Appl Intell
DOI 10.1007/s10489-017-0950-z

Hybrid artificial bee colony algorithm based approaches for two ring loading problems

Alok Singh[1] · **Jayalakshmi Banda**[1]

Abstract This paper presents hybrid artificial bee colony algorithm based approaches for two $\mathcal{NP}$-hard problems arising in optical ring networks. These two problems falls under the category of ring loading problems. Given a set of data transfer demands between different pair of nodes, the first problem consists in routing the demands on the ring in either clockwise or counter-clockwise directions so that the maximum data transmitted through any link in either directions is minimized. The second problem, on the other hand, discriminates between the data transmitted in one direction from the other and consists in minimizing the maximum data transmitted in one particular direction through any link. The first problem is referred to as weighted ring edge-loading problem in the literature, whereas the latter as weighted ring arc-loading problem. Computational results on the standard benchmark instances show the effectiveness of our approaches on both the problems.

Keywords Artificial bee colony algorithms · Communication networks · Ring loading · Swarm intelligence · Weighted ring arc-loading problem · Weighted ring edge-loading problem

✉ Alok Singh
alokcs@uohyd.ernet.in

Jayalakshmi Banda
bjayalakshmi@uohyd.ac.in

[1] School of Computer and Information Sciences, University of Hyderabad, Hyderabad 500046, India

1 Introduction

Synchronous Optical NETworking (SONET) in USA and Canada, and Synchronous Digital Hierarchy (SDH) in the rest of the world are the current standards for transmitting and multiplexing high speed signals over optical fibre communication networks. Basically, SONET/SDH enforce a ring based topology where nodes are connected via a ring of optical fibre cables. Each node is equipped with an Add-Drop-Multiplexer (ADM) that acts as an interface between the node and the ring and that performs the following functions

- It sends the message originating at its associated node over the ring
- It receives the message meant for its associated node over the ring and removes that message from the ring
- It relays all other messages

SONET/SDH rings are bi-directional, i.e., a message can be transmitted in either directions (clockwise or counter-clockwise) over the ring and the messages routed through clockwise direction compete with those messages routed through counter-clockwise direction for the common bandwidth. In addition to the type of optical fibre cable, the bandwidth available along any edge of SONET/SDH ring depends on the ADM [16]. The amount of data transmitting through an edge in either directions at a particular instant is called its load at that instant. Obviously, the load on any edge can not exceed the available bandwidth. Weighted Ring Edge-Loading Problem (WRELP) is an important problem in this context that seeks to minimize the maximum load on any edge. Given a set of communication

Published online: 22 May 2017

demands between various pairs of nodes, the WRELP consists in routing the demands on the ring in either clockwise or counter-clockwise directions so that the maximum load over all the edges is minimized.

IEEE 802.17 standard for Resilient Packet Ring (RPR) combines the benefits of SONET/SDH and Ethernet networks for significantly enhancing the performance of optical fibre ring networks with regard to data traffic [12, 30, 36]. Like SONET/SDH, RPR rings are also bidirectional. However, unlike SONET/SDH where there is a single bi-directional ring, RPR consists of two distinct uni-directional rings (one clockwise and another counter-clockwise) each with its own bandwidth and the messages sent through clockwise ring do not compete with those messages sent through counter-clockwise ring for the common bandwidth. So bi-directionality, RPR is achieved by making use of these two rings while sending the messages. Clearly, in RPR, we have to deal with directed edges or arcs, where an arc, depending on its direction, belongs to clockwise or counter-clockwise ring. The amount of data transmitting through an arc at a particular instant is called its load at that instant. Weighted Ring Arc-Loading Problem (WRALP) in RPR is equivalent of WRELP in SONET/SDH. Given a set of communication demands between various pairs of nodes, the WRALP problem consists in routing the demands either through clockwise ring or counter-clockwise ring so that the maximum load over all the arcs of both the rings is minimized.

Depending on whether demands can be split or not, there are two variants of WRELP and WRALP. In the first variant, demands can be split into two parts with each part routed through a different direction, whereas the second variant does not allow splitting of demands, i.e., each demand has to be routed into through one of the two directions. The first variant can be solved in polynomial time, whereas the latter variant is $\mathcal{NP}$-Hard. For those interested in the first variant may refer to [10, 26, 27, 35, 36]. In this paper, we have considered the second variant only. WRELP and WRALP with non-split demands are referred to as non-split WRELP and non-split WRALP respectively in the literature. Hereafter, in this paper, WRELP and WRALP will refer to their respective non-split variants only even if we don't use the qualifier "non-split".

In this paper, we present artificial bee colony (ABC) algorithm based hybrid approaches for WRELP and WRALP. The best solution obtained through ABC algorithm is improved further by two local searches which are applied one after the other. The ABC algorithm is same for WRELP and WRALP, but the two local searches differ according to the problem. We have compared the performance of our hybrid approaches with four state-of-the-art approaches available in the literature on the standard benchmark instances. Computational results show the superiority of our proposed approaches over these approaches. We have also evaluated the performance of our ABC approaches without the use of two local searches and found that they are capable of finding high quality solutions on their own and local searches improves the solution quality only slightly.

The remaining part of this paper is organized in the following manner: Section 2 provides an overview of ABC algorithm, whereas Section 3 defines the two ring loading problems in a formal manner. Section 4 provides a brief survey of related works in the literature. Our hybrid ABC approaches for WRELP and WRALP are presented in Section 5. Section 6 reports the computational results and provide a comparative analysis of our hybrid approaches vis-à-vis state-of-the-art approaches available in the literature. Finally, Section 7 outlines some concluding remarks and directions for future research.

2 Overview of ABC algorithm

The artificial bee colony (ABC) algorithm is a population based metaheuristic algorithm inspired from the intelligent foraging behavior of the honey bees. It is developed by D. Karaboga in 2005 [17]. There exist three groups of bees in a colony of real bees, viz. scout, employed and onlooker. Scout bees look for new food sources in the proximity of the hive. Their status is changed to employed at the moment new food sources are discovered by them. Employed bees are responsible for exploiting a food source. They bring loads of nectar to the hive and share the information about their food sources with onlooker bees which wait in the hive. This information is used by the onlooker bees in selecting the food sources which take into consideration the information shared by several employed bees. Each onlooker choose a food source with a probability directly proportional to the quality of that food source and then it becomes employed. An employed bee whose food source is exhausted will leave that food source and change its status to either scout or onlooker.

Motivated by this intelligent behavior of foraging bees, Karaboga developed ABC algorithm with the intention of solving optimization problems in continuous domain. Later, ABC algorithm was extended for solving discrete optimization problems also. Now, numerous variants of ABC algorithm exist in the literature [1, 9, 17–21, 28, 29, 32–34]. Karaboga et al. [21] provides a good survey on ABC algorithm and its applications.

