International Journal on Artificial Intelligence Tools
Vol. 28, No. 1 (2019) 1950004 (28 pages)

DOI: 10.1142/S0218213019500040

Solving Traveling Salesman Problem by Using Combinatorial Artificial Bee Colony Algorithms

Dervis Karaboga* and Beyza Gorkemli†

Computer Engineering Department, Erciyes University
Kayseri, Kayseri 38700/Melikgazi, Turkey
**karaboga@erciyes.edu.tr*
†*bgorkemli@erciyes.edu.tr*

Received 9 March 2016
Accepted 11 February 2019
Published 28 February 2019

Artificial bee colony (ABC) is a quite popular optimization approach that has been used in many fields, with its not only standard form but also improved versions. In this paper, new versions of ABC algorithm to solve TSP are introduced and described in detail. One of these is the combinatorial version of standard ABC, called combinatorial ABC (CABC) algorithm. The other one is an improved version of CABC algorithm, called quick CABC (qCABC) algorithm. In order to see the efficiency of the new versions, 15 different TSP benchmarks are considered and the results generated are compared with different well-known optimization methods. The simulation results show that, both CABC and qCABC algorithms demonstrate good performance for TSP and also the new definition in quick ABC (qABC) improves the convergence performance of CABC on TSP.

Keywords: Combinatorial optimization; TSP; artificial bee colony algorithm; combinatorial artificial bee colony algorithm.

1. Introduction

Combinatorial optimization tries to find the optimal solutions to the problems on discrete space. A large set of practical problems can be defined as the form of combinatorial optimization problems, such as traveling salesman problem (TSP), minimum spanning tree, vehicle routing problem, quadratic assignment problem, shortest path tree, assembly line balancing problem, scheduling problem, etc. Among them, TSP is generally used to test the performance of newly developed approaches for combinatorial optimization problems and related to many important fields such as logistics, production, transportation and semiconductor industries.

†Corresponding author

TSP is an NP hard problem about finding a Hamiltonian path with minimum cost.[1] According to the basic principle of TSP, the salesman starts from some city, visits all other cities exactly once and returns to the starting city trying to obtain a closed tour with minimum cost. The cost of the tour directly depends on the tour length. In order to calculate the distance between the city i and the following city, $d(T[i], T[i+1])$, Euclidean distance is used in this study and defined by Eq. (1).

$$d(T[i], T[i+1]) = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}\,. \tag{1}$$

Total tour length f can be given using the Euclidean distances between the cities as in Eq. (2)

$$\sum_{i=1}^{n-1} d(T[i], T[i+1]) + d(T[n], T[1]) \tag{2}$$

where n represents the total number of cities.

Many researchers proposed to solve TSP which is an interesting problem with its easy definition and very hard solution. Firstly, some traditional methods consisting of exact and heuristic methods were introduced for solving TSP. For example, cutting planes[2] and branch and cut[3] methods are able to solve small TSPs, optimally. On the other hand, heuristic methods like Markov chain[4] and 2-opt[5] are more preferable for large TSPs. Additionally, there are some greedy principle based algorithms which can be used to solve TSP, like nearest neighbor algorithm.[6] However, the traditional methods generally result in exponential computing times or solve the problems with unsatisfactory qualities. In order to overcome these shortcomings, in other words because of the high importance of achieving better solutions in acceptable computational time, several metaheuristic algorithms have been developed for TSP in literature. They can be grouped as single solution based and population based algorithms. Simulated annealing (SA)[7] and tabu search (TS)[8] are well known single solution based algorithms which take an initial solution and try to improve it by producing neighbor solutions through the optimization process. On the other hand, genetic algorithm (GA),[9] ant colony system (ACS),[10] particle swarm optimization (PSO),[11] differential evolution (DE)[12] and artificial bee colony (ABC) algorithm[13] are most popular population based metaheuristics which try to improve a set of solutions called population in parallel. Some of these metaheuristic methods were originally proposed for searching on discrete space such as ACS, bee colony optimization (BCO), GA, SA and TS algorithm. Dorigo and Gambardella solved TSP using the ACS.[10] Zhang and Feng used a max-min ant system for solving TSP.[14] An ant colony optimization (ACO) algorithm was proposed for TSP by Gan *et al.*.[15] In Ref. 15, to solve the stagnation behavior and premature convergence problem of the basic ACO algorithm on TSP, a scout characteristic was introduced. Ilie and Badica proposed a new distributed approach for ACO algorithm on a distributed architecture.[16] In order to improve the quality of the solutions of ACO, Puris *et al.* used a two-stage approach.[17] Tuba and Jovanovic presented an improved ACO algorithm with novel pheromone correction strategy

for TSP.[18] Wong *et al.* described BCO algorithm for TSP[19] and then, in order to improve prior solutions generated by the BCO model, they integrated 2-opt heuristic into BCO.[20] Also, there are many different versions of TS,[21,22] SA,[23] GA[24,9] for TSP. In addition to these metaheuristics, there are some other metaheuristics which were originally introduced for numerical problems rather than combinatorial ones. However, having inspired of their success on numerical problems, many researchers attempted to describe the combinatorial variants of these numerical optimization approaches to solve TSP, such as PSO[25,26] and DE.[27–29]

ABC was also originally proposed to solve numerical problems by Karaboga.[13] It is a swarm intelligence based metaheuristic algorithm that models the foraging behavior of honey bees.[13,30] Since 2005, ABC has been used in many fields as an effective optimization technique. There are several survey studies related to ABC in the literature.[31–35] Some of the researchers proposed ABC's discrete versions for TSP. Li *et al.* developed a discrete ABC algorithm which uses the concept of swap operator to solve TSP.[36] By combining the modified nearest neighbor approach and the improved inner-over operation, Li *et al.* described a variant of ABC for TSP.[37] Being inspired of the success of ABC's optimization on numerical problems,[38] and a successful mutation operator of GA introduced for TSP,[39] the authors defined a new neighborhood searching mechanism and proposed a new combinatorial version of ABC for TSP and named this algorithm as Combinatorial ABC (CABC).[40] In 2012, the authors introduced a new definition for onlooker bees' foraging behavior and called this new variant as quick ABC (qABC).[41,42] These studies showed that the new definition significantly improves the convergence performance of standard ABC on numerical optimization problems. Therefore, they integrated this new definition into CABC algorithm which is one of the combinatorial versions of ABC for TSP.[43] This was the first time that qABC model is applied to a combinatorial optimization problem. However, these studies about CABC and qCABC only provide the basic information about the algorithms and the experimental studies were carried out only on two test problems: 150 and 200 cities. Only the best and average errors are presented as results and they are compared with the results of GA variants. Also, convergence graphics are given. Since these limited studies are not sufficient for examining the performance of these algorithms, in this article, CABC and qCABC algorithms are described in a more detailed way and their performances are deeply examined on a larger set of TSP benchmarks using statistical tests. Their results are not only compared with the results of GA variants but also with the results of some other well known state of art optimization algorithms like ACS and BCO that are also swarm intelligence based optimization algorithms. And also some tuning studies on the control parameter limit are carried out for both CABC and qCABC algorithms. Additionally, some analysis about the CPU times that algorithms require is provided in this study. The rest of the paper is organized as follows; in Section 2, CABC algorithm is presented, and in Section 3 it describes the qCABC algorithm. The simulation results are presented in Section 4 and in Section 5, the conclusion is given.

2. Combinatorial Artificial Bee Colony Algorithm — CABC

CABC algorithm is a variant of ABC that works on discrete space and introduced into the literature in 2011.[40] In CABC algorithm, basic phases of standard ABC algorithm which are also suitable for discrete search space are fully protected. Only the neighborhood searching mechanism is adapted to combinatorial optimization for TSP. It is a known fact that the neighborhood search mechanism has a significant effect on the performance of an optimization algorithm. Sometimes it is difficult to determine whether a good or bad performance is due to the algorithm itself or due to the neighborhood search mechanism used within it. One of the successful mutation operators of GA for TSP, which was proposed by Albayrak and Allahverdi,[39] is used for this adaptation. This mutation operator is called Greedy Sub Tour Mutation (GSTM). So, while neighbor solutions are produced by using this powerful mechanism to increase a general success of optimization process proposed, at the same time it is also possible to compare the CABC's *pure algorithmic performance* with respect to GA for combinatorial optimization problems.

The flowchart of CABC algorithm is given in Fig. 1. It should be stated that the basic steps of the CABC are similar to standard ABC. The adaptation of ABC to discrete space is mainly based on bees' neighbor production mechanism used in the steps of the employed and onlooker bees for TSP. The steps of neighbor production mechanism are given in Fig. 2 for a solution x_i. For a clear understanding of this mechanism, the details of GTSM operator can be examined in Ref. 39.

