tree is rewarded, if the cost of the tree is the minimum cost seen so far, otherwise, the constructed tree is penalized. Author (Torkestani 2012) demonstrates the performance of LABD by comparing the results of LABD with the results reported in (Gruber and Raidl 2005), (Lucena et al. 2010). Vuppuluri and Patvardhan (2021) presented a serial memetic algorithm (2PMA) for BD-MST problem. 2PMA is a two-phase memetic algorithm that uses two recombination operators based on order recombination operator and neighborhood-based recombination operator in order to direct the exploration of the search space into regions containing better solutions. Also, authors presented a parallel memetic algorithm for BD-MST problem. In addition, many of the heuristics for the BD-MST problem have been also recast for solving the related bi-objective BD-MST problem (Saha et al. 2010; Prakash et al. 2018; Saha and Kumar 2011; Prakash et al. 2020).

Table 1 summarizes the work done in the literature—exact methods, problem-specific heuristic approaches, metaheuristic techniques and parallel implementations—for the BD-MST problem.

In this paper, we present an artificial bee colony (ABC) algorithm for the BD-MST problem. Artificial bee colony (ABC) algorithm is a swarm-based metaheuristic technique which emulates the intelligent foraging behavior of honey bees. ABC algorithm was introduced by Karaboga (2005) and has been successfully applied for various optimization problems (Karaboga et al. 2014; Pan et al. 2011;

**Table 1** Summary of the work done in the literature for the BD-MST problem

| Exact methods | Problem-specific heuristic approaches | Metaheuristic techniques | Parallel implementations |
|---|---|---|---|
| | Abdalla et al. (2000) | Raidl and Julstrom (2003) | |
| | Raidl and Julstrom (2003) | Julstrom (2003) | |
| | Julstrom (2009) | Julstrom (2004) | |
| Achuthan et al. (1994) | Singh and Gupta (2007) | Singh and Gupta (2007) | Vuppuluri and Patvardhan (2021) |
| Gouveia et al. (2011) | Saxena and Singh (2009) | Binh et al. (2008) | Patvardhan et al. (2014) |
| Gouveia and Magnanti (2003) | Gruber and Raidl (2009) | Gruber et al. (2006) | |
| Gruber and Raidl (2008) | Gruber et al. (2006) | Lucena et al. (2010) | |
| | Steitz (2015) | Torkestani (2012) | |
| | Patvardhan et al. (2015) | | |

Singh 2009; Sundar and Singh 2010). The proposed ABC algorithm employs permutation encoding. To exploit this encoding structure, two neighborhood strategies that help ABC algorithm in faster convergence towards finding high quality solutions are applied. On a set of Euclidean and non-Euclidean benchmark instances for various diameter bounds, the proposed approach has been compared with state-of-the-art approaches. Computational results demonstrate the effectiveness of the proposed approach to the other extant approaches in the literature in terms of solution quality and computational time.

The remaining of this paper is organized as follows: Sect. 2 discusses the proposed ABC algorithm for the BD-MST problem; Sect. 3 discusses about the computational results of ABC algorithm for the BD-MST problem; and finally Sect. 4 presents some concluding remarks.

## 2 ABC algorithm for the BD-MST

Inspired by the foraging behaviour of honey bees, Karaboga (Karaboga 2005; Karaboga et al. 2014) modeled ABC algorithm for the solutions of optimization problems. ABC algorithm also consists of three categories of artificial bees called employed bees, scout bees and onlooker bees which exhibit a variety of intelligent behaviours in their respective phases, i.e. *employed bee phase*, *scout bee phase* and *onlooker bee phase*. Each category of artificial bees performs stochastic behaviour in order to interact locally with one another, leading to the emergence of high quality solutions for an optimization problem. To readers, a general introduction on ABC algorithm and its applications can be found in (Karaboga and Basturk 2007; Karaboga et al. 2014; Singh 2009; Sundar and Singh 2010).

This subsection presents an ABC algorithm (PABC_BD) for the BD-MST. Hereafter, the proposed ABC algorithm for the BD-MST will be referred to as PABC_BD.

The description of each component of PABC_BD is as follows:

### 2.1 Solution representation

PABC_BD uses a permutation encoding to represent a solution ($E_i$), where $E_i$ is encoded as a linear permutation of all vertices of $V$. Such ordering of vertices in $E_i$ helps in decoding (see Sect. 2.3) $E_i$ into a feasible bounded-diameter spanning tree. In this encoding, if $D$ is even, then the first vertex in the ordering of vertices is considered as the center of $E_i$. If $D$ is odd, then the first two vertices in the ordering of vertices are considered as the centers of $E_i$. Note that such permutation encoding is also used in (Julstrom 2003, Julstrom 2004, Singh and Gupta 2007).

### 2.2 Solution initialization

Each initial solution $E_i$ which is encoded as a linear permutation of all vertices of $V$ is generated randomly in the following iterative process. Initially, $E_i$ represents an array of $|V|$ empty positions. Create a copy (say $S$) of $V$. A vertex $v_i$ is selected randomly from $S$. The selected vertex $v_i$ is first inserted into the first vacant position in $E_i$ and then is deleted from $S$. In a similar way, the next vertex $v_j$ that is selected randomly from $S$ is inserted into the next vacant position in the order of $|V|$ positions in $E_i$. After this, $v_j$ is deleted from $S$. This whole procedure is repeatedly called until $S$ becomes empty.

Once an initial solution $E_i$ is generated, its association becomes with a unique employed bee.

### 2.3 Solution decoding

We follow a slight modified version of randomized greedy heuristic (RGH) (Raidl and Julstrom 2003) as a decoder to decode a solution $E_i$ into a feasible bounded-diameter spanning tree. In this decoding, instead of picking a ver-

tex randomly, the decoder picks a vertex in the order from the linear arrangement of vertices in $E_i$. After decoding, a feasible bounded-diameter spanning tree for $E_i$ is obtained. Similar decoding is also applied in (Julstrom 2003).

### 2.4 Fitness of a solution

Once the solution $E_i$ is decoded as a bounded-diameter spanning tree, its fitness is computed as the sum of weights of all edges in $E_i$.

### 2.5 Selection of a solution

We use binary tournament selection method—probability based selection method—for PABC_BD. As per this selection method, two solutions which are different are selected randomly from the employed bee population. Between these two selected solutions, a best one (in terms of fitness) is selected with probability $P_{bt}$, otherwise, a worst one (in terms of fitness) is selected with probability 1-$P_{bt}$.

Algorithm 1 presents the pseudocode of binary tournament selection method—i.e. $BTS(E_1, E_2, \ldots, E_{NE})$ function—which returns the index of the selected solution in the current employed bee population.

**Algorithm 1:** The pseudocode of binary tournament selection method – i.e. $BTS(E_1, E_2, \ldots, E_{NE})$ function

**Input** : The current employed bee population $E_1, E_2, \ldots, E_{NE}$;
**Output**: The index ($s_i$) of the selected solution in the current employed bee population;

```
Two different solutions (say E_x and E_y) are selected uniformly at random from the current employed population; // x and y are, respectively, indices of E_x and E_y in the employed bee population
if u01 < P_bt then
    // u01 is a uniform variate
    if the fitness of E_x ≤ the fitness of E_y then
        s_i ← the index of E_x;
    else
        s_i ← the index of E_y;
else
    if the fitness of E_x > the fitness of E_y then
        s_i ← the index of E_x;
    else
        s_i ← the index of E_y;
Return s_i;
```

### 2.6 Determination of a neighboring solution

Since the problem structure of a solution in PABC_BD is a linear random permutation of vertices of $V$, therefore, to exploit this structure effectively so that PABC_BD converges faster towards the high solution quality, PABC_BD uses two different neighborhood strategies—*insertion on multiple positions* (*IMP*) and *swap on multiple positions* (*SMP*). Such neighborhood strategy is applied on the current solution $E_i$ in order to determine a neighboring solution ($E^{'}$). *IMP* follows the principle of utilization of solution components from another solution of employed bee population ( Sundar and Singh 2012). The principle is that if a vertex is positioned correctly in a high quality solution, then there is a high likelihood that this vertex would hold the same position or would be in the vicinity of this same position in many other high quality solutions. The role of *SMP* is to avoid the current solution from the local optima trap and to help in exploration of the search space. Note that both *IMP* and *SMP* are applied in a mutually exclusive way. With probability $P_{nbr}$, *IMP* is applied, otherwise *SMP* is applied. $P_{nbr}$ is a parameter to be decided experimentally. The description of each neighborhood strategy is as follows:

To apply *insertion on multiple positions* (*IMP*) neighborhood strategy on the current solution $E_i$ in order to determine a new permutation of vertices of $V$ (a new neighboring solution $E^{'}$), first a solution $E^r$, different from $E_i$, is picked from the employed bee population with help of binary tournament selection method (see Sect. 2.5). Also, *IMP* initializes $E^{'}$ with an empty solution. A number (say $(|V| \times P_{mpi})$ of different positions are picked randomly from $E^r$. Insert all vertices assigned to these picked positions of $E^r$ into the exact positions of $E^{'}$. Now, insert those vertices of $E_i$, which are not in the current $E^{'}$, one-by-one in the same order in which these remaining vertices seem to be in $E_i$. Note that $P_{mpi}$ is a parameter to be decided experimentally. Also note that while picking a solution $E^r$ with the help of binary tournament selection method, if it is found that $E_i$ and $E^r$ represent the same solution, i.e a collision happens (Singh 2009), then it shows that the employed bee population is facing the diversity issue. To handle this, if this situation happens while determining a new solution in the neighborhood of current solution in the employed bee phase, then the employed bee associated with this solution (food source) $E_i$ abandons $E_i$ to become a scout. This scout instantly becomes employed by associating it with a newly generated random solution (see Sect. 2.2). However, if this situation happens while determining a new neighboring solution for an onlooker bee in the onlooker bee phase, then this onlooker bee is associated with a dummy solution whose fitness value is assigned a value artificially higher than the fitness value of its associated employed bee solution so that it cannot be selected in the next iteration. Note that we have not generated a random solution for such onlooker bee. The reason behind this one is that this newly generated random solution has to strive to win against the actual solution as well as with the solutions of all those onlookers which are associated with this actual

solution. Hence, it is highly likely that such a newly generated random solution will be lost immediately. This is similar to the concept of "collision" coined in (Singh 2009).