In ABC algorithm, there are also three groups of bees, viz. employed, onlooker and scout with functions similar to real bees. In ABC algorithm, the food sources represent the solutions to the problem under consideration and the quality of a food source represents the fitness of the solution being represented by that food source. Every employed bee

is assigned an unique food source, which means, the number of food sources is equal to the number of employee bees. The number of onlooker bees is also taken to be equal to number of employed bees usually (but not always). Throughout this paper, we will use food source and solution interchangeably. The ABC algorithm starts with initialization, then iterates through the cycles of the employed bee and onlooker bee phases until some stopping criteria is met.

In the initialization, each employed bee is assigned a solution which is usually generated randomly. In the employed bee phase, each employed bee determines a new food source in the vicinity of its associated food source and evaluates its quality. The manner in which a new food source is determined depends on the problem under consideration. If the new food source determined by the employee bee is of better quality in comparison to its currently associated food source, then it leaves the currently associated food source and proceeds to the new food source. If the quality of the new food source is not as good as the current one, then the employee bee remains at the currently associated food source. Once all employee bees complete this process, they initiate the onlooker bee phase by sharing the information about their food sources.

In onlooker bee phase, onlookers select food sources (employed bee solutions) as per their quality. This selection is carried out in such a manner that solutions having higher fitness will have more chance of selection. Such a selection policy results in better chances of finding good quality solutions. Then, they determine the new food sources in the vicinity of their selected food sources and compute their fitness. The onlookers determine the new food sources in the same manner as employed bees. Next, the best food source among all the new food sources determined by all the onlookers who have selected food source i is found. If this food source is better than food source i, then this best food source will be the new location of food source i for the next iteration, otherwise there is no change in the location of food source i for the next iteration. Once all food sources are updated in this manner, the onlooker bee phase ends.

The ABC algorithm repeats itself through the cycles of the afore-mentioned two phases as long as the termination condition is not met. If a solution associated with an employed bee has not improved over certain number of iterations, then that solution is discarded by the associated employed bee and it becomes scout. Immediately this scouts is turned into an employed bee by associating it with a newly generated solution. This new solution is usually generated in the same manner as an initial employed bee solution.

The success of any metaheuristic algorithm will mainly depend on two factors: exploration and exploitation. A balance between these two need to be maintained to achieve good results. In the ABC algorithm, during employed bee phase, every employed bee solution is given a fair chance to improve itself. However, in the onlooker bee phase, owing to the selection policy used by the onlookers as described already, good quality solutions get more chance to improve themselves in comparison to poor quality solutions. This bias towards good quality solutions may produce better quality solutions, as there are higher chances of finding even better solutions in the proximity of good solutions. However, if a solution is locally optimal, then no better solution exists in its proximity and any attempt to improve it is futile. The concept of scout bees helps in such a situations. Instead of determining whether a solution is locally optimal or not which involves exploring the whole neighborhood and which can be computationally expensive, the ABC algorithm assumes a solution to be locally optimal when it has not improved over certain number of iterations. To get rid of this solution, its associated employee bee is made a scout. This is equivalent to the phenomenon of abandoning an exhausted food source by an employed bee associated with that food source in real bee colonies. A locally optimal solution is treated like an exhausted food source. However, unlike the real employed bee, in case of ABC algorithm, the employed bee of a locally optimal solution always becomes scout only, but never an onlooker. As explained already, a new solution is generated for this scout bee to turn it into an employed bee again. Hence by utilizing the concept of scout bees, solutions which have not improved since long are replaced with new solutions. Clearly, in case of the ABC algorithm, exploitation is carried out by employed bees and onlooker bees, whereas scouts do the job of exploration. A proper balance is maintained between exploration and exploitation by appropriately setting the number of iterations without improvement in the ABC algorithm after which an employed bee abandons a solution and becomes scout.

3 Problem formulation

Given a n node bi-directional ring $R_n = \{n_1, n_2, \ldots, n_n\}$ such that $e_i =< n_i, n_{i+1} > \forall i \in \{1, 2, \ldots, n-1\}$ and $e_n =< n_n, n_1 >$ are edges of this ring. Each edge $e_i =< n_i, n_{i+1} >$ corresponds to two directed edges or arcs represented by ordered pairs $e_i^+ = (n_i, n_{i+1})$ and $e_i^- = (n_{i+1}, n_i)$. Likewise $e_n =< n_n, n_1 >$ corresponds to two arcs $e_n^+ = (n_n, n_1)$ and $e_n^- = (n_1, n_n)$. The direction $n_1, n_2, \ldots, n_n$ corresponds to clockwise direction and $n_n, n_{n-1}, \ldots, n_1$ corresponds to counter-clockwise direction. So, e_i^+, $\forall i \in \{1, 2, \ldots, n\}$ are arcs in clockwise direction, whereas e_i^-, $\forall i \in \{1, 2, \ldots, n\}$ are arcs in counter-clockwise direction. A set of m demands is given where each demand j is represented by a triple (s_j, d_j, w_j) where s_j is the source node, $d_j \neq s_j$ is the destination

node and $w_j > 0$ is the weight associated with the demand j, which can be interpreted as the size of the data to be transmitted or amount of traffic generated to fulfil this demand. Demands can not be split, i.e., each demand needs to be routed into one of the two directions. By associating binary variables x_j with each demand j such that $x_j = 1$, means demand j is routed through clockwise direction and $x_j = 0$ means demand j is routed through counter-clockwise direction, the load $L(e_i)$ on an edge e_i can be determined as follows: Clearly, $L(e_i) = L(e_i^+) + L(e_i^-)$ where
$L(e_i^+) = \sum_{j=1}^{m} f(i,j) \times w_j$
and
$L(e_i^-) = \sum_{j=1}^{m} g(i,j) \times w_j$
The value of $f(i,j)$ and $g(i,j)$ can be determined in the following manner:

$$f(i,j) = \begin{cases} 0 \text{ if } x_j = 0 \\ 1 \text{ if } ((i < n) \text{ and } (x_j = 1) \text{ and } ((s_j \leq n_i \text{ and } d_j > n_i) \text{ or } (s_j > d_j \text{ and } s_j > n_{i+1} \text{ and } d_j > n_i))) \\ 1 \text{ if } ((i = n) \text{ and } (x_j = 1) \text{ and } (s_j > d_j \text{ and } s_j > n_1 \text{ and } d_j \geq n_1)) \\ 0 \text{ otherwise} \end{cases}$$

and

$$g(i,j) = \begin{cases} 0 \text{ if } x_j = 1 \\ 1 \text{ if } ((i < n) \text{ and } (x_j = 0) \text{ and } ((s_j > n_i \text{ and } d_j \leq n_i) \text{ or } (s_j < d_j \text{ and } s_j < n_i \text{ and } d_j \leq n_i))) \\ 1 \text{ if } ((i = n) \text{ and } (x_j = 0) \text{ and } (s_j < d_j \text{ and } s_j < n_i \text{ and } d_j \leq n_i)) \\ 0 \text{ otherwise} \end{cases}$$

The WRELP seeks a route for each of these m demands such that the maximum load over all the edges is minimized. In other words, the objective of WRELP is to find the values of m binary variables $x_1, x_2, \ldots x_m$ in such a manner that minimizes $max(L(e_1), L(e_2), \ldots, L(e_n))$

The WRALP seeks a route for each of these m demands such that the maximum load over all the arcs is minimized. In other words, the objective of WRELP is to find the values of m binary variables $x_1, x_2, \ldots x_m$ in such a manner that minimizes $max(\ell^+, \ell^-)$, where
$\ell^+ = max(L(e_1^+), L(e_2^+), \ldots, L(e_n^+))$
and
$\ell^- = max(L(e_1^-), L(e_2^-), \ldots, L(e_n^-))$.

