Soft Computing (2024) 28:8497–8511
https://doi.org/10.1007/s00500-023-09509-7

APPLICATION OF SOFT COMPUTING

An artificial bee colony algorithm for the minimum edge-dilation *K*-center problem

Manisha Israni[1] · **Shyam Sundar**[1]

Accepted: 22 November 2023 / Published online: 11 July 2024

Abstract
This paper studies the minimum edge-dilation K-center (MEDKC) problem for edge-weighted, undirected and connected graphs. This problem which is $\mathcal{NP}$-hard holds significant relevance in designing efficient routing schemes for computer networks. To the best of our knowledge, there exists only two variants of genetic algorithm that have been developed for this problem. In this paper, we propose an artificial bee colony algorithm (ABC_MEDKC) for this problem. The proposed ABC_MEDKC incorporates two adaptable neighborhood operators specifically tailored for this problem in which the first neighborhood operator utilizes solution components of another solution, and the second neighborhood operator follows swapping of center vertices with non-center vertices in a mixed strategies of greedy and random approach. On available benchmark instances, computational results of ABC_MEDKC indicate that ABC_MEDKC overall outperforms the existing two variants of genetic algorithm in both solution quality and computational time. ABC_MEDKC also outperforms an existing polynomial-time approximation algorithm developed for this problem in terms of solution quality. ABC_MEDKC finds new values for 17 instances out of 91 instances. In addition, the convergence behavior of ABC_MEDKC and the statistical analysis are also studied.

Keywords Edge-dilation K-center problem · Network · Artificial bee colony algorithm · Neighborhood operators

1 Introduction

The MEDKC problem, proven to be a $\mathcal{NP}$-hard (Könemann et al. 2004), can be outlined as follows: In a given edge-weighted, undirected and connected graph $G = (V, E, w)$—with V denoting the vertex-set and E representing the edge-set, and w as the edge-weight function—let $\Pi \subset V$ refer to a set of K center vertices and every other vertex $v \in V$ is allotted to only one vertex $\pi_v \in \Pi$, where $K > 0$ is a parameter. The objective of the MEDKC problem is to identify a set Π of K center vertices and assign each remaining vertex to a unique vertex in Π in a manner that minimizes the stretch of the solution $(\Pi, \pi_{v\,v\in V})$. The stretch of a solution $(\Pi, \{\pi_v\}_{v\in V}) = \max_{(u,v)\in V\times V}\{\frac{d_\pi(u,v)}{d_w(u,v)}\}$ is described as the maximum stretch among all pairs of vertices in G.

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

Manisha Israni
misrani.phd2018.mca@nitrr.ac.in

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

The stretch for a pair of vertices $(u, v \in V)$ is the ratio of the center distance $(d_\pi(u, v))$ to the shortest distance $(d_w(u, v))$ in G, represented as $\frac{d_\pi(u,v)}{d_w(u,v)}$, where $d_\pi(u, v)$ with respect to Π is defined as

$$d_\pi(u, v) = d_w(u, \pi_u) + d_w(\pi_u, \pi_y) + d_w(\pi_y, v). \quad (1)$$

Here, $d_w(u, v)$ signifies the shortest path between u and v.

The optimal solution for a given G is denoted as $Opt(G) = \min(\Pi, \{\pi_v\}_{v\in V})$ over all possible subsets Π and assignment $\{\pi_v\}_{v\in V}$.

In the literature, the MEDKC problem exhibits connections to various problems (Garcia-Diaz et al. 2019). Some of these related problems include:

- p-Center problem aims to seek a set of p vertices $\in V$, such that the maximum distance from any vertex to its closest center is as minimum as possible (Davidović et al. 2011; Garcia-Diaz et al. 2019).
- The capacitated K-center problem allows each center to attend only a certain number of vertices (Khuller and Sussmann 2000)

- The fault tolerant K-center problem in which each selected center must belong to a set of $\alpha \leq K$ centers close to it (Khuller et al. 2000).
- The p-next center problem that aims to minimize the total sum of the distance from the farthest vertex to its nearest center and the distance between this center to its nearest alternative center (López-Sánchez et al. 2019).
- The mixed K-center problem in which q centers must be in the set of vertices, and the remaining vertices can be anywhere ($q < K$) (Xu et al. 2018).

Könemann et al. (2004) introduced the MEDKC problem, inspired by a practical application in routing schemes within computer networks. In such schemes, each source node has the capability to route messages to a destination node. A host (node) stores information about routing paths to every other host in its routing table. To ensure shortest-path routing for the entire network, each node requires $\mathcal{O}(n)$ entries in its routing table to retain information about the shortest-path routing to every other node.

Considering computer networks in modern times comprising of millions of nodes poses a challenge for each node to store information on shortest-path routing for other nodes because of memory limitations. Attempts have been made to develop an efficient routing scheme, prioritizing considerations for memory constraints. Researchers focus on two main objectives: minimize the size of routing tables and/or minimize the stretch of a routing scheme. Modern routing protocols such as OSPF (Moy 1998) enable the division of a network into areas, where each node retains information about paths within the same area. Routing between nodes in distinct areas is accomplished through a backbone network of area border routers connecting these areas. This results in a decrease in the routing table size for each node, with the anticipation that the overall network stretch remains acceptable.

The objective of the MEDKC problem is based on this principle that K center nodes in computer networks are exactly the area border routers, while each remaining node in the network is assigned to a router. The task is to construct a compact routing scheme that minimizes the maximal stretch of the entire network.

2 Literature review

The literature has seen numerous papers addressing compact routing schemes, with a primary emphasis on balancing the trade-off between the size of routing tables and the stretch. In the initial stages, (Peleg and Upfal 1989) pioneered compact routing schemes tailored for undirected graphs. Subsequently, a routing scheme (Awerbuch et al. 1990; Awerbuch and Peleg 1990) was proposed for weighted graphs. Thorup and Zwick (2001) introduced a labeled compact routing scheme with a stretch of $4K$-5 using a routing table of size $\mathcal{O}(\frac{1}{k})$-bit by each vertex. Chechik (2013) enhanced the outcome of Thorup and Zwick (2001). Eilam et al. (2003) employed the pivot interval routing (PIR) scheme, achieving a stretch factor of at most 5 and an average stretch factor of at most 3. Each vertex utilized a routing table of size $\mathcal{O}(\sqrt{n}\log^{3/2} n)$ bits. Cowen (2001) demonstrated that if $\mathcal{O}(n/2/3\log^{4/3} n)$ space is allowed at each vertex in a given weighted undirected network (G), a solution can be found where the routing path between any pair of vertices $((u, v \in V))$ is at most three times as long as the shortest u, v-path in G. Krioukov et al. (2007) established that using sub-linear sized routing tables, a stretch of less than 3 is achievable. Enachescu et al. (2008) applied Internet-like graphs and proposed a routing scheme with a stretch value of less than 3 with high probability while retaining $o(n)$ memory. Later, (Abraham et al. 2004) presented a routing scheme for a given weighted undirected network achieving a stretch of 3 using a $\mathcal{O}(\sqrt{n})$ space at each vertex. Roditty and Tov (2015) introduced a routing scheme that surpasses previous results achieving a stretch of 5+ϵ using $\mathcal{O}(\frac{1}{\epsilon}n^{1/3}\log D)$ memory at each node, the label of each node of $\mathcal{O}(\log n)$ size, and $\mathcal{O}(\frac{1}{\epsilon}\log D)$-bit headers.

Based on the literature discussed above, it is evident that there is currently no existing solution approach for the MEDKC problem applicable to practical scenarios. Moreover, the methods developed for creating a compact routing scheme cannot be directly employed, given the distinct characteristics of the two problems. Compact routing scheme problems are concerned with balancing the trade-off between stretch and the size of routing tables. In contrast, the MEDKC problem, where memory that is used to store distance information is not an issue, aims to find a set of K central nodes and to arrange the paths over them in a manner that minimizes the maximum stretch (Könemann et al. 2004).