In this mechanism, T_i refers to the original tour that represents the solution x_i without any neighborhood searches. *random* is used as a randomly generated number in (0,1). Food sources (solutions) are strings of cities which represent feasible solutions. Parameters of GSTM operator are presented in more detail in Ref. 39. These parameters can be listed as: P_{RC} (reconnection probability), P_{CP} (correction and perturbation probability), P_L (linearity probability), L_{MIN} (minimum sub tour length), L_{MAX} (maximum sub tour length), NL_{MAX} (neighborhood list size). Gain of point R, G_R is calculated by using Eq. (3) similarly as in Ref. 39

$$\begin{aligned} G_R = {} & d(T[R], T[R-1]) + d(T[NL_R], T[NL_R - 1]) \\ & - (d(T[R], T[NL_R]) + d(T[R-1], T[NL_R - 1])) \end{aligned} \tag{3}$$

where $d(T[R], T[R-1])$ represents the distance between the point R and the point that visited just before the R in the tour T.

Detailed steps of CABC algorithm are given in Fig. 3.

TSP optimization aims to minimize the total closed tour length. So, the quality of each tour fit_i is calculated by using Eq. (4)

$$fit_i = \frac{1}{1 + f_i} \tag{4}$$

where f_i refers to the objective value, in other words the tour length of the solution x_i.

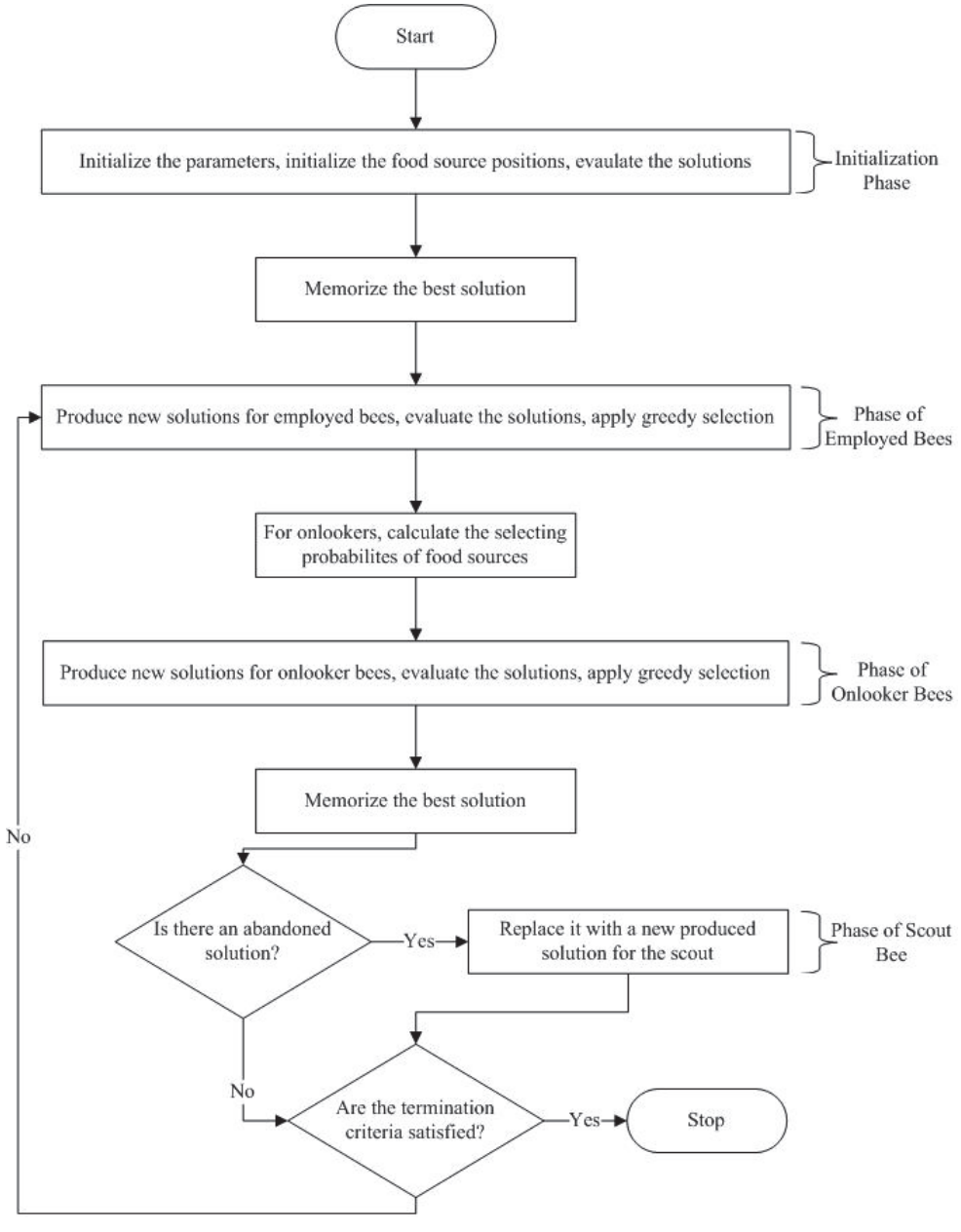

Fig. 1. Flow diagram of CABC algorithm.

Randomly select a solution x_k in the population. $(k \neq i)$
Randomly select a city j.
Determine the value of direction of the search parameter ϕ randomly. $(\phi = -1, 1)$
if $(\phi = -1)$ **then**

The city visited just before the city j in tour T_i is set as the city visited just before the city j in tour T_k.

else

The city visited immediately after the city j in tour T_i is set as the city visited immediately after the city j in tour T_k.

end if// A new closed tour $T_i^\sharp$ is produced with this operation.
As a result of this new connection, there will be an open sub tour T_i^*. This sub tour's first city is assigned as R_1 and last city is assigned as R_2
if $(random \leq P_{RC})$ **then**
Add T_i^* to $T_i^\sharp$ so that a minimum extension is generated.
else

if $(random \leq P_{CP})$ **then**

Start from the position of R_1 in T_i and add each city of T_i^* to the $T_i^\sharp$ by rolling or mixing with P_L probability.(If $random \leq P_L$ then add the city by mixing, else add the city by rolling)

else

Randomly select one neighbor from neighbor lists for the points R_1 and R_2 (NL_{R_1} and NL_{R_2}) . (To guarantee the inversion, these selected points must not be the immediately preceding cities of R_1 and R_2 in tour T_i.)

Invert the points NL_{R_1} or NL_{R_2} which provides maximum gain by arranging these points to be the neighbors (immediately preceding cities) of the points R_1 or R_2.

end if
end if

Fig. 2. Neighbor production mechanism.

Probability values of being selected by an onlooker bee is formulated with Eq. (5) for the tours.

$$p_i = \frac{0.9 \times fit_i}{fit_{best}} + 0.1\,. \tag{5}$$

The limit value, l, is calculated by Eq. (6)

$$l = \frac{cs \times D}{L} \tag{6}$$

Initialization phase:

Set initial values of the control parameters: colony size cs, maximum number of iterations *MaxNumber*.

Initialize the food sources' positions (initial tours) x_i. $i = 1, 2, \ldots, SN$.

Evaluate the tour lengths.

Memorize the best tour.

$c = 0$

repeat

Employed bees phase: for each employed bee;

Generate a new candidate solution (tour) υ_i in the neighborhood of x_i (Fig. 2) and evaluate this tour.

Select the better one between x_i and υ_i.

Using their fitness values computed by Eq. (4), calculate the probability values p_i for the tours x_i with Eq. (5).

Onlooker bees phase: for each onlooker bee;

Depending on p_i values, select a tour x_i.

Generate a new candidate solution (tour) υ_i in the neighborhood of x_i (Fig. 2) and evaluate this solution.

Select the better one between x_i and υ_i.

Memorize the best tour achieved so far.

Scout bee phase: If there is an abandoned tour (The tour which could not be improved through a predetermined number of trials. This number is called "limit"), replace it with a new tour for the scout (this tour will be generated by using the same strategy as in the initialization phase).

$c = c + 1$

until $(c = MaxNumber)$

Fig. 3. Detailed steps of CABC algorithm.

where cs represents the colony size and D is the dimension of the problem ($D = n$). L is a dividing coefficient that has an important role while determining the value of limit parameter.

3. Quick Combinatorial Artificial Bee Colony Algorithm — qCABC

Comparing to standard ABC algorithm, qABC simulates the behavior of real honey bee foragers more closely.[41,42] In real honey bee colonies, employed bees and onlooker bees use different searching ways while determining their food sources. Employed bees exploit the food sources they have already visited before. However,

foraging behavior of onlooker bees is not based on their own knowledge about the food sources. An onlooker decides a region of food sources by watching the dances of employed bees. When she arrives at the chosen region where she has not visited and seen before, she visually examines the sources from above and chooses the fittest one. In other words, before making a decision about a food source, the onlooker bee evaluates the information of neighbor sources which are similar in terms of position, since she visits that region first time. However, in basic ABC algorithm, since it is assumed that both employed and onlooker bees determine their food sources in the same way, the same formula is defined and used for both foragers. For the foraging behavior of onlooker bees, a new definition was introduced and described a new variant of ABC. This new variant was called as quick ABC-qABC[41,42] and used to solve numerical optimization problems. In the literature some researchers proposed to overcome the weakness of the convergence performance of ABC by introducing different strategies like modification, hybridization, etc.[31] Using the qABC idea, the local search ability of ABC is improved since more trials are carried out by onlookers to improve their solutions which are the most promising solutions. Details of the qABC algorithm proposed for numerical problems can be found in Refs. 41 and 42. It is very clear that, if an appropriate similarity measure is defined for the solutions to TSP, this useful idea can be easily incorporated into CABC proposed for combinatorial problems, too.

