Optik 127 (2016) 2771–2779

Contents lists available at ScienceDirect

Optik

journal homepage: www.elsevier.de/ijleo

A novel artificial bee colony algorithm for the QoS based multicast route optimization problem

Jiaxu Ning, Changsheng Zhang, Bin Zhang*

College of Information Science & Engineering, Northeastern University, Shenyang 110819, PR China

ARTICLE INFO

Article history:
Received 21 June 2015
Accepted 24 November 2015

Keywords:
ABC
Pareto dominance
QoS
Route optimization
Multi-objective

ABSTRACT

The QoS based multicast route optimization is a key issue of the network communication, which is a NP-hard multi-objective optimization problem. In this paper, an efficient multi-objective artificial bee colony optimization algorithm based on Pareto dominance called NSABC (Non-dominated Sorting Artificial Bee Colony) is proposed to tackle this problem. A novel strategy is defined for evaluating the non-dominated food sources. The proposed algorithm is evaluated on a set of different scale real and synthesized test problems and compared with the recently proposed related algorithms for this problem. The experimental results reveal very encouraging results in terms of the solution quality and the processing time required.

1. Introduction

The increasing demand of real-time wireless communication has led to the needs of quality-of-service (QoS) based routing. Most of the wireless multimedia applications require strict QoS guarantee during the communications. This gives rise to the need for an efficient QoS oriented routing strategy used to determine the optimal multicast paths that satisfies a set of constraints. The objective functions related to cost, delay and free space loss are appropriated for selecting the most satisfactory multicast routes in many wireless communication network optimization problems and should be considered simultaneously [1]. Therefore, it is an intrinsically multi-objective optimization problem.

Due to the computational complexity, the most related researches are concentrated on heuristic-based algorithms especially the meta-heuristic approaches aiming to find near-optimal solutions. The SPEA algorithm was applied for this problem by Donoso and Fabregat [2]. A strength Pareto evolutionary algorithm based multi-objective optimization algorithm was proposed for this problem in 2011 by Potti and Chinnasamy [3]. Then, a non-dominated sorting based multi-objective genetic algorithm was proposed for this problem in 2012 by Chitra and Subbaraj [4]. These algorithms are all based on genetic algorithm, and little research has been done for using other recently proposed meta-heuristic algorithms. As a novel meta-heuristic approach, the ABC algorithm is defined by Karaboga and Basturk [5], motivated by the intelligent behavior of honey bees and has been applied to solve many problems and obtained satisfying results [6–11]. But, the most existing ABC based algorithms focus on single objective optimization problem and little research has been done to apply this algorithm to discrete multi-objective optimization problem.

In this paper, the ABC algorithm is extended and a Pareto-based multi-objective artificial bee colony algorithm called NSABC is proposed to solve the QoS based multicast route optimization problem. Different foraging strategies and selecting rules for the different kinds of bees are designed, and a novel method for computing the food source chosen probability used by onlookers is designed, which takes for each individual into account how many individuals it dominates and it is dominated by other candidates. The proposed algorithm was evaluated on a set of different scale test problems and compared with the recently proposed algorithms proposed by Donoso and Fabregat [2], Potti and Chinnasamy [3] and Chitra and Subbaraj [4]. This paper is organized as follows. In Section 2, we give the definition of the QoS based multi-objective route optimization problem and an introduction of the artificial bee colony algorithm. The NSABC algorithm including its model and concrete algorithm description is provided in Section 3. The experiments and comparative studies are given in Section 4. Finally, Section 5 summarizes the contribution of this paper along with some future research directions.

* Corresponding author. Tel.: +086 024 83688338.
E-mail address: paper820@sohu.com (B. Zhang).

http://dx.doi.org/10.1016/j.ijleo.2015.11.223
0030-4026/© 2015 Elsevier GmbH. All rights reserved.

2. The QoS based multicast route optimization problem and ABC algorithm

2.1. Problem definition

For a given wireless network which is modeled as a undirected graph $G=(N,E)$, where N is the set of nodes and E is the set of links, the QoS based route optimization problem in this paper is to find a multicast tree from the source node $s\in N$ to the destination nodes set T that meets different optimization criteria and satisfies the specified constraints. The attribute delay, cost and free space loss related criteria are considered simultaneously in this paper, and the problem is formulated as follows:

$$\text{Min}:\quad f(p)=(f_1(p),f_2(p),f_3(p))$$

where

$$f_1(p)=\sum_{t\in T}\sum_{(i,j)\in E} d_{ij}\times x_{ij}^t \tag{1}$$

$$f_2(p)=\sum_{(i,j)\in E} w_{ij}\times \max\left(x_{ij}^t\right)_{t\in T} \tag{2}$$

$$f_3=\max\left(\left(\frac{4\pi l_{ij}}{\lambda_{ij}}\right)^2\times\max\left(x_{ij}^t\right)_{t\in T}\right)_{(i,j)\in E} \tag{3}$$

Subject to the constraints:

$$\sum_{(i,j)\in E} x_{ij}^t=1,\ i=s,t\in T$$

$$\sum_{(i,j)\in E} x_{ij}^t=1,j=d,t\in T$$

$$\sum_{(i,j)\in E} x_{ij}^t-\sum_{(j,i)\in E} x_{ji}^t=0,\quad i\neq s,j\neq d,t\in T$$

The f_1, f_2 and f_3 are the objective functions related to the delay, cost and free space loss attributes respectively. The p denotes the multicast tree from node s up to destination nodes set T. The w_{ij} and d_{ij} denote the cost and delay time of the link $(i,j)\in E$ respectively. The wavelength and the distance between the antennas i and j are represented as λ_{ij} and l_{ij}. The $x^t{}_{ij}$ is a binary variable, that takes the value of 1 if the link $(i,j)\in E$ is used to transport data stream from the source node s to the destination t, $t\in T$; otherwise, it takes the value of 0.