Könemann et al. (2004) proved that the MEDKC problem is a $\mathcal{NP}$-hard problem. They also introduced a polynomial-time approximation algorithm (referred to as $Approx\_Kon$), which is at most 4.Opt + 3, where Opt represents the optimal stretch value. The study on the results for the MEDKC problem (Könemann et al. 2004) is of theoretical importance, as their proposed approximation algorithm provides only an upper bound for any solution. Given its status as an $\mathcal{NP}$-hard problem (Könemann et al. 2004), employing metaheuristic techniques becomes a favorable approach to obtain high-quality solutions within a reasonable computational timeframe. Readers are suggested to study a tutorial on metaheuristic techniques (Osaba et al. 2021). To the best of our knowledge, among metaheuristic techniques, only Matic et al. (2017) proposed two variants (GA1 and GA2) of genetic algorithm which are evolutionary algorithms. In both variants of GA, authors used binary coding and used modified

crossover and mutation operators, particularly for each GA variant, to maintain feasibility of solutions throughout the optimization process. Based on the literature survey, it is clear that the MEDKC problem is an under-studied problem in terms of metaheuristic techniques.

Swarm intelligence techniques have been attracting significant attention from researchers due to their capability to find high-quality solutions for numerous hard optimization problems. This capability arises from the collective behavior of decentralized and self-organized swarms. The foraging behavior of honey bees (swarms) is one of them, such as BeeHive (Wedde et al. 2004), bees algorithm (Pham et al. 2006), bee colony optimization algorithm (Lučić and Teodorović 2001), honeybee search algorithm (Olague and Puente 2006), and artificial bee colony (ABC) algorithm (Karaboga 2005). All of them came up with different concepts for designing algorithms. Readers can find a survey on the bees' behavior inspiring algorithms in (Karaboga and Akay 2009; Rajasekhar et al. 2017). Among them, ABC algorithm, introduced by (Karaboga 2005), has received world-wide researchers' attention for solving complex optimization problems across diverse domains (Karaboga et al. 2014; Singh 2009; Sundar et al. 2017; Singh and Sundar 2018a, b; Ghoshal and Sundar 2021, 2020a) since its inception. This motivated us to work on ABC algorithm for the MEDKC problem. In this paper, the proposed ABC algorithm uses two adaptable neighborhood operators tailored to the MEDKC problem. On available benchmark instances, computational results indicate that the proposed ABC algorithm overall performs superior to the existing two variants of genetic algorithm in both solution quality and computational time. The proposed ABC algorithm also exhibits superiority over an existing polynomial-time approximation algorithm ($Approx\_Kon$) in terms of solution quality.

The remainder of the paper is summarized as follows: Sect. 3 provides a concise introduction to the artificial bee colony algorithm; Sect. 4 discusses an ABC algorithm tailored for the MEDKC problem; Sect. 5 discusses the computational results of ABC algorithm in comparison to the existing approaches; and Sect. 6 discusses concluding remarks.

3 Artificial bee colony algorithm

Artificial Bee Colony (ABC) algorithm was developed by Karaboga (2005), inspired by the intelligent foraging behavior of honey bee swarms. Initially designed for optimizing numerical problems, ABC algorithm has shown its capability in tackling various optimization problems since its inception. Relevant studies include (Karaboga et al. 2014; Singh 2009; Sundar et al. 2017; Singh and Sundar 2018a, b; Ghoshal and Sundar 2020a, 2021). In ABC algorithm, artificial bees, as per their task, are classified into three groups—employed bees, onlooker bees, and scout bees—work in cooperation to find high-quality solutions within the search space. ABC algorithm presumes that each artificial employed bee is associated with a food source, representing a solution to the optimization problem at hand. Consequently, the number of employed bees in the colony equals the number of food sources. The nectar amount of a food source is associated with the solution quality (fitness) of its associated solution. ABC algorithm starts with a set of generated initial solutions (food sources), and the size of this set is referred to as the employed bee population. Subsequently, ABC algorithm iteratively (generation) progresses through three phases—employed bee phase, scout bee phase, and onlooker bee phase. ABC algorithm continues its iteration by moving toward better solutions (in terms of fitness) through neighborhood search mechanism while abandoning inferior solutions. The iteration process concludes when a termination criterion is met. In ABC algorithm, employed bees and onlooker bees exploit the search space by determining new neighboring solutions, while the scout bee explores the search space. One can find an in-depth understanding of the framework of ABC algorithm in Karaboga et al. (2014); Singh (2009).

4 ABC algorithm for the MEDKC problem

This section describes our proposed ABC algorithm for the MEDKC problem (referred to as ABC_MEDKC). In the beginning of ABC_MEDKC, precompute the shortest paths between all pairs of vertices of a given input graph (G).

The salient features of ABC_MEDKC are discussed in the following subsections:

4.1 Solution representation

Each feasible solution is denoted as a set of K vertices acting as centers. Vertices that are not included in the K center vertices of the solution are classified as non-center vertices.

4.2 Initial solution generation

Algorithm 1: Pseudo-code of generation of an initial solution E_i

```
E_i ← ∅;
U ← ∅;
U ← U ∪ {i}  ∀i ∈ V;
for i ← 1 to K do
    Select a random vertex r as a center vertex from U;
    E_i ← E_i ∪ {r};
    U ← U \ {r};
```

Each initial solution, denoted as E_i, within the employed bee population, is randomly generated, where the size of the employed bee population is E_{pop}. The initial solution E_i, designed to contain K center vertices, begins as an empty set. Initially, all vertices are assigned to a set called U. In the subsequent steps, the procedure, at each iteration, randomly selects a vertex (denoted as r) from U and assigns it as the center vertex for E_i. The set U is then updated by removing the selected vertex r. This iterative process continues until K vertices are assigned as center vertices to E_i. Vertices not included in E_i remain in set U and are considered non-center vertices. Algorithm 1 provides the pseudo-code for the procedure outlining the generation of the initial solution E_i.

4.3 Fitness computation

Upon the completion of constructing a feasible solution (denoted as S), the fitness, represented by the stretch of S, is computed by determining the maximum stretch among all pairs of vertices in G. This is defined as follows:

$$\max_{(u,v)\in V\times V}\left\{\frac{d_\pi(u,v)}{d_w(u,v)}\right\}.$$

Readers are suggested to refer to the introduction section (Sect. 1) for more details of the stretch of a feasible solution in the context of the MEDKC problem.

4.4 Probability of selecting a solution

In ABC_MEDKC, each onlooker bee uses a binary tournament selection method to select a solution from the employed bee population. This selection mechanism involves randomly selecting two different solutions from the employed bee population. The preferred solution, based on the fitness defined in Sect. 4.3, is chosen with a probability P_{bt}, while the less favorable one is chosen with a probability of $1 - P_{bt}$. This method prioritizes better solutions over inferior ones, encouraging more onlooker bees to opt for higher quality solutions. Additionally, this selection method is employed to choose a distinct solution in the first neighborhood operator (refer to Sect. 4.5.1) when determining a new neighboring solution.

4.5 Neighborhood operators

Neighborhood operators in ABC algorithm play a significant role in finding high-quality solutions for $\mathcal{NP}$-hard combinatorial optimization problems (Sundar et al. 2017; Singh and Sundar 2018a; Pan et al. 2011; Singh 2009; Ghoshal and Sundar 2020b). For the MEDKC problem, we apply two neighborhood operators in a mutually exclusive way to determine a new neighboring solution (say E_i') within the vicinity of the current employed bee solution (say E_i) of employed bee population. The first neighborhood operator (*Nbr_cv*) is based on this fact that if a center vertex is present in a solution with high fitness, the chance of that center vertex being present in many other solutions of the employed bee population is high, whereas the second neighborhood (*Nbr_swp*) is based on the swapping of center vert(ex/ices) with non-center vert(ex/ices) in a mixed strategies of greedy and random approach. The details of each neighborhood are outlined as follows.

4.5.1 First neighborhood operator (Nbr_cv)

The first neighborhood operator (*Nbr_cv*) first creates a copy (say E_i') of the current employed bee solution (say E_i), and then, it selects a solution (say E_j), different from E_i, using binary tournament selection method (see Sect. 4.4 in detail). A check is performed whether E_i and E_j are identical. If both are different, then a certain number (say *nocv_1*) of center vertices of E_j are selected randomly, where *nocv_1* = max(1, (int)(Para1 $\times$ cnt)). Here, *cnt* in E_j is the number of those center vertices which are different from the center vertices in E_i. Para1 is a parameter whose value is determined empirically. All these selected center vertices from E_j are assigned to E_i', making solution E_i' infeasible. To make E_i' feasible, the same number (*nocv*) of center vertices which are originally part of E_i is removed from E_i'. This way makes E_i' feasible, resulting in a new feasible neighboring solution E_i'.