For both WRELP and WRALP, the size of the search space is 2^m as each of m demands can be routed through one of the two possible directions. Please note that the size of the search space is independent of the number of nodes in the ring and depends only on the number of demands.

To illustrate WRELP and WRALP, let us consider the 8 nodes ring shown in Fig. 1. Suppose 10 communication demands occur between different pairs of nodes in this ring. These demands, along with their route and values of their associated variables x_i in a routing scheme (a feasible solution) are shown below:

$(1,5,4)$	*clockwise*	$x_1 = 1$
$(2,7,3)$	*counter-clockwise*	$x_2 = 0$
$(3,7,2)$	*clockwise*	$x_3 = 1$
$(4,8,5)$	*clockwise*	$x_4 = 1$
$(5,2,7)$	*counter-clockwise*	$x_5 = 0$
$(5,8,3)$	*clockwise*	$x_6 = 1$
$(6,1,5)$	*clockwise*	$x_7 = 1$
$(6,4,4)$	*counter-clockwise*	$x_8 = 0$
$(7,5,8)$	*counter-clockwise*	$x_9 = 0$
$(8,2,3)$	*clockwise*	$x_{10} = 1$

The route for each demand is also shown in Fig. 1. In this routing scheme, loads on various arcs and edges are as follows:

$L(e_1^+) = 7$	$L(e_1^-) = 3$	$L(e_1) = 10$
$L(e_2^+) = 4$	$L(e_2^-) = 7$	$L(e_2) = 11$
$L(e_3^+) = 6$	$L(e_3^-) = 7$	$L(e_3) = 13$
$L(e_4^+) = 11$	$L(e_4^-) = 11$	$L(e_4) = 22$
$L(e_5^+) = 10$	$L(e_5^-) = 12$	$L(e_5) = 22$
$L(e_6^+) = 15$	$L(e_6^-) = 8$	$L(e_6) = 23$
$L(e_7^+) = 13$	$L(e_7^-) = 3$	$L(e_7) = 16$
$L(e_8^+) = 8$	$L(e_8^-) = 3$	$L(e_8) = 11$

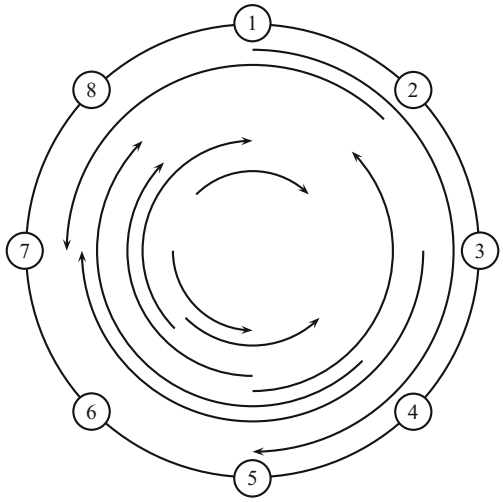

Fig. 1 Illustrating WRELP and WRALP

If the above routing scheme is used for WRELP then objective function value is 23. If the above routing scheme is used for WRALP then objective function value is 15 ($max(\ell^+, \ell^-)$, where $\ell^+ = 15$, $\ell^- = 12$).

4 Related work

Cosares and Saniee [11] introduced non-split WRELP and proved its $\mathscr{NP}$-hardness. This problem is known to be polynomially solvable only in cases where all demands are one [14]. The non-split WRALP was introduced and proved $\mathscr{NP}$-hard by Yuan et al. [36] and further studied in [10, 25].

Schrijver et al. [31] proposed a highly efficient greedy heuristic for WRELP, which returns a solution that can exceed the optimal by at most $\frac{3}{2}$ times the maximum demand and which performs much better in practice. Moreover, this heuristic finds the optimal solution when all demands are equal to one. Extending this work, Khanna [23] developed a polynomial time approximation scheme for this problem. Dell' Amico et al. [13] presented efficient lower and upper bounding procedures, and a branch-and-bound based exact algorithm for WRELP.

Among the metaheuristics techniques for WRELP and WRALP, Karunanithi and Carpenter [22] proposed a genetic algorithm for WRELP and solved the small instances of WRELP. Kim et al. [24] presented different variations of Ant Colony Optimization (ACO) Algorithm for WRELP. [2–6, 8] proposed several evolutionary algorithms based approaches and a tabu search based approach for WRELP and several evolutionary algorithms and swarm intelligence based approaches for WRALP. In particular, [5] describes an artificial bee colony algorithm based approach for WRALP. As explained in Section 5, our proposed ABC algorithm is altogether different from this ABC algorithm except for solution encoding. Further, unlike the ABC approach of [5], no local search is used inside the neighboring solution generation procedure in our ABC approach. Our neighboring solution generation procedure is based on a very simple, but effective strategy. Hence, our ABC approach is computationally more efficient.

Bernardino et al. [7] presented a genetic algorithm (GA), a hybrid differential evolution algorithm (HDEM) and a hybrid discrete particle swarm optimization algorithm (HDPSO) for WRELP and WRALP and compared the performance of these three approaches on both the problems with best performing previously proposed approaches mentioned above. The HDEM performed the best followed by ABC approach of [5] and HDPSO on both the problems. It is to be noted, that ABC approach was presented in [5] for WRALP. However, [7] presented the results of ABC approach for WRELP also. We have compared our proposed approaches with these four approaches, viz. GA, HDEM, HDPSO and ABC on the same benchmark instances as used in [7].

As mentioned already, we have applied the two local searches only on the best solution obtained through ABC algorithm. This strategy has already been used in the ABC literature, e.g. [29, 34]. However, this strategy deviates from the widely used strategy of combining local search with ABC algorithm where local search is used inside ABC algorithm to improve the neighboring solutions. Actually, applying a local search on every/some neighboring solutions inside ABC algorithm is computationally expensive in comparison to using it once on the best solution. This latter strategy is useful in circumstances where either the local search is computationally too expensive to be used inside ABC algorithm or the best solution produced by ABC algorithm without any local search is already closed to optimal/best known solution values and applying the local search only on the best solution can produce the results of desired quality. We have used the strategy of applying local search only on the best solution due to latter reason.