In order to use the definition in qABC for the onlooker bees in the CABC algorithm, a similarity measure is defined for the solutions of TSP in Ref. 43. This new variant is called the quick combinatorial ABC-qCABC. In qABC, the Euclidean distance was proposed to determine the difference between two solutions for numerical optimization. Since it is aimed to obtain the optimal combination of the cities to be visited, similarity of the solutions can be defined based on the same connections of the cities in the closed tours in TSP. Hence, Expression 7 and Eq. (8) can be used to determine the distance between two solutions x_i and x_m,

$$dv_{im}(j) = \begin{cases} 0\,, & \textit{if } T_i[j+1] = T_m[j+1] \;\; \textit{or} \;\; T_i[j+1] = T_m[j-1]) \\ 1\,, & \textit{else} \end{cases} \tag{7}$$

where $T_i[j+1]$ and $T_m[j+1]$ refer to the cities visited immediately after the city j in tour T_i and T_m, respectively. Similarly, $T_m[j-1]$ refers to the city that visited just before the city j in tour T_m which is related to the solution x_m and dv_{im} represents a distance vector between two closed tours x_i and x_m. As explained above, similarity definition is based on the same connections of the cities in the closed tours. To calculate the distance between two solutions, $d(i,m)$, the following equation is used.

$$d(i,m) = \sum_{j=1}^{n} dv_{im}(j)\,. \tag{8}$$

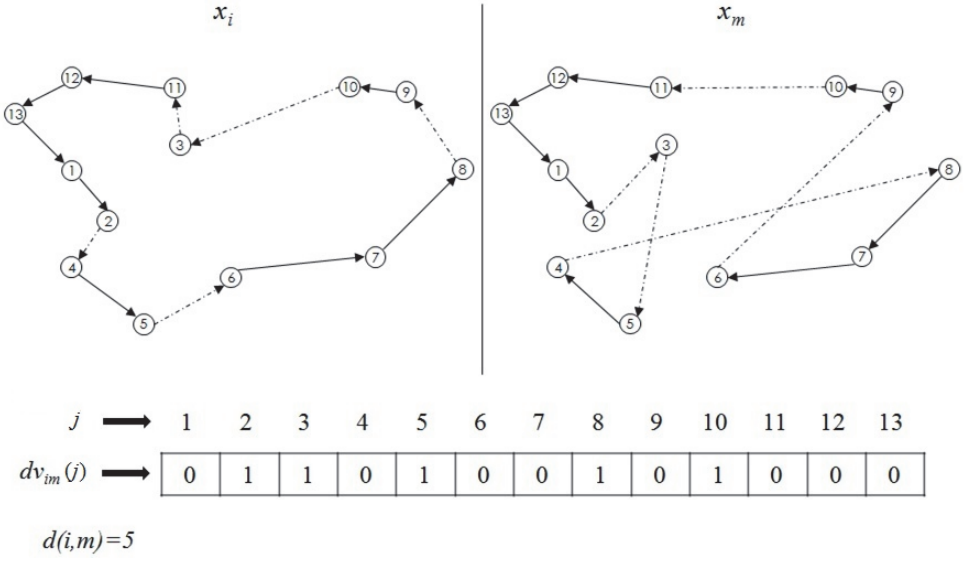

Fig. 4. An example of the similarity measure between two tours.

A simple example for qCABC's similarity measure is given in Fig. 4. In this example a 13 citied TSP is used. By using the expression 7, distance vector dv_{im} is produced for two different solutions of this problem, x_i and x_m. While obtaining this vector, actually, it is checked if there is a connection between the same cities in the solutions, x_i and x_m. If there is, the related cell of the vector is set with the value "0", otherwise the value "1" is assigned to this cell. After producing this distance vector, the total distance $d(i,m)$ is calculated by using Eq. (8).

The neighborhood of x_i is described depending on the neighborhood radius coefficient r and the mean distance calculated for x_i, md_i. md_i is the mean distance between x_i and the rest of solutions in the population and defined by Eq. (9).

$$md_i = \frac{\sum_{m=1}^{SN} d(i,m)}{SN-1} \tag{9}$$

where, SN represents the number of solutions in the population.

Finally, the following rule is used for determining the neighbor solutions of x_i.

$$\begin{aligned} &\textit{if } d(i,m) \leq r \times md_i \textit{ then } x_m \textit{ is a neighbor of } x_i\,, \\ &\textit{else not} \end{aligned} \tag{10}$$

Detailed steps of qCABC algorithm are given in Fig. 5.

Notice that only difference between CABC and qCABC variants is with the phase of onlooker bees. While employed and onlooker bees determine their new tours in the same way in CABC, they do differently in qCABC.

Initialization phase:

Set initial values of the control parameters: colony size cs, maximum number of iterations *MaxNumber*.

Initialize the food sources' positions (initial tours) x_i. $i = 1, 2, \ldots, SN$.

Evaluate the tour lengths.

Memorize the best tour.

$c = 0$

repeat

Employed bees phase: for each employed bee;

Generate a new candidate solution (tour) υ_i in the neighborhood of x_i (Fig. 2) and evaluate this tour.

Select the better one between x_i and υ_i.

Using their fitness values computed by Eq. (4), calculate the probability values p_i for the tours x_i with Eq. (5).

Onlooker bees phase: for each onlooker bee;

Depending on p_i values, select a tour x_i.

Determine the best tour $x_{N_i}^{best}$ among the neighbors determined by Expression 10 of the tour x_i and itself.

Generate a new candidate solution (tour) $\upsilon_{N_i}^{best}$ in the neighborhood of $x_{N_i}^{best}$ (Fig. 2) and evaluate this solution.

Select the better one between $x_{N_i}^{best}$ and $\upsilon_{N_i}^{best}$.

Memorize the best tour achieved so far.

Scout bee phase: If there is an abandoned tour (the tour which could not be improved through a predetermined number of trials. This number is called "limit"), replace it with a new tour for the scout (this tour will be generated by using the same strategy as in the initialization phase).

$c = c + 1$

until ($c = MaxNumber$)

Fig. 5. Detailed steps of qCABC algorithm.

4. Experimental Study and Simulation Results

In this study, firstly a set of experiments is conducted about the limit parameter value for both CABC and qCABC algorithms since the limit parameter is a significant control parameter of ABC.[38] And, some comparison studies are carried out among ten different optimization methods. These algorithms are CABC, qCABC, and eight different variants of GA having different mutation operators (Exchange Mutation (EXC), Displacement Mutation (DISP), Inversion Mutation (INV), Insertion Mutation (INS), Simple Inversion Mutation (SIM), Scramble

Mutation (SCM), Greedy Swap Mutation (GSM) and GTSM). In these comparison studies, the results obtained by GAs are taken from the study of Ref. 39 that provides GTSM operator which is adapted to ABC's neighborhood searching mechanism in both CABC and qCABC. So, in this study the same test problems are considered as in Ref. 39. Additionally, CABC and qCABC algorithms are run for the FL1577 problem in order to see the ABC based algorithms' performances on a problem having much more cities than other 14 benchmarks having various number of cities. These test problems are symmetric type TSP benchmarks. Test problems, the city numbers of the problems and the optimum tour lengths are given in Table 1. They can be also found in the TSPLIB: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. In our study, distances between the cities are measured by integer numbers.

Table 1. Optimum tour lengths of the TSP benchmarks.

Problem	Number of Cities	Optimum Tour Length
Berlin52	52	7542
KroA100	100	21282
Pr144	144	58537
Ch150	150	6528
KroB150	150	26130
Pr152	152	73682
Rat195	195	2323
D198	198	15780
KroA200	200	29368
Ts225	225	126643
Pr226	226	80369
Pr299	299	48191
Lin318	318	42029
Pcb442	442	50778
Fl1577	1577	22249

For all experiments, algorithms were run ten times with random seeds as in Ref. 39. For a fair comparison, the nearest neighbor tour construction heuristic is used to produce the initial solutions and the same parameter values are taken as in Ref. 39. Considering the searching mechanisms of CABC and qCABC algorithms, only the parameter L_{MAX} is set as $n/2$, not $Int(\sqrt{(n)})$ as in Ref. 39. The parameter settings of CABC and qCABC algorithms are given in Table 2.