It is important to emphasize that if both E_i and E_j are identical, the continuation of the first neighborhood operator (Nbr_cv) is halted, a scenario referred to as a collision (Singh 2009). Such duplication of two solutions within the employed bee population poses an obstacle to effectively searching for high-quality solutions in the search space. To handle this issue, the employed bee associated with E_i is designated as a scout. The scout bee then explores and identifies a new solution (refer to Sect. 4.6). This approach aids in eliminating a duplicated solution from the employed bee population, promoting greater diversity within the population. However, in the event of a collision during the onlooker bee phase, the first neighborhood operator (Nbr_cv) is similarly discontinued for the associated solution, and a very large fitness value is assigned to that particular solution associated by its onlooker bee.

4.5.2 Second neighborhood operator (Nbr_swp)

The second neighborhood operator (*Nbr_swp*) is based on the swapping of center vert(ex/ices) with non-center vert(ex/ices) in a mixed strategies of greedy and random approach. *Nbr_swp* initiates with creating a copy (say E_i') of the current employed solution E_i. Subsequently, *Nbr_swp* randomly selects a certain number (say *nocv_2*) of cen-

ter vertices from E_j, where $nocv\_2$ = max(1, (int)(Para2 $\times K$)). Here, K is the number of center vertices. *Para2* is a parameter whose value is to be determined empirically. Each selected random center vertex (say v_c) of E'_i is replaced with a candidate non-center vertex (v_{nc}) of E'_i in either two ways. With probability (Para3), *Nbr_swp* applies a greedy approach in which a candidate non-center vertex is selected from a subset of total number of non-center vertices of E'_i (say $no\_ncv = (int)(val \times (V\text{-}K))$, where val is set to 0.1 based on our preliminary experiments to balance a trade-off between running time and solution quality) based on having minimum fitness. Otherwise, *Nbr_swp* applies a random approach wherein a candidate non-center vertex is selected randomly from the total number of non-center vertices of E'_i with probability (1-$Para3$).

4.6 Scout bee phase

During the scout bee phase, if an employed bee solution ceases to show improvement for a specific number of generations (referred to as limit), the employed bee promptly abandons its non-improving solution, transitioning its status to that of a scout bee. The scout bee then seeks a new solution through a prescribed procedure (refer to Sect. 4.5.2). Once the process for finding a new solution concludes, the scout bee reverts to the state of an employed bee. The parameter limit to be determined empirically holds substantial importance in the ABC algorithm. In addition, a collision (as described in Sect. 4.5.1) can also prompt the transformation of an employed bee into a scout. It is crucial to note that the ABC algorithm does not impose an upper limit on the number of scouts in a single generation. A generation may witness multiple scouts if the conditions mentioned earlier (limit and collision conditions) are met; otherwise, no scouts are encountered.

Algorithm 2 outlines the pseudo-code of ABC_MEDKC in which the procedure $BTSM(E_1, E_2, \ldots, E_{E_{pop}})$ is a binary tournament selection method that provides an index of a solution in $< E_1, E_2, \ldots, E_{E_{pop}} >$; $Nbr\_cv(X_1, X_2)$ is the first neighborhood operator that provides a new neighboring solution (say X'); and $Nbr\_swp(X)$ is the second neighborhood operator that provides a new neighboring solution (say X').

It is important to highlight that the most time-consuming part of ABC_MEDKC is the computation of the fitness for each generated solution. This process involves computing shortest paths between all pairs of vertices in a given input $G = (V, E, w)$, incurring a time complexity of $\mathcal{O}(V^3)$. Additionally, it encompasses (i) the computation of center distance for each pair of vertices, with a complexity of $\mathcal{O}((V\text{-}K) \times K)$, and (ii) the computation of stretch for each pair of vertices, with a complexity of $\mathcal{O}((V\text{-}K)^2)$ (refer to the first two paragraphs of Sect. 1). To alleviate this computational load, the approach precomputes the shortest paths between all pairs of vertices in a given input $G = (V, E, w)$ once. Consequently, the fitness function procedure for each generated solution utilizes this precomputed information of shortest paths between all pairs of vertices, reducing the running time from $\mathcal{O}(V^3)$ to $\mathcal{O}((V\text{-}K)^2)$. Another time-consuming part of ABC_MEDKC involves a greedy approach applied by Nbr_swp with probability ($Para3$). In this procedure, a selected random center vertex is tried for swap with a subset of total number of non-center vertices (no_ncv) one-by-one, and the swap that results in the minimum fitness is chosen. The running time for this greedy approach is $\mathcal{O}(no\_ncv \times (V\text{-}K)^2)$, where $\mathcal{O}((V\text{-}K)^2)$ is the running time for computing the fitness of a newly generated solution after a swap.

Algorithm 2: The pseudo-code of ABC_MEDKC

```
Generate initial solutions < E_1, E_2, ..., E_{E_pop} >;
Best ← best solution in < E_1, E_2, ..., E_{E_pop} >;
while Termination criterion is not reached do
    for i ← 1 to E_pop do
        if u01 < P_nbr then
            In_i ← BTSM(E_1, E_2, ..., E_{E_pop});
            if E_{In_i} is identical to E_i then
                goto Jump_1;
            E'_i ← Nbr_cv(E_i, E_{In_i});
        else
            E'_i ← Nbr_swp(E_i);
        if F(E'_i) < F(E_i) then
            E_i ← E'_i;
            if F(Best) > F(E_i) then
                Best ← E_i;
        else if There is no improvement in E_i for a certain
        number of generations (i.e., limit) then
            Jump_1;
            Scout bee phase;
            if F(Best) > F(E_i) then
                Best ← E_i;
    for i ← 1 to On_pop do
        ind_i ← BTSM(E_1, E_2, ..., E_{E_pop});
        if u01 < P_nbr then
            In_j ← BTSM(E_1, E_2, ..., E_{E_pop});
            if E_{In_j} is identical to E_{ind_i} then
                F(O_i) ← ∞;
            else
                O_i ← Nbr_cv(E_{ind_i}, E_{In_j});
        else
            O_i ← Nbr_swp(E_{ind_i});
    for i ← 1 to On_pop do
        if F(O_i) < F(E_{ind_i}) then
            E_{ind_i} ← O_i;
        if F(Best) > F(E_{ind_i}) then
            Best ← E_{ind_i};
```

5 Computational results

C language is used to code ABC_MEDKC, and experiments are conducted on a Linux system equipped with an Intel Core *i*5-4570 CPU @ 3.2 GHz × 4 processor and 4 GB of RAM using a single thread. Similar to Matic et al. (2017), the performance of ABC_MEDKC is tested on the same set of benchmark instances. ABC_MEDKC is allowed to execute 20 independent runs for each instance. The termination criterion for ABC_MEDKC allows to execute for either at least 150 generations or until the best-so-far obtained solution does not improve after ($K \times 30$) generations.

In the subsequent subsections, we discuss on instances, parameter settings, and computational results of ABC_MEDKC in comparison to existing approaches, including a polynomial-time approximation algorithm ($Approx\_Kon$) (Könemann et al. 2004) and two variants of genetic algorithms, namely GA1 and GA2 (Matic et al. 2017).

5.1 Instances

Similar to GA1 and GA2 (Matic et al. 2017), we have also used the same two set of instances available in the public domain. The first set pertains to the BX instances, comprising 7 graphs that are 4-regular and possess varying vertex sizes from the set {25, 50, 75, 100, 200, 400, 1000} (Blesa and Xhafa 2000). The second set corresponds to the BB instances, including 7 graphs: 5 edge-weighted graphs of grid with sizes ranging from $< 15 \times 15, 33 \times 33, 45 \times 5, 50 \times 50$ and $100 \times 10 >$, and two large Steiner tree benchmark instances, namely steinc5 and steind5, each having 500 vertices and 1000 vertices, respectively (Blum and Blesa 2005). Each instance is examined for different values of K with in the set $\in \{2, 3, 4, 5, 10, 15, 20\}$. It is important to note that for K=1, the MEDKC problem is not $\mathcal{NP}$-hard (Matic et al. 2017).

5.2 Parameter settings

Given its stochastic nature, ABC_MEDKC encompasses a set of parameters, and the configuration of each parameter is critical for its effectiveness. While tuning these parameters can be challenging, it is feasible to approximate the setting of each parameter value in such a way that it helps ABC_MEDKC in finding solutions of high quality. To set the value of each parameter used in ABC_MEDKC, ABC_MEDKC was examined on ten instances, i.e., bb15×15_1 (k = 20), bb15×15_1 (k = 4), g25-4-01 (k = 10), g50-4-01 (k = 3), g100-4-01 (k = 15), g200-4-01 (k = 15), bb45×5_1 (k = 20), bb15×15_1 (k = 15), g50-4-01 (k = 15), and g50-4-01 (k = 10). Drawing from existing literature and our prior experience with ABC algorithm, we considered a range of potential values for each parameter (refer to Table 1). Initial experiments indicated that ABC_MEDKC exhibited the best overall performance across the ten instances under consideration when $E_{pop} = 10$, $On_{pop} = 30$, $limit = 25$, $P_{bt} = 0.85$, $P\_nbr = 0.7$, $Para1 = 0.1$, $Para2 = 0.1$, and $Para3 = 0.5$. One can observe the best potential value (highlighted in bold) of each parameter and its solution quality obtained (highlighted in bold) in Table 1.