To explain $IMP$, we take the help of an example that is depicted by a series of Fig. 1a–c. Figure 1a depicted in *white color* represents a solution $E_i$—a linear permutation of vertices, i.e. {1, 5, 7, 2, 4, 3, 6}. Figure 1b depicted in *light grey color* represents another solution $E^r$—a linear permutation of vertices, i.e. {4, 7, 1, 2, 3, 6, 5}—picked randomly from the employed bee population. One can notice that $E^r$ is different from $E_i$. Initially, $E^{'}$ is an empty solution. Two different positions are picked randomly from $E^r$. Insert all vertices assigned to these picked positions of $E^r$ into the exact positions of $E^{'}$. These selected positions of $E^r$ are represented by *light grey* color in $E^{'}$. Now, insert those vertices of $E_i$, which are not in the current $E^{'}$, one-by-one in the same order in which these remaining vertices seem to be in $E_i$. Finally, Fig. 1c depicts a new neighboring solution $E^{'}$—a linear permutation of vertices, i.e. {1, 7, 5, 2, 4, 6, 3}.

To apply *swap on multiple positions* (*SMP*) neighborhood strategy on the current solution $E_i$ in order to determine a new permutation of vertices of $V$ (a new neighboring solution $E^{'}$), first create a copy (say $E^{'}$) of $E_i$. Hereafter, a number (say $|V| \times P_{msw}$) of swap is performed by swapping two vertices associated with two different positions ($x$ and $y$) which are selected uniformly at random from $E^{'}$. Here, $P_{msw}$ is a parameter to be decided experimentally. It is to be noted that the distance between positions $x$ and $y$ must be greater than two.

To explain *SMP*, we take the help of an example that is depicted by a series of Fig. 2a–b. Figure 2a depicts a solution

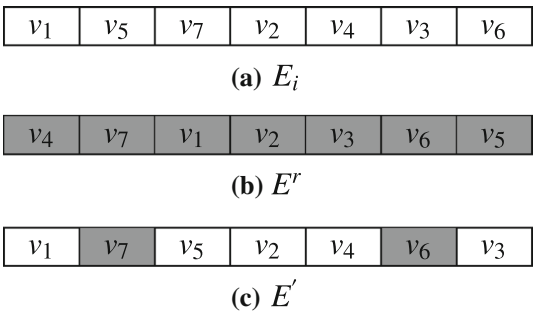


**Fig. 1** An illustration for $IMP$

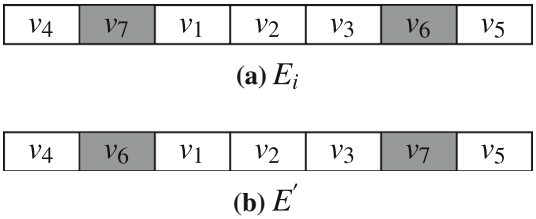


**Fig. 2** An illustration for $SMP$

$E_i$—a linear permutation of vertices, i.e. {4, 7, 1, 2, 3, 6, 5}. Figure 2b depicts a solution ($E^{'}$) which is a copy of $E_i$. Two positions in *light grey* color with distance greater than two are selected in $E^{'}$. Vertices assigned to these two positions are 7 and 6 that are swapped in $E^{'}$. After this swap, the resultant solution is depicted in Fig. 2b.

**Algorithm 2:** The pseudocode of PABC_BD

```
1  Generate a set of NE solutions, i.e., E_1, E_2, ..., E_NE;
2  Best ← Best solution in NE solutions;
3  while Termination criteria is not met do
4      for i ← 1 to NE do
5          E' ← DNbring_Sol(E_i);
6          if E' == Φ then
7              Replace E_i with a newly generated random
               solution;// See Section 2.2
8          else
9              if E' is better than E_i then
10                 E_i ← E';
11             else if E_i is not improving last limit iterations then
12                 Scout bee;
13         if E_i is better than Best then
14             Best ← E_i;
15     for i ← 1 to NO do
16         s_i ← BTS(E_1, E_2, ..., E_NE);
17         O_i ← DNbring_Sol(E_{s_i});
18         if O_i == Φ then
19             Assign O_i a fitness value artificially which is higher
               than the fitness value of E_{s_i};
20     for i ← 1 to NO do
21         if O_i is better than E_{s_i} then
22             E_{s_i} ← O_i;
23         if O_i is better than Best then
24             Best ← O_i;
```

## 2.7 Other features

If a solution (say $E_i$) of the employed bee population does not improve over a certain number of iterations (say *limit*), then it is assumed that $E_i$ is trapped into a local optima. To handle this situation, the employed bee becomes a scout bee by abandoning its associated $E_i$. This scout bee, then, generates a new random solution (see 2.2). This newly generated solution instantly replaces this abandoned solution, and the status of this scout bee becomes an employed bee on this newly generated random solution. *limit* is a parameter that is to be decided experimentally.

Algorithm 2 presents the pseudocode of PABC_BD, where binary tournament selection method is called by $BTS(E_1, E_2, \ldots, E_{NE})$ function which returns a selected solution (see Algorithm 1 in Sect. 2.5); and the method for

determining a solution in the neighborhood of a solution, say $X$, is called by $DNbring\_Sol(X)$ function which returns a neighboring solution, say $X'$. Note that $DNbring\_Sol(X)$ contains two different neighborhood strategies, i.e., *IMP* and *SMP* that are applied in a mutually exclusive way. If a collision occurs in *IMP*, then $DNbring\_Sol(X)$ will return $\Phi$ for $X'$.

## 3 Computational results

PABC_BD is implemented in C and executed on a Linux-based operating system with the configuration of Intel Core i5 processor 3.3 GHz × 4 with 4 GB RAM. Subsequent subsections discuss about benchmark instances (see Sect. 3.1); parameter setting (see Sect. 3.2); a comparison study of PABC_BD with state-of-the-art approaches such as PEA-I (Singh and Gupta 2007) and HGA (Binh et al. 2008) (see Sect. 3.3) and VNS Gruber et al. 2006, EA (Gruber et al. 2006), ACO (Gruber et al. 2006), LRS (Lucena et al. 2010) and LABD (Torkestani 2012) (see Sect. 3.4); and a statistical analysis is performed to find significant difference between PABC_BD and other approaches (PEA-I, VNS, EA, ACO, LRS and LABD) (see Sect. 3.5).

### 3.1 Benchmark instances

As in (Gruber et al. 2006, Lucena et al. 2010, Singh and Gupta 2007, Torkestani 2012), PABC_BD is also tested on the same benchmark instances. The description of the set of 45 benchmark instances that are classified into two different data-sets with varying sizes of $|V|$ from 50 to 1000 vertices are as follows:

- **Euclidean data-set:** This data-set consists of graphs whose sizes vary from 50 to 1000 vertices. There are 5 instances for each value of $V = \{50, 100, 250, 500, 1000\}$, leading to a total of 25 instances. One can use OR-library to download these instances. These instances are listed there as instances of the Euclidean Steiner tree problem. Euclidean instances consist of $V$ points randomly chosen in the unit square. The diameter bound ($D$) is set to {5, 10, 15, 20, 25} for problems of size $V$ = {50, 100, 250, 500, 1000} respectively.

- **Non-Euclidean data-set:** There are five instances for each value of $V$ = {100, 250, 500, 1000}, leading to a total of 20 non-Euclidean instances. All of these instances are complete graphs with weights randomly chosen from the range[0.01, 0.99]. The diameter bound $D$ is set to {10, 15, 20, 25} for problems of size $V$ = {100, 250, 500, 1000} respectively.

### 3.2 Parameter tuning

In all our experiments with PABC_BD, we have used $NE$, $NO$, $limit$, $P_{bt}$, $P_{nbr}$, $P_{mpi}$ and $P_{msw}$ parameters. For parameter setting, we consider three Euclidean instances $V$ = {250, 500, 1000} with the instance number {3, 4, 2} and the diameter-bound ($D$) {15, 20, 25} respectively.

Table 2 reports the results on various values of each parameter. In Table 2, the column denotes *Parameter* used in PABC_BD; the column *Value* denotes various potential values of a particular parameter; each next two columns report the best value (*Best*) and the average solution quality (*Avg*) obtained over 50 runs. The best values are highlighted in bold fonts. Note that graph instances are represented by notation $NV\_IN$, where $NV$ is the total number of vertices of considered graph and $IN$ is the instance number from the set of five instances of $V$ = {250, 500, 1000}. Also note that for parameter setting, each time one potential value of considered parameter is tested, whereas values of other parameters are kept fixed. Experimental observations based on the performance of PABC_BD on different combinations of various values of parameters that can be seen in Table 2 led to choose the best combination of value of each parameter, i.e. $NE$ = 50, $NO$ = 150, $limit$ = 50, $P_{bt}$ = 0.85, $P_{nbr}$ = 0.80, $P_{mpi}$ = 0.04, $P_{msw}$ = 0.05.

