Bi-Velocity Discrete Particle Swarm Optimization and Its Application to Multicast Routing Problem in Communication Networks

Meie Shen, Zhi-Hui Zhan, *Member, IEEE*, Wei-Neng Chen, *Member, IEEE*, Yue-Jiao Gong, *Student Member, IEEE*, Jun Zhang, *Senior Member, IEEE*, and Yun Li, *Member, IEEE*

Abstract—This paper proposes a novel bi-velocity discrete particle swarm optimization (BVDPSO) approach and extends its application to the NP-complete multicast routing problem (MRP). The main contribution is the extension of PSO from continuous domain to the binary or discrete domain. Firstly, a novel bi-velocity strategy is developed to represent possibilities of each dimension being 1 and 0. This strategy is suitable to describe the binary characteristic of the MRP where 1 stands for a node being selected to construct the multicast tree while 0 stands for being otherwise. Secondly, BVDPSO updates the velocity and position according to the learning mechanism of the original PSO in continuous domain. This maintains the fast convergence speed and global search ability of the original PSO. Experiments are comprehensively conducted on all of the 58 instances with small, medium, and large scales in the OR-library (Operation Research Library). The results confirm that BVDPSO can obtain optimal or near-optimal solutions rapidly as it only needs to generate a few multicast trees. BVDPSO outperforms not only several state-of-the-art and recent heuristic algorithms for the MRP problems, but also algorithms based on GA, ACO, and PSO.

Index Terms—Communication networks, multicast routing problem, particle swarm optimization, Steiner tree problem

I. INTRODUCTION

MULTICAST routing problem (MRP) has drawn much attention worldwide for a number of decades owing to its significance in communication and networking systems [1]. MRP is a problem that, given a communication network with a source node, a set of destination nodes, a set of intermediate nodes, and a set of edges that make the network connected, find a tree to connect the source node and all the destination nodes, so that the data can be sent to all destination nodes by multicasting. Work during recent years' on research and applications have witnessed the significance of MRP in various communication networks, providing services such as IPTV, distributed data process, internet telephone, interactive multimedia conference, and video broadcast [2].

An MRP can be treated as an optimization problem. For example, its objective can be to minimize delays from the source to destinations, crucial to latency-sensitive multicasting. In this sense, we can consider the delay of each edge as the cost of the edge and optimize the multicast tree with a minimal cost. We can also take other network properties as the cost of each edge, such as the reserved bandwidth and the utility price [3]. Without loss of generality, this paper does not consider the physical significance of the cost. Therefore optimizing the cost of MRP is also referred to the Steiner tree problem (STP) in graph theory.

The challenge in solving the MRP or STP lies in that the STP is known to be an NP-complete problem [4]. Therefore, deterministic algorithms are inapplicable, while approximation or nondeterministic algorithms are promising and of practical value to solve the MRP [4]. In early years, researchers proposed some state-of-the-art greedy heuristic algorithms for STP, such as shortest path heuristic (SPH), distance network heuristic (DNH), and average distance heuristic (ADH) [5]. In recent years, modern heuristics like directed convergence heuristic (DCH) [6] and greedy randomized adaptive search procedure (GRASP) [7] have also been developed. However, these heuristic algorithms appear not promising in constructing an optimal multicast tree when the network scale becomes large with many nodes and edges. This is because that the greedy heuristic algorithms are based on local information, and are hence easily trapped in local optima in a complex environment.

With the development of evolutionary computation (EC), many studies have shown that EC algorithms like genetic algorithm (GA) [8][9], ant colony optimization (ACO) [10], particle swarm optimization (PSO) [11]-[13], and others [14][15] are promising to solve various complex optimization problems. These also motivate researchers to apply EC algorithms for

M. Shen is with the School of Computer Science, Beijing Information Science and Technology University, Beijing, China.

Z.-H. Zhan, W.-N. Chen, Y.-J. Gong, and J. Zhang are with Sun Yat-Sen University, Guangzhou, 510275, China, with the Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China, with the Engineering Research Center of Supercomputing Engineering Software (Sun Yat-sen University), Ministry of Education, China, and also with the Key Laboratory of Software Technology, Education Department of Guangdong Province, China.