5.3 Comparison of results of ABC_MEDKC against state-of-the-art approaches

This section presents a comparative analysis of the results achieved by ABC_MEDKC against the results of an existing polynomial-time approximation algorithm ($Approx\_Kon$) (Könemann et al. 2004) and state-of-the-art approaches (GA1 and GA2 (Matic et al. 2017)) on the BX and BB instances.

Tables 2 and 3 presents the results of $Approx\_Kon$, GA1, GA2 and ABC_MEDKC on the 49 BX instances and 42 BB instances respectively. In these Tables 2 and 3, $Instance$ column indicates the name of the instance; $|V|$ column represents the number of vertices in G; the column $Value$ represents the value of solution returned by $Approx\_Kon$; the next two three columns, i.e., $Best$, Avg, $ATTB$, and $ATET$, respectively, represent the best-so-far value, average value, average time to reach the best solution (in seconds), and average total execution time (in seconds) over 20 runs obtained by GA1 and GA2; the last five columns, i.e., $Best$, Avg, SD, $ATTB$, and $ATET$, respectively, represent the best-so-far value, average value, standard deviation, average time to reach the best solution (in seconds), and average total execution time over 20 runs obtained by ABC_MEDKC. The best Avg results are highlighted in bold in both Tables 2 and 3. The code for the polynomial-time approximation algorithm ($Approx\_Kon$) (Könemann et al. 2004) for the MEDKC problem is available to us; we executed this code on considered instances using the same computer configuration used for ABC_MEDKC and reported the results of $Approx\_Kon$ in Tables 2 and 3.

Table 2 presents the results of 49 BX weighted instances. In comparison with GA1, ABC_MEDKC, in terms of $Best$, is better on 10, same on 26 and worse on 13; ABC_MEDKC, in terms of Avg, is better on 35, same on 9 and worse on 5. comparing with GA2, ABC_MEDKC, in terms of $Best$, is better on 27, same on 16 and worse on 6; ABC_MEDKC, in terms of Avg, is better on 45, same on 2 and worse on 2. The results in Table 2 indicate that ABC_MEDKC, in terms of Avg, demonstrates higher robustness compared to GA1 and GA2. In comparison to $Approx\_Kon$, ABC_MEDKC outperforms in both $Best$ and Avg on all 49 BX weighted instances.

Table 3 reports the results of 42 BB weighted instances. The results reveal that ABC_MEDKC dominates GA1 and GA2, not only in terms of solution quality, but also in terms of

Table 1 Influence of parameter setting on the solution quality

Parameter	Value	bb15x15_1 (k=20)		bb15x15_1 (k=4)		g25-4-01 (k=10)		g50-4-01 (k=3)		g100-4-01 (k=15)		g200-4-01 (k=15)		bb45x5_1 (k=20)		bb15x15_1 (k=15)		g50-4-01 (k=15)		g50-4-01 (k=10)	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
E_{pop}	5	18.33	**19.11**	13400	139	2.16	2.32	11.73	11.79	7.38	7.7	10.69	10.99	17.67	18.25	24.71	25.59	4.32	4.35	5.46	5.48
	10	**18.33**	19.71	**134.00**	138	**2.16**	**2.16**	**11.73**	**11.73**	**7.38**	**7.61**	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	20	19	20.14	134	135.43	2.16	2.17	11.73	11.73	7.38	7.68	10.92	11.24	18	19.01	24.71	25.68	4.32	4.32	5.46	5.47
	30	19.5	20.41	134	**134.12**	2.16	2.16	11.73	11.73	7.38	7.76	11	11.27	18.33	19.44	24.71	26.11	4.32	4.32	5.46	5.46
On_{pop}	10	19.5	20.82	134	140	2.16	2.19	11.73	11.77	7.38	7.88	10.69	11.53	19	20.48	25	27.14	4.32	4.4	5.46	5.57
	20	18.33	20.12	134	141.02	2.16	2.18	11.73	11.78	7.38	7.73	10.79	11.14	18	19.52	24.71	25.73	4.32	4.35	5.46	5.5
	30	**18.33**	**19.71**	**134.00**	**134.88**	**2.16**	**2.16**	11.73	**11.73**	**7.38**	**7.61**	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
limit	10	18.33	20.51	134	**135.68**	2.16	2.17	**11.73**	11.73	7.38	7.75	11	11.15	17.67	19.93	24.71	26.36	4.32	4.33	5.46	5.48
	20	19.5	20.82	134	140.3	2.16	2.17	11.73	11.74	7.38	7.68	10.62	11.08	17.67	18.86	24.71	25.44	4.32	4.34	5.46	5.51
	25	**18.33**	19.71	**134.00**	138.88	**2.16**	**2.16**	11.73	**11.73**	**7.38**	**7.61**	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	50	18.33	**19.27**	134	139.6	2.16	2.21	**11.73**	11.73	7.38	7.62	10.69	10.96	17.67	18.52	24.71	25.41	4.32	4.34	5.46	5.47
P_{bt}	0.8	18.33	**19.27**	134	139.6	2.16	2.17	11.73	11.73	7.38	**7.57**	10.69	11.03	17.67	18.47	24.71	25.16	4.32	4.33	5.46	5.46
	0.85	**18.33**	19.71	**134.00**	**138.88**	**2.16**	**2.16**	11.73	**11.73**	**7.38**	7.61	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	0.9	18.33	19.88	134	139.6	2.16	2.18	**11.73**	11.83	7.38	7.63	10.69	11.03	18	18.73	24.71	25.87	4.32	4.33	5.46	5.46
P_nbr	0.6	18.33	**19.50**	134	**135.40**	2.16	2.18	11.73	11.73	7.38	**7.56**	10.69	**10.95**	17.67	18.62	24.71	25.23	4.32	4.33	5.46	5.46
	0.7	**18.33**	19.71	**134.00**	138.88	**2.16**	**2.16**	11.73	**11.73**	**7.38**	7.61	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	0.8	18.33	19.88	134	141.27	2.16	2.22	**11.73**	11.74	7.38	7.7	10.62	11.09	17.67	18.62	24.71	25.87	4.32	4.33	5.46	5.49
*Para*1	0.05	18.57	**19.68**	134	138.88	2.16	2.16	11.73	11.74	7.38	7.61	10.62	11.01	17.67	18.55	24.71	25.16	4.32	4.32	5.46	5.46
	0.1	**18.33**	19.71	**134.00**	**138.88**	**2.16**	**2.16**	11.73	**11.73**	**7.38**	**7.61**	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	0.2	18.33	19.79	134	138.88	2.16	2.2	**11.73**	11.73	7.38	7.75	10.62	11	17.67	19.17	24.71	25.18	4.32	4.34	5.46	5.46
*Para*2	0.05	18.33	**18.84**	134	138.88	2.16	2.16	11.73	11.73	7.38	7.61	10.62	11.01	17.67	**17.95**	24.71	25.16	4.32	4.32	5.46	5.46
	0.1	**18.33**	19.71	**134.00**	**138.88**	**2.16**	**2.16**	11.73	**11.73**	**7.38**	**7.61**	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	0.2	20.00	20.86	134	141.27	2.16	2.18	**11.73**	11.73	7.5	7.93	11.18	12.3	18.08	20.11	24.71	25.87	4.32	4.37	5.46	5.51
*Para*3	0.4	18.33	19.71	134	138.88	2.16	2.16	11.73	11.73	7.38	7.61	10.62	11.01	17.67	18.54	24.71	25.16	4.32	4.32	5.46	5.46
	0.5	**18.33**	**19.71**	**134.00**	**138.88**	**2.16**	**2.16**	11.73	**11.73**	**7.38**	**7.61**	**10.62**	11.01	**17.67**	18.54	**24.71**	**25.16**	**4.32**	**4.32**	**5.46**	**5.46**
	0.6	18.33	19.71	134	138.88	2.16	2.16	**11.73**	11.73	7.38	7.61	10.62	11.01	17.67	18.54	24.71	25.16	4.32	4.32	5.46	5.46

computational time. Comparing with GA1, ABC_MEDKC, in terms of $Best$, is better on 8 and is same on 34; ABC_MEDKC, in terms of Avg, is better on 33, same on 8 and worse on 1. Comparing with GA2, ABC_MEDKC, in terms of $Best$, is better on 34 and is same on 8; ABC_MEDKC, in terms of Avg, is better on 41 and is same on 1. In comparison to $Approx\_Kon$, ABC_MEDKC outperforms in both $Best$ and Avg on all 42 BB weighted instances.