### 3.3 Comparison of PABC_BD, HGA and 2PMA with PEA-I

This subsection discusses about comparison of PABC_BD with PEA-I (Singh and Gupta 2007), HGA (Binh et al. 2008) and 2PMA (Vuppuluri and Patvardhan 2021) on a set of standard benchmark instances. Since authors of PEA-I, HGA and 2PMA used a termination criterion when the best-so-far solution obtained does not improve over 100,000 generations on a given instance, therefore, to allow a fair comparison of PABC_BD, HGA and 2PMA with PEA-I, we have also used the same termination criterion for PABC_BD, i.e. PABC_BD is allowed to execute till the best-so-far solution does not improve over 500 generations. As PEA-I which is a steady state genetic algorithm generates a single solution in each generation, hence, 1,00,000 generations generate 1,00,000 child solutions. In a similar way, PABC_BD generates ≈ 200 neighboring solutions due to mainly employed bee phase ($NE$ = 50) and onlooker phase ($NO$ = 150), hence 500 generations generate ≈ 1,00,000 neighboring solutions. As in PEA-I, HGA and 2PMA, PABC_BD is also given 50 runs for each problem instance.

Tables 3 and 4 report the results of 25 Euclidean instances and 20 non-Euclidean instances for PEA-I, HGA, 2PMA and PABC_BD. In Tables 3 and 4, first three columns $|V|$, $D$ and $N$ under *Instances* respectively denote the total number of vertices of each Euclidean/non-Euclidean instance, the

**Table 2** Influence of parameter values on a set of graph instances

| Parameter | Value | 250_3 | | 500_4 | | 1000_2 | |
|---|---|---|---|---|---|---|---|
| | | Best | Avg | Best | Avg | Best | Avg |
| *NE* | 25 | **11.98** | 12.07 | 16.92 | 17.02 | 23.63 | 23.79 |
| | **50** | **11.98** | **12.06** | **16.86** | **16.96** | **23.51** | **23.68** |
| | 100 | **11.98** | **12.06** | 16.89 | 16.97 | 23.53 | 23.76 |
| *NO* | 100 | 12.00 | 12.07 | **16.84** | 16.97 | 23.58 | 23.69 |
| | **150** | **11.98** | 12.06 | 16.86 | **16.96** | **23.51** | **23.68** |
| | 200 | **11.98** | **12.05** | 16.85 | **16.96** | 23.55 | 23.69 |
| $P_{bt}$ | 0.80 | 11.99 | **12.06** | 16.89 | **16.96** | **23.50** | 23.69 |
| | **0.85** | **11.98** | **12.06** | **16.86** | **16.96** | 23.51 | 23.68 |
| | 0.90 | 12.00 | **12.06** | 16.88 | 16.98 | 23.51 | **23.66** |
| $P_{nbr}$ | 0.70 | **11.98** | **12.06** | 16.87 | 16.96 | 23.52 | 23.68 |
| | **0.80** | **11.98** | **12.06** | **16.86** | **16.96** | **23.51** | **23.68** |
| | 0.90 | **11.98** | **12.06** | 16.90 | 16.97 | 23.52 | 23.69 |
| *limit* | 25 | 12.00 | 12.07 | **16.84** | 16.97 | 23.58 | 23.69 |
| | **50** | **11.98** | **12.06** | 16.86 | **16.96** | **23.51** | **23.68** |
| | 100 | 11.99 | 12.06 | 16.87 | 16.96 | 23.55 | 23.69 |
| $P_{mpi}$ | 0.03 | **11.97** | 12.10 | 16.89 | 16.98 | 23.52 | **23.65** |
| | **0.04** | 11.98 | **12.06** | 16.86 | **16.96** | **23.51** | 23.68 |
| | 0.05 | **11.97** | 12.07 | **16.85** | 16.97 | 23.55 | 23.70 |
| $P_{msw}$ | 0.04 | 11.99 | **12.06** | 16.89 | 16.97 | 23.52 | 23.69 |
| | **0.05** | **11.98** | **12.06** | **16.86** | **16.96** | **23.51** | **23.68** |
| | 0.06 | 11.99 | **12.06** | 16.87 | **16.96** | 23.52 | 23.69 |

diameter-bound and the instance number from each set of five instances; and for PEA-I, HGA, 2PMA and PABC_BD, each next four columns *Best*, *Avg*, *SD* and *ATET* respectively denote the best-so-far solution obtained in terms of fitness, average solution quality, standard deviation and average total execution time. Tables 3 and 4 highlight the best value (*Best*) and the best average solution quality (*Avg*) among PEA-I, HGA, 2PMA and PABC_BD in bold fonts. Comparing PABC_BD with PEA-I, one can observe the results of 25 Euclidean instances from Table 3 that PABC_BD, in terms of *Best*, is better on 14, is same on 9 and is worse on 2, and that PABC_BD, in terms of *Avg*, is better on 24 and is same on 1. In a similar way, comparing PABC_BD with PEA-I, one can observe the results of 20 non-Euclidean instances from Table 4 that PABC_BD, in terms of *Best*, is better on 13, is same on 6 and is worse on 1, and that PABC_BD, in terms of *Avg*, is better on 18, is same on 1 and is worse on 1. Comparing PABC_BD with HGA, one can observe the results of 25 Euclidean instances from Table 3 that PABC_BD, in terms of *Best*, is better on 11, is same on 2 and is worse on 12, and that PABC_BD, in terms of *Avg*, is better on 14, is same on 4 and is worse on 7. In a similar way, comparing PABC_BD with HGA, one can observe the results of 20 non-Euclidean instances from Table 4 that PABC_BD, in terms of *Best*, is better on 6, is same on 4 and is worse on 10, and that PABC_BD, in terms of *Avg*, is better on 18, is same on 1 and is worse on 1. Comparing PABC_BD with 2PMA, one can observe the results of 25 Euclidean instances from Table 3 that PABC_BD, in terms of *Best*, is better on 13, is same on 5 and is worse on 7, and that PABC_BD, in terms of *Avg*, is better on 20, is same on 3 and is worse on 2. In a similar way, comparing PABC_BD with 2PMA, one can observe the results of 20 non-Euclidean instances from Table 4 that PABC_BD, in terms of *Best*, is better on 6, is same on 2 and is worse on 12, and that PABC_BD, in terms of *Avg*, is better on 8, is same on 1 and is worse on 11. One can also observe from the overall results of PABC_BD and 2PMA on 45 instances (25 Euclidean instances and 20 non-Euclidean instances) in Tables 3 and 4 that PABC_BD is better than 2PMA in terms of robustness, as comparing with 2PMA, PABC_BD, in terms of *Avg*, is better on 28, is same on 4 and is worse on 13.

Since configuration of computer system (Pentium 4, 2.4 GHz with 512 MB RAM) used for PEA-I (Singh and Gupta 2007) is different from that of computer system (Intel Core i5, 3.3 GHz × 4 with 4 GB RAM) used for PABC_BD, therefore, to give a fair comparison based on computational time, a scaling factor mechanism is used according to the public CPU benchmark provided by PassMark Software ( https://www.cpubenchmark.net/singleThread.html). As per this website, we find that CPU Mark for the computer system used for PABC_BD is 2114, whereas CPU Mark for the computer

**Table 3** Results of **PEA-I** (Singh and Gupta 2007), **HGA** (Binh et al. 2008), **2PMA** (Vuppuluri and Patvardhan 2021) and **PABC_BD** on 25 Euclidean instances having 50, 100, 250, 500 and 1000 vertices

| Instances | | | PEA-I | | | | HGA | | | | 2PMA | | | | PABC_BD | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \|V\| | *D* | *N* | Best | Avg | SD | ATTB | Best | Avg | SD | ATET* | Best | Avg | SD | ATET* | Best | Avg | SD | ATET |
| **50** | 5 | 1 | **7.60** | 7.64 | 0.10 | 0.39 | **7.60** | 7.64 | 0.06 | NR | **7.60** | **7.62** | 0.05 | NR | **7.60** | **7.62** | 0.03 | 0.30 |
| | | 2 | 7.75 | 7.75 | 0.01 | 0.17 | **7.68** | 7.75 | 0.01 | NR | 7.70 | **7.71** | 0.03 | NR | 7.75 | 7.75 | 0.01 | 0.26 |
| | | 3 | 7.25 | 7.27 | 0.05 | 0.41 | **7.24** | 7.27 | 0.04 | NR | 7.24 | 7.26 | 0.03 | NR | 7.25 | **7.25** | 0.00 | 0.25 |
| | | 4 | 6.62 | 6.63 | 0.02 | 0.20 | **6.59** | 6.63 | 0.02 | NR | 6.62 | **6.62** | 0.01 | NR | 6.62 | **6.62** | 0.00 | 0.21 |
| | | 5 | 7.39 | 7.42 | 0.04 | 0.42 | **7.32** | 7.42 | 0.03 | NR | 7.33 | **7.36** | 0.01 | NR | 7.39 | 7.39 | 0.00 | 0.23 |
| **100** | 10 | 1 | 7.76 | 7.82 | 0.03 | 2.88 | **7.62** | **7.80** | 0.02 | NR | 7.76 | 7.84 | 0.04 | NR | 7.76 | **7.80** | 0.03 | 0.61 |
| | | 2 | 7.85 | 7.89 | 0.04 | 2.31 | **7.71** | **7.80** | 0.04 | NR | 7.73 | 7.87 | 0.08 | NR | 7.85 | 7.86 | 0.01 | 0.65 |
| | | 3 | 7.90 | 7.97 | 0.04 | 3.74 | **7.73** | **7.82** | 0.03 | NR | 7.90 | 7.97 | 0.04 | NR | 7.92 | 7.95 | 0.02 | 0.61 |
| | | 4 | 7.98 | 8.04 | 0.03 | 3.16 | **7.72** | **7.94** | 0.03 | NR | 7.98 | 8.04 | 0.03 | NR | 7.98 | 8.02 | 0.02 | 0.55 |
| | | 5 | 8.16 | 8.21 | 0.03 | 3.22 | **7.92** | **8.02** | 0.02 | NR | 8.11 | 8.19 | 0.05 | NR | 8.16 | 8.19 | 0.02 | 0.63 |
| **250** | 15 | 1 | 12.24 | 12.36 | 0.05 | 26.35 | **12.22** | 12.32 | 0.05 | NR | 12.27 | 12.39 | 0.06 | NR | **12.22** | **12.31** | 0.06 | 3.01 |
| | | 2 | 12.04 | 12.13 | 0.04 | 25.18 | **12.00** | **12.08** | 0.05 | NR | 12.03 | 12.18 | 0.07 | NR | 12.01 | **12.08** | 0.04 | 2.83 |
| | | 3 | 12.03 | 12.11 | 0.05 | 26.66 | 12.00 | **12.06** | 0.05 | NR | 12.01 | 12.15 | 0.08 | NR | **11.98** | **12.06** | 0.04 | 2.63 |
| | | 4 | 12.42 | 12.57 | 0.05 | 28.44 | **12.40** | **12.47** | 0.03 | NR | 12.43 | 12.58 | 0.06 | NR | 12.45 | 12.51 | 0.04 | 2.71 |
| | | 5 | 12.28 | 12.39 | 0.05 | 23.52 | 12.25 | 12.30 | 0.05 | NR | 12.28 | 12.42 | 0.07 | NR | **12.21** | **12.28** | 0.04 | 2.96 |
| **500** | 20 | 1 | 16.96 | 17.13 | 0.06 | 124.63 | 16.93 | 17.01 | 0.04 | NR | 16.92 | 17.12 | 0.08 | NR | **16.85** | **16.94** | 0.04 | 9.53 |
| | | 2 | 16.81 | 16.99 | 0.07 | 134.12 | 16.78 | 16.85 | 0.05 | NR | 16.81 | 16.97 | 0.07 | NR | **16.71** | **16.83** | 0.05 | 9.68 |
| | | 3 | 16.89 | 17.04 | 0.06 | 137.94 | 16.83 | 16.89 | 0.03 | NR | 16.95 | 17.08 | 0.08 | NR | **16.79** | **16.88** | 0.05 | 10.00 |
| | | 4 | 16.96 | 17.10 | 0.06 | 155.62 | **16.84** | **16.94** | 0.05 | NR | 17.00 | 17.12 | 0.06 | NR | 16.86 | 16.96 | 0.06 | 9.71 |
| | | 5 | 16.58 | 16.72 | 0.06 | 130.08 | **16.47** | **16.55** | 0.06 | NR | 16.51 | 16.72 | 0.07 | NR | 16.51 | 16.59 | 0.04 | 9.30 |
| **1000** | 25 | 1 | 23.97 | 24.19 | 0.10 | 1148.72 | 23.92 | 24.20 | 0.12 | NR | 23.92 | 24.18 | 0.11 | NR | **23.80** | **23.97** | 0.08 | 31.86 |
| | | 2 | 23.70 | 23.98 | 0.13 | 990.63 | 23.65 | 24.09 | 0.13 | NR | 23.78 | 23.96 | 0.10 | NR | **23.51** | **23.68** | 0.08 | 31.56 |
| | | 3 | 23.61 | 23.76 | 0.08 | 1120.10 | 23.55 | 23.70 | 0.08 | NR | 23.53 | 23.76 | 0.12 | NR | **23.37** | **23.50** | 0.07 | 30.41 |
| | | 4 | 24.04 | 24.16 | 0.07 | 1124.68 | 24.04 | 24.16 | 0.07 | NR | 23.91 | 24.16 | 0.10 | NR | **23.72** | **23.89** | 0.07 | 31.53 |
| | | 5 | 23.75 | 23.90 | 0.07 | 1086.25 | 23.75 | 23.89 | 0.07 | NR | 23.56 | 23.87 | 0.12 | NR | **23.48** | **23.64** | 0.07 | 33.28 |

⋆NR means *Not Reported*

**Table 4** Results of **PEA-I** (Singh and Gupta 2007), **HGA** (Binh et al. 2008), **2PMA** (Vuppuluri and Patvardhan 2021) and **PABC_BD** on 20 non-Euclidean instances having 100, 250, 500 and 1000 vertices

| Instances | | | PEA-I | | | | HGA | | | | 2PMA | | | | PABC_BD | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \|V\| | *D* | Number | Best | Avg | SD | ATTB | Best | *Avg* | SD | ATET* | Best | Avg | SD | ATET* | Best | Avg | SD | ATET |
| **100** | 10 | 1 | 2.33 | 2.37 | 0.03 | 2.16 | **2.32** | 2.38 | 0.03 | NR | **2.32** | 2.37 | 0.03 | NR | 2.33 | **2.35** | 0.01 | 0.44 |
| | | 2 | 2.20 | **2.22** | 0.02 | 2.17 | **2.18** | **2.22** | 0.02 | NR | 2.24 | 2.29 | 0.04 | NR | 2.20 | **2.22** | 0.01 | 0.42 |
| | | 3 | 2.40 | 2.43 | 0.04 | 1.76 | 2.40 | **2.41** | 0.04 | NR | **2.34** | 2.42 | 0.04 | NR | 2.40 | 2.44 | 0.02 | 0.45 |
| | | 4 | 2.18 | 2.23 | 0.02 | 1.81 | 2.16 | 2.21 | 0.04 | NR | **2.14** | **2.17** | 0.02 | NR | 2.17 | 2.20 | 0.02 | 0.42 |
| | | 5 | 2.35 | 2.42 | 0.04 | 1.95 | 2.34 | 2.40 | 0.03 | NR | **2.29** | **2.34** | 0.03 | NR | 2.36 | 2.39 | 0.02 | 0.47 |
| **250** | 15 | 1 | 3.73 | 3.79 | 0.03 | 17.41 | 3.72 | 3.76 | 0.03 | NR | **3.71** | 3.76 | 0.02 | NR | **3.71** | **3.74** | 0.02 | 2.00 |
| | | 2 | 3.79 | 3.83 | 0.03 | 18.70 | 3.77 | 3.82 | 0.03 | NR | **3.60** | **3.65** | 0.02 | NR | 3.78 | 3.80 | 0.01 | 1.97 |
| | | 3 | **3.69** | 3.76 | 0.03 | 18.24 | **3.69** | 3.77 | 0.03 | NR | 3.81 | 3.85 | 0.02 | NR | **3.69** | **3.72** | 0.02 | 2.34 |
| | | 4 | 3.76 | 3.82 | 0.03 | 15.36 | 3.75 | 3.80 | 0.03 | NR | **3.67** | **3.72** | 0.03 | NR | 3.75 | 3.78 | 0.02 | 2.04 |
| | | 5 | 3.88 | 3.95 | 0.04 | 16.03 | 3.87 | 3.96 | 0.04 | NR | **3.69** | **3.72** | 0.02 | NR | 3.88 | 3.91 | 0.01 | 2.22 |
| **500** | 20 | 1 | 6.24 | 6.29 | 0.03 | 79.87 | 6.23 | 6.27 | 0.02 | NR | 6.26 | 6.28 | 0.01 | NR | **6.22** | **6.25** | 0.02 | 6.74 |
| | | 2 | 6.30 | 6.36 | 0.03 | 81.48 | 6.28 | 6.35 | 0.03 | NR | **6.15** | **6.19** | 0.02 | NR | 6.29 | 6.31 | 0.02 | 7.13 |
| | | 3 | 6.16 | 6.22 | 0.03 | 87.48 | 6.16 | 6.24 | 0.03 | NR | 6.23 | 6.27 | 0.02 | NR | **6.14** | **6.17** | 0.01 | 7.47 |
| | | 4 | 6.25 | 6.32 | 0.03 | 71.39 | 6.23 | 6.32 | 0.04 | NR | **6.18** | **6.21** | 0.02 | NR | 6.25 | 6.28 | 0.02 | 6.84 |
| | | 5 | 6.25 | 6.29 | 0.04 | 77.39 | 6.22 | 6.27 | 0.04 | NR | 6.27 | 6.30 | 0.02 | NR | **6.21** | **6.24** | 0.02 | 7.36 |
| **1000** | 25 | 1 | 11.26 | 11.31 | 0.03 | 609.97 | **11.25** | 11.30 | 0.03 | NR | 11.26 | 11.30 | 0.02 | NR | **11.25** | **11.27** | 0.01 | 24.26 |
| | | 2 | 11.30 | 11.34 | 0.03 | 620.88 | 11.27 | 11.32 | 0.03 | NR | **11.22** | **11.26** | 0.02 | NR | 11.28 | 11.31 | 0.02 | 23.45 |
| | | 3 | 11.30 | 11.35 | 0.03 | 585.26 | 11.28 | 11.34 | 0.04 | NR | **11.26** | **11.30** | 0.02 | NR | 11.29 | 11.31 | 0.01 | 23.75 |
| | | 4 | 11.22 | 11.26 | 0.03 | 628.37 | 11.22 | 11.25 | 0.03 | NR | **11.19** | **11.22** | 0.02 | NR | **11.19** | **11.22** | 0.02 | 24.73 |
| | | 5 | 11.39 | 11.42 | 0.03 | 608.97 | 11.37 | 11.41 | 0.03 | NR | **11.21** | **11.25** | 0.02 | NR | 11.35 | 11.38 | 0.02 | 22.78 |

⋆NR means *Not Reported*

system used for PEA-I is 561. Higher CPU Mark result represents higher performance. Hence, the performance of our computer system used for PABC_BD is ≈ 3.8 times (scaling factor) better than the computer system used for PEA-I (see Table 5). Also, authors of PEA-I reported only average time till best (i.e. ATTB) needed by PEA-I to reach the best solution, not average total execution time (i.e. ATET) needed by PEA-I. Even considering this fact about average time till best (ATTB) versus average total execution time (ATET) (see Tables 3 and 4) of each instance obtained by PEA-I and PABC_BD respectively, we can safely say that PABC_BD converges much faster in terms of finding better solution quality in comparison to PEA-I.

It is to be noted and is already discussed in Sect. 1 that authors (Binh et al. 2008) did not discuss or report the computational time needed for obtaining the solution quality of each Euclidean and non-Euclidean instances (see *N R* that refers to *Not Reported* in Tables 3 and 4). Also, one can infer that HGA obtains overall better results than that of PEA-I (Singh and Gupta 2007) at the cost of much higher computational time. Since our proposed PABC_BD is much faster in terms of finding better solution quality in comparison to PEA-I, therefore, it implies that PABC_BD is even much more faster in terms of finding better solution quality in comparison to HGA. It is clear from the results reported in Tables 3 and 4 that PABC_BD is comparable with HGA in terms of *Best* on Euclidean instances, but is better than HGA in terms of *Avg* on Euclidean instances as well as in terms of *Best* and *Avg* on non-Euclidean instances. Since authors of 2PMA (Vuppuluri and Patvardhan 2021) did not report the computational time (ATTB or ATET) on each instance while comparing with PEA-I in their paper, therefore, we have not provided the details of comparison of our proposed PABC_BD with 2PMA in terms of computational time.

### 3.4 Comparison of PABC_BD with VNS, EA, ACO, LRS and LABD

The performance of VNS (Gruber et al. 2006), EA (Gruber et al. 2006), ACO (Gruber et al. 2006) and LRS (Lucena et al. 2010) has been tested on only a set of Euclidean benchmark instances. To give a fair comparison, we have also compared our PABC_BD with these approaches on only Euclidean instances. Also, to test VNS, EA, ACO and LRS approaches, their authors used different-2 criterion for termination. As in (Gruber et al. 2006), authors used different CPU time limits which depend on the size of instances. For example, they used CPU time limits of 2000, 3000, and 4000 seconds for instances with $|V| = 100, 250, 500$, respectively. In addition, in case of the VNS and ACO approaches, they terminated their approaches after 1000 iterations without further improvement of the best-so-far solution obtained. For instances with $|V| = 1000$, they used a CPU time limit of 1000 seconds. Similarly, authors of LRS also used a CPU time limit as the termination criterion. They used CPU time limits of 200, 1500, 3000, and 4500 seconds for instances with $|V| = 100, 250, 500$ and 1000, respectively. In addition to these different-2 termination criterion, configurations of the computer system used for VNS, EA, ACO (Pentium 4 with 2.8 GHz) and LRS (Pentium 4 with 2.0 GHz) approaches are also different from that of computer system used for PABC_BD.

To know the performance of computer system used for PABC_BD in comparison to that of computer system used for VNS, EA, ACO and LRS approaches, we have again used a scaling factor according to the public CPU benchmark provided by PassMark Software. Hence, the performance of our computer system used for PABC_BD is ≈ 3.4 times (scaling factor) better than the computer system used for VNS, EA and ACO approaches (Gruber et al. 2006) (see Table 5). In case of LRS (Lucena et al. 2010), the performance of our computer system used for PABC_BD is ≈ 4.6 times (scaling factor) better than the computer system used for LRS (see Table 5). Looking at all these aspects, i.e. different-2 criteria for termination of VNS, EA, ACO and LRS approaches and their computer system configuration, it is difficult to exactly compare PABC_BD with VNS, EA, ACO and LRS approaches in terms of solution quality and computational time. To overcome this difficulty, we have used another termination criterion for PABC_BD which is different from the termination criterion used for comparing with PEA-I (see Sect. 3.3). PABC_BD is allowed to execute till the best-so-far solution obtained does not improve over 5000 generations in order to compare VNS, EA, ACO and LRS approaches in terms of solution quality and computational time. Like VNS, EA and ACO approaches, PABC_BD is also given 50 runs for each problem instance.

For LABD (Torkestani 2012), author allowed 100 runs to execute their approach on same machine as used in (Lucena et al. 2010) (i.e. (Pentium 4 with 2.0 GHz)). However, results of LABD (Torkestani 2012) are reported in terms of best value and average total execution time only. To compare PABC_BD with LABD, we have used the same termination criterion for PABC_BD as discussed in the previous paragraph.

Table 6 reports the results of 20 Euclidean instances for VNS, EA, ACO, LRS, LABD and PABC_BD approaches. In this table, first three columns $|V|$, *D* and *N* under *Instances* respectively denote the total number of vertices of each Euclidean instance, the diameter-bound and the instance number from each set of five instances; for VNS, EA and ACO approaches, each next four columns *Best*, *Avg*, *SD* and *ATET* respectively denote the best-so-far solution obtained in terms of fitness, average solution quality, standard deviation and average total execution time; for LRS, next three columns *Best*, *Avg* and *ATET* respectively denote the best-so-far solution obtained in terms of fitness, average solution

**Table 5** Scaling factors of various computer systems used in existing approaches with respect to the computer system (Intel Core i5 processor 3.3 GHz × 4) used in **PABC_BD**

| Existing approaches | Configuration of computer system | CPU mark | Scaling factor |
|---|---|---|---|
| PEA-I | Pentium 4 with 2.4 GHz | 561 | ≈ 3.8 |
| VNS, EA, ACO | Pentium 4 with 2.8 GHz | 631 | ≈ 3.4 |
| LRS | Pentium 4 with 2.0 GHz | 464 | ≈ 4.6 |
| LABD | Pentium 4 with 2.0 GHz | 464 | ≈ 4.6 |

quality and average total execution time ; for LABD, next two columns *Best* and *ATET* respectively denote the best-so-far solution obtained in terms of fitness and average total execution time; for PABC_BD, next four columns *Best*, *Avg*, *SD* and *ATET* respectively denote the best-so-far solution obtained in terms of fitness, average solution quality, standard deviation and average total execution time.

Table 6 highlights the best value (*Best*) and the best average solution quality (*Avg*) among VNS, EA, ACO, LRS, LABD and PABC_BD approaches in bold fonts. Comparing the results of PABC_BD with the results of VNS on 20 Euclidean instances, PABC_BD, in terms of *Best*, is better on 16 and is equal on 4. In terms of *Avg*, PABC_BD is better than VNS in all instances. Comparing the results of PABC_BD with the results of EA on 20 Euclidean instances, PABC_BD, in terms of *Best*, is better on 13, is equal on 6 and is worse on 1. In terms of *Avg*, PABC_BD is better on 19 and is worse on 1 in comparison to EA. Similarly, comparing the results of PABC_BD with the results of ACO approach on 20 Euclidean instances, PABC_BD, in terms of *Best*, is better on 12, is equal on 4 and is worse on 4. In terms of *Avg*, PABC_BD is better on 17 and is worse on 3 in comparison to ACO. Comparing the results of PABC_BD with the results of LRS on 20 Euclidean instances, PABC_BD, in terms of *Best*, is better in all instances. In terms of *Avg*, PABC_BD is also better than LRS in all instances. Similarly, comparing the results of PABC_BD with the results of LABD, on 20 Euclidean instances, PABC_BD, in terms of *Best*, is better on 15, is equal on 3 and is worse on 2.

In terms of computational time (*ATET*) in Table 6, even considering the points—the performance of computer system used for PABC_BD is ≈ 3.4 times (scaling factor) better than the computer system used for VNS, EA and ACO approaches and ≈ 4.6 times (scaling factor) better than the computer system used for LRS and LABD approaches—discussed in the second paragraph of Sect. 3.4, we can safely say that PABC_BD converges much faster towards better solution quality in comparison to VNS, EA, ACO, LRS and LABD approaches.

### 3.5 Statistical analysis

For statistical analysis, we have performed a non-parametric *Wilcoxon's signed rank test* (García et al. 2009) on each data-set in order to compare PABC_BD with PEA-I, VNS, EA, ACO, LRS and LABD approaches in terms of the best (*Best*) and the average solution quality (*Avg*). We have used *Wilcoxon Signed-Rank Test Calculator* which is available at http://www.socscistatistics.com/tests/signedranks/Default2.aspx. For this statistical test, we first calculate the difference between the results obtained by each two compared approaches on each data-set and then rank them according to their absolute value. For each data-set, $R^+$ is the sum of ranks in which the second approach outperforms the first, while $R^-$ denotes the sum of ranks for the opposite case. If $min\{R^+; R^-\}$ is less than or equal to the critical value, then this test detects significant difference between the two compared approaches. The critical values are taken from the statistical table available at http://users.stat.ufl.edu/~athienit/Tables/tables. Tables 7 and 8 report the results of *Wilcoxon's signed rank test* with a level of significance $\alpha = 0.05$ for the best value (*Best*) and the average solution quality (*Avg*) over 50 runs respectively.

In Tables 7 and 8, the column *Data-set* denotes the name of each data-set; the column *Comparison* denotes the name of two compared approaches; the next five columns *Sample Size*, *Critical Value*, $R^+$, $R^-$ and *Significant* respectively denote the sample size, critical value, $R^+$, $R^-$ and significant difference between the two compared approaches ("**yes**" if there exists a significant difference between two compared approaches, otherwise "**no**") for each *Best* and *Avg* on each data-set. The results in Tables 7 and 8 confirm that PABC_BD significantly performs better with respect to the other approaches because the value of $R^-$ (corresponding to the other approaches) is lower than the value of $R^+$ (corresponding to the PABC_BD) as well as the critical value in all performed comparisons.

### 3.6 Analysis of convergence behavior of PABC_BD

This subsection analyzes the convergence behavior of PABC_BD in order to gain a useful insight into PABC_BD. For this, we consider four instances, i.e. 100_1, 250_1, 500_2 and 1000_5 which are picked randomly from the set of Euclidean instances. Figure 3a–d exhibit the average solution quality over 50 runs versus the number of generations on considered instances. In each figure, PABC_BD is allowed to execute over 5000 generations. X-axis represents the "*Num-*

**Table 6** Results of **VNS** (Gruber et al. 2006), **EA** (Gruber et al. 2006), **ACO** (Gruber et al. 2006), **LRS** (Lucena et al. 2010), **LABD** (Torkestani 2012) and **PABC_BD** on 20 Euclidean instances having 100, 250, 500 and 1000 vertices

| Instances | | | VNS | | | | EA | | | | ACO | | | | LRS | | | LABD | | ABC | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \|V\| | *D* | Number | Best | Avg | SD | ATET | Best | Avg | SD | ATET | Best | Avg | SD | ATET | Best | Avg | ATET | Best | ATET | Best | Avg | SD | ATET |
| **100** | 10 | 1 | **7.76** | 7.81 | 0.03 | 37.35 | **7.76** | 7.79 | 0.03 | 678.70 | **7.76** | 7.77 | 0.02 | 27.78 | 7.83 | 7.88 | 200.00 | **7.76** | 45.01 | **7.76** | **7.76** | 0.01 | 3.97 |
| | | 2 | **7.85** | 7.89 | 0.03 | 41.52 | **7.85** | 7.86 | 0.02 | 734.65 | **7.85** | 7.86 | 0.01 | 25.10 | 7.94 | 7.96 | 200.00 | **7.85** | 43.47 | **7.85** | **7.85** | 0.00 | 3.67 |
| | | 3 | **7.90** | 7.96 | 0.04 | 38.66 | **7.90** | 7.96 | 0.04 | 897.58 | 7.91 | 7.94 | 0.04 | 28.48 | 7.98 | 8.00 | 200.00 | **7.90** | 35.16 | **7.90** | **7.92** | 0.01 | 4.09 |
| | | 4 | **7.98** | 8.05 | 0.03 | 34.27 | **7.98** | 8.00 | 0.03 | 732.83 | **7.98** | 8.00 | 0.01 | 38.24 | 8.04 | 8.08 | 200.00 | **7.90** | 32.98 | 7.98 | **7.99** | 0.01 | 3.90 |
| | | 5 | 8.17 | 8.20 | 0.03 | 39.31 | **8.16** | 8.18 | 0.03 | 410.17 | **8.16** | 8.17 | 0.00 | 25.45 | 8.21 | 8.22 | 200.00 | 8.18 | 26.12 | **8.16** | **8.16** | 0.00 | 4.06 |
| **250** | 15 | 1 | 12.30 | 12.43 | 0.05 | 1584.31 | 12.28 | 12.38 | 0.05 | 1992.70 | 12.23 | 12.28 | 0.02 | 174.17 | 12.45 | 12.53 | 1500.00 | 12.30 | 110.21 | **12.18** | **12.26** | 0.04 | 12.69 |
| | | 2 | 12.02 | 12.17 | 0.06 | 1678.90 | 12.05 | 12.16 | 0.06 | 1969.42 | 12.02 | 1.04 | 0.01 | 156.71 | 12.31 | 12.36 | 1500.00 | **12.00** | 102.43 | 12.01 | **12.04** | 0.02 | 13.44 |
| | | 3 | 12.04 | 12.11 | 0.04 | 1309.21 | 12.03 | 12.10 | 0.04 | 1897.87 | 12.00 | 12.02 | 0.01 | 145.29 | 12.13 | 12.18 | 1500.00 | 12.03 | 98.61 | **11.97** | **12.01** | 0.02 | 13.73 |
| | | 4 | 12.51 | 12.62 | 0.06 | 1572.39 | 12.49 | 12.59 | 0.05 | 1742.48 | 12.46 | 12.49 | 0.01 | 159.41 | 12.70 | 12.75 | 1500.00 | 12.50 | 94.92 | **12.39** | **12.45** | 0.03 | 14.20 |
| | | 5 | 12.28 | 12.42 | 0.07 | 1525.39 | 12.31 | 12.42 | 0.06 | 1712.16 | 12.23 | 12.29 | 0.04 | 211.11 | 12.44 | 12.51 | 1500.00 | 12.28 | 90.36 | **12.18** | **12.23** | 0.03 | 13.22 |
| **500** | 20 | 1 | 16.97 | 17.13 | 0.07 | 3718.54 | 16.87 | 16.97 | 0.06 | 2609.28 | **16.78** | **16.85** | 0.03 | 906.17 | 17.20 | 17.25 | 3000.00 | 16.37 | 210.24 | 16.83 | 16.91 | 0.04 | 36.80 |
| | | 2 | 16.88 | 17.05 | 0.07 | 3762.02 | 16.76 | 16.86 | 0.05 | 2472.59 | **16.63** | **16.70** | 0.03 | 1012.91 | 17.06 | 17.25 | 3000.00 | 16.86 | 186.90 | 16.65 | 16.79 | 0.05 | 38.67 |
| | | 3 | 16.98 | 17.15 | 0.07 | 3849.42 | 16.86 | 16.98 | 0.05 | 2808.15 | 16.79 | 16.84 | 0.03 | 1069.84 | 17.22 | 17.28 | 3000.00 | 16.98 | 160.39 | **16.76** | **16.83** | 0.03 | 41.67 |
| | | 4 | 16.99 | 17.17 | 0.06 | 3687.97 | 16.94 | 17.04 | 0.06 | 2837.81 | **16.80** | 16.92 | 0.04 | 1010.91 | 17.24 | 17.31 | 3000.00 | 17.00 | 153.21 | 16.81 | **16.91** | 0.05 | 44.01 |
| | | 5 | 16.57 | 16.79 | 0.07 | 3693.13 | 16.50 | 16.59 | 0.05 | 2294.43 | **16.42** | **16.46** | 0.02 | 947.26 | 16.94 | 17.02 | 3000.00 | 16.56 | 144.10 | 16.44 | 16.54 | 0.04 | 38.04 |
| **1000** | 25 | 1 | 25.18 | 25.57 | 0.14 | 905.50 | **23.43** | **23.57** | 0.08 | 565.38 | 24.84 | 25.03 | 0.07 | 812.78 | 24.66 | 24.83 | 4500.00 | 24.60 | 392.13 | 23.76 | 23.88 | 0.06 | 117.56 |
| | | 2 | 25.01 | 25.34 | 0.14 | 930.04 | **23.46** | 23.67 | 0.08 | 561.49 | 24.63 | 24.83 | 0.06 | 847.79 | 24.46 | 24.77 | 4500.00 | 24.48 | 399.35 | **23.46** | **23.58** | 0.06 | 115.58 |
| | | 3 | 24.82 | 25.09 | 0.11 | 956.06 | 23.62 | 23.79 | 0.08 | 524.21 | 24.50 | 24.62 | 0.06 | 838.68 | 24.32 | 24.56 | 4500.00 | 24.62 | 365.19 | **23.27** | **23.40** | 0.07 | 115.65 |
| | | 4 | 25.29 | 25.57 | 0.11 | 928.97 | 23.79 | 23.96 | 0.09 | 602.30 | 24.99 | 25.09 | 0.06 | 793.41 | 24.61 | 24.71 | 4500.00 | 24.60 | 340.74 | **23.68** | **23.83** | 0.06 | 122.48 |
| | | 5 | 25.03 | 25.25 | 0.12 | 935.85 | 23.82 | 23.98 | 0.10 | 516.74 | 24.57 | 24.73 | 0.06 | 844.67 | 24.27 | 24.50 | 4500.00 | 24.27 | 327.16 | **23.37** | **23.58** | 0.07 | 109.01 |

**Table 7** Results of statistical comparison for the best value (*Best*)

| Data-set | Comparison | Best | | | | |
|---|---|---|---|---|---|---|
| | | Sample size | Critical value | $R^+$ | $R^-$ | Significant |
| Euclidean | PEA-I-PABC_BD | 16 | 29 | 131 | 5 | **Yes** |
| | VNS-PABC_BD | 16 | 29 | 136 | 0 | **Yes** |
| | EA-PABC_BD | 14 | 21 | 93 | 12 | **Yes** |
| | ACO-PABC_BD | 16 | 29 | 116 | 20 | **Yes** |
| | LRS-PABC_BD | 20 | 52 | 210 | 0 | **Yes** |
| | LABD-PABC_BD | 17 | 34 | 136 | 17 | **Yes** |
| Non-euclidean | PEA-I-PABC_BD | 14 | 21 | 105 | 0 | **Yes** |

**Table 8** Results of statistical comparison for the average solutions quality (*Avg*)

| Data-set | Comparison | Avg | | | | |
|---|---|---|---|---|---|---|
| | | Sample size | Critical value | $R^+$ | $R^-$ | Significant |
| Euclidean | PEA-I-PABC_BD | 24 | 81 | 300 | 0 | **Yes** |
| | VNS-PABC_BD | 20 | 52 | 210 | 0 | **Yes** |
| | EA-PABC_BD | 20 | 52 | 192 | 18 | **Yes** |
| | ACO-PABC_BD | 19 | 46 | 151.5 | 38.5 | **Yes** |
| | LRS-PABC_BD | 20 | 52 | 210 | 0 | **Yes** |
| Non-euclidean | PEA-I-PABC_BD | 19 | 46 | 189 | 1 | **Yes** |

*ber of Generations*" generated, whereas Y-axis represents the "*Average Solution Quality*" obtained over 50 runs after carrying out the "*Number of Generations*" generated. One can observe from Fig. 3a–d that the PABC_BD converges rapidly towards better solution quality as the number of generations increases.

### 3.7 Collective picture

Table 9 gives a collective picture of an overall comparison of PABC_BD on Euclidean and non-Euclidean Instances with PEA-I, VNS, EA, ACO, LRS and LABD approaches in terms of best value and average solution quality obtained. From Table 9, one can observe the superiority of PABC_BD over PEA-I, VNS, EA, ACO, LRS and LABD approaches in terms of best value (*Best*) and average solution quality (*Avg*).

### 3.8 Performance of PABC_BD on 15 Euclidean instances of 1000 nodes with various diameter bounds

The subsection discusses the performance of PABC_BD on 15 Euclidean Steiner tree instances of Beasley's OR-Library with 1000 nodes for various diameter bounds that range from 4 to 25 whose results obtained by two hierarchical clustering heuristics—$Cd^A$ and $Cd^B$—and ACO approach are reported in (Gruber and Raidl 2009) (see discussion on literature survey in Sect. 1). It is mentioned in Gruber and Raidl (2009) that being problem-specific heuristics, $Cd^A$ and $Cd^B$ takes very few seconds to find high quality solutions in general, whereas ACO approach which is taken from (Gruber et al. 2006) needs computation times of one hour and more. Table 10 reports the results of $Cd^A$, $Cd^B$, ACO and PABC_BD on such Euclidean instances, where the results on instances obtained by $Cd^A$, $Cd^B$ and ACO approach are taken from (Gruber and Raidl 2009). In this table, first column denotes instances with 1000 nodes for various bounds; for $Cd^A$, $Cd^B$ and ACO, each next two columns *Avg* and *SD* respectively denote the average solution quality and standard deviation obtained over 30 runs; and the last three columns *Avg*, *SD* and *ATET* respectively denote the average solution quality, standard deviation and average total execution time. Table 10 shows that the performance of PABC_BD decreases with decreasing diameter bound against $Cd^A$, $Cd^B$ and ACO approach. The possible reason behind poor performance of PABC_BD particularly on smaller diameter bounds is the use of RGH decoding. It is mentioned in (Gruber and Raidl 2009) that RGH in comparison to clustering heuristics lacks in finding a good backbone. For more details, readers are suggested to read (Gruber and Raidl 2009).

## 4 Conclusion

In this paper, we present an artificial bee colony algorithm (PABC_BD) for the bounded diameter minimum spanning tree (BD-MST) problem. The proposed approach employs a permutation encoding of solutions. To exploit this encod-

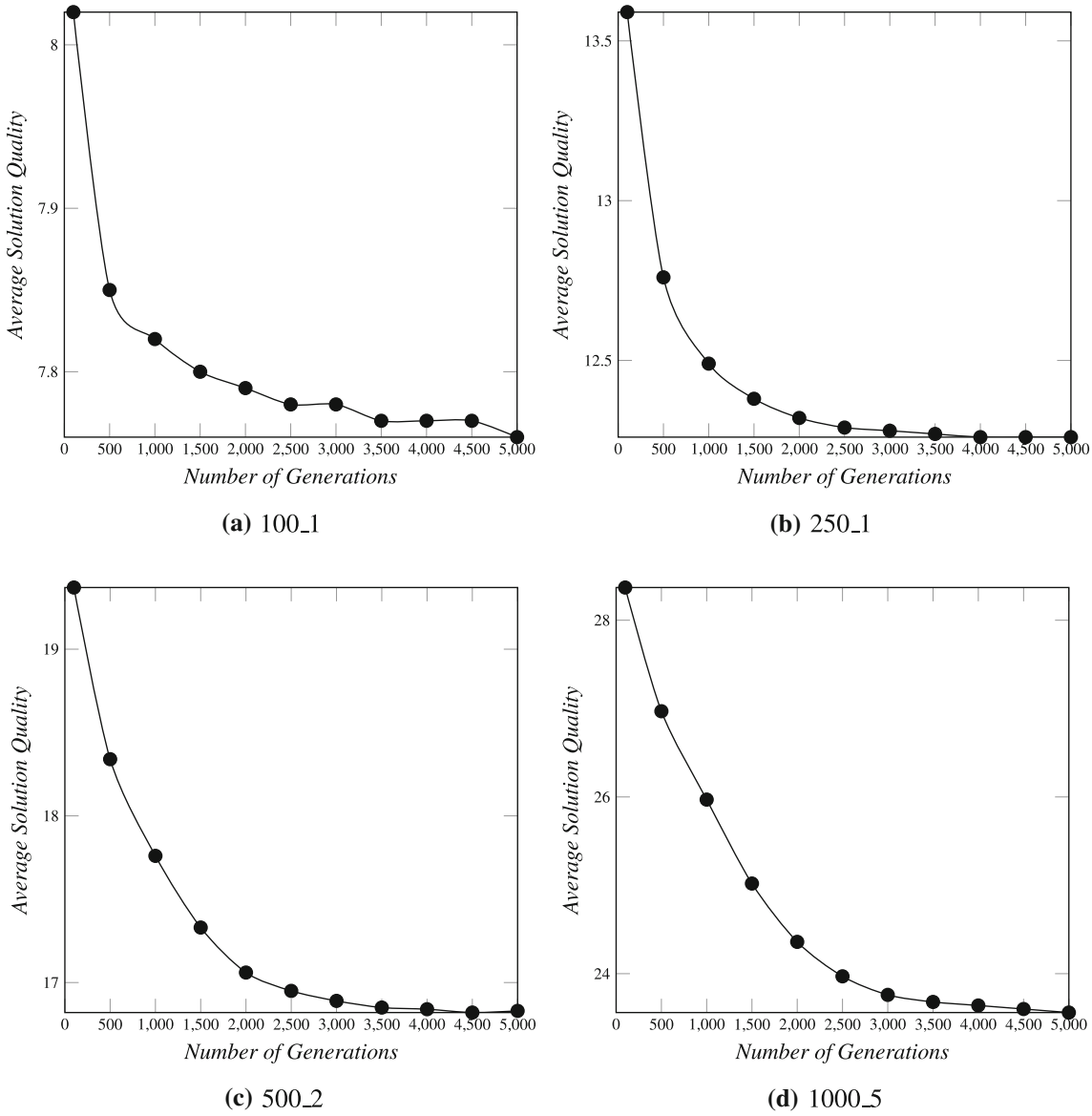


**Fig. 3** Convergence curve of PABC_BD for the average solution quality

**Table 9** Overall comparison of PABC_BD with **PEA-I** (Singh and Gupta 2007), **VNS** (Gruber et al. 2006), **EA** (Gruber et al. 2006), **ACO** (Gruber et al. 2006), **LRS**, (Lucena et al. 2010) and **LABD** (Torkestani 2012)

| Comparison | Euclidean-instances | | | | | | | | | | | Non-euclidean-instances | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | | | | | | Avg | | | | | Best | Avg |
| State-of-the-art approaches | PEA-I | VNS | EA | ACO | LRS | LABD | PEA-I | VNS | EA | ACO | LRS | PEA-I | PEA-I |
| Total instances | 25⋆ | 20 | 20 | 20 | 20 | 20 | 25⋆ | 20 | 20 | 20 | 20 | 20 | 20 |
| Better | 14 | 16 | 13 | 12 | 20 | 15 | 24 | 20 | 19 | 17 | 20 | 13 | 18 |
| Same | 9 | 4 | 5 | 4 | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 6 | 1 |
| Worse | 2 | 0 | 2 | 4 | 0 | 2 | 0 | 0 | 1 | 3 | 0 | 1 | 1 |

⋆ See Sect. 3.3

**Table 10** Comparison of PABC_BD with $Cd^A$, $Cd^B$ (Gruber and Raidl 2009) and ACO (Gruber et al. 2006)

| Diameter | $Cd^A$ | | $Cd^B$ | | ACO | | PABC_BD | | |
|---|---|---|---|---|---|---|---|---|---|
| | Best | SD | Best | SD | Avg | SD | Avg | SD | ATET |
| **4** | 68.32 | 0.72 | 68.32 | 0.70 | **65.80** | 0.48 | 108.09 | 1.44 | 79.93 |
| **6** | 47.40 | 4.85 | 47.17 | 4.61 | **42.12** | 0.26 | 52.98 | 1.02 | 27.58 |
| **8** | 37.07 | 1.35 | 36.94 | 1.34 | **34.75** | 0.23 | 39.02 | 0.40 | 13.85 |
| **10** | 33.55 | 0.67 | 33.34 | 0.66 | **31.04** | 0.20 | 33.66 | 0.20 | 9.91 |
| **12** | 32.26 | 0.48 | 31.96 | 0.44 | **28.64** | 0.23 | 31.12 | 0.15 | 8.59 |
| **14** | 31.38 | 0.37 | 31.02 | 0.33 | **26.65** | 0.32 | 29.35 | 0.20 | 9.06 |
| **16** | 30.7937 | 0.33 | 30.43 | 0.29 | 25.58 | 0.19 | **25.48** | 1.15 | 32.08 |
| **18** | 30.52 | 0.29 | 30.13 | 0.27 | 24.88 | 0.16 | **24.86** | 0.09 | 34.07 |
| **20** | 30.31 | 0.31 | 30.04 | 0.28 | **24.37** | 0.15 | 24.41 | 0.06 | 32.42 |
| **22** | 30.24 | 0.30 | 30.07 | 0.28 | **24.01** | 0.17 | 24.06 | 0.06 | 32.49 |
| **24** | 30.02 | 0.23 | 30.16 | 0.27 | **23.77** | 0.20 | **23.77** | 0.07 | 32.27 |
| **5** | 62.29 | 0.76 | 62.06 | 0.67 | **59.59** | 0.49 | 89.16 | 1.28 | 60.44 |
| **7** | 46.73 | 3.92 | 46.41 | 3.73 | **39.99** | 0.25 | 48.47 | 0.65 | 20.60 |
| **9** | 37.02 | 1.25 | 36.89 | 1.27 | **33.59** | 0.23 | 37.19 | 0.29 | 11.46 |
| **11** | 33.41 | 0.70 | 33.17 | 0.66 | **30.27** | 0.19 | 32.85 | 0.17 | 8.91 |
| **13** | 32.11 | 0.43 | 31.80 | 0.41 | **28.12** | 0.20 | 30.65 | 0.134 | 8.10 |
| **15** | 31.27 | 0.35 | 30.89 | 0.32 | **26.39** | 0.25 | 28.60 | 0.26 | 9.76 |
| **17** | 30.77 | 0.33 | 30.37 | 0.30 | 25.38 | 0.23 | **25.26** | 0.52 | 35.36 |
| **19** | 30.54 | 0.29 | 30.08 | 0.27 | 24.77 | 0.18 | **24.70** | 0.06 | 33.30 |
| **21** | 30.30 | 0.30 | 30.04 | 0.27 | 24.77 | 0.18 | **24.28** | 0.06 | 31.83 |
| **23** | 30.06 | 0.24 | 30.12 | 0.31 | 23.97 | 0.21 | **23.93** | 0.07 | 31.6 |
| **25** | 29.95 | 0.21 | 30.14 | 0.24 | 23.77 | 0.25 | **23.65** | 0.07 | 31.96 |

ing structure, two neighborhood strategies based on *insertion on multiple positions* and *swap on multiple positions* are applied. PABC_BD has been tested on a set of benchmark instances for various diameter bounds. Computational results justify the effectiveness and robustness of our proposed PABC_BD as the diameter bound increases. On these benchmark instances, PABC_BD is much faster in finding better solution quality in comparison to state-of-the-art metaheuristic approaches except smaller diameter bounds.

The performance of PABC_BD against two hierarchical clustering heuristics on 15 Euclidean instances of 1000 nodes suggested us that in the near future, we will focus our efforts in the development of a better decoder that can be applied to decode a solution encoded as a linear permutation of all vertices of $V$ for this problem while developing a metaheuristic technique.

**Acknowledgements** This work is supported in part by a grant (grant number YSS/2015/000276) from the Science and Engineering Research Board – Department of Science & Technology, Government of India.

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

## References

Abdalla A, Deo N, Gupta P (2000) Random-tree diameter and the diameter constrained mst. Cong Numer 144:161–182

Achuthan NR, Caccetta L, Caccetta P, Geelen JF (1994) Computational methods for the diameter restricted minimum weight spanning tree problem. Aust J Comb 10:51–72

Anderson R, Stenger B, Cipolla R (2014) Using bounded diameter minimum spanning trees to build dense active appearance models. Int J Comput Vis 110(1):48–57

Bala K, Petropoulos K, Stern TE (1993) Multicasting in a linear lightwave network. In: Proceedings of IEEE INFOCOM'93, pp 1350–1358

Binh HTT, Hoai NX, McKay RI (2008) A new hybrid genetic algorithm for solving the bounded diameter minimum spanning tree problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1–6, 2008, Hong Kong, China, pp 3128–3134

Bookstein A, Klein ST (1996) Compression of correlated bit-vectors. Inf Syst 16:110–118

Deo N, Abdalla AM (2000) Computing a diameter-constrained minimum spanning tree in parallel. In: Bongiovanni GC, Gambosi G, Petreschi R (eds) Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, Proceedings, Springer, Lecture Notes in Computer Science, vol 1767, pp 17–31

García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the cec 2005 special session on real parameter optimization. J Heuristics 15(6):617–644

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco

Gouveia L, Magnanti TL (2003) Network flow models for designing diameter-constrained minimum-spanning and steiner trees. Networks 41(3):159–173

Gouveia L, Simonetti L, Uchoa E (2011) Modeling hop-constrained and diameter-constrained minimum spanning tree problems as steiner tree problems over layered graphs. Math Program 128(1–2):123–148

Gruber M, Raidl GR (2005) Variable neighborhood search for the bounded diameter minimum spanning tree problem. In: Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, Tenerife, Spain

Gruber M, Raidl GR (2008) Heuristic cut separation in a branch and cut approach for the bounded diameter minimum spanning tree problem. In: Proceedings of the 2008 International Symposium on Applications and the Internet, SAINT 2008, 28 July - 1 August 2008, Turku, Finland, IEEE Computer Society, pp 261–264, https://doi.org/10.1109/SAINT.2008.68

Gruber M, Raidl GR (2009) Exploiting hierarchical clustering for finding bounded diameter minimum spanning trees on euclidean instances. In: Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8–12, 2009, ACM, pp 263–270

Gruber M, An Hemert JI, Raidl GR (2006) Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. In: Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8–12, 2006, pp 1187–1194

Julstrom BA (2003) A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In: Proceedings of the 2003 Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations, pp 2–7

Julstrom BA (2004) Encoding bounded-diameter spanning trees with permutations and with random keys. Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York 3102:1272–1281

Julstrom BA (2009) Greedy heuristics for the bounded diameter minimum spanning tree problem. ACM J Exp Algorithmics. doi. https://doi.org/10.1145/1498698.1498699

Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Engineering Faculty, Erciyes University, Turkey, Computer Engineering Department

Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numeric function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39:459–471

Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif Intell Rev 42(1):21–57

Lucena A, Ribeiro CC, Santos AC (2010) A hybrid heuristic for the diameter constrained minimum spanning tree problem. J Global Optim 46(3):363–381

Pan QK, Tasgetiren MF, Suganthan PN, Chua TJ (2011) A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf Sci 181:2455–2468

Patvardhan C, Prakash VP (2009) Novel deterministic heuristics for building minimum spanning trees with constrained diameter. In: Pattern Recognition and Machine Intelligence, LNCS, pp 68–73

Patvardhan C, Prakash VP, Srivastav A (2014) Parallel heuristics for the bounded diameter minimum spanning tree problem. In: Annual IEEE India conference (INDICON), pp 1–5

Patvardhan C, Prakash VP, Srivastav A (2015) Fast heuristics for large instances of the euclidean bounded diameter minimum spanning tree problem. Inform (Slovenia) 39(3):281–292

Prakash VP, Patvardhan C, Srivastav A (2018) Effective heuristics for the bi-objective euclidean bounded diameter minimum spanning tree problem. In: Bhattacharyya P, Sastry H, Marriboyina V, Sharma R (eds) Smart and Innovative Trends in Next Generation Computing Technologies. NGCT 2017. Communications in Computer and Information Science, Springer, Singapore, vol 827, pp 580–589, https://doi.org/10.1007/978-981-10-8657-1_44

Prakash VP, Patvardhan C, Srivastav A (2020) A novel hybrid multi-objective evolutionary algorithm for the bi-objective minimum diameter-cost spanning tree (bi-mdcst) problem. Eng Appl Artif Intell. https://doi.org/10.1016/j.engappai.2019.103237

Prim R (1957) Shortest connection networks and some generalizations. Bell Syst Tech J 36:1389–1401

Raidl GR, Julstrom BA (2003) Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In: Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), March 9–12, 2003, Melbourne, FL, USA, pp 747–752

Raymond K (1989) A tree-based algorithm for distributed mutual exclusion. ACM Trans Comput Syst 7:61–77

Saha S, Kumar R (2011) Improvement of bounded-diameter MST instances with hybridization of multi-objective EA. In: Jena SK, Kumar R, Turuk AK, Dash M (eds) Proceedings of the 2011 International Conference on Communication, Computing and Security, ICCCS 2011, Odisha, India, February 12–14, 2011, ACM, pp 468–473

Saha S, Aslam M, Kumar R (2010) Assessing the performance of bi-objective MST for euclidean and non-euclidean instances. In: Ranka S, Banerjee A, Biswas KK, Dua S, Mishra P, Moona R, Poon S, Wang C (eds) Contemporary Computing: Third International Conference, IC3 2010, Noida, India, August 9–11, 2010. Proceedings, Part I, Springer, Communications in Computer and Information Science, vol 94, pp 229–240

Saxena R, Singh A (2009) Solving bounded diameter minimum spanning tree problem with improved heuristics. ADCOM 2009:90–95

Singh A (2009) An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. Appl Soft Comput 9(2):625–631

Singh A, Gupta AK (2007) Improved heuristics for the bounded-diameter minimum spanning tree problem. Soft Comput 11(10):911–921

Steitz W (2015) New heuristic approaches for the bounded-diameter minimum spanning tree problem. INFORMS J Comput 27(1):151–163

Sundar S, Singh A (2010) A swarm intelligence approach to the quadratic minimum spanning tree problem. Inf Sci 180(17):3182–3191

Sundar S, Singh A (2012) A swarm intelligence approach to the early/tardy scheduling problem. Swarm Evolut Comput 4:25–32

Torkestani JA (2012) An adaptive heuristic to the bounded-diameter minimum spanning tree problem. Soft Comput 16(11):1977–1988

Vuppuluri PP, Patvardhan C (2021) Serial and parallel memetic algorithms for the bounded diameter minimum spanning tree problem. Expert Syst J Knowl Eng. https://doi.org/10.1111/exsy.12610

Wang S, Lang SD (1994) A tree-based distributed algorithm for the k-entry critical section problem. In: Proceedings of the 1994 International Conference on Parallel and Distributed Systems, IEEE, pp 592–597