Y. Li is with the School of Engineering, University of Glasgow, Glasgow G12 8LT, U.K.

communication and networking optimization. For example, GAs have been reported successful in solving MRP for decades [5]. ACO was also reported to solve some kinds of MRP [16]. However, the GA and ACO based methods may encounter their inherent low convergence speed, which is criticized in the communication community on inefficiency in meeting high demands of real-time communication services [17]. As an efficient EC variant, PSO is featured by its simpler implementation and faster convergence speed when compared with other EC algorithms, yet offering strong global search ability [18]. Therefore, we focus on applying PSO to solve STP, which is an MRP without QoS constraints. Wang *et al.* [19] used tree-based scheme to code the solution (particle). However, the method needs to code all multicast trees in potential solution, making the algorithm complex for implementation. Qu *et al.* [20] have recently proposed to use jumping PSO (JPSO) to solve STP. Although they used a discrete PSO, the velocity of original PSO was not considered, being inefficient to maintain the advantages of original PSO in continuous domain.

In order to keep the simple implementation and efficiency of original PSO in continuous domain, and also to maintain the fast convergence speed and global search advantages of PSO, this paper extends the origin PSO to the binary domain and optimizes MRP by solving STP as a binary optimization problem. The solution (particle) is coded as a 0/1 string (whose length equals to the number of nodes in the network) where 1 means a node is selected to construct the multicast tree and 0 means the node is not selected. Based on this coding scheme, a novel bi-velocity discrete PSO (BVDPSO) is proposed for both maintaining the search advantages of PSO and matching the binary characteristics of MRP. Therefore, BVDPSO is different from other existing discrete or binary PSO algorithms that use random jumping strategy [20], sigmoid function strategy [21], or set-based strategy [22], because it uses a very simple and straightforward bi-velocity strategy that utilizes two vectors to represent the possibilities of being 0 and 1 respectively for each node. Moreover, BVDPSO modifies the velocity and position update equations to keep the learning mechanism of original PSO in continuous domain. BVDPSO is also different from our previous discrete PSO (DPSO) for MRP optimization [23][24], as DPSO does not keep the original PSO framework but adds a mutation operator to PSO. BVDPSO does not change the PSO algorithm structure, but takes the advantages of the ring topological structure to avoid local optimal, being still as simple as the original PSO. In this paper, BVDPSO is comprehensively studied in local networks and rigorously tested for not only small or medium scale networks, but also large scale networks with a large number of nodes, edges, and destination nodes.

Therefore, the main contributions of this paper are twofold: one is in the algorithm design aspect and the other is in the practical application aspect.

Firstly, the novelty of BVDPSO is that it uses a general bi-velocity scheme to make PSO suitable for solving a class of binary optimization problem in the discrete domain. More importantly, such a bi-velocity scheme makes BVDPSO still as simple as the original PSO in continuous domain, and also maintains the fast global search behaviors of the PSO.

Secondly, BVDPSO can overcome the local optimum issue of many heuristic algorithms and has much faster speed to find the optimal or near-optimal multicast solution than other EC algorithms. This can provide researchers and engineers a new and practical approach to multicast design in communication network. Moreover, to the best of our knowledge, BVDPSO is the first EC-based algorithm that is tested on all the 58 problems (with small, medium, and large scales) in the OR-library (Operation Research Library) [25]. The experimental results can provide a baseline for future research on these MRP problems.

The remainder of the paper is organized as follows. Section II formulates the MRP and describes the framework of PSO. Section III proposes BVDPSO for optimizing MRP. Section IV presents the experimental results and comparisons. Conclusions and future work are drawn in Section V.