In terms of computational time (ATET), ABC_MEDKC is much faster than GA1 and GA2 on 49 BX weighted instances except one instance (g1000-1-01 ($K = 20$)) on GA1 reported in Table 2; however, ABC_MEDKC is much better than GA1 in terms of Avg on this instance (g1000-1-01 ($K = 20$)). Also, ABC_MEDKC is much faster than GA1 and GA2 on 42 BB weighted instances except one instance (steind5 ($K = 20$)) reported in Table 3; however, ABC_MEDKC is much better than GA1 in terms of Avg, and is much better than GA2 in terms of $Best$ and Avg on this instance (steind5 ($K = 20$)). It is important to note that the execution environment for GA1 and GA2 in (Matic et al. 2017) was a PC with a 3.40 GHz Intel Core $i7$-4770 processor and 8 GB RAM, while ABC_MEDKC is executed on a PC with a 3.2 GHz $\times$ 4 Intel Core $i5$ processor and 4 GB RAM.

In summary, the results of ABC_MEDKC on 49 BX weighted instances and 42 BB weighted instances highlight its superior robustness compared to GA1 and GA2, in both solution quality (Avg) and computational time ($ATET$). Additionally, ABC_MEDKC identifies new $Best$ values in 17 out of 91 instances, as indicated by the instances highlighted in bold in Tables 2 and 3.

5.4 Convergence behavior of ABC_MEDKC

This subsection discusses the convergence behavior of ABC_MEDKC. To carry out this experimentation, three instances $<$ g400_4_01 ($k = 10$), bb33$\times$33_1 ($k = 15$), steinc5 ($k = 10$) $>$ are considered. Figure 1a–c illustrates the evolution of solution quality (Avg) against the successive average number of generations. The X-axis represents the “Average Number of Generations" over 20 runs, while the Y-axis represents the “Average Solution Quality" over 20 runs. It is noteworthy that we opted for successive average number of generations instead of average total execution time (ATET) or the termination criterion of ABC_MEDKC, as the total number of generations executed within a given termination criterion of ABC_MEDKC can be easily computed. The curves in Fig. 1a–c depict a continuous decrease in the fitness of the solution obtained by ABC_MEDKC with the increase in generations.

5.5 Impact of neighborhood operators (*Nbr_cv* and *Nbr_swp*)

Before considering the use of both neighborhood operators (*Nbr_cv* and *Nbr_swp*) in a mutually exclusive way in ABC_MEDKC, a preliminary experiment was conducted. Two variants, namely, ABC_Nbr_cv and ABC_Nbr_swp, were created by disabling *Nbr_swp* and *Nbr_cv*, respectively, in ABC_MEDKC. These variants were executed on some benchmark instances using a setup similar to ABC_MEDKC. The results, including the average solution quality (Avg) and average total execution time ($ATET$) of ABC_Nbr_cv, ABC_Nbr_swp, and ABC_MEDKC, are reported in Table 4. The findings indicate that ABC_MEDKC, utilizing both neighborhood operators (*Nbr_cv* and *Nbr_swp*) in a mutually exclusive way, achieves better average solution quality (highlighted in bold) on most of the considered instances while requiring less computational time. It is noteworthy that the *Nbr_swp* operator is computationally more expensive compared to *Nbr_cv*. The use of both operators (*Nbr_cv* and *Nbr_swp*) in a mutually exclusive way in ABC_MEDKC demonstrates its effectiveness in obtaining high-quality solutions within a reasonable computational time.

5.6 Statistical analysis

For the statistical analysis, a two-tailed nonparametric Wilcoxon’s signed-rank test was conducted on the best ($Best$) and average solution quality (Avg) values across 91 benchmark instances (49 BX instances and 42 BB instances, as reported in Tables 2 and 3) using Wilcoxon’s Signed-Rank test calculator (Wilcoxon 1945). The significance criterion was set at 0.05. The results of Wilcoxon’s signed-rank test, reported in Tables 5 and 6, show that ABC_MEDKC finds significant difference with GA1 and GA2.

6 Conclusion

This paper presents an artificial bee colony algorithm, denoted as ABC_MEDKC, for the minimum edge-dilation K-Center (MEDKC) problem on a connected, undirected and edge-weighted graph. ABC_MEDKC incorporates two adaptable neighborhood operators specifically tailored for this problem. The synergistic coordination of all components within ABC_MEDKC contributes to achieving solutions of high quality for the MEDKC problem. To evaluate its performance, ABC_MEDKC is compared with state-of-the-art approaches, namely two variants (GA1 and GA2) of the genetic algorithm (Matic et al. 2017), using available benchmark instances. Computational results show that ABC_MEDKC dominates the existing two variants of genetic algorithm in both solution quality and computational

Table 2 Results of $Approx\_Kon$, GA1, GA2, and ABC_MEDKC on BX instances

Instance	$\|V\|$	K	$Approx\_Kon$	GA1				GA2				ABC_MEDKC				
			Value	Best	Avg	ATTB	ATET	Best	Avg	ATTB	ATET	Best	Avg	SD	ATTB	ATET
g25-4-01	25	2	20.600000	10.333333	**10.333333**	0.033	0.607	10.333333	**10.333333**	0.006	0.676	10.33	**10.33**	0	0.00	0.01
		3	8.344828	8	**8**	0.022	0.773	8	**8**	0.011	0.614	8	**8**	0	0.00	0.01
		4	7.166667	5.4	**5.4**	0.216	1	5.4	5.435135	0.056	0.746	5.96	5.96	0	0.00	0.01
		5	5.965517	5	**5.114479**	0.198	1.112	5.142857	5.196571	0.04	0.747	5.14	5.14	0	0.00	0.01
		10	4.916667	2.6	2.651429	0.684	2.023	2.6	2.821022	0.406	1.713	**2.16**	**2.16**	0	0.00	0.02
		15	4.916667	2.111111	2.111111	0.42	2.341	2.111111	2.117111	0.556	2.746	**1.74**	**1.74**	0	0.00	0.01
		20	4.916667	1.823529	1.823529	0.136	2.289	1.823529	1.828186	0.192	3.377	**1.13**	**1.13**	0	0.00	0.02
g50-4-01	50	2	22.428571	16	16.7	0.285	3.683	16	**16.245238**	0.06	0.735	17	17	0	0.00	0.03
		3	14.428571	11.222222	11.95235	1.04	4.485	11.222222	11.911111	0.098	0.909	11.73	**11.73**	0	0.00	0.04
		4	14.428571	9.588235	10.623648	0.731	4.421	9.588235	10.73225	0.142	1.019	10.27	**10.29**	0.07	0.01	0.04
		5	14.428571	9	9.177745	1.105	4.672	9	9.618384	0.203	1.272	9	**9.05**	0.15	0.01	0.05
		10	8.708333	5	5.751302	2.267	6.623	5	6.118029	1.073	4.099	5.46	**5.46**	0	0.03	0.11
		15	7.166667	4.234043	4.55421	2.623	9.184	4.234043	4.531934	2.468	7.547	4.32	**4.32**	0	0.04	0.15
		20	5.363636	3.195652	3.655856	3.774	11.288	3.328358	3.781173	3.824	10.986	3.29	**3.32**	0.04	0.11	0.3
g75-4-01	75	2	27.750000	**22.666667**	**22.666667**	0.303	4.149	22.666667	23.214583	0.156	1.287	22.67	**22.67**	0	0.01	0.09
		3	31.750000	17.75	**17.7625**	0.753	4.732	17.75	17.964167	0.137	1.837	17.8	17.8	0	0.01	0.09
		4	19.666667	16.2	**16.2025**	0.143	4.397	16.2	16.4075	0.122	2.101	16.2	**16.2**	0.01	0.02	0.1
		5	18.333333	11.666667	12.420449	0.722	4.257	11.666667	13.66903	0.643	2.951	11.67	**11.67**	0	0.04	0.12
		10	11.047619	7.666667	8.489212	2.507	7.629	8.304348	9.260038	1.086	6.075	7.67	**7.94**	0.5	0.09	0.27
		15	10.740741	5.466667	6.468417	4.575	11.043	5.466667	6.425334	3.212	10.324	5.47	**5.69**	0.1	0.17	0.45
		20	9.000000	4.116279	4.786497	6.772	14.976	4.482143	4.969506	11.035	19.892	4.33	**4.59**	0.18	0.36	0.86