Firstly, since the parameter limit has a significant effect on the performance of ABC algorithm,[38] we carried out some experiments about the limit parameter of both CABC and qCABC algorithms for the TSP benchmarks presented in Table 1. In these experiments, for each problem, we searched the appropriate values of the limit l by changing the dividing coefficient L. The mean value, standard deviation and best value of finally obtained tour lengths of these ten runs are given in Tables 3–6 for CABC algorithm, and in Tables 7–10 for qCABC algorithm. When

Table 2. Parameter settings.

Parameter	CABC	qCABC
cs	40	40
Max Number of Evaluations	800 000	800 000
P_{RC}	0.5	0.5
P_{CP}	0.8	0.8
P_L	0.2	0.2
L_{MIN}	2	2
L_{MAX}	$n/2$	$n/2$
NL_{MAX}	5	5
r	—	1
L	1, 2, 3, 4	1, 2, 3, 4

Tables 3–10 are examined, it can be seen from the results that, limit parameter has an important effect on the performance of ABC algorithms. Best values of table columns are written in bold and the optimum values are written in italic.

If we compare the mean objective value of ten independent runs, the CABC algorithm gives better results for Pr144 when $L = 1$. For KroA100, Pr152, D198, KroA200 and Pcb442 problems this approach obtains better results when $L = 2$. Ch150, Krob150, Rat195 and Lin 318 problems solved with less errors when $L = 3$ and Pr226, Pr299, Fl1577 problems are solved with less errors when L is set as $L = 4$. The CABC algorithm achieves optimal tour length and the standard deviation value is zero for Berlin52 problem for all tried limit values and for Ts225 when $L = 4$. KroA100, Ts225 and Pr152 problems' optimum tour lengths are also found by CABC for four different L values. Additionally, when the $L = 3$, optimum tour length of Pr144 is also achieved.

When the results of qCABC are evaluated, considering the mean objective value of ten independent runs, it can be seen that this algorithm obtains better results for Rat195, KroA200 and Lin318 if L is set as $L = 1$. Pr144 problem is solved with less error when $L = 2$. When $L = 3$, the algorithm produces better results for KroA100, Krob150, Pr152, Pcb442. However when the limit value of the qCABC algorithm is adjusted with $L = 4$, Ch150, D198, Pr226, Pr299 and Fl1577 problems solved with less error. Similar to the CABC algorithm, the qCABC algorithm also achieves optimal tour length and the standard deviation value is zero for Berlin52 problem for all tried limit values and for Ts225 when $L = 2$ and $L = 4$. Optimum tour lengths of KroA100, Pr152 and Ts225 problems are also obtained by qCABC for four different L values. Optimum tour length is found for Pr144 when $L = 3$ by qCABC.

Considering the mean objective function values, the most appropriate L values are determined for each problem. And using these results, the performances of CABC, qCABC and eight different variants of GA are compared in Tables 11 and 12. In this comparison, the best and average percentage error values are used

Table 3. Limit experiments of CABC algorithm.

	Berlin52			KroA100			Pr144			Ch150		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	***7542***	***0***	***7542***	21297.4	29.4184	***21282***	**58631**	**35.3072**	58590	6565	14.7037	6549
2	***7542***	***0***	***7542***	**21291**	**8.5790**	***21282***	58641.6	41.0638	58590	6561.2	**7.7175**	6549
3	***7542***	***0***	***7542***	21291.4	11.4123	***21282***	58649.4	66.6126	***58537***	**6556.6**	8.8566	**6542**
4	***7542***	***0***	***7542***	21292.6	13.6250	***21282***	58632.9	38.2582	58590	6566.4	8.9353	6549

Table 4. Table 3 continued.

	KroB150			Pr152			Rat195			D198		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	26350.1	52.2904	26265	73855.7	142.6464	***73682***	2358.2	**3.3704**	2353	15866.5	28.3275	**15825**
2	26338	59.6406	**26186**	**73792**	142.2385	***73682***	2355.4	6.0033	2343	**15854.4**	25.8503	**15825**
3	**26311.6**	49.6955	26211	73797.1	109.6654	***73682***	**2350**	7.2250	**2337**	15862.4	**18.6022**	15838
4	26334.6	**44.2384**	26217	73842.2	**93.5252**	***73682***	2352.3	4.7550	2345	15863.9	24.3740	15826

Table 5. Table 3 continued.

	KroA200			Ts225			Pr226			Pr299		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	29551.5	35.3674	29487	126706.8	127.6	***126643***	81152.3	144.8199	80929	48775	**53.9870**	48693
2	**29517.5**	46.5623	**29467**	126751.7	169.4007	***126643***	81133	123.4480	80872	48743.1	98.0617	48640
3	29545.2	**34.6087**	29503	126706.8	127.6	***126643***	81005.9	**89.3705**	80908	48657.2	138.3523	**48470**
4	29532.6	37.7603	29503	***126643***	***0***	***126643***	**80999.6**	98.1745	**80860**	**48649.2**	86.0451	48512

Table 6. Table 3 continued.

	Lin318			Pcb442			Fl1577		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	43155	123.1073	42947	51607.3	156.9159	**51285**	22852	105.5036	22737
2	43037.2	186.2089	**42756**	**51539.1**	108.3138	51309	22813.7	92.8031	**22665**
3	**43018.1**	127.4523	42810	51555.3	**76.0369**	51424	22814.3	80.5122	22702
4	43126.7	**118.0432**	42898	51556.3	89.3611	51389	**22791.1**	**45.7918**	22703

Table 7. Limit experiments of qCABC algorithm.

	Berlin52			KroA100			Pr144			Ch150		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	***7542***	***0***	***7542***	21291.4	10.0817	***21282***	58656.9	65.7076	58590	6566.3	9.9	6549
2	***7542***	***0***	***7542***	21288.9	10.6626	***21282***	**58624.2**	**39.7437**	58590	6563.6	7.2966	6553
3	***7542***	***0***	***7542***	**21284.4**	**4.8**	***21282***	58626.1	49.1558	***58537***	6563.3	9.9905	**6544**
4	***7542***	***0***	***7542***	21289	9.2520	***21282***	58632.6	41.4782	58570	**6561.9**	**4.9285**	6549

Table 8. Table 7 continued.

	KroB150			Pr152			Rat195			D198		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	26384.2	72.8146	26257	73872.3	147.1292	***73682***	**2350.4**	5.3889	**2340**	15872.6	21.9691	**15812**
2	26344	43.9500	26251	73903.5	116.2327	***73682***	2353.2	**5.0160**	2347	15875.7	**20.6109**	15845
3	**26317.1**	68.4872	**26193**	**73843.2**	**107.3665**	***73682***	2353.4	6.5146	**2340**	15862.1	31.7945	15819
4	26358.3	**33.0395**	26284	73850.3	116.7810	***73682***	2350.5	6.8739	**2340**	**15856.1**	24.8775	15820

Table 9. Table 7 continued.

	KroA200			Ts225			Pr226			Pr299		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	**29508.4**	43.3917	**29420**	126674.9	95.7	***126643***	81119.6	160.5143	80829	48754.6	134.6820	48524
2	29526.6	43.2856	29461	***126643***	***0***	***126643***	81057.7	118.4399	80876	48723	**114.1622**	48513
3	29512.5	**28.4824**	29477	126651.3	24.9	***126643***	81108.3	**57.0159**	81040	48690.7	154.7043	48452
4	29534.3	35.5276	29477	***126643***	***0***	***126643***	**81051.6**	131.8523	**80804**	**48617.2**	133.5049	**48347**

Table 10. Table 7 continued.

	Lin318			Pcb442			Fl1577		
L	Mean	Std	Best	Mean	Std	Best	Mean	Std	Best
1	**42986.4**	146.4494	42809	51537	116.2188	51387	22841.3	95.4694	22707
2	43037.1	110.8958	42829	51542.3	**71.0972**	51403	22793.7	83.6935	**22677**
3	43069.5	129.1551	42837	**51524.2**	151.7859	**51240**	22758.7	**40.7015**	22688
4	43052.8	**99.2510**	**42774**	51543.8	84.6638	51419	**22803.9**	50.4251	22742

Table 11. Performance comparison of algorithms.