5 Hybrid ABC approaches for WRELP and WRALP

We have developed hybrid approaches combining artificial bee colony (ABC) algorithm with local search procedures for WRELP and WRALP. Except for fitness function, the

ABC algorithm is same for both the problems. The local search procedure consists of two local searches that are applied one after the other. These two local searches vary according to the problem. The local search procedure is applied only to the best solution obtained through ABC algorithm in a bid to further improve its quality. Hereafter, our hybrid approach for WRELP will be referred to as HABC-E, whereas our hybrid approach for WRALP will be referred to as HABC-A.

Following subsections describe other salient features of proposed approaches.

5.1 Solution encoding

We have used a bit vector of length m to represent a solution, where m is the total number of communication demands. A value of 1 at the i^{th} position indicates that demand i to be routed in clockwise direction, whereas a value of 0 at the same place indicates that demand i to be routed in counter-clockwise direction. Bernardino et al. [7] and [5] also used the same encoding.

5.2 Fitness

We have used the objective function as the fitness function. So, HABC-E uses objective function of WRELP as fitness function, whereas HABC-A uses objective function of WRALP as fitness function. As WRELP and WRALP are minimization problems, so for these problems, a lower value of the fitness function indicates a more fit solution.

5.3 Initial employed bee solutions

Among the initial employed bee solutions, the first solution is generated by following a shortest path strategy where each demand is routed through the direction where it has to traverse lesser number of links in comparison to the other direction (ties are broken arbitrarily). All other initial employed bee solutions are generated by following either a purely random strategy or a strategy that is a mix of greediness and randomness. The first strategy is used with probability ρ_{ir}, otherwise the second strategy is used. In the first strategy, each demand is routed uniformly at random in one of the two directions. In the second strategy, each demand is routed through shortest path with probability ρ_{ig}, otherwise it is routed uniformly at random in one of the two directions. ρ_{ir} and ρ_{ig} are two parameters whose values need to be determined empirically. The values for these parameters are chosen in such a manner so that the initial population of employed bee solutions is a proper mix of greediness and randomness.

The ABC approach of [5] generates initial employed bee solutions either in a completely random manner or in a completely deterministic manner. The deterministic strategy was again based on shortest path algorithm.

5.4 Policy used by onlookers to select a food source

We have used probabilistic binary tournament selection method to select a food source for each onlooker bee. This method works by choosing two different food sources uniformly at random and then comparing their fitness. With probability ρ_o, more fit food source is selected, otherwise the lesser fit food source is selected. ρ_o is a parameter to be determined empirically. The reason for using probabilistic binary tournament selection method in place of roulette wheel selection method, which is used commonly in ABC algorithm is that probabilistic binary tournament selection method performs better in general and at the same time computationally less expensive [15].

The ABC algorithm of [5] assigns $P_i \times NO$ onlookers to a food source i, where $P_i = \frac{\sum_{j=1}^{NE} F_j - F_i}{\sum_{j=1}^{NE} F_j}$, F_i is the fitness of food source i, and NE is the number of employed bees and NO is the upper limit on the number of onlookers that can be sent to any food source. Usually, $\sum_{j=1}^{NE} F_j$ is much larger than the fitness of any solution and hence P_i is near to 1 for all $i = 1, \ldots, NE$. NO is taken to be greater than NE in [5], so a large number of onlookers are deputed for every solution in their approach. In contrast, we have used total number of onlookers for all food sources to be twice as much as there are food sources.

5.5 Neighboring solution generation

Our neighboring solution generation method is based on the fact that if a demand is routed through a particular direction in one good solution, then it is highly likely that this demand is routed through the same direction in many good solutions. Hence, to generate a new solution X' in the neighborhood of a solution X, another solution Y is utilized. The solution Y is chosen randomly from all employed bee solutions other than X. Then to create X', each demand is considered one-by-one and it is routed in X' through the same direction as in Y with small probability ρ_{ns}, otherwise it is routed through the same direction as in X. Pseudo-code of neighboring solution generation process is given in Algorithm 1, where $X[i]$ refers to the i^{th} element (bit) of solution X which is a bit vector.

Algorithm 1 Pseudo-code of neighboring solution generation process

```
Input: A solution X
Output: A new solution X' in the neighborhood of X
Select a solution Y randomly from all employed bee solutions other than X;
for i ← 1 to m do
    if (u_01 < ρ_ns) then
        X'[i] ← Y[i];
    else
        X'[i] ← X[i];
return X';
```

We have applied the afore-mentioned neighboring solution generation method in both employed and onlooker bee phases. Bernardino et al. [5] used different neighboring solution generation methods in employed and onlooker bee phases. Unlike our neighboring solution generation method which does not use any form of local search, these methods do partial local search around a solution to generate its neighboring solution. Hence, all these methods are computationally much more expensive than our method. In the employed bee phase, two neighboring solution generation methods are used. In the first method called "Exchange Direction", some demand pairs are randomly selected and each pair is checked one-by-one to see whether the exchange of directions between the two demands in the pair can improve the original solution. If more than one such exchanges are possible, then the exchange which results in the solution of least cost is performed and solution thus obtained replaces the original solution. If no such exchange exists, then the original solution does not change. The second method called "Exchange Max Arc" randomly selects some demands routed through the arc having the highest load and tries to flip the direction of each of these demands one-by-one so as to improve the solution. Again the best improvement strategy is followed like the previous method. In onlooker bee phase, the neighboring solution generation method tries to improve a solution by changing the direction of a randomly chosen demand in the solution. With probability 0.5, the direction of the chosen demand is set to shortest path, otherwise the direction of the chosen demand is set to its direction in the best solution. If the solution improves, the methods stops, otherwise it repeats itself. The method stops only after some fixed number of unsuccessful attempts.

5.6 Other features

If an employed bee solution X has not improved over S_{it} number of consecutive iterations, then it is assumed to be locally optimal and replaced with a new solution X'. However, instead of generating X' like the initial population members, which is usually the case with most ABC algorithms including the one described in [5], we have generated X' by perturbing X. For perturbing X to generate X', each demand is considered one-by-one and its direction in X is flipped with probability ρ_s. We have followed this perturbation strategy because solution thus generated retains major part of the original highly fit solution, and as a result, it is expected to be of better quality in comparison to a solution generated in a manner similar to initial population members.