2.2. The ABC algorithm

Artificial bee colony (ABC) is one of the most recently defined algorithms by Karaboga and and Basturk [5], motivated by the intelligent forage behavior of honey bees. In ABC algorithm, the colony of artificial bees consists of three groups of bees: employed bees, onlookers and scouts. A food source represents a possible solution to the problem to be optimized. The nectar amount of a food source corresponds to the quality of the solution represented by that food source. For every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources around the hive. The employed bee whose food source has been abandoned by the bees becomes a scout.

As other social foragers, bees search for food sources in a way that maximizes the ration E/T where E is the energy obtained and T is the time spent for foraging. In the case of artificial bee swarms, E is proportional to the nectar amount of food sources discovered by bees. In a maximization problem, the goal is to find the maximum of the objective function $F(\theta)$, $\theta\in R^P$. Assume that θ_i is the position of the ith food source; $F(\theta_i)$ represents the nectar amount of the food source located at θ_i and is proportional to the energy $E(\theta_i)$. Let $P(c)=\{\theta_i(c)|i=1,2,\ldots,S\}$($c$: cycle, S: number of food sources being visited by bees) represent the population of food sources being visited by bees.

As mentioned before, the preference of a food source by an onlooker bee depends on the nectar amount $F(\theta)$ of that food source. As the nectar amount of the food source increases, the probability with the preferred source by an onlooker bee increases proportionally. Therefore, the probability with the food source located at θ_i will be chosen by a bee can be expressed as

$$P_i=\frac{F\left(\theta_i\right)}{\sum_{k=1}^{S}F\left(\theta_k\right)} \tag{4}$$

After watching the dances of employed bees, an onlooker bee goes to the region of food source located at θ_i by this probability and determines a neighbor food source to take its nectar depending on some visual information, such as signs existing on the patches. In other words, the onlooker bee selects one of the food sources after making a comparison among the food sources around θ_i. The position of the selected neighbor food source can be calculated as $\theta_i(c+1)=\theta_i(c)\pm\phi_i(c)$. $\phi_i(c)$ is a randomly produced step to find a food source with more nectar around θ_i. $\phi(c)$ is calculated by taking the difference of the same parts of $\theta_i(c)$ and $\theta_k(c)$ (k is a randomly produced index) food positions. If the nectar amount $F(\theta_i(c+1))$ at $\theta_i(c+1)$ is higher than that at $\theta_i(c)$, then the bee goes to the hive and share her information with others and the position $\theta_i(c)$ of the food source is changed to be $\theta_i(c+1)$, otherwise $\theta_i(c)$ is kept as it is.

Every food source has only one employed bee. Therefore, the number of employed bees is equal to the number of food sources. If the position θ_i of the food source i cannot be improved through the predetermined number of trials "*limit*", then that food source θ_i is abandoned by its employed bee and then the employed bee becomes a scout. The scout starts to search a new food source, and after finding a new source, the new position is accepted to be θ_i. Every bee colony has scouts that are the colony's explorers. The explorers do not have any guidance while looking for food. They are primarily concerned with finding any kind of food source. As a result of such behavior, the scouts are characterized by low search costs and a low average in food source quality. Occasionally, the scouts can accidentally discover rich, entirely unknown food sources. In the case of artificial bees, the artificial scouts could have the fast discovery of the group of feasible solutions as a task.

It is clear from the above explanation that there are four control parameters used in the ABC algorithm: The number of food sources which is equal to the number of employed bees (S), the value of *limit* and the maximum cycle number (MCN). The main steps of the algorithm can be described as follows:

Step 1. Initialize the population of solutions θ_i, $i=1,\ldots,S$ and evaluate them.

Step 2. Produce new solutions for the employed bees, evaluate them and apply the greedy selection process.

Step 3. Calculate the probabilities of the current sources with which they are preferred by the onlookers.

Step 4. Assign onlooker bees to employed bees according to probabilities, produce new solutions and apply the greedy selection process.

Step 5. Stop the exploitation process of the sources abandoned by bees and send the scouts in the search area for discovering new food sources, randomly.

s	n_{11}	n_{12}	n_{13}	…	t_1
s	n_{21}	n_{22}	n_{23}	…	t_2
s	n_{31}	n_{32}	n_{33}	…	t_3

Fig. 1. Multicast tree.

Step 6. Memorize the best food source found so far.

Step 7. If the termination condition is not satisfied, go to *Step 2*, otherwise stop the algorithm.

After each candidate source position being produced and evaluated by the artificial bee, its performance is compared with that of its old one. If the new food has an equal or better nectar amount than the old one, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory. In other words, a greedy selection mechanism is employed as the selection operation between the old and the candidate one.