Method		BERLIN52	KROA100	PR144	CH150	KROB150	PR152	RAT195
EXC	Best Err. (%)	**0**	0.1692	0.2426	1.9761	2.9277	2.0358	4.477
	Ave. Err. (%)	2.0353	2.8714	0.533	2.8232	5.0919	3.99	5.9836
DISP	Best Err. (%)	0.0663	4.2947	1.4845	3.4161	7.8263	4.7216	5.0366
	Ave. Err. (%)	1.424	8.0552	1.7628	3.9859	10.4635	5.5932	6.9393
INV	Best Err. (%)	**0**	2.6125	1.0284	2.9412	7.1183	3.1202	3.7021
	Ave. Err. (%)	1.1854	4.7965	1.9755	3.6596	9.0521	5.0318	6.3108
INS	Best Err. (%)	**0**	0.4652	0.1879	0.6434	1.8178	1.3938	4.2617
	Ave. Err. (%)	0.1525	3.0726	1.3802	2.3943	4.5488	2.8468	5.7426
SIM	Best Err. (%)	**0**	0.1692	0.2255	0.8425	2.8397	1.5282	2.3676
	Ave. Err. (%)	0.6099	1.3114	1.1717	1.5809	3.8596	2.7976	4.1886
SCM	Best Err. (%)	**0**	0.8176	0.1589	0.9344	3.2836	1.5146	4.6492
	Ave. Err. (%)	**0**	3.6966	1.3016	2.1078	7.0609	2.694	5.9923
GSM	Best Err. (%)	**0**	0.3007	0.2426	1.6544	3.452	2.2977	4.3048
	Ave. Err. (%)	0.9931	3.698	0.6891	2.4908	4.9541	2.9112	6.5476
GSTM	Best Err. (%)	**0**	**0**	**0**	0.4596	0.9644	0.7695	**0.6027**
	Ave. Err. (%)	**0**	1.1836	1.0809	0.6357	1.7616	1.6202	1.8425
CABC	Best Err. (%)	**0**	**0**	0.0905	**0.2145**	0.3100	**0**	**0.6027**
	Ave. Err. (%)	**0**	0.0423	0.1606	**0.4381**	**0.6950**	**0.1493**	**1.1623**
qCABC	Best Err. (%)	**0**	**0**	0.0905	0.3217	**0.2411**	**0**	0.7318
	Ave. Err. (%)	**0**	**0.0113**	**0.1490**	0.5193	0.7160	0.2188	1.1795

in order to consider the results presented in Ref. 39. The percentage errors are calculated by using Eq. (11)

$$Error(\%) = \frac{L_F - L_O}{L_O} \times 100 \tag{11}$$

where L_F is the tour length found by the algorithm and L_O is the optimum tour length for the problem.

From Tables 11 and 12 we can see that, results of ABC based algorithms are generally close to each other but highly better than other GA variants. Only for Lin318, Rat195 and Pr144 problems GSTM gives better results in terms of the best of ten independent runs. Notice that, when $L = 3$, both CABC and qCABC achieve 0% error for Pr144, however, in this comparison, limit values which give better mean values of ten runs are chosen.

In order to show the robustness of ABCs, in Tables 13 and 14, the results of CABC and qCABC are also compared to GA results when the limit values of both CABC and qCABC are calculated by setting the value of L as same for all problems ($L = 2$) which gives the original equation defined in the literature for ABC,[44] and also used for qABC.[42] The table shows that, even if a parameter adjusting study is not carried out for each problem, both CABC and qCABC algorithms demonstrate better performance on the considered problems and the solutions are generally better than those found by GA variants.

Both of the comparison tables show that GSTM, CABC and qCABC algorithms outperform the other algorithms clearly. Considering the average percentage errors

Table 12. Table 11 continued.

Method		D198	KROA200	TS225	PR226	PR299	LIN318	PCB442	FL1577
EXC	Best Err. (%)	1.9709	2.5061	3.0424	4.9186	10.2198	10.1026	7.3437	—
	Ave. Err. (%)	4.9728	4.9877	3.8174	7.2295	15.1775	11.7195	10.4774	—
DISP	Best Err. (%)	3.891	6.7284	3.3693	4.1322	12.2616	10.9996	11.3435	—
	Ave. Err. (%)	5.9835	8.8276	3.8916	7.993	16.849	12.9013	12.3053	—
INV	Best Err. (%)	2.6743	6.2245	3.1640	8.1512	10.9543	9.541	10.5124	—
	Ave. Err. (%)	4.9823	8.3594	3.7355	8.8249	15.706	13.0386	11.6322	—
INS	Best Err. (%)	2.0279	2.1554	3.1751	3.8423	3.8804	6.3099	8.9685	—
	Ave. Err. (%)	3.41	4.2819	3.8767	6.484	11.4187	10.3935	11.0585	—
SIM	Best Err. (%)	1.6984	1.5766	1.6930	2.5769	6.439	6.3932	6.3728	—
	Ave. Err. (%)	2.5241	3.2457	2.5818	4.1795	9.1264	8.714	8.124	—
SCM	Best Err. (%)	3.1179	3.9873	2.7945	3.7875	10.8776	9.1651	7.8597	—
	Ave. Err. (%)	4.5818	5.4502	3.8029	5.0694	13.8686	10.4456	10.6251	—
GSM	Best Err. (%)	3.0482	4.4675	3.0424	5.0517	8.8191	9.4126	5.7643	—
	Ave. Err. (%)	4.2421	5.7018	4.4254	6.4485	13.6295	11.0036	8.0389	—
GSTM	Best Err. (%)	0.3866	0.8683	0.2527	0.7242	1.2326	**0.9827**	2.0501	—
	Ave. Err. (%)	1.2193	1.5432	0.4994	1.5287	2.9169	3.3099	2.7758	—
CABC	Best Err. (%)	0.2852	0.3371	**0**	0.6109	0.6661	1.8582	1.0457	**2.0405**
	Ave. Err. (%)	**0.4715**	0.5090	**0**	**0.7846**	0.9508	2.3534	1.4989	**2.4365**
qCABC	Best Err. (%)	**0.2535**	**0.1771**	**0**	**0.5412**	**0.3237**	1.8557	**0.9098**	2.2158
	Ave. Err. (%)	0.4822	**0.4781**	**0**	0.8493	**0.8844**	**2.2780**	**1.4695**	2.4940

1950004-18

Table 13. Performance comparison of algorithms when $L = 2$.

Method		BERLIN52	KROA100	PR144	CH150	KROB150	PR152	RAT195
EXC	Best Err. (%)	**0**	0.1692	0.2426	1.9761	2.9277	2.0358	4.477
	Ave. Err. (%)	2.0353	2.8714	0.533	2.8232	5.0919	3.99	5.9836
DISP	Best Err. (%)	0.0663	4.2947	1.4845	3.4161	7.8263	4.7216	5.0366
	Ave. Err. (%)	1.424	8.0552	1.7628	3.9859	10.4635	5.5932	6.9393
INV	Best Err. (%)	**0**	2.6125	1.0284	2.9412	7.1183	3.1202	3.7021
	Ave. Err. (%)	1.1854	4.7965	1.9755	3.6596	9.0521	5.0318	6.3108
INS	Best Err. (%)	**0**	0.4652	0.1879	0.6434	1.8178	1.3938	4.2617
	Ave. Err. (%)	0.1525	3.0726	1.3802	2.3943	4.5488	2.8468	5.7426
SIM	Best Err. (%)	**0**	0.1692	0.2255	0.8425	2.8397	1.5282	2.3676
	Ave. Err. (%)	0.6099	1.3114	1.1717	1.5809	3.8596	2.7976	4.1886
SCM	Best Err. (%)	**0**	0.8176	0.1589	0.9344	3.2836	1.5146	4.6492
	Ave. Err. (%)	**0**	3.6966	1.3016	2.1078	7.0609	2.694	5.9923
GSM	Best Err. (%)	**0**	0.3007	0.2426	1.6544	3.452	2.2977	4.3048
	Ave. Err. (%)	0.9931	3.698	0.6891	2.4908	4.9541	2.9112	6.5476
GSTM	Best Err. (%)	**0**	**0**	**0**	0.4596	0.9644	0.7695	**0.6027**
	Ave. Err. (%)	**0**	1.1836	1.0809	0.6357	1.7616	1.6202	1.8425
CABC	Best Err. (%)	**0**	**0**	0.0905	**0.3217**	**0.2143**	**0**	0.8610
	Ave. Err. (%)	**0**	0.0423	0.1787	**0.5086**	**0.7960**	**0.1493**	1.3947
qCABC	Best Err. (%)	**0**	**0**	0.0905	0.3830	0.4631	**0**	1.0331
	Ave. Err. (%)	**0**	**0.0324**	**0.1490**	0.5453	0.8190	0.3006	**1.3000**

Table 14. Table 13 continued.