Unlike the traditional ABC algorithm where there is an upper limit of only one scout in an iteration, we have not imposed any limit on the number of employed bee solutions replaced in an iteration, i.e., number of scouts. This number can be more than one or can be zero depending on how many employed bee solutions in an iteration has not improved over last S_{it} number of consecutive iterations. This is also different from ABC approach of [5], where in every iteration, the number of scouts is equal to 10% of the number of employed bees. The solutions for these scouts bee are created in the same manner as the initial population members. Instead of replacing those employed bee solutions which has not improved since long, these scout bee solutions replace worst employed bee solutions in case scout bee solutions are better. However, this is a severe flaw because if an employed bee solution is locally optimal, it will forever remain in the population and waste computational efforts without offering any opportunity for further improvement.

5.7 Local search procedure

As mentioned already, our local search procedure is composed of two local searches which are applied one after the other on the best solution obtained through ABC algorithm. Our first local search repeatedly applies a direction flip based heuristic as long as there is improvement in solution quality. This heuristic considers each demand one-by-one in their natural order and flips its direction in the solution, if doing so reduces the objective function value. It is to be noted that in case of WRELP problem, flipping the direction of only those demands can possibly reduce the objective function value which are currently routed through the edge having the maximum load. Similarly, in case of WRALP problem, flipping the direction of only those demands can possibly reduce the objective function value which are currently routed through the arc having the maximum load. Hence, for efficiency considerations, only such potential demands should be tried for flipping.

Our second local search repeatedly applies a heuristic, which considers a pair of demands at a time, as long as there is improvement in solution quality. This heuristic considers each demand one-by-one in some random order (for implementing the random ordering, every time the heuristic begins execution, a random sequence of demands is generated) and if the demand under consideration, say i, contributes to the maximum load, then we try to pair it with some other demand so that flipping the directions of both the demands together reduces the objective function values. The demands are tried for pairing with demand i in their natural order. If such a pairing is found, then we immediately do the required flipping and then again all demands are tried one-by-one for pairing with i. Only when i can not be paired with any other demand to reduce the objective function value, next demand in the random sequence is considered. Demands in a pair can have the same direction or opposite directions. Again for efficiency considerations, we can curtail the number of demands that need to be tried for pairing

with i. Only those demands can possibly pair with i which does not contribute to the maximum load and which have weight less than that of i. In case of WRELP, a demand contributes to the maximum load if it is routed through the edge having the maximum load. In case of WRALP, a demand contributes to the maximum load if it is routed through the arc having the maximum load.

The pseudo-code of hybrid ABC approach is presented in Algorithm 2. In this algorithm, E_n and O_n are the number of employed bees and onlooker bees respectively. *DNS(X)* is a function that determines a new solution X' in the neighborhood of the solution X and returns X' (Section 5.5). Pseudo-code for *DNS(X)* is given in Algorithm 1. Function *PBTS*(e_1, e_2,...,e_E) implements the probabilistic binary tournament selection method (Section 5.4). This function returns the index of the solution selected. Function *Perturb(X)* perturbs a solution X to generate a new solution X' as per Section 5.6. *LS_1(best)* and *LS_2(best)* are two functions that implement the two local searches described in Section 5.7. These two local searches are applied once on the best solution returned by ABC algorithm.

Algorithm 2 Pseudo-code of hybrid ABC approach

```
Input: Set of parameters for ABC algorithm and a WRALP/WRELP instance
Output: Best solution found
Generate E_n initial employed bee solutions e_1, e_2, ..., e_{E_n};
best ← Best solution among e_1, e_2, ..., e_{E_n};
while best has improved over specified number of previous iterations do
    for i ← 1 to E_n do
        e_i' ← DNS(e_i);
        if e_i' is better than e_i then
            e_i ← e_i';
        else if e_i has not improved over last S_it iterations then
            e_i ← Perturb(e_i);
        if e_i is better than best then
            best ← e_i;
    for i ← 1 to O_n do
        j ← PBTS(e_1, e_2, ..., e_{E_n});
        o_i ← DNS(e_j);
        if o_i is better than e_j then
            e_j ← o_i;
            if o_i is better than best then
                best ← o_i;
best ← LS_1(best);
best ← LS_2(best);
return best;
```

6 Computational results

To evaluate the performance of HABC-A and HABC-E, we have used the same 19 instances as used in [7] and [5]. The number of nodes (n) in these instances vary from 5 to 30 and the number of demands (m) in these instances vary from 6 to 435. To generate these instances, six equally spaced values of n in the interval [5, 30] are considered, i.e., $n \in \{5, 10, 15, 20, 25, 30\}$. Hence, with respect to n, there are six cases which are called case 1 ($n = 5$), case 2 ($n = 10$) and so on. The rings with 5/10/15 nodes are characterised as ordinary sized rings and rings with 20/25/30 as extremely large rings. For a n node ring, $\frac{n(n-1)}{2}$ demand pairs are possible. So if the demand set contains all these pairs then the demand set is said to be complete, otherwise it is said to be partial. Four different cases are considered with respect to the type of demand set. The first three cases are applicable for all values of n, whereas the last case is applicable to only $n = 30$. The first case consists of complete set of demands, the second case consists of partial set of demands with $max(\lceil \frac{n(n-1)}{4} \rceil, 8)$ demand pairs randomly chosen from complete set of demands and the third case consists of partial sets of demands with $max(\lceil \frac{n(n-1)}{8} \rceil, 6)$ demand pairs randomly chosen from complete set of demands. The weights associated with demands is considered to be uniformly distributed in the interval [5, 100] in these three cases. For each combination of n and these three demand cases, a single instance is generated leading to a total of 18 instances. The last demand case is applicable for $n = 30$ only and consists of a complete set of demands with weights uniformly distributed in [1, 500]. Again, a single instance is generated for this case, thereby, leading to a grandtotal of 19 instances. These instances have the name of the form Cxy where $x \in \{1, 2, 3, 4, 5, 6\}$ is the case with respect to the value of n and $y \in \{1, 2, 3, 4\}$ is the case with respect to demand set. Please also note, that $y = 4$ only when $x = 6$. Except for larger ring sizes and higher variance in demand weights, these instances are similar to the instances used previously in the literature for ring loading problems. Table 1 presents the characteristics of these test instances along with their best known values (BKV) for WRALP and WRELP.