3. The NSABC algorithm

Since the basic ABC algorithm is proposed and used to solve the continuous single objective optimization problems, it cannot be directly applied to solve the discrete multi-objective route optimization problem. From Section 2.2, we know that there exists a population of individuals (bees) in the ABC algorithm. Each individual consists of an encoding of a candidate solution (food source) and a fitness that indicates its quality. In order to apply it to solve the route optimization problem, we have used an encoded variable integer matrix to represent a candidate solution, which denotes a multicast tree from the source node s to the destination nodes set T expressed as Fig. 1, where T contains three destination nodes – t_1, t_2, t_3. In order to generate a feasible multicast tree during search, a random depth-first searching process called *Search_Path* is proposed and used to find a path from the node s to the destination node t_i, $t_i \in T$, after finding these paths for the destination nodes set T, then using *Spanning_Tree* algorithm strategy to merge these paths into a multicast tree. In searching path process a mark array is used to store some nodes which have been added to the path. Then a node that connects with the current node n_i is selected randomly. If there is no nodes connecting with n_i, then delete n_i from the path, mark the n_i as a no access node and mark the link between n_i and n_{i-1} in the path disconnected. Because the network is connected, the algorithm will be able to find a path eventually. The detail description of this process and the spanning tree algorithm process are as follows:

```
Algorithm Search_Path(s,t)
Input:
    N:            The number of the networks nodes
    Adj[N][ N]:   The adjacent matrix
Output:
    path:         A path from the node s up to the node t
Begin
    Mark[s]=1;              //The mark array, marks the nodes have been added to the path
    while(path.last_node!=t) do{
        node=path.last_node;
        for i=0 to N-1 do{
            if(Mark[i]==0 && Adj[node][i]==1){
                SelectSet.add(i);   /*SelectSet stores the nodes connecting with
                                      the current node and out of the path*/
            }endif
        }endfor
        if(SelectSet.size==0) {
            node1=path.last_node;
            path.pop(node1);
            node2=path.last_node;
            Mark[node1]=0;
            Adj[node1][node2]=Adj[node2][node1]=0;
            } else{
                pos=rand(0, SelectSet.size-1);
                Mark[SelectSet [pos]]=1;
                path.add(SelectSet [pos]);
        } endif
    }endwhile
End.
Algorithm Spanning_Tree(s,T)
Input:
    T:            The destination nodes set.
    N:            The number of the networks nodes.
    Adj[N][ N]:   The adjacent matrix.
Output:
    Tree:         A multicast tree from the node s to the destination nodes set T.
Begin
    for each t∈T do{
        path=Search_Path(s,t);
        path_set.add(path);         // path_set stores these paths to destination nodes set T.
    }endfor
    flag=0;                         //a mark
    for i=0 to path_set.size-1 do{
        for j=0 to path_set[i].size-2 do{
            edge.node1= path_set[i][j];
            edge.node2= path_set[i][j+1];
            if edge.node1 ∉ Tree||edge.node2 ∉ Tree{
                Tree.add(edge);
            }endif
            if Tree contains each t∈T{
                flag=1;
                break;
            }endif
        }endfor
        if (flag==1){
            break;
        }endif
    }endfor
End
```

Furthermore, since more than one objective is considered, the way to compare two feasible solutions described in Section 2.2 and the way to computing the choosing probability used by onlookers with the food source located at θ_i defined by Eq. (4) cannot be directly used. In order to solve these problems, the dominance concept [12] is introduced and defined as follows:

Definition1 *((Dominance)).* For the two feasible solutions X and Y of a minimization optimization problem: min $F=(f_1, f_2, f_3, \ldots, f_k)$, say X dominate Y ($X \prec Y$), *iff* $\forall i, f_i(X) <= f_i(Y)$, and $\exists i, f_i(X) < f_i(Y)$, $i \in [1,k]$.

Based on above definition, we can see that more than one solution that is not dominated by others may exist at the same time. To preserve these candidates and use them to provide valuable information to guide the foraging process, a variable archive *Arch* is used to hold them. During each generation, the new discovered food sources are used to update the archive based on the dominance. To evaluate the current food sources S and compute their attractions for onlookers, a novel strategy is proposed, which considers both dominating and dominated candidates coming from the current food sources and the archive. For each current food source $\theta_i \in S$, its choosing probability $P(\theta_i)$ is computed as follows:

$$P\left(\theta_i\right) = \frac{1}{R(\theta_i)+D(\theta_i)} \quad (5)$$

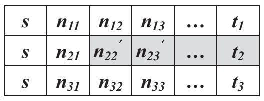
$$D\left(\theta_i\right)=\frac{1}{2+\sum_{k=1}^{\sqrt{|S\cup Arch|}}\left\|f\left(\theta_i\right)-f\left(a_k\right)\right\|},\quad a_k\in S\cup Arch \tag{6}$$

$$R\left(\theta_i\right)=\sum_{\beta\prec\theta_i}\left|\left\{\omega|\beta\prec\omega\wedge\omega\in S\right\}\right|,\ \beta\in S\cup Arch \tag{7}$$

where $D(\theta_i)$ is used to evaluate the candidates density around θ_i based on the $\sqrt{|S\cup Arch|}$ nearest neighbors, and $\left(\theta_i\right)$ is used to measure the importance of θ_i in terms of how many candidates it is dominated by and dominated. The ||.|| denotes the distance of two candidates in the objective space. It is important to note that if θ_i is not dominated by others, the value $\left(\theta_i\right)$ is zero, while a high $\left(\theta_i\right)$ value means that θ_i is dominated by many individuals. Obviously, $D(\theta_i)<1$, so through this way, the less a candidate is dominated by others, the more is its choosing probability, and for two candidates with same R value, the one with smaller D value has a higher choosing probability. The runtime of the choosing probability computation procedure is dominated by the density estimation $O(M^2\log M)$, and $M=|S\cup Arch|$. Furthermore, the objective value must be normalized when calculating the choosing probability in the proposed algorithm.

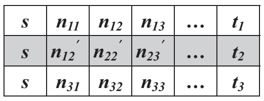
In the proposed algorithm, onlooker does not fly the food source chosen by it, the NSABC algorithm uses the following strategy: Selecting a destination node t_i randomly from the multicast tree that corresponding to the food source θ_i, Selecting a node n_j randomly from the path that is from the source node s to the destination node t_i, reserving the nodes before n_j, and continuing to search a new path start from the node n_j using the *Search_Path* process(Fig. 2), after finishing it, then using the *Spanning_Tree* algorithm strategy to form a new multicast tree.