Method		D198	KROA200	TS225	PR226	PR299	LIN318	PCB442	FL1577
EXC	Best Err. (%)	1.9709	2.5061	3.0424	4.9186	10.2198	10.1026	7.3437	—
	Ave. Err. (%)	4.9728	4.9877	3.8174	7.2295	15.1775	11.7195	10.4774	—
DISP	Best Err. (%)	3.891	6.7284	3.3693	4.1322	12.2616	10.9996	11.3435	—
	Ave. Err. (%)	5.9835	8.8276	3.8916	7.993	16.849	12.9013	12.3053	—
INV	Best Err. (%)	2.6743	6.2245	3.1640	8.1512	10.9543	9.541	10.5124	—
	Ave. Err. (%)	4.9823	8.3594	3.7355	8.8249	15.706	13.0386	11.6322	—
INS	Best Err. (%)	2.0279	2.1554	3.1751	3.8423	3.8804	6.3099	8.9685	—
	Ave. Err. (%)	3.41	4.2819	3.8767	6.484	11.4187	10.3935	11.0585	—
SIM	Best Err. (%)	1.6984	1.5766	1.6930	2.5769	6.439	6.3932	6.3728	—
	Ave. Err. (%)	2.5241	3.2457	2.5818	4.1795	9.1264	8.714	8.124	—
SCM	Best Err. (%)	3.1179	3.9873	2.7945	3.7875	10.8776	9.1651	7.8597	—
	Ave. Err. (%)	4.5818	5.4502	3.8029	5.0694	13.8686	10.4456	10.6251	—
GSM	Best Err. (%)	3.0482	4.4675	3.0424	5.0517	8.8191	9.4126	5.7643	—
	Ave. Err. (%)	4.2421	5.7018	4.4254	6.4485	13.6295	11.0036	8.0389	—
GSTM	Best Err. (%)	0.3866	0.8683	0.2527	0.7242	1.2326	**0.9827**	2.0501	—
	Ave. Err. (%)	1.2193	1.5432	0.4994	1.5287	2.9169	3.3099	2.7758	—
CABC	Best Err. (%)	**0.2852**	0.3371	**0**	**0.6259**	0.9317	1.7298	**1.0457**	**1.8697**
	Ave. Err. (%)	**0.4715**	**0.5090**	0.0858	0.9506	1.1456	2.3988	**1.4989**	2.5381
qCABC	Best Err. (%)	0.4119	**0.3167**	**0**	0.6308	**0.6682**	1.9034	1.2308	1.9237
	Ave. Err. (%)	0.6065	0.5400	**0**	**0.8569**	**1.1039**	**2.3986**	1.5052	**2.4482**

Table 15. Results of statistical tests.

	Compared Pairs	Mean Difference	N	p Value of Paired t Test	p value of Wilcoxon Test
Case 1	GSTM-CABC	0.907	14	0.000	0.000
	GSTM-qCABC	0.906	14	0.000	0.000
	CABC-qCABC	−0.005	15	0.685	0.839
Case 2	GSTM-CABC	0.842	14	0.000	0.000
	GSTM-qCABC	0.840	14	0.000	0.000
	CABC-qCABC	0.004	15	0.834	0.715

in Tables 11–14, some statistical analyses are carried out for GSTM, CABC and qCABC algorithms to see if there is a significant difference between the performances of these algorithms. Since the test data is approximately normally distributed, paired t test, one of the parametric tests, is used to compare the means of two samples. Null hypothesis is that the mean difference of the samples is equal to 0. However, the sample size may be small for paired t test. So, Wilcoxon signed rank test which is one of the nonparametric tests is also applied. Null hypothesis of this test is that the median difference between pairs of observations is equal to 0. The compared pairs, mean difference of the percentage errors, sample sizes (N) and p values are given in Table 15. In this table, case 1 presents the test results for ABC algorithms having most appropriate L values while case 2 presents the results for ABC algorithms having the same L values ($L = 2$). When the results are examined, both of the statistical tests state that CABC and qCABC algorithms have significantly better performance than GSTM in both cases. When the mean difference of the percentage errors are considered CABC produces better results than qCABC in case 1 and qCABC produces better results than CABC in case 2. However these differences are not significant according to the paired t test with $p = 0.685$ and $p = 0.834$ and Wilcoxon signed rank test with $p = 0.839$ and $p = 0.715$. Note that, these tests are carried out for the final results of 800 000 evaluations.

In order to present the convergence performance of CABC and qCABC, four test problems are selected as having different and various number of cities. The convergence graphics of the algorithms are shown in Figs. 6–9 for KroA100, KroA200, Pcb442 and Fl1577 problems, respectively. These figures demonstrate that qCABC is quicker than CABC for the all considered problems having 100, 200, 442 and 1577 cities.

Studies show that ABC provides better performance than many evolutionary computation based optimization algorithms, especially in the meaning of exploration since it has an effective mechanism in the scout bee phase. Also, in employed bees phase all solutions in the population are tried to improve one time by performing a neighborhood search around them not by looking whether this solution is good or not. However, every food source might not be considered in the same way in onlooker bees phase. More neighborhood searches are performed for the better solutions in this phase. Actually, exploitation of better solutions are carried

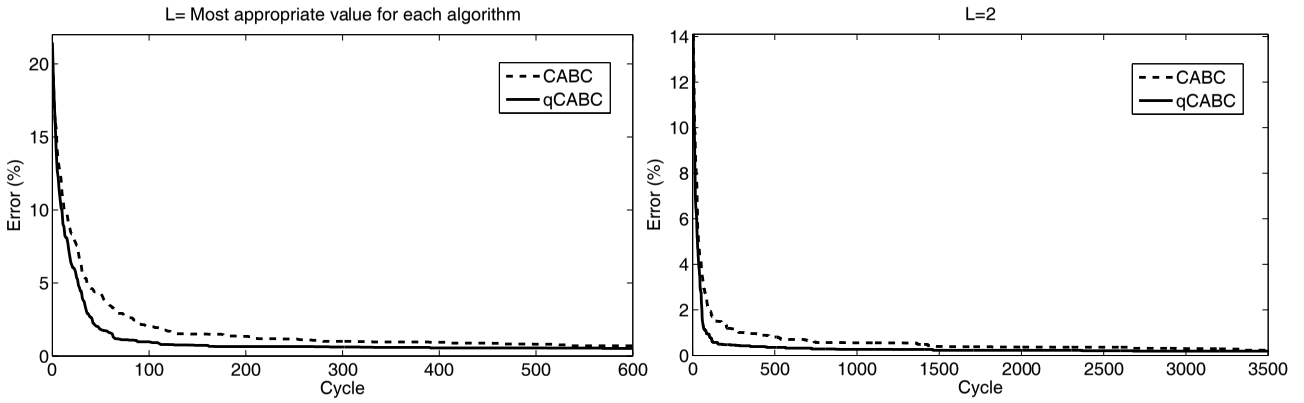

Fig. 6. Convergence performance for KroA100.

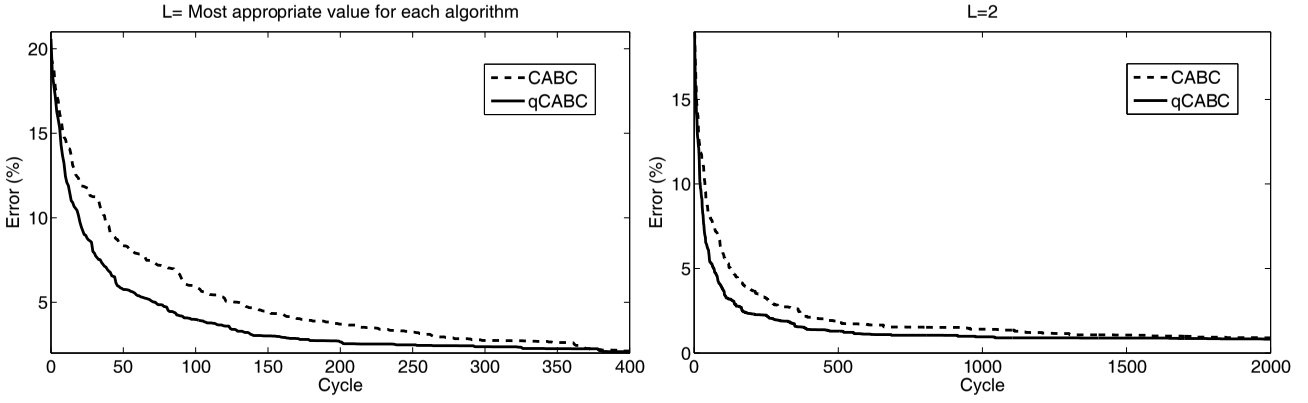

Fig. 7. Convergence performance for KroA200.

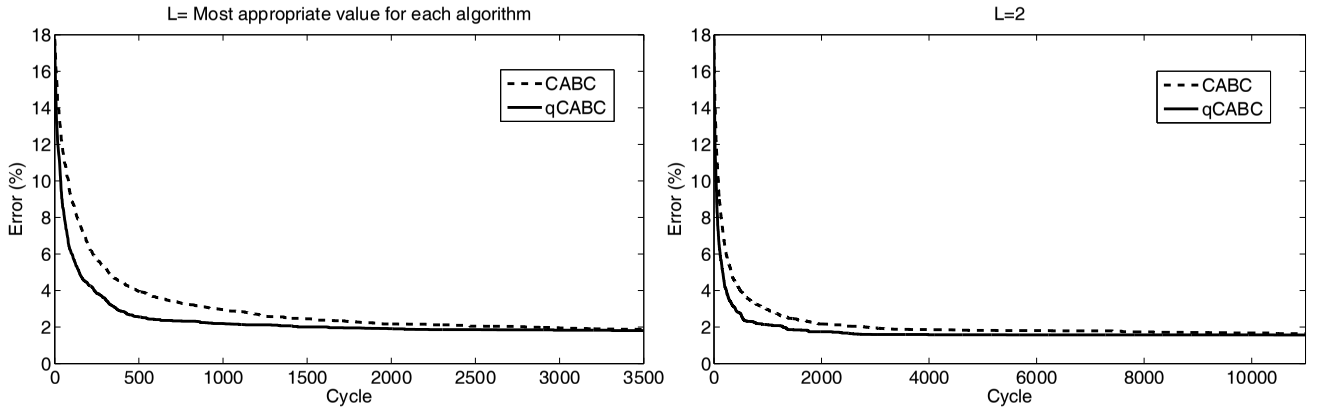

Fig. 8. Convergence performance for Pcb442.