II. BACKGROUND OF MRP AND PSO

A. Formulation of MRP

To define an MRP, suppose $Netw=\{A, E\}$ is an undirected, connected, weighted network where A is the node set and E is the edges set. A positive function $c(e)$ is used to denote the cost of each edge e in E. The source node s and all multiple destination nodes in the network make up the set $R=\{s\}\cup D$, where D stands for the destination nodes set. $S=A/R$ stands for the rest nodes (which are also named as the intermediate nodes or the Steiner nodes). The functionality of MRP is to send the data from the source node (s) to all the destination nodes in the set D. Therefore, the objective of MRP is to find a minimal cost tree T that connects the source node to all the destination nodes through some of the intermediate nodes.

For $T=\{A^*, E^*\}$, where $A^* \subseteq A, E^* \subseteq E, R \subseteq A^*$, the objective of MRP is formulated as:

$$f = \text{Min } T = \min \sum_{e \in E^*} c(e), \text{ where } A^* \subseteq A, E^* \subseteq E, R \subseteq A^* \quad (1)$$

B. PSO in Continuous Domain

PSO is an EC algorithm paradigm that emulates the swarm behaviors of birds flocking [17]. Optimizing a N-dimensional continuous optimization problem, each particle i has a velocity vector $V_i=[v_{i1}, v_{i2}, …, v_{iN}]$ and position vector $X_i=[x_{i1}, x_{i2}, …, x_{iN}]$ to indicate the current status. Moreover, the particle i keeps its personal historical best position vector $P_i=[p_{i1}, p_{i2}, …, p_{iN}]$. The best position of all the P_i in the ring topology is regarded as the neighborhood local best position $L_i=[l_{i1}, l_{i2}, …, l_{iN}]$. The V_i and X_i are initialized randomly and are updated in every generation by the guidance of P_i and L_i as:

$$v_{ij} = \omega \times v_{ij} + c_1 \times r_{1j} \times (p_{ij} - x_{ij}) + c_2 \times r_{2j} \times (l_{ij} - x_{ij}) \quad (2)$$

$$x_{ij} = x_{ij} + v_{ij} \quad (3)$$

where ω is the inertia weight linearly decreasing from 0.9 to 0.4 during the running time. c_1 and c_2 are acceleration coefficients set as 2.0. r_{1j} and r_{2j} are two random values in the range of [0, 1] for the jth dimension.

As the ring topology is not easy to be trapped into local optima while can still keep very fast optimization speed [26], it is adopted herein. Interested readers can refer to [27][28] for enhanced PSOs and [29] for PSO industrial applications.

III. BVDPSO

A. Particle Code

The MRP can be regarded to find a set of nodes (include the source node, all the destination nodes, and some intermediate nodes) to construct an optimal multicast tree. Therefore, it is natural and intuitive to code the solution as a binary string, whose length is the same as the number of total nodes, and all the destination nodes are always coded with a bit 1 to indicate that they are always in the tree. Moreover, if an intermediate node is with value 1, it means that the node is used to construct the multicast tree; otherwise, the node is not used to construct the tree. Therefore, suppose there are totally N nodes in the network, the position code of each particle i is as:

$$X_i = [x_{i1}, x_{i2}, \ldots, x_{iN}], \text{ where } x_{ij}=0 \text{ or } 1 \quad (4)$$

while the velocity is coded by a novel bi-velocity fashion as:

$$V_i = \begin{bmatrix} v_{i1}^0, v_{i2}^0, \ldots v_{ij}^0, \ldots, v_{iN}^0 \\ v_{i1}^1, v_{i2}^1, \ldots v_{ij}^1, \ldots, v_{iN}^1 \end{bmatrix}, \text{ where } 0 \leq v_{ij}^0 \leq 1, 0 \leq v_{ij}^1 \leq 1 \quad (5)$$

In (5), it is interesting that the jth dimension of the V_i is associated with bi-values. The v_{ij}^0 is the possibility of x_{ij} being 0 and v_{ij}^1 is the possibility of x_{ij} being 1. Note that v_{ij}^0 and v_{ij}^1 are calculated according to the difference between two positions (e.g., the P_i and the X_i, or the L_i and the X_i), and are independent to stand for the possibilities of the x_{ij} being 0 and 1. Therefore, the sum of v_{ij}^0 and v_{ij}^1 is unnecessarily equal to 1.