In order to produce a candidate food position based on the current food source θ_i for the employed bees, the NSABC algorithm uses the following strategy: Selecting a destination node t_i randomly from the multicast tree that corresponding to the food source θ_i, using the *Search_Path* process to find a new path from the source node s up to the destination node t_i(Fig. 3), after finishing it, then using the *Spanning_Tree* algorithm strategy to form a new multicast tree. If the new food source dominates the old one, the new one will replace the old one, and its trial will be changed to 0. If the old one dominates the new one, the employed bee will continue to use the old one and its trial will be increased by 1. If they do not dominate each other, the new one will replace the old one, but its trial will not be changed. This greedy selection process is described as follows:



s	n_{11}	n_{12}	n_{13}	…	t_1
s	n_{21}	n_{22}'	n_{23}'	…	t_2
s	n_{31}	n_{32}	n_{33}	…	t_3

Fig. 2. Onlooker exploitation.



s	n_{11}	n_{12}	n_{13}	…	t_1
s	n_{12}'	n_{22}'	n_{23}'	…	t_2
s	n_{31}	n_{32}	n_{33}	…	t_3

Fig. 3. Employed bee exploitation.

```
Algorithm Greedy_Selection($\theta_i$, $\theta_g$)
Input:
    $\theta_i$:   The original food source used to generated $\theta_g$;
    $\theta_g$:   The new generated food source;
    S:     The set of food sources corresponding to the current employed bees;
    Trial: An array used to count the trial value of the food sources;
    Arch:  A variable archive used to hold the best solutions ever found
Output:
    $S_{new}$: The set of obtained food sources after this greedy selection
Begin
    if ($\theta_i$ dominate $\theta_g$) {
        $S_{new}$=S;            //$\theta_i$ is better than $\theta_j$
        Trial[i] = trial[i]+1;
    }else
    {
        Update the Arch using $\theta_g$ based on the defined dominance relationship;
        if($\theta_g$ dominate $\theta_i$){
            Replace $\theta_i$ in S with$\theta_g$ to form the $S_{new}$;
            Trial[i] = 0;
        }else{
            $S_{new}$=S;
            Replace $\theta_i$ in $S_{new}$ with$\theta_g$;
        }endif
    }endif
End
```

In real bee colony, the employed bee whose food source has been exhausted becomes a scout. If a scout discovered a rich food source, it would be employed. In order to simulate this behavior of real bees, the following strategy is used in ABC: providing that a position cannot be improved further through a predetermined number of cycles which is called "limit" for abandonment, the food source is assumed to be abandoned and the corresponding employed bee becomes a scout for exploration. Since the scout does not have any guidance while looking for food, the scout will find a multicast tree from the source node s up to the destination nodes set T using the random *Search_Path* process and the *Spanning_Tree* algorithm strategy in the NSABC algorithm.

Based on the above descriptions, the flowchart of the NSABC algorithm used in this paper is shown in Fig. 4. It is clear that there exist three control parameters used in this algorithm: the swarm size M, the value of *limit* and the maximum cycle number *MCN*. Detailed description of each step is given below:

Step 1 (: Initialization). Set the control parameter values. Make the first half of the colony consist of the employed bees and the second half include the onlookers. Use the *Search_Path* process and the *Spanning_Tree* process to randomly generate a food position for each employed bees. Then, find these solutions that do not dominate each other to update the external archive.

Step2 (: Employed bees exploitation). Produce new solution for each employed bee and update the food source using the *Greedy_Selection* process. If the "limit" for abandonment is reached, the employed bee forgets its memory and becomes a scout for exploration.

Step3 (: Scouts exploration). Send each scout into the search area for discovering a new multicast tree from the source node s to the destination nodes set T. After it searches a new multicast tree, it becomes an employed bee again.

Step 4 (: Preferences computation for the current food sources). Calculate choosing probability values of the current food sources with which they are preferred by the onlookers according to Eq. (5).

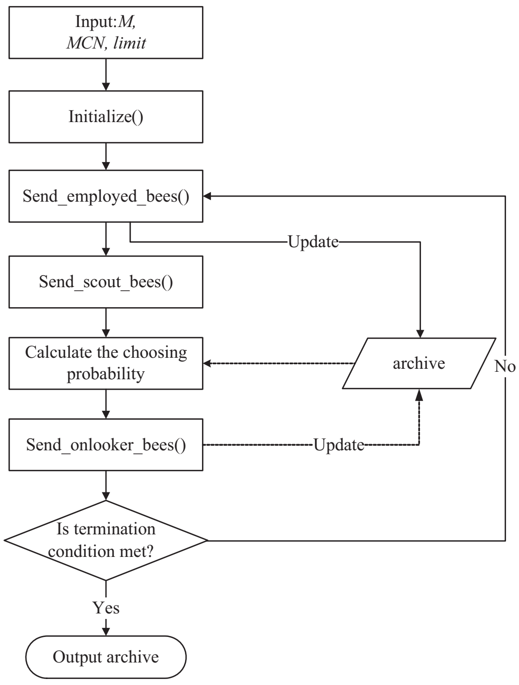


Fig. 4. The NSABC flowchart.

Step5 (: Onlookers exploitation). For each onlooker, produce a new solution from the current food sources selected depending on the computed probabilities and update the food source using the *Greedy_Selection* process.