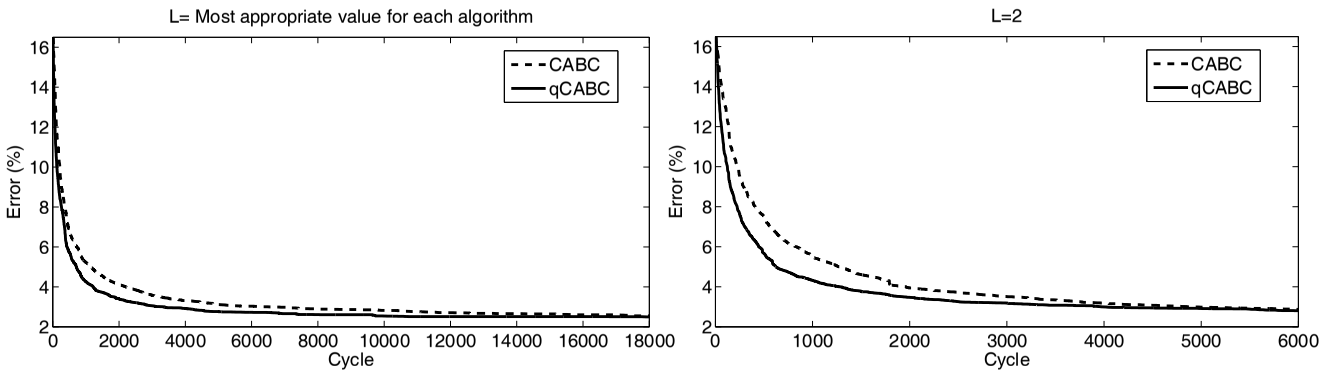

Fig. 9. Convergence performance for Fl1577.

out in the algorithm in a powerful way like exploration is stated. So, the success of ABC is not only related with the type of optimization problem (e.g. numerical or combinatorial), the power of the algorithm is from its search structure which is based on the good balance between exploration and exploitation mechanisms. Although the standard ABC is introduced for numerical problems, if we can protect its exploration-exploitation balanced mechanisms while adapting it to discrete problems the success of algorithm will continue. Since CABC is improved by employing an effective mutation operator considering the original ABC's neighborhood searching mechanism it keeps its superiority. And also qABC model proposes an idea to increase the exploitation of better solutions and this causes a quicker local search through cycles. Using an appropriate similarity measure, it is expected that this idea will speed up all kind of optimization processes like this one.

4.1. *CPU times of CABC and qCABC algorithms*

In this section CPU times of CABC and qCABC algorithms are given for 50, 100, 200, 500 and 1000 city problems when the colony size is 40, $limit = (Colony\ size * Dimension)/2$ and the stopping criteria is achieving total 4000 evaluations. 30 independent runs are carried out for each case and mean values and standard deviations of 30 runs are presented in Table 16. CPU times of the algorithms can be assessed considering the change in dimension. This change is shown in a graphic by demonstrating the mean values of the CPU Times on Fig. 10 for both CABC and qCABC algorithms. These analyses were performed on Windows 10 Education on a Intel(R) Core(TM) M-5Y51 1.20 GHz processor with 8 GB RAM and the algorithms were coded by using C Sharp programming language and .net framework 4 client profile was used.

Table 16 shows that, the needed CPU time of qCABC is higher than CABC algorithm to complete 4000 function evaluations since there is an additional part in the onlooker bees phase. However, the increment rate on qCABC's complete computing time is lower than the increment rate of the dimension, like CABC algorithm for dimension rate 100/50. For greater dimension values, increment rates on both algorithms are higher than the increment rate of the dimension.

Table 16. CPU times for problems with different number of cities.

	CABC		qCABC	
Number of Cities	Mean	Std	Mean	Std
50	0.2787	0.0952	0.5521	0.1398
100	0.3115	0.0188	1.0844	0.1822
200	1.0100	0.2325	2.2947	0.1855
500	3.5468	0.1118	5.9587	0.7467
1000	8.4452	1.6377	18.5469	0.9318

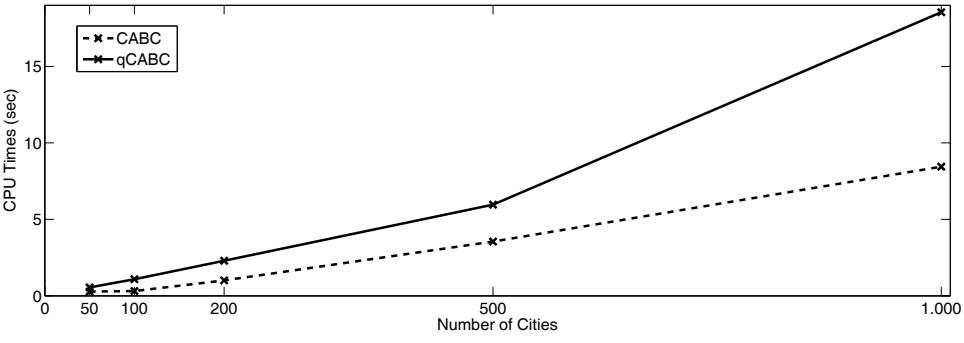

Fig. 10. Mean CPU times of CABC and qCABC.

4.2. *Additional study*

In order to compare the performance of CABC and qCABC to other swarm intelligence based algorithms, we present the results of ABCs and ACS in Table 17 and BCO in Table 18. Notice that, ACS and BCO which were originally introduced for searching on discrete spaces were considered in these comparisons. We used the same results as in Table 11 for CABC and qCABC.

Table 17 reports the best tour lengths and number of tours required to find the best tour (in parentheses) for ACS, CABC and qCABC. The results are taken from Ref. 10 for ACS. As seen in Table 17, qCABC produces better solutions

Table 17. Comparison of ACS, CABC and qCABC.

Problems	ACS	CABC	qCABC
KROA100	***21282***	***21282***	***21282***
	(4820)	(61688)	**(1680)**
D198	15888	15825	**15820**
	(585000)	(408388)	**(269680)**
PCB442	51268	51309	**51240**
	(595000)	**(145440)**	(339055)
FL1577	22977	**22703**	22742
	(942000)	(651242)	**(328763)**

Table 18. Comparison of BCO, CABC and qCABC.

Problems		BCO	CABC	qCABC
BERLIN52	Best Err. (%)	**0**	**0**	**0**
	Ave. Err. (%)	**0**	**0**	**0**
KROA100	Best Err. (%)	0.23	**0**	**0**
	Ave. Err. (%)	0.65	0.0423	**0.0113**
KROB150	Best Err. (%)	1.66	0.3100	**0.2411**
	Ave. Err. (%)	2.22	**0.6950**	0.7160

with smaller number of tour constructions than ACS for all considered problems. Also comparing the results of CABC, better solutions with smaller number of tour constructions are produced by qCABC for most of the problems except Fl1577. For KroA100 problem, although all of the algorithms give the optimum tour length, CABC algorithm finds it by constructing more tours. Examining the table, it can be said that, CABC is more successful than ACS for D198 and Fl1577 problems. However, for Pcb442, ACS performs better than CABC.

Table 18 presents the best and average (%) errors of BCO, CABC, and qCABC algorithms for Berlin52, KroA100 and KroB150 problems. We used the results of BCO given in Ref. 20. In Ref. 20, number of cycles is 10 000 and colony size is equal to total number of cities for BCO algorithm (520 000 evaluations for Berlin52, 1 000 000 evaluations for Kroa100 and 1 500 000 evaluations for Krob150). Since the parameters of CABC and qCABC algorithms are set as in Table 2, evaluation number of qCABC and CABC is 800 000 for all problems. Although ABCs are run for more evaluations (800 000), they found the optimum tour length for all runs much earlier than 520 000 evaluations for Berlin52 problem. For KroA100 and KroB150 problems, CABC and qCABC algorithms demonstrate much better performances than BCO algorithm.

The performance of ABCs is generally better than well known algorithms, GA and its variants, ACS and BCO for considered test problems. Consequently, it can be concluded that CABC and qCABC can be employed to solve TSP type problems, effectively.

5. Conclusions

In this study, two different variants of ABC approach are explained for TSP. A combinatorial version of standard ABC, called CABC algorithm and an improved version of CABC algorithm, named qCABC. The experimental studies were carried out on a set of 15 TSP test problems. Then the performance of these two algorithms and eight different GA variants are compared. Additionally, the performance of ABCs was also compared with ant colony system (ACS) and bee colony optimization (BCO) algorithms which were originally proposed for combinatorial type problems. Also, some analyses on the CPU times of CABC and qCABC algorithms and some parameter tuning studies of *limit* are provided. When the results in the comparison tables were examined, it is clearly understood that both CABC and qCABC algorithms can solve the problems quite successfully. The convergence performance of CABC is improved by applying the qABC idea for TSP.