Table 1 Characteristics of test instances along with their best known values (BKV)

Instance	#Nodes(n)	#Demands(m)	BKV(WRALP)	BKV(WRELP)
C11	5	10	161	185
C12	5	8	116	137
C13	5	6	116	137
C21	10	45	525	583
C22	10	23	243	352
C23	10	12	141	199
C31	15	105	1574	1657
C32	15	50	941	941
C33	15	25	563	618
C41	20	190	2581	2745
C42	20	93	1482	1760
C43	20	40	612	683
C51	25	300	4265	4304
C52	25	150	2323	2488
C53	25	61	912	1015
C61	30	435	5762	5953
C62	30	201	2696	2901
C63	30	92	1453	1506
C64	30	435	27779	29245

We have implemented HABC-A and HABC-E approaches in C and executed them on a 2.83 GHz Intel Q9550 processor based system under Linux environment. In all our computational experiments with HABC-A, we have used $E_n = 100$, $O_n = 200$, $\rho_{ir} = 0.95$ if $m < 250$, otherwise $\rho_{ir} = 0.75$, $\rho_{ig} = 0.25$, $\rho_o = 0.8$, $\rho_{ns} = max(0.15, \frac{2}{m})$, $\rho_s = max(0.15, \frac{4}{m})$. If $m < 250$ then S_{it} is set to 60 and HABC-A terminates when best solution has not improved over 100 consecutive iterations, otherwise ($m \geq 250$) S_{it} is set to 90 and HABC-A terminates when best solution has not improved over 150 consecutive iterations. In all our computational experiments with HABC-E, we have used $E_n = 125$, $O_n = 175$, $\rho_{ir} = 0.75$, $\rho_{ig} = 0.25$, $\rho_o = 0.8$, $\rho_{ns} = max(0.15, \frac{2}{m})$, $\rho_s = max(0.1, \frac{4}{m})$. If $m < 250$ then S_{it} is set to 15 and HABC-E terminates when best solution has not improved over 100 consecutive iterations, otherwise ($m \geq 250$) S_{it} is set to 25 and HABC-E terminates when best solution has not improved over 150 consecutive iterations. These parameter values are chosen empirically after performing a large number of trials. Approaches proposed in [7] and [5] were designed with the intention of obtaining best known solution value in every run and their termination conditions were chosen accordingly. To allow a fair comparison with the approaches of [7] and [5], the termination conditions for HABC-A and HABC-E are chosen with the same intention. Please also note that HABC-E and HABC-A both use 300 bees in total ($E_n + O_n$) in an iteration, though the number of employed and the number of onlooker bees used by them differ, and hence, their termination conditions are equivalent in terms of total number of solutions generated without improvement in quality of best solution. Like the approaches of [7] and [5], HABC-A and HABC-E approaches are executed 100 times independently on each instance. As termination conditions are chosen for various methods to reach the best known solution value in every run, therefore, comparison is done on execution time. However, on instance C51, we found value of 4283 for WRELP which is better than BKV of 4304 and no matter how we tried, we failed to get this value of 4283 in every run. Please refer to Table 5 for average solution quality in case of this instance.

Table 2 compares the average execution time of HABC-A on WRALP instances with that of genetic algorithm (GA), hybrid differential evolution algorithm (HDEM), hybrid particle swarm optimisation algorithm (HDPSO) and artificial bee colony algorithm (ABC). The first three approaches are from [7], whereas the last approach is from [5]. Data for GA, HDEM, HDPSO and ABC approaches have been taken from [7] where all the approaches were executed on a 2.66 GHz Intel Q9450 processor based system and average time and iterations to reach the best solution were reported for each method. From average time and iteration to reach the best solution, we got average time per iteration and multiplying it with total number of iterations allotted yielded the execution time. This is done to follow the standard practice

Table 2 Execution times of various approaches in seconds on WRALP instances

Instance	GA	HDEM	HDPSO	ABC	HABC-A
C11	<0.10	<0.10	<0.10	<0.10	<0.10
C12	<0.10	<0.10	<0.10	<0.10	<0.10
C13	<0.10	<0.10	<0.10	<0.10	<0.10
C21	<0.10	<0.10	<0.10	<0.10	<0.10
C22	<0.10	<0.10	<0.10	<0.10	<0.10
C23	<0.10	<0.10	<0.10	<0.10	<0.10
C31	0.33	1.00	0.50	0.67	0.23
C32	<0.10	<0.10	<0.10	<0.10	<0.10
C33	<0.10	<0.10	<0.10	<0.10	<0.10
C41	0.60	0.90	0.53	1.20	0.53
C42	0.19	0.30	0.16	0.3	0.22
C43	<0.10	<0.10	<0.10	<0.10	<0.10
C51	4.69	10.00	2.50	5.00	1.15
C52	1.00	2.00	1.00	2.13	0.39
C53	0.10	0.19	0.09	0.19	0.16
C61	20.19	64.29	15.63	28.13	2.03
C62	3.33	7.50	3.00	5.00	0.65
C63	1.25	2.50	1.25	2.50	0.22
C64	5.00	16.67	10.00	10.00	2.99

Table 3 Execution times of various approaches in seconds on WRELP instances

Instance	GA	HDEM	HDPSO	ABC	HABC-A
C11	<0.10	<0.10	<0.10	<0.10	<0.10
C12	<0.10	<0.10	<0.10	<0.10	<0.10
C13	<0.10	<0.10	<0.10	<0.10	<0.10
C21	<0.10	<0.10	<0.10	<0.10	<0.10
C22	<0.10	<0.10	<0.10	<0.10	<0.10
C23	<0.10	<0.10	<0.10	<0.10	<0.10
C31	0.33	0.67	0.50	0.40	0.19
C32	<0.10	<0.10	<0.10	<0.10	<0.10
C33	<0.10	<0.10	<0.10	<0.10	<0.10
C41	0.45	0.75	0.50	0.40	0.45
C42	0.19	0.33	0.20	0.40	0.18
C43	<0.10	<0.10	<0.10	<0.10	<0.10
C51	5.00	6.25	3.33	6.25	1.58
C52	1.00	2.00	0.80	1.33	0.39
C53	0.10	0.25	0.13	0.17	0.13
C61	22.5	60.00	17.50	18.50	2.00
C62	4.29	10.00	5.00	8.33	0.55
C63	1.25	3.75	1.25	2.50	0.19
C64	3.75	12.50	3.13	4.17	2.67

Table 4 Relative contribution of two local searches and artificial bee colony framework on WRALP instances