Step 6 (: Check the termination criteria). If the termination condition is not satisfied, *go to Step 2*, otherwise stop the algorithm

4. Results and discussion

In order to evaluate the effectiveness of the proposed NSABC algorithm, it was evaluated on six different scale test instances and compared with the recently proposed algorithms called SPEA1 [2] proposed in 2007, SPEA2 [3] proposed by Potti and and Chinnasamy in 2011 and NSGA [4] proposed by Chitra and Subbaraj in 2012. Two of the six test instances were constructed from the U.S.Net with 24 nodes and 43 links [13] and Japan NTT network with 55 nodes and 144 links [14], and the other four were randomly generated according to the methods used by Donoso and Fabregat [2]. In this paper, these test instances are named based on its number of nodes and links, and called G(24,43), G(20, 72),G(40,140),G(55,144), G(70, 280) and G(100, 568) respectively. For conveniently, if not specially pointed, the swarm size is set as 150, the maximum generation is set as 4000 during comparisons for all the algorithms, and the value of the threshold "*limit*" of the NSABC algorithm are set as 8. The other parameters of the compared algorithms are set the same as in their original papers. All algorithms are implemented in C++ language and executed on a Core(i7), 2.93 GHz, 2 GB RAM computer. In the next part, we first compare them from the solving efficiency and quality aspects respectively, and then discuss the effect of the threshold "*limit*" on the performance of the proposed algorithm.

4.1. Compared from the running efficiency aspect

In order to compare the two algorithm**s'** running efficiency, we run each algorithm 20 times for each test instance. The obtained maximum time, minimum time and average time (the unit is *ms*) for once run spent on each test instance is given in Table 1.

Table 1
Running time.

Scale	Algorithm	Max_Time	Min_Time	Average
G(20,72)	SPEA1	66.72	61.14	64.17
	SPEA2	43.13	42.55	42.95
	NSGA	198.75	190.79	192.79
	NSABC	36.89	35.27	36.40
G(24, 43)	SPEA1	59.23	56.95	57.59
	SPEA2	40.23	48.92	41.23
	NSGA	188.23	179.89	184.58
	NSABC	33.56	32.98	33.25
G(40,140)	SPEA1	169.20	166.10	168.15
	SPEA2	146.99	146.12	146.46
	NSGA	200.86	192.90	195.55
	NSABC	130.64	128.93	130.12
G(55, 144)	SPEA1	155.41	151.28	153.45
	SPEA2	144.58	141.89	142.58
	NSGA	192.89	190.29	191.59
	NSABC	130.56	128.49	129.58
G(70,280)	SPEA1	196.80	188.21	193.34
	SPEA2	175.60	174.32	174.88
	NSGA	211.85	202.55	205.86
	NSABC	155.81	153.82	154.92
G(100,568)	SPEA1	215.74	200.75	207.34
	SPEA2	188.65	183.46	186.90
	NSGA	226.10	194.01	208.75
	NSABC	173.22	172.15	172.61

From Table 1, we can see that the NSABC algorithm is fastest among these compared algorithms for all the test instances. This is mainly because there are two populations with the same size, the parent population and offspring population in the compared GA based algorithms. Furthermore, a clustering process is included in the SPEA1 algorithm and a non-dominated sorting process is included in the compared NSGA algorithm at each generation. So, when the swarm size is set the same of the two algorithms, these two algorithms consumed more time.

4.2. Compared from the solving quality and convergence aspects

In order to provide a quantitative assessment for the performances of the two algorithms, the metrics called IDG [15] and C-metric [16] which is made as a complementary of the IDG are used. The IDG measure is defined as follows: assume that P^* be a set of uniformly distributed points along the Pareto front in the objective space. Let A be a set of approximated points to the Pareto front. The average distance from P^* to A is computed using the following equation:

$$IGD(A,P*) = \frac{\sum\limits_{\vartheta \in P*} d(\vartheta, A)}{|P*|} \tag{8}$$

where $d(\vartheta, A)$ is the minimum Euclidean distance between ϑ and the points in A. If $|P^*|$ is large enough to represent the Pareto front very well, both the diversity and convergence of the approximated set A could be measured using $IGD(A, P^*)$. An optimization algorithm will try to minimize the value of $IGD(A, P^*)$ measure. To obtain smaller values of this measure, the approximated set A must be very close to the Pareto front and cannot miss any part of the whole Pareto front.

However, for multi-objective discrete optimization problem, the approximated sets obtained by two algorithms may have the same IDG value, but one set may contain more non-dominate candidate solutions. Obviously, the one with more non-dominate candidate solutions is better than the other. But this cannot be reflected by the IDG values of the two sets. In order to tackle this

Table 2
The comparisons based on the IGD metric.

Scale/ Algorithm	G(20, 72)	G(24, 43)	G(40, 140)	G(55,144)	G(70, 280)	G(100, 568)
SPEA1	1.1261	6.2634	2.4596	5.7143	9.6237	11.1456
SPEA2	0.2112	**0.0000**	0.3179	**0.0000**	2.0913	5.1270
NSGA	**0.0000**	3.3644	0.0486	1.1731	3.1677	9.0169
NSABC	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**1.7442**	**3.5226**

case, the C-metric is used as a complementary of the IDG metric and defined as follows:

$$C(A \prec B) = \frac{\left|\left\{a \in A | \forall b \in B, b \ \text{does not} \ \text{dominate} \ a\right\}\right|}{\left|A\right| + \left|B\right|} \tag{9}$$

In this formula, we calculate the proportion of solutions contained in set A that are not dominated by any solution contained in set B contrast to the size of all solutions. Obviously, the $C(A \prec B) > C(B \prec A)$ means A contained more effective solutions than B.

During comparison, we run each algorithm twenty times for each test instance. The solutions obtained by NSABC algorithm for each test instance are merged together first. Then make these non-dominate solutions in this set as the Pareto fronts obtained by this algorithm for this test instance and denoted as M. The same processing is applied to the solutions obtained by SPEA1, SPEA2 and NSGA for twenty runs and the Pareto fronts corresponding to these algorithms are denoted as $S1$, $S2$ and N respectively. For each test instance, in order to approximate its ideal Pareto front, the obtained Pareto fronts by each algorithm are all merged together, and the non-dominate solutions in this set are made as the ideal Pareto fronts P^*. Based on the obtained Pareto fronts M, $S1$, $S2$, N and P^* for each test instance, the values of IDG and C-metric are computed. The comparative results based on the two metrics are given in Tables 2 and 3.