Acknowledgment

This work was supported by Research Fund of the Erciyes University, Project Number: FBA-11-3536.

References

1. G. Laporte, The traveling salesman problem: An overview of exact and approximate algorithms, *European Journal of Operational Research* **59**(2) (1992) 231–247.
2. P. Miliotis, Using cutting planes to solve the symmetric travelling salesman problem, *Mathematical Programming* **15**(1) (1978) 177–188.
3. M. Padherg and R. Rinaldi, Optimization of a 532-city symmetric travelling salesman problem by branch and cut, *Operations Research Letters* **6**(1) (1987) 1–7.
4. O. Martin, S. Otto and E. Felten, Large-step Markov chains for the traveling salesman problem, *Complex Systems* **5**(3) (1991) 299–326.
5. G. A. Croes, A method for solving traveling salesman problems, *Operations Research* **6**(6) (1958) 791–812.
6. G. Gutin, A. Yeo and A. Zverovich, Traveling salesman should not be greedy: Domination analysis of greedy-type heuristics for the TSP, *Discrete Applied Mathematics* **117**(1–3) (2002) 81–86.
7. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science* **220**(4598) (1983) 671–680.
8. F. Glover, Tabu search — A tutorial, *Interfaces* **20**(4) (1990) 74–94.
9. Y. Nagata and S. Kobayashi, A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem, *Informs Journal on Computing* **25**(4) (2013) 346–363.
10. M. Dorigo and L. M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* **1**(1) (1997) 53–66.
11. J. Kennedy and R. Eberhart, A novel max-min ant system algorithm for traveling salesman problem, in *Proc. of the IEEE Int. Conf. on Neural Network*, Vol. 4 (1995), pp. 1942–1948.
12. R. Storn and K. Price, Differential evolution — A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11**(4) (1997) 341–359.
13. D. Karaboga, An idea based on honey bee swarm for numerical optimization, Tech. Rep. TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005).
14. Z. Zhang and Z. Feng, A novel max-min ant system algorithm for traveling salesman problem, in *Proc. of the Intelligent Computing and Intelligent Systems* (*ICIS 2009*) (2009), pp. 508–511.
15. R. Gan, Q. Guo, H. Chang and Y. Yi, Improved ant colony optimization algorithm for the traveling salesman problems, *Journal of Systems Engineering and Electronics* **21**(2) (2010) 329–333.
16. S. Ilie and C. Badica, Effectiveness of solving traveling salesman problem using ant colony optimization on distributed multi-agent middleware, in *Proc. of the Int. Multiconference on Computer Science and Information Technology* (2010), pp. 197–203.
17. A. Puris, R. Bello and F. Herrera, Analysis of the efficacy of a two-stage methodology for ant colony optimization: Case of study with TSP and QAP, *Expert Systems with Applications* **37**(7) (2010) 5443–5453.
18. M. Tuba and R. Jovanovic, Improved ACO algorithm with pheromone correction strategy for the traveling salesman problem, *International Journal of Computers Communications & Control* **8**(3) (2013) 477–485.
19. L. P. Wong, M. Y. H. Low and C. S. Chong, A bee colony optimization algorithm for traveling salesman problem, in *Proc. of the Second Asia Int. Conf. on Modelling & Simulation* (*AMS 2008*) (2008), pp. 818–823.

20. L. P. Wong, M. Y. H. Low and C. S. Chong, Bee colony optimization with local search for traveling salesman problem, in *Proc. of the Industrial Informatics* (*INDIN 2008*) (2008), pp. 1019–1025.
21. Y. He, Y. Qiu, G. Liu and K. Lei, A parallel adaptive tabu search approach for traveling salesman problems, in *Proc. of the IEEE Int. Conf. on Natural Language Processing and Knowledge Engineering* (2005), pp. 796–801.
22. N. Yang, P. Li and B. Mei, An angle-based crossover tabu search for the traveling salesman problem, in *Proc. of the Int. Conf. on Natural Computation* (2007), pp. 512–516.
23. J. W. Pepper, B. L. Golden and E. A. Wasil, Solving the traveling salesman problem with annealing-based heuristics: A computational study, *Systems, Man and Cybernetics, Part A: Systems and Humans* **32**(1) (2002) 72–77.
24. J. D. Wei and D. T. Lee, A new approach to the traveling salesman problem using genetic algorithms with priority encoding, in *Proc. of the Congress on Evolutionary Computation* (2004), pp. 1457–1464.
25. W. L. Zhong, J. Zhang and W. N. Chen, A novel discrete particle swarm optimization to solve traveling salesman problem, in *Proc. of the IEEE Congress on Evolutionary Computation* (2007), pp. 3283–3287.
26. Z. Yuan, L. Yang, Y. Wu, L. Liao and G. Li, Chaotic particle swarm optimization algorithm for traveling salesman problem, in *Proc. of the IEEE Int. Conf. on Automation and Logistics* (2007), pp. 1121–1124.
27. R. S. Prado, R. C. P. Silva, F. G. Guimaraes and O. M. Neto, Using differential evolution for combinatorial optimization: A general approach, in *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics* (*SMC 2010*) (2010).
28. J. G. Sauer and L. dos S. Coelho, Discrete differential evolution with local search to solve the traveling salesman problem: Fundamentals and case studies, in *Proc. of the 7th IEEE Int. Conf. on Cybernetic Intelligent Systems*, eds. R. Comley, B. Amavasai, X. Cheng, M. O'Grady, C. Huyck and N. Siddique (2008), pp. 299–304.
29. X. Wang and G. Xu, Hybrid differential evolution algorithm for traveling salesman problem, in *Proc. of the CEIS 2011*, eds. C. Ran and G. Yang, **15** (2011).
30. D. Karaboga, Artificial bee colony (ABC) algorithm, *Scholarpedia* **5**(3) (2010) 6915.
31. D. Karaboga, B. Gorkemli, C. Ozturk and N. Karaboga, A comprehensive survey: Artificial bee colony (ABC) algorithm and applications, *Artificial Intelligence Review* (2012).
32. J. C. Bansal, H. Sharma and S. S. Jadon, Artificial bee colony algorithm: A survey, *International Journal of Advanced Intelligence Paradigms* **5**(1/2) (2013) 123–159.
33. A. L. Bolaji, A. T. Khader, M. A. Al-Betar and M. A. Awadallah, Artificial bee colony algorithm, its variants and applications: A survey, *Journal of Theoretical & Applied Information Technology* **47**(2) (2013) 434–459.
34. B. Kumar and D. Kumar, A review on artificial bee colony algorithm, *International Journal of Engineering and Technology* **2**(3) (2013) 175–186.
35. K. Balasubramani and K. Marcus, A comprehensive review of artificial bee colony algorithm, *International Journal of Computers & Technology* **5**(1) (2013) 15–28.
36. L. Li, Y. Chong, L. Tan and B. Niu, A discrete artificial bee colony algorithm for TSP problem, in *Proc. of the 7th Int. Conf. on Intelligent Computing* (*ICIC*) (2011), pp. 566–573.
37. W. Li, W. Li, Y. Yang, H. Liao, J. L. Li and X. Zheng, Artificial bee colony algorithm for traveling salesman problem, in *Proc. of the Int. Conf. on Advanced Design and Manufacturing Engineering* (*ADME 2011*) (2011), pp. 2191–2196.

38. D. Karaboga and B. Basturk, On the performance of artificial bee colony (ABC) algorithm, *Applied Soft Computing* **8**(1) (2008) 687–697.
39. M. Albayrak and N. Allahverdi, Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms, *Expert Systems with Applications* **38**(3) (2011) 1313–1320.
40. D. Karaboga and B. Gorkemli, A combinatorial artificial bee colony algorithm for traveling salesman problem, in *Proc. of the INISTA 2011: Int. Symp. on Innovations in Intelligent Systems and Application* (2011), pp. 50–53.
41. D. Karaboga and B. Gorkemli, A quick artificial bee colony -qABC- algorithm for optimization problems, in *Proc. of the INISTA 2012: Int. Symp. on Innovations in Intelligent Systems and Applications* (2012).
42. D. Karaboga and B. Gorkemli, A quick artificial bee colony (qABC) algorithm and its performance on optimization problems, *Applied Soft Computing* **23** (2014) 227–238.
43. B. Gorkemli and D. Karaboga, Quick combinatorial artificial bee colony -qCABC- optimization algorithm for TSP, in *Proc. of the Second Int. Symp. on Computing in Informatics and Mathematics* (*ISCIM 2013*) (2013).
44. D. Karaboga and B. Akay, A comparative study of artificial bee colony algorithm, *Applied Mathematics and Computation* **214**(1) (2009) 108–132.