B. Velocity Update

When updating the particle's velocity in BVDPSO, it is important to keep the learning concept in original PSO. In our implementation of the velocity update in BVDPSO, the same equation as (2) is used. However, three modifications are taken to fit the 0/1 position code and bi-velocity code in BVDPSO. The details are described as follows and given in Fig. 1.

1) Velocity=Position1–Position2: Suppose that the *Position1* is X_1 and the *Position2* is X_2, our BVDPSO keeps the original PSO learning concept and makes up the *Velocity* ($V_i=X_1–X_2$) by considering the difference between X_1 and X_2 when X_2 learns from X_1. For the jth dimension of V_i, if x_{1j} is b but x_{2j} is not b (b is 0 or 1), this means that X_2 is different from X_1 on this j^{th} dimension, then the particle X_2 should learn from the X_1 (because X_1 is the better position). Hence $v_{ij}^b = 1$ and $v_{ij}^{1-b} = 0$; if x_{1j} is the same as x_{2j}, it means it is unnecessary for X_2 to learn from X_1 on this corresponding dimension, hence $v_{ij}^0 = v_{ij}^1 = 0$. For example, if X_1=[1,1,1,0,1,0,1,0] and X_2=[1,1,1,0,1,1,0,0], then

$$V_i = X_1 - X_2 = \begin{bmatrix} 0,0,0,0,0,1,0,0 \\ 0,0,0,0,0,0,1,0 \end{bmatrix}.$$

2) Velocity=Coefficient×Velocity: This operation is to multiply *Coefficient* ω, or $c \times r$ with each element of the current *Velocity* to obtain each element of the final *Velocity*. As velocity is to denote the possibility for the position being 0 and 1, if any dimension of the final *Velocity* is larger than 1, this value is set to 1. Suppose $c \times r$=[1.5, 1.8, 0.9, 0.5, 1.2, 1.3, 0.8, 0.5] and $V = \begin{bmatrix} 0,0,0,0,0,1,0,0 \\ 0,0,0,0,0,0,1,0 \end{bmatrix}$, then $V=(c \times r) \times V = \begin{bmatrix} 0,0,0,0,0,1.3,0,0 \\ 0,0,0,0,0,0,0.8,0 \end{bmatrix} \begin{bmatrix} 0,0,0,0,0,1,0,0 \\ 0,0,0,0,0,0,0.8,0 \end{bmatrix}.$

3) Velocity=Velocity1+Velocity2: Suppose the *Velocity1* and *Velocity2* are V_1 and V_2, then $V_i=V_1+V_2$ is the final *Velocity*. The jth dimension v_{ij}^b in the velocity V_i is the same as the larger one between v_{1j}^b and v_{2j}^b, where b=0, 1. For example, suppose

$$V_1 = \begin{bmatrix} 0.2, 0, 0.8, 0.1, 0, 0, 0.5, 0 \\ 0.3, 0, 0.1, 0.4, 0, 0, 0, 0.5 \end{bmatrix} \text{ and } V_2 = \begin{bmatrix} 0.4, 0.5, 0.6, 0, 0, 1, 0.3, 0 \\ 0.1, 0, 0.9, 0, 0.2, 0, 0.8, 0 \end{bmatrix},$$

then $V_i=V_1+V_2 = \begin{bmatrix} 0.4, 0.5, 0.8, 0.1, 0, 1, 0.5, 0 \\ 0.3, 0, 0.9, 0.4, 0.2, 0, 0.8, 0.5 \end{bmatrix}.$

```
Procedure Velocity_Update (Xi)
Begin
   For j=1 to N Do //Calculate vij=ωvij+c1r1j(pij−xij)+c2r2j(lij−xij)
      //Step 1: Calculate c1r1j(pij − xij)
      If pij==xij      Then vp_0=0; vp_1=0;//do not learn
      Else If pij==1  Then vp_0=0; vp_1=1;//learn from vpj1
      Else If pij==0  Then vp_0=1; vp_1=0;//learn from vpj