From Tables 2 and 3, we can see that the IDG values obtained by NSABC algorithm are all less bigger than the IDG values obtained by others algorithms, and the values of $C(M \prec N)$, $C(M \prec S1)$ and $C(M \prec S2)$ are all bigger than the values of $C(S1 \prec M)$, $C(S2 \prec M)$ and $C(N \prec M)$. So, we can say that the Pareto front obtained by NSABC algorithm is the closer to the ideal Pareto front than the Pareto fronts obtained by other compared algorithms, and the NSABC algorithm has acquire more effective solutions than the compared algorithms for all test instances. This means the NSABC algorithm has achieved better performances for all test instances from the solving quality aspect. This can be further proved by the Pareto fronts explicitly shown in Fig. 5 on the representative test instances.

TO compare the convergence speed, for each representative test instance, we record the solutions generated at iterations 500, 1000, 1500, 2000, 2500, 3000, 3500 and 4000 during twenty time runs of these algorithms. Then, the IDG values corresponding to the iterations 500, 1000, 1500, 2000 2500, 3000, 3500 and 4000 for each algorithm are computed. Based on these IDG values, we can approximately depict the approaching process of the non-dominated solutions obtained by each algorithm to the ideal Pareto fronts shown as in Fig. 6. We can see that the NSABC algorithm has achieved fastest convergent speed on the last three representative test instances, and converged to better positions. For the test instances – G(70, 280), G(100, 568), the obtained IDG values corresponding to the SPEA algorithm show a wave state in the early iterations, for example, the iterations at 500, 1000 for instance G(70, 280), IGD(500) < IGD(1000), this is mainly because when the iterations are 500, this algorithm achieved more solutions than when the iterations are 1000; however, it has less solutions on the ideal Pareto fronts P^*.

Table 3
The comparisons based on the C metric.

	G(20, 72)				G(24, 43)				G(40, 140)				G(55, 144)				G(70, 280)				G(100, 568)			
	S1	S2	N	M	S1	S2	N	M	S1	S2	N	M	S1	S2	N	M	S1	S2	N	M	S1	S2	N	M
S1	–	0.4845	0.4796	0.4796	–	0.2667	0.2667	0.2667	–	0.4405	0.4458	0.4568	–	0.4545	0.4545	0.4545	–	0.2857	0.3043	0.2917	–	0.2308	0.3333	0.2558
S2	0.5155	–	0.4951	0.4951	0.6	–	0.5	0.5	0.5476	–	0.5055	0.5169	0.5455	–	0.5	0.5	0.6327	–	0.5254	0.4590	0.5641	–	0.5000	0.4583
N	0.5204	0.5050	–	0.5000	0.6	0.5	–	0.5	0.5421	0.4945	–	0.5114	0.5445	0.5	–	0.5	0.6087	0.4576	–	0.4310	0.4872	0.3182	–	0.3333
M	0.5204	0.5050	0.5000	–	0.6	0.5	0.5	–	0.5309	0.4831	0.4886	–	0.5454	**0.5**	**0.5**	–	0.6250	0.4918	0.5172	–	0.6047	0.4792	0.5208	–

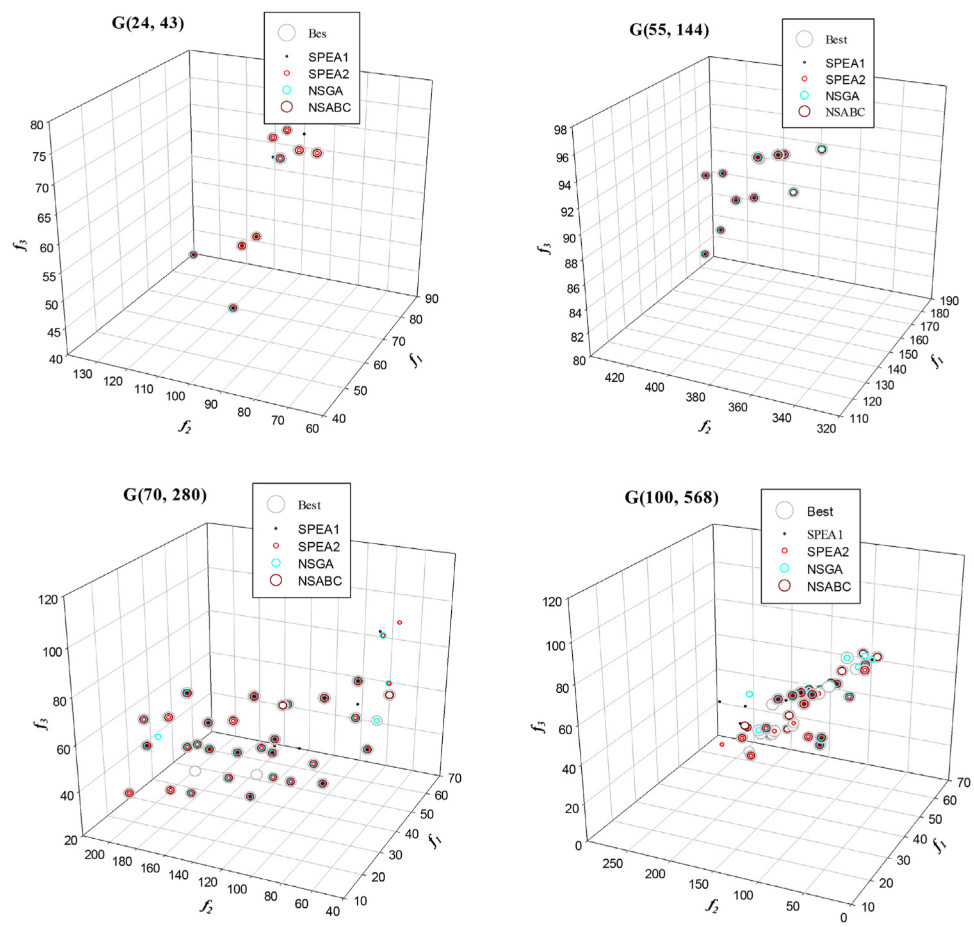


Fig. 5. The obtained Pareto fronts on different scale problems.