Table 2 continued

Instance	$\|V\|$	K	*Approx_Kon*	GA1				GA2				ABC_MEDKC				
			Value	Best	Avg	ATTB	ATET	Best	Avg	ATTB	ATET	Best	Avg	SD	ATTB	ATET
g100-4-01	100	2	33.333333	30.25	**30.25**	0.025	3	30.25	30.404167	0.008	1.562	30.25	**30.25**	0	0.00	0.17
		3	29.230769	24.333333	24.566666	0.792	4.879	24.555556	24.977778	0.019	2.348	24.33	**24.43**	0.24	0.04	0.18
		4	29.230769	13.333333	14.229982	0.644	4.946	14	17.560806	0.41	2.926	14	**14.04**	0.18	0.05	0.19
		5	15.500000	12.24	13.3365	0.745	5.44	13.333333	14.051923	0.458	3.508	12.24	**13.01**	0.5	0.04	0.2
		10	12.538462	8.9375	9.747968	3.383	10.062	9.307692	10.117313	1.573	9.162	8.94	**9.14**	0.15	0.22	0.57
		15	13.461538	7.375	8.114372	6.535	15.902	7.545455	8.262583	4.019	16.56	7.38	**7.61**	0.18	0.33	0.87
		20	8.526316	6.052632	7.159488	9.689	21.323	5.931818	7.09543	8.284	25.843	6.26	**6.47**	0.14	0.83	1.84
g200-4-01	200	2	78.000000	40.142857	**40.142857**	0.245	9.493	40.142857	41.125714	0.333	6.945	40.14	**40.14**	0	0.04	1.04
		3	31.800000	31.8	**31.8**	0.49	11.786	31.8	32.365714	0.283	9.298	31.8	**31.8**	0	0.05	1.04
		4	31.800000	27.111111	28.811111	1.039	13.348	29	29	1.393	11.477	27.11	**27.96**	0.94	0.27	1.13
		5	31.923077	22.454545	23.662093	2.353	15.649	23	26.813889	4.445	15.804	22.6	**22.62**	0.09	0.31	1.14
		10	27.142857	16.6	17.532976	5.991	25.666	17	18.521628	12.229	32.072	**15.62**	**16.22**	0.42	1.78	3.8
		15	20.000000	11	12.805148	17.528	40.44	11.181818	14.953014	11.752	40.52	**10.62**	**11.01**	0.17	2.18	5.06
		20	13.461538	10.393939	11.074202	19.736	48.763	10.685714	11.22531	18.098	56.671	**10.09**	**10.41**	0.18	5.45	11.63
g400-4-01	400	2	441.000000	93	**93**	1.648	36.537	93	157.1	5.895	31.497	93	**93**	0	0.34	7.3
		3	91.000000	71.5	**71.5**	5.497	47.988	72	82.908333	8.562	41.13	71.5	**71.5**	0	0.93	7.17
		4	91.000000	55	55.375	8.245	52.675	55.666667	66.233333	7.844	43.916	55	**55**	0	1.77	7.24
		5	91.000000	51.4	54.64	10.132	60.328	55	65.275	6.457	46.797	51.4	**53.56**	1.76	1.64	7.56
		10	54.000000	29	32.121905	32.156	95.382	31.5	38.248571	22.096	70.535	**25.86**	**29.48**	1.11	12.03	23.85
		15	47.444444	21	24.448889	41.624	119.684	24.6	29.68627	31.325	91.147	21.1	**22.34**	0.47	20.40	38.73
		20	25.400000	17.769231	20.512649	95.592	193.912	19.4	25.071429	30.828	105.145	17.8	**19.05**	0.65	41.22	85.04
g1000-4-01	1000	2	196.000000	138	138.2	38.195	467.519	143	159.9	29.203	224.296	138	**138**	0	13.94	102.3
		3	196.000000	114	114.4	33.021	483.743	122	142.05	40.486	248.211	114	**114**	0	10.26	100.7
		4	143.000000	90	94.05	121.116	558.249	107.666667	121.35	84.462	284.886	90	**90.45**	1.96	39.83	110.94
		5	143.000000	78	88.05	160.485	594.203	90	110.85	94.691	521.564	**76.33**	**80.48**	4.93	63.52	142.05
		10	118.750000	55	63.008333	371.548	891.103	70.5	79.366667	263.645	758.729	**54.33**	**57.01**	2.2	182.29	356.4
		15	102.000000	45.666667	55.4	330.472	908.486	57	65.8	241.235	814.247	**43.67**	**45.27**	1.81	358.84	690.12
		20	70.833333	38.5	46.197619	429.589	988.091	53	64.7	276.545	905.573	38.5	**40.5**	1.33	620.23	1395.73

Table 3 Results of $Approx\_Kon$, GA1, GA2, and ABC_MEDKC on BB weighted instances

Instance	$\|V\|$	K	$Approx\_Kon$	GA1				GA2				ABC_MEDKC				
			Value	Best	Avg	ATTB	ATET	Best	Avg	ATTB	ATET	Best	Avg	SD	ATTB	ATET
15 × 15	255	2	855.000000	403	**403**	0.226	12.207	423	423	0.019	6.137	403	**403**	0	0.05	1.22
		3	247.000000	157	159.4	1.636	15.148	167	176.05	0.217	9.865	157	**157**	0	0.15	1.23
		4	209.000000	134	142.4	2.071	15.952	148	152.35	0.517	11.676	134	**138.88**	6.64	0.39	1.31
		5	209.000000	98	100.508333	3.722	18.017	98	116.593333	7.432	21.248	98	**98**	0	0.38	1.29
		10	81.400000	50.555556	53.266667	7.223	28.85	51	57.5	10.516	29.562	50.56	**50.67**	0.19	0.74	2.31
		15	57.375000	25	31.906667	14.364	41.638	26.25	36.463373	11.321	36.433	**24.71**	**25.16**	0.56	2.85	5.18
		20	62.714286	20.666667	23.19	20.656	53.142	21	23.460833	22.303	52.267	**18.33**	**19.71**	0.75	6.76	12.47
33 × 33	1089	2	1863.000000	1087	**1088**	105.541	772.058	1087	1134.3	32.731	591.818	1087	1091.2	7.53	31.44	117.11
		3	1863.000000	707	**707**	159.031	910.835	707	743	83.901	673.265	707	**707**	0	23.2	116.78
		4	931.000000	609	613.7	363.148	1060.107	643	688.7	208.041	808.344	609	**611**	4	66.57	131.73
		5	795.000000	601	607	195.902	1012.91	595	628.4	148.951	821.428	595	**600.6**	4.13	44.34	129.09
		10	581.000000	319	368.35	650.386	1354.273	343	383.7	222.947	968.63	319	**327.1**	9.01	265.81	415.42
		15	423.000000	195	258.416667	902.792	1548.094	227	284.316667	383.558	1130.592	195	**211.9**	9.65	419.97	645.36
		20	281.000000	185	206.15	741.422	1519.992	185	215.15	348.991	1149.519	**148**	**168.1**	6.61	890.89	1470.59
45 × 5	225	2	963.000000	483	**483**	0.293	12.429	483	**483**	0.085	10.03	483	**483**	0	0.04	1.23
		3	337.000000	208.333333	**208.333333**	0.638	15.07	208.333333	246.6	2.285	14.231	208.33	**208.33**	0	0.12	1.22
		4	337.000000	183	**183**	1.549	16.77	183	198.533333	3.478	16.429	183	**183**	0	0.15	1.23
		5	337.000000	146.333333	150.9	3.222	18.221	146.333333	175.033333	1.976	17.122	146.33	**146.83**	2.18	0.62	1.49
		10	85.666667	59	64.926667	4.193	23.06	61	73.94	9.116	25.9	59	**59.1**	0.44	0.83	2.39
		15	49.000000	32	37.15	8.848	32.795	34	40.733333	14.371	40.808	32	**33.03**	1.25	2.02	4.33
		20	38.000000	19	25.608333	19.13	44.755	395	473.3	2260.595	6677.6	**17.67**	**18.54**	0.63	7.31	12.96