Instance	HABC-A				FABC-A				ABC-A			
	Best	Avg	SD	AvT	Best	Avg	SD	AvT	Best	Avg	SD	AvT
C11	161	161.00	0.00	0.01	161	161.00	0.00	0.01	161	161.00	0.00	0.01
C12	116	116.00	0.00	0.01	116	116.00	0.00	0.01	116	116.00	0.00	0.01
C13	116	116.00	0.00	0.01	116	116.00	0.00	0.01	116	116.00	0.00	0.01
C21	525	525.00	0.00	0.07	525	525.00	0.00	0.07	525	525.00	0.00	0.07
C22	243	243.00	0.00	0.05	243	243.00	0.00	0.05	243	243.09	0.51	0.05
C23	141	141.00	0.00	0.02	141	141.00	0.00	0.02	141	141.00	0.00	0.02
C31	1574	1574.00	0.00	0.23	1574	1574.00	0.00	0.23	1574	1574.00	0.00	0.23
C32	941	941.00	0.00	0.09	941	941.00	0.00	0.09	941	941.00	0.00	0.09
C33	563	563.00	0.00	0.04	563	563.00	0.00	0.04	563	563.00	0.00	0.04
C41	2581	2581.00	0.00	0.53	2581	2581.00	0.00	0.52	2581	2581.00	0.00	0.52
C42	1482	1482.00	0.00	0.22	1482	1482.00	0.00	0.22	1482	1482.00	0.00	0.22
C43	612	612.00	0.00	0.07	612	612.00	0.00	0.07	612	612.00	0.00	0.07
C51	4265	4265.00	0.00	1.15	4265	4265.00	0.00	1.15	4265	4265.00	0.00	1.15
C52	2323	2323.00	0.00	0.39	2323	2323.00	0.00	0.39	2323	2323.00	0.00	0.39
C53	912	912.00	0.00	0.16	912	912.00	0.00	0.16	912	912.00	0.00	0.16
C61	5762	5762.00	0.00	2.03	5762	5762.00	0.00	2.02	5762	5762.00	0.00	2.02
C62	2696	2696.00	0.00	0.65	2696	2696.00	0.00	0.65	2696	2696.00	0.00	0.65
C63	1453	1453.00	0.00	0.22	1453	1453.00	0.00	0.22	1453	1453.00	0.00	0.22
C64	27779	27779.00	0.00	2.99	27779	27779.06	0.28	2.98	27779	27779.23	0.58	2.98

Table 5 Relative contribution of two local searches and artificial bee colony framework on WRELP instances

Instance	HABC-E				FABC-E				ABC-E			
	Best	Avg	SD	AvT	Best	Avg	SD	AvT	Best	Avg	SD	AvT
C11	185	185.00	0.00	0.01	185	185.00	0.00	0.01	185	185.00	0.00	0.01
C12	137	137.00	0.00	0.01	137	137.00	0.00	0.01	137	137.00	0.00	0.01
C13	137	137.00	0.00	0.01	137	137.00	0.00	0.01	137	137.00	0.00	0.01
C21	583	583.00	0.00	0.07	583	583.00	0.00	0.07	583	583.00	0.00	0.07
C22	352	352.00	0.00	0.03	352	352.00	0.00	0.03	352	352.00	0.00	0.03
C23	199	199.00	0.00	0.02	199	199.00	0.00	0.02	199	199.00	0.00	0.02
C31	1657	1657.00	0.00	0.19	1657	1657.00	0.00	0.19	1657	1657.00	0.00	0.19
C32	941	941.00	0.00	0.08	941	941.00	0.00	0.08	941	941.00	0.00	0.08
C33	618	618.00	0.00	0.04	618	618.00	0.00	0.04	618	618.00	0.00	0.04
C41	2745	2745.00	0.00	0.45	2745	2745.00	0.00	0.45	2745	2745.00	0.00	0.45
C42	1760	1760.00	0.00	0.18	1760	1760.00	0.00	0.18	1760	1760.00	0.00	0.18
C43	683	683.00	0.00	0.07	683	683.00	0.00	0.07	683	683.00	0.00	0.07
C51	4283	4284.11	1.16	1.58	4283	4284.55	1.24	1.58	4283	4284.59	1.30	1.58
C52	2488	2488.00	0.00	0.39	2488	2488.00	0.00	0.39	2488	2488.00	0.00	0.39
C53	1015	1015.00	0.00	0.13	1015	1015.00	0.00	0.13	1015	1015.00	0.00	0.13
C61	5953	5953.00	0.00	2.00	5953	5953.00	0.00	2.00	5953	5953.00	0.00	2.00
C62	2901	2901.00	0.00	0.55	2901	2901.00	0.00	0.55	2901	2901.00	0.00	0.55
C63	1506	1506.00	0.00	0.19	1506	1506.00	0.00	0.19	1506	1506.00	0.00	0.19
C64	29245	29245.00	0.00	2.67	29245	29245.02	0.14	2.67	29245	29245.06	0.24	2.67

of comparing execution times of various approaches. No matter how fast the best solution is obtained, it is returned only when the program finishes execution, and hence, comparison of time required to reach the best solution does not make sense. For small instances C11, C12, C13, C21, C22, C23, C32, C33 and C41, [7] did not report the average time till best precisely and mentioned that they are < 0.001 seconds, and hence, for these instances we have not reported the execution times precisely in Table 2. However, precise execution times for HABC-A are reported in Table 4. In a manner similar to Table 2, Table 3 compares the average execution time of HABC-E on WRELP instances with that of GA, HDEM, HDPSO and ABC approaches. These tables clearly show the superiority of HABC-A and HABC-E in terms of execution times over other methods on large instances of WRALP and WRELP both. Moreover, the difference in speed between our approaches and other methods widens with increase in number of demands. Please note that we have executed our approaches on a 2.83 GHz Intel Q9550 processor based system which is different from the system used to execute GA, HDEM, HDPSO and ABC approaches (2.66 GHz Intel Q9450 processor). However, Q9550 and Q9450 processors belong to the same series of processors and Q9550 is the immediate successor of Q9450. Hence, there is only a slight difference is processing speeds of the two systems and conclusions drawn here take into account this difference.

To show the relative contribution of two local searches and artificial bee colony framework, we have implemented HABC-A and HABC-E approaches without any local search and with only first local search. The versions without any local search will be referred to as ABC-A and ABC-E, whereas the versions with only first local search will be referred to as FABC-A and FABC-E respectively. Tables 4 & 5 report the results. For each instance, we report the best solution (column Best), average solution quality (column Avg) and standard deviation of solution values (column SD) over 100 runs found by HABC-A, FABC-A, ABC-A or HABC-E, FABC-E, ABC-E as the case may be along with their respective average execution times (column AvT) in seconds. These tables clearly show that local searches contribute little to the success of artificial bee colony algorithm which is capable of finding best solutions on its own in most runs.

7 Conclusions

In this paper, we have proposed hybrid artificial bee colony algorithm based approaches for two real-world $\mathscr{NP}$-hard problems belonging to the domain of optical ring networks. Computational results on the standard benchmark instances of the problems show the effectiveness of the proposed approaches. Our ABC approaches are capable of finding high quality solutions on their own even without the use of local search.

As a future work, we would like to work on a multi-objective version of ring loading problems involving multiple costs. Approaches similar to our approaches can be designed for other related assignment problems.

Acknowledgments Authors are grateful to Dr. A. M. Bernardino for providing the test instances for WRELP/WRALP. Authors would like to thank two anonymous reviewers also for their valuable comments and suggestions which helped in improving the quality of this manuscript.

References

1. Banda J, Singh A (2015) A hybrid artificial bee colony algorithm for the terminal assignment problem. In: Swarm, evolutionary, and memetic computing, lecture notes in computer science, vol 8947, Springer-Verlag, pp 134–144
2. Bernardino A, Bernardino E, Sánchez-Pérez J, Gómez-Pulidoua J, Vega-Rodríguez M (2009) Solving the ring loading problem using genetic algorithms with intelligent multiple operators. In: International symposium on distributed computing and artificial intelligence 2008 (DCAI 2008), Springer, pp 235–244
3. Bernardino A, Bernardino E, Sánchez-Pérez J, Gómez-Pulidoua J, Vega-Rodríguez M (2009) Solving the weighted ring edge-loading problem without demand splitting using a hybrid differential evolution algorithm. In: 2009 IEEE 34th Conference on local computer networks, IEEE, pp 562–568
4. Bernardino A, Bernardino E, Sánchez-Pérez J, Gómez-Pulidoua J, Vega-Rodríguez M (2010) A discrete differential evolution algorithm for solving the weighted ring arc loading problem. In: International conference on industrial, engineering and other applications of applied intelligent systems, Springer, pp 153–163
5. Bernardino A, Bernardino E, Sánchez-Pérez J, Gómez-Pulidoua J, Vega-Rodríguez M (2010) Efficient load balancing for a resilient packet ring using artificial bee colony. In: Proceedings of applications of evolutionary computation: EvoApplications 2010, lecture notes in computer science, vol 6025, Springer-Verlag, pp 61–70
6. Bernardino A, Bernardino E, Sánchez-Pérez J, Gómez-Pulidoua J, Vega-Rodríguez M (2010) A hybrid ant colony optimization algorithm for solving the ring arc-loading problem. In: Hellenic conference on artificial intelligence, Springer, pp 49–59
7. Bernardino A, Bernardino E, Sánchez-Pérez J, Gómez-Pulido J, Vega-Rodríguez M (2011) Solving ring loading problems using bio-inspired algorithms. J Netw Comput Appl 34(2):668–685
8. Bernardino AM, Bernardino EM, Sánchez-Pérez JM, Pulido JAG, Vega-Rodríguez MA (2009c) Solving the non-split weighted ring arc-loading problem in a resilient packet ring using particle swarm optimization. In: Proceedings of the international conference in evolutionary computations, pp 230–236
9. Chaurasia S, Singh A (2015) A hybrid swarm intelligence approach to the registration area planning problem. Inf Sci 302:50–69
10. Cho K, Joo U, Lee H, Kim B, Lee W (2005) Efficient load balancing algorithms for a resilient packet. ETRI J 27(1):110–113
11. Cosares S, Saniee I (1994) An optimization problem related to balancing loads on sonet rings. Telecommun Syst 3(2):165–181
12. Davik F, Yilmaz M, Gjessing S, Uzun N (2004) Ieee 802.17 resilient packet ring tutorial. IEEE Commun Mag 42(3):112–118

13. Dell'Amico M, Labbé M, Maffioli F (1999) Exact solution of the {SONET} ring loading problem. Oper Res Lett 25(3):119–129
14. Frank A, Nishiseki T, Saito N, Suzuki H, Tardos E (1992) Algorithms for routing around a rectangle. Discret Appl Math 40:363–378
15. Goldberg DE, Deb K (1991) A comparative analysis of selection schemes used in genetic algorithms. In: Proceedings of the 1991 conference on foundations of genetic algorithms, Morgan Kaufmann, pp 69–93
16. Goralski W (2002) SONET. McGraw-Hill Professional
17. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. In: Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey
18. Karaboga D, Akay B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471
19. Karaboga D, Akay B (2008) On the performance of artificial bee colony (ABC) algorithm. Appl Soft Comput 8(1):687–697
20. Karaboga D, Akay B (2009) A comparative study of artificial bee colony algorithm. Appl Math Comput 214(1):108–132
21. Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif Intell Rev 42(1):21–57
22. Karunanithi N, Carpenter T (1994) A ring loading application of genetic algorithms. In: Proceedings of the 1994 ACM symposium on applied computing, ACM, New York, NY, USA, SAC '94, pp 227–231
23. Khanna S (1997) A polynomial time approximation scheme for the sonet ring loading problem. Bell Labs Tech J 2(2):36–41
24. Kim SS, Kim IH, Mani V, Kim H (2008) Ant colony optimization for sonet ring loading problem. Int J Innov Comput Inf Control 4(7):1617–1626
25. Kubat P, Smith J (2005) Balancing traffic flows in resilient packet rings. Springer, US, pp 125–140
26. Myung YS, Kim HG (2004) On the ring loading problem with demand splitting. Oper Res Lett 32(2):167–173
27. Myung YS, Kim HG, Tcha DW (1997) Optimal load balancing on sonet bidirectional rings. Oper Res 45(1):148–152
28. Pan Q, Tasgetiren M, Suganthan P, Chua T (2011) A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf Sci 181(12):2455–2468
29. Pandiri V, Singh A (2015) Two metaheuristic approaches for the multiple traveling salesperson problem. Appl Soft Comput 26:74–89
30. RPR Alliance (2004) A summary and overview of the IEEE 802.17 resilient packet ring standrad
31. Schrijver A, Seymour P, Winkler P (1998) The ring loading problem. SIAM J Discret Math 11(1):1–14
32. Sharma TK, Pant M (2013) Enhancing the food locations in an artificial bee colony algorithm. Soft Comput 17(10):1939–1965
33. Singh A (2009) An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. Appl Soft Comput 9(2):625–631
34. Singh A, Sundar S (2011) An artificial bee colony algorithm for the minimum routing cost spanning tree problem. Soft Comput 15(12):2489–2499
35. Wang B (2005) Linear time algorithms for the ring loading problem with demand splitting. J Algorithms 54(1):45–57
36. Yuan P, Gambiroza V, Knightly E (2004) The ieee 802.17 media access protocol for high-speed metropolitan-area resilient packet rings. IEEE Netw 18(3):8–15

Alok Singh received the B.Sc. (Hons.) and M.Sc. degrees in Computer Science from the Banaras Hindu University, Varanasi, India in 1996 and 1998 respectively and the D.Phil. degree in Science from the University of Allahabad, Allahabad, India in 2006. He is currently an Associate Professor in the School of Computer and Information Sciences at the University of Hyderabad, Hyderabad, India. His primary research interests lie in the area of combinatorial optimization using heuristic and metaheuristic techniques. He has published more than 80 papers including 45 journal papers. He is a senior member of IEEE, USA, an elected life member of the National Academy of Sciences, India, an associate editor for the journal Swarm and Evolutionary Computation, and, a member of the editorial board for the journal Applied Soft Computing.

Jayalakshmi Banda received B.E. degree in Computer Science and Engineering from Osmania University, Hyderabad, India in 2005 and M.Tech. degree in Computer Science from University of Hyderabad, Hyderabad, India in 2010. She is currently pursuing the Ph.D. degree in Computer Science in the School of Computer and Information Sciences at the University of Hyderabad, Hyderabad, India. Her research interests include study and development of heuristics and metaheuristics for solving combinatorial optimization problems.