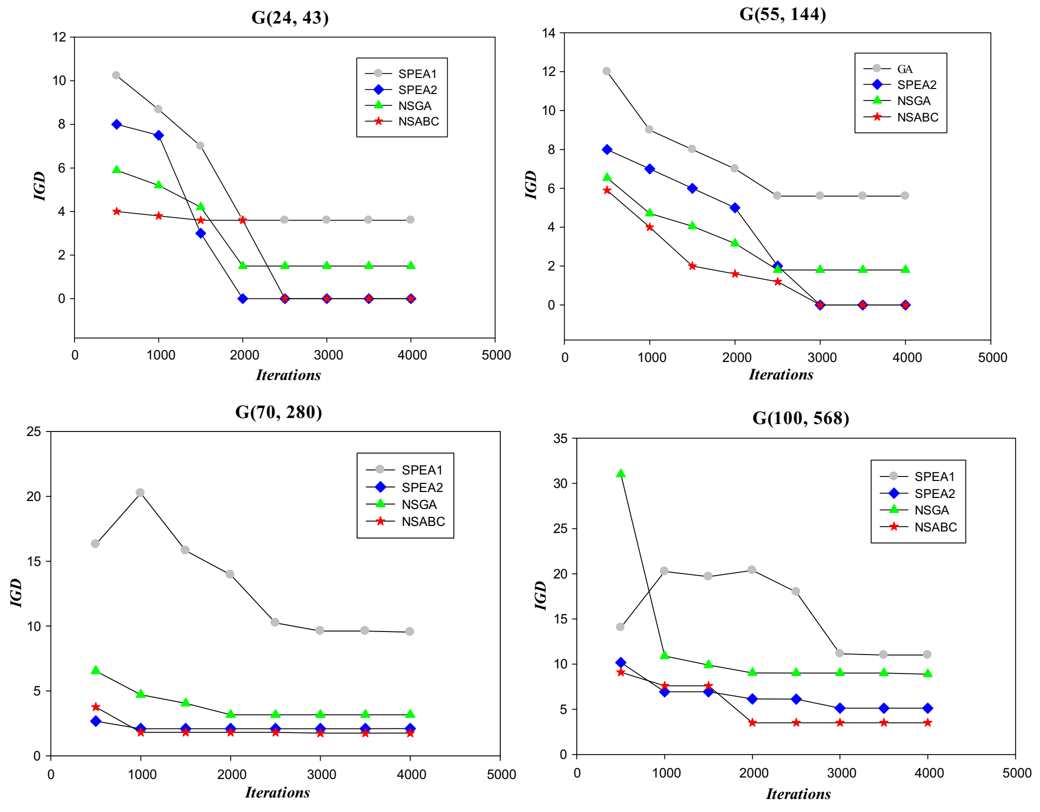


Fig. 6. The comparisons of the Pareto front convergence on different scale problem.

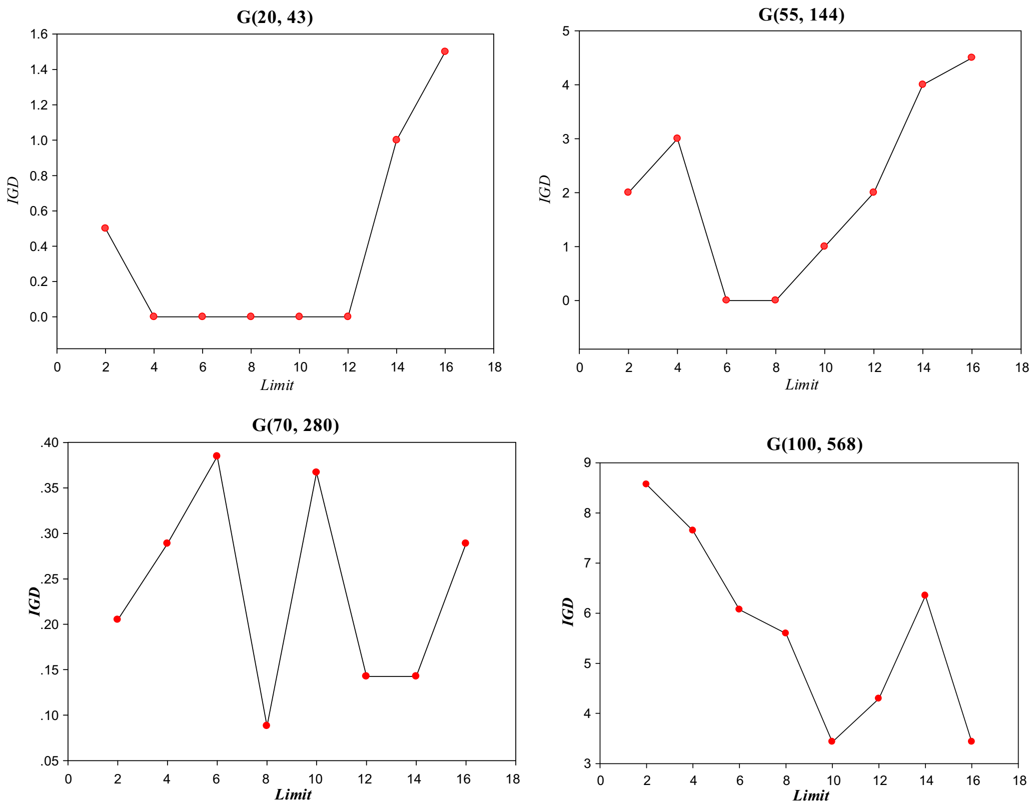


Fig. 7. The effects of the limit values on the NSABC algorithm's performance.