Table 3 continued

Instance	$\|V\|$	K	$Approx\_Kon$	GA1				GA2				ABC_MEDKC				
			Value	Best	Avg	ATTB	ATET	Best	Avg	ATTB	ATET	Best	Avg	SD	ATTB	ATET
100 × 10	1000	2	3037.000000	1689	**1689**	35.178	484.535	1689	1733	22.128	271.1	1689	**1689**	0	7.71	96.51
		3	1497.000000	1041	1041.1	167.014	623.227	1121	1146.3	30.088	351.567	1041	**1041**	0	22.38	90.32
		4	1399.000000	867	**869.1**	163.167	623.724	937	1041	59.017	366.924	867	**869.1**	5	39.58	96
		5	1293.000000	703	718.7	127.257	626.312	743	831.4	41.486	388.885	703	**703.6**	2.62	41.57	103.33
		10	633.000000	395	405.1	194.575	760.973	397	487.1	88.935	492.907	**375**	**391.9**	8.8	111.23	228.9
		15	527.000000	265	297.3	333.968	886.385	326	359	169.162	604.22	**259**	**269.35**	10.75	331.72	507.5
		20	347.000000	189	228.123333	532.296	1062.399	223	261.4	173.97	647.442	**183**	**194.6**	6.91	667.07	1120.4
steinc5	500	2	463.000000	257	257.7	10.194	75.483	261	278.6	5.967	54.861	257	**257**	0	0.52	12.4
		3	537.000000	203	204.6	13.131	82.895	221	251.1	14.068	72.982	203	**203**	0	2.26	12.47
		4	357.000000	179	182.85	8.911	83.491	186	218.85	5.656	68.237	179	**179.35**	1.53	4.82	12.61
		5	275.000000	169	170.55	24.268	103.021	176	200.45	24.627	93.12	169	**169**	0	4.71	13.18
		10	135.000000	88	96.75	38.433	138.411	99	129.275	48.709	136.838	88	**89.35**	3.21	13.42	29.58
		15	97.000000	52.428571	62.527381	88.057	199.472	61.5	97.975	56.565	161.846	52.43	**54**	2.54	43.43	66.93
		20	100.000000	47.666667	54.978571	88.144	231.482	49.571429	68.400794	71.507	200.101	47.67	**48.53**	0.75	67.43	127.67
steind5	1000	2	815.000000	473	**473**	27.734	485.418	509	538.8	50.692	395.576	473	**473**	0	4.7	90.23
		3	795.000000	457	457.1	50.096	551.608	459	477.5	33.335	477.755	457	**457**	0	16.53	90.75
		4	737.000000	367	378.8	144.643	652.206	435	449.4	72.868	504.82	367	**374.2**	11	28.1	95.95
		5	541.000000	349	351.6	75.588	606.527	367	413.8	82.437	530.674	349	**349.2**	0.87	22.47	92.58
		10	425.000000	197	237.4	341.623	901.829	271	316.9	205.511	688.88	197	**213**	12.76	161.52	280.12
		15	303.000000	139	162.7	630.996	1072.29	201	247.9	217.661	747.034	139	**146.85**	5.49	271.93	450.91
		20	201.000000	117	137.85	551.433	1123.258	145	204.6	319.596	854.022	117	**119.11**	3.27	705.07	1166.8

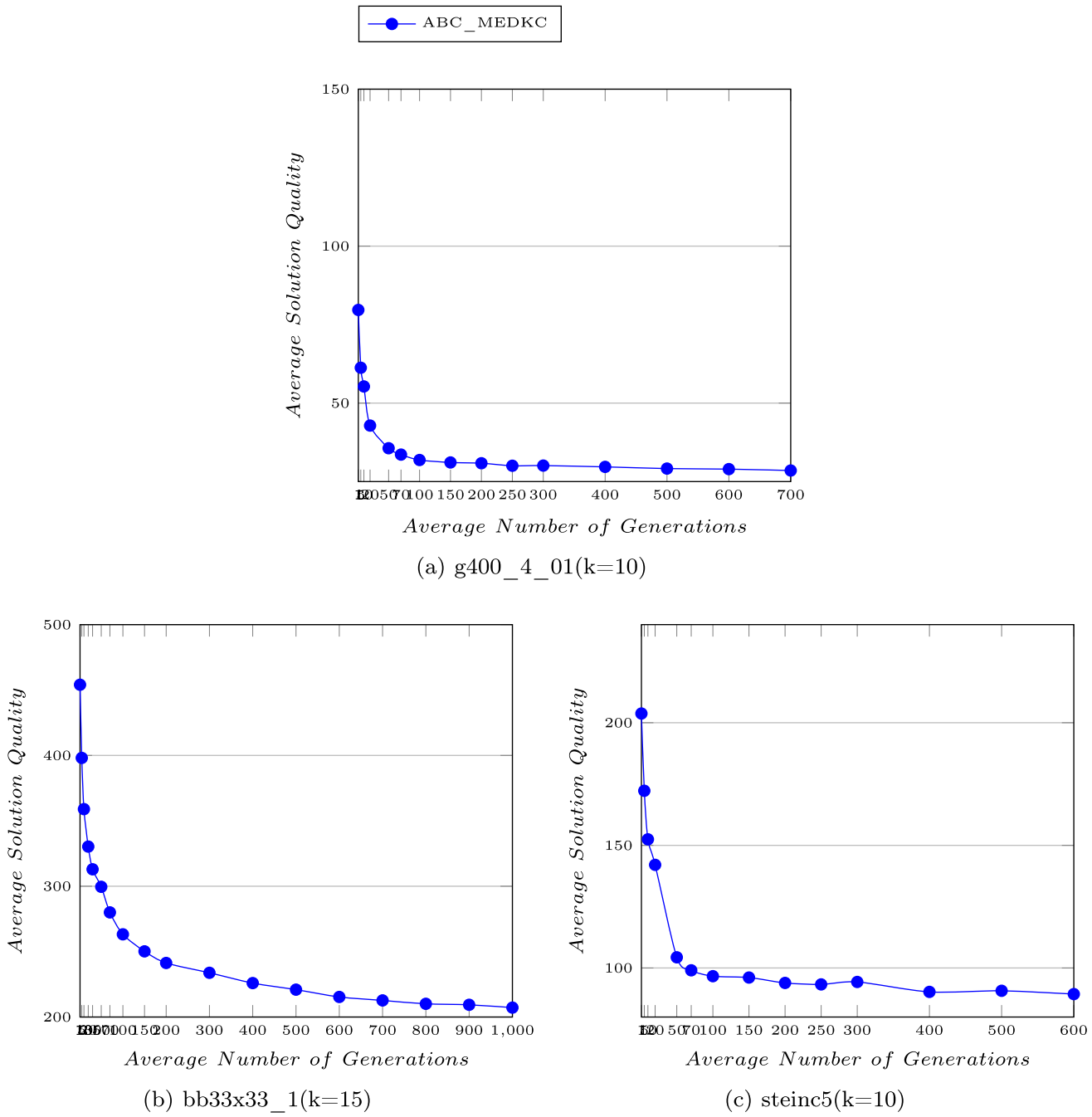

Fig. 1 Evolution of average solution quality over average number of generations

time. Notably, ABC_MEDKC exhibits greater robustness (in terms of average solution quality) compared to the genetic algorithm variants (GA1 and GA2). ABC_MEDKC also dominates an existing polynomial-time approximation algorithm ($Approx\_Kon$) in terms of finding solutions of high quality. Moreover, ABC_MEDKC finds new values for 17 instances out of 91 instances. The paper also explores the convergence behavior of ABC_MEDKC and conducts a statistical test, revealing a significant difference with GA1 and GA2.