4.3. The effect of the parameter "limit" on the NSABC algorithm's performance

The parameter "*limit*" is very important to the NSABC algorithm. Its value will have great impact on the algorithm's performance. If the value is set too big, the employed bees will do lots of exploitations which will make the search process quickly converge to a local minimum. If the value is set too small, the employed bees do few local exploitations and quickly become scouts to do more explorations, which will decrease the algorithm's solving quality and convergent speed. In order to test the impacts of the parameter "*limit*" on the solving processes of the NSABC algorithm for our used instances, we set its thresholds as 2, 4, 6, 8, 10, 12, 14 and 16 respectively. For each different setting, we run this algorithm twenty times on the representative test instances and compute the corresponding IDG value. The results are presented in Fig. 7. We can see that this algorithm achieved the smallest IDG values for instance G(20, 43) when the "*limit*" value is set as 4, 6, 8 and 10 and 12, gets the smallest IDG value when the "*limit*" value is set as 6 and 8 for instance G(55, 144), gets the smallest IDG value when the "*limit*" value is set as 8 for instance G(70, 280) and gets the smallest IDG value when the "*limit*" value is set as 10 and 16 for instance G(100, 568). This may be because these instances possessing different difficulties.

From the above results, we can conclude that the proposed NSABC algorithm has achieved better performance than the compared GA algorithm in terms of processing time, solving quality and convergent speed for the tackled QoS based route optimization problem in this paper. Its superiority is evident and can be considered as a viable and a promising efficient way to solve the discrete multi-objective optimization problem.

5. Conclusion

Modeling the behavior of social insects, such as ants, birds or bees, for the purpose of search and problem solving has been the emerging area of swarm intelligence. In this paper, an artificial bee colony based multi-objective optimization algorithm is developed to solve the QoS based multicast route optimization problem which is inspired by the bees' forage behavior. To evaluate the performance of this algorithm, it is compared with the GA based multi-objective optimization algorithms and the experimental results reveal very encouraging results in terms of the quality of solution and the processing time required. There are a number of research directions that can be considered as useful extensions of this research. We can combine it with some local search strategy or hybrid it with other meta-heuristic algorithms properly, and apply it to solve other multi-objective optimization problems. Furthermore, how to make the threshold "*limit*" adaptively change with the running states is another problem deserving to further research.

Acknowledgments

This work was supported by NSFC Major Research Program (61100090) and the Special Fund for Fundamental Research of Central Universities of Northeastern University (N120804001, N120604003).

References

[1] M.A. Razzaque, M.H.U. Ahmed, C.S. Hong, S. Lee, QoS-aware distributed adaptive cooperative routing in wireless sensor networks, Ad Hoc Networks 19 (2014) 28–42.

[2] Y. Donso, R. Fabregat, Multi-objective Optimization in Computer Networks Using Metaheuristics, Auerbach Publications, Boca Raton, NY, 2007.
[3] S. Potti, C. Chinnasamy, Strength Pareto evolutionary algorithm based multi-objective optimization for shortest path routing problem in computer networks, J. Comput. Sci. 7 (2011) 17–26.
[4] C. Chitra, P. Subbaraj, A non-dominated sorting genetic algorithm solution for shortest path routing problem in computer networks, Expert Syst. Appl. 39 (2012) 1518–1528.
[5] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (2008) 687–697.
[6] X. Zhang, S.L. Ho, W.N. Fu, A modification of artificial bee colony algorithm applied to loudspeaker design problem, IEEE Trans. Magn. 50 (2014) 4811–4816.
[7] M. Fatih Tasgetiren, et al., A discrete artificial bee colony algorithm for the total flow-time minimization in permutation flow shops, Inf. Sci. 181 (2011) 3459–3475.
[8] P. Tsai, M.K. Khan, J. Pan, B. Liao, Interactive artificial bee colony supported passive continuous authentication system, IEEE Syst. J. 8 (2014) 395–405.
[9] D. Karaboga, C. Ozturk, A novel clustering approach: artificial bee colony (ABC) algorithm, Appl. Soft Comput. 11 (2011) 652–657.
[10] W.C. Yeh, T.J. Hsieh, Solving reliability redundancy allocation problems using an artificial bee colony algorithm, Comput. Oper. Res. 38 (2011) 1465–1473.
[11] Q.K. Pan, L. Wang, K. Mao, J.H. Zhao, M. Zhang, An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process, IEEE Trans. Autom. Sci. Eng. 10 (2013) 307–322.
[12] K. Deb, A. Pratap, S. Agarwal, A fast and elitist multi-objective genetic algorithm: GA, IEEE Trans. Evol. Comput. 6 (2002) 182–197.
[13] B. Marwan, A.D. Schupke, On routing and transmission-range determination of multi-bit-rate signals over mixed-line-rate WDM optical networks for carrier Ethernet, IEEE/ACM Trans. Networking 19 (2011) 1304–1316.
[14] J. Crichigno, B. Barán, Multi-objective multicast routing algorithm for traffic engineering, in: IEEE International Conference on Computer and Communications ICCCN'2004, Chicago, US, 2004.
[15] K. Deb, M. Mohan, S. Mishra, Evaluating the epsilon-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions, Evol. Comput. 13 (2005) 501–525.
[16] S.K. Mishra, Comparative performance evaluation of multiobjective optimization algorithms for portfolio management, in: World Congress on Nature & Biologically Inspired Computing, Coimbatore, India, 2009, pp. 1338–1342.