Table 4 Results of ABC_Nbr_cv, ABC_Nbr_swp, and ABC_MEDKC on some benchmark instances

Instance	$\|V\|$	K	ABC_Nbr_cv		ABC_Nbr_swp		ABC_MEDKC	
			Avg	ATET	Avg	ATET	Avg	ATET
g25-4-01	25	10	2.21	0.01	2.22	0.02	**2.16**	0.02
g50-4-01	50	10	5.67	0.07	5.47	0.13	**5.46**	0.08
		15	4.37	0.09	4.33	0.15	**4.32**	0.1
g75-4-01	75	4	16.39	0.04	**16.2**	0.18	**16.2**	0.08
		5	13.05	0.06	11.73	0.19	**11.67**	0.1
		20	4.62	0.36	4.79	1.25	**4.59**	0.62
g200-4-01	200	3	31.97	0.29	**31.8**	2.5	**31.8**	0.89
		10	18.33	0.98	16.27	7.43	**16.22**	2.99
g400-4-01	400	3	73.03	1.65	**71.5**	19.41	**71.5**	6.47
		4	62.56	1.77	**55**	19.61	**55**	6.48
g1000-4-01	1000	2	139.3	17.33	**138**	292.42	**138**	92.79
		3	116.43	12.31	**114**	293.46	**114**	90.88
15×15	255	2	411.6	0.33	**403**	3.52	**403**	1.22
		3	167.1	0.39	**157**	3.51	**157**	1.23
		5	108.26	0.45	**98**	3.56	**98**	1.29
33×33	1089	3	718.6	14.11	**707**	379.51	**707**	116.78
		10	358.95	36.94	334.55	484.45	**327.1**	415.42
		15	244.83	66.49	224.35	830.83	**211.9**	645.36
		20	184.02	95.05	168.27	1822.32	**168.1**	1470.59
45×5	225	2	483	0.33	**483**	3.63	**483**	1.23
		20	20.24	2.64	19.43	36.22	**18.54**	12.96
100×10	1000	2	1689	23.32	**1689**	300.97	**1689**	96.51
		3	1058	13.65	**1041**	302	**1041**	90.32
		5	757.3	13.31	703.8	117.55	**703.6**	103.33
		10	405.4	30.55	398.6	268.4	**391.9**	228.9
		15	290.42	48.83	273.05	830.52	**269.35**	507.5
steinc5	500	2	257.4	3.52	**257**	38.35	**257**	12.4
		3	211.7	2.93	**203**	38.39	**203**	12.47
		5	177.6	2.85	**169**	38.87	**169**	13.18
		10	99.8	6.85	**89.35**	87.89	**89.35**	29.58
steind5	1000	2	480.2	11.11	**473**	300.87	**473**	90.23
		3	458.7	9.81	**457**	303.1	**457**	90.75

Table 5 Statistical test on the results of *Best* values

Approaches	*p*-value	Significant
ABC_MEDKC vs GA1	0.02444	Yes
ABC_MEDKC vs GA2	<0.00001	Yes

Table 6 Statistical test on the results of *Avg* values

Approaches	*p*-value	Significant
ABC_MEDKC vs GA1	0.00001	Yes
ABC_MEDKC vs GA2	<0.00001	Yes

As future work, the research can be extended for the development of new metaheuristic techniques specifically tailored for the MEDKC problem.

Author Contributions MI: conceptualization, programming, validation, and writing—original draft. SS: supervision, and writing—review & editing.

Funding No funding is available.

Data availability All the data used in this study are available at Matic et al. (2017) and can be made available on request.

Declarations

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

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abraham I, Gavoille C, Malkhi D, Nisan N, Thorup M (2004) Compact name-independent routing with minimum stretch. In: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, pp 20–24

Awerbuch B, Bar-Noy A, Linial N, Peleg D (1990) Improved routing strategies with succinct tables. J Algorithms 11(3):307–341

Awerbuch B, Peleg D (1990) Sparse partitions. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science, pp 503–513

Blesa MJ, Xhafa F (2000) A c++ implementation of tabu search for k- cardinality tree problem based on generic programming and component reuse. Young Researchers Workshop, Citeseer

Blum C, Blesa MJ (2005) New metaheuristic approaches for the edge-weighted k-cardinality tree problem. Comput Oper Res 32(6):1355–1377

Chechik S (2013) Compact routing schemes with improved stretch. In: Proceedings of the 2013 ACM symposium on Principles of distributed computing, pp 33–41

Cowen LJ (2001) Compact routing with minimum stretch. J Algorithms 38(1):170–183

Davidović T, Ramljak D, Šelmić M, Teodorović D (2011) Bee colony optimization for the p-center problem. Comput Oper Res 38(10):1367–1376

Eilam T, Gavoille C, Peleg D (2003) Compact routing schemes with low stretch factor. J Algorithms 46(2):97–114

Enachescu M, Wang M, Goel A (2008) Reducing maximum stretch in compact routing. In: IEEE INFOCOM 2008-The 27th Conference on Computer Communications, pp 336–340. IEEE

Garcia-Diaz J, Menchaca-Mendez R, Menchaca-Mendez R, Hernández SP, Pérez-Sansalvador JC, Lakouari N (2019) Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation. IEEE Access 7:109228–109245

Ghoshal S, Sundar S (2020) Two heuristics for the rainbow spanning forest problem. Eur J Oper Res 285(3):853–864

Ghoshal S, Sundar S (2020) Two heuristics for the rainbow spanning forest problem. Eur J Oper Res 285(3):853–864

Ghoshal S, Sundar S (2021) Two approaches for the min-degree constrained minimum spanning tree problem. Appl Soft Comput 111:107715

Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Technical report, Citeseer

Karaboga D, Akay B (2009) A survey: algorithms simulating bee swarm intelligence. Artif Intell Rev 31(1–4):61

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

Khuller S, Pless R, Sussmann YJ (2000) Fault tolerant k-center problems. Theor Comput Sci 242(1–2):237–245

Khuller S, Sussmann YJ (2000) The capacitated k-center problem. SIAM J Discr Math 13(3):403–418

Könemann J, Li Y, Parekh O, Sinha A (2004) An approximation algorithm for the edge-dilation k-center problem. Oper Res Lett 32(5):491–495

Krioukov D, Claffy K, Fall K, Brady A (2007) On compact routing for the internet. ACM SIGCOMM Comput Commun Rev 37(3):41–52

López-Sánchez AD, Sánchez-Oro J, Hernández-Díaz AG (2019) Grasp and vns for solving the p-next center problem. Comput Oper Res 104:295–303

Lučić P, Teodorović D (2001) Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. u: Preprints of the triennial symposium on transportation analysis tristan iv. Azores, Portugal, June, pp 13–19

Matic D, Kratica J, Maksimovic Z (2017) Solving the minimum edge-dilation k-center problem by genetic algorithms. Comput Ind Eng 113:282–293

Moy J (1998) Ospf version 2, ietf rfc 2328,1998 (at http://www.ietf.org/rfc)

Olague G, Puente C (2006) The honeybee search algorithm for three-dimensional reconstruction. In: Workshops on applications of evolutionary computation, Springer, pp 427–437

Osaba E, Villar-Rodriguez E, Del Ser J, Nebro AJ, Molina D, LaTorre A, Suganthan PN, Coello C AC, Herrera F (2021) A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems. Swarm Evolut Comput, p 100888

Pan Q-K, Tasgetiren MF, Suganthan PN, Chua TJ (2011) A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf Sci 181(12):2455–2468

Peleg D, Upfal E (1989) A trade-off between space and efficiency for routing tables. J ACM (JACM) 36(3):510–530

Pham DT, Ghanbarzadeh A, Koç E, Otri S, Rahim S, Zaidi M (2006) The bees algorithm-a novel tool for complex optimisation problems. In Intelligent production machines and systems, Elsevier, pp 454–459

Rajasekhar A, Lynn N, Das S, Suganthan PN (2017) Computing with the collective intelligence of honey bees-a survey. Swarm Evolut Comput 32:25–48

Roditty L, Tov R (2015) New routing techniques and their applications. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, pp 23–32

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

Singh K, Sundar S (2018) Artifical bee colony algorithm using problem-specific neighborhood strategies for the tree t-spanner problem. Appl Soft Comput 62:110–118

Singh K, Sundar S (2018) Two new heuristics for the dominating tree problem. Appl Intell 48(8):2247–2267

Sundar S, Suganthan PN, Chua TJ, Cai TX, Soon CC (2017) A hybrid artificial bee colony algorithm for the job-shop scheduling problem with no-wait constraint. Soft Comput 21(5):1193–1202

Thorup M, Zwick U (2001) Compact routing schemes. In: 13th annual acm symposium on parallel algorithms and architectures (spaa)

Wedde HF, Farooq M, Zhang Y (2004) Beehive: an efficient fault-tolerant routing algorithm inspired by honey bee behavior. In: International Workshop on Ant Colony Optimization and Swarm Intelligence, Springer, pp 83–94

Wilcoxon F (1945) Wilcoxon signed-rank test calculator. https://www.socscistatistics.com/tests/signedranks/default2.aspx

Xu Y, Peng J, Xu Y (2018) The mixed center location problem. J Combin Optim 36(4):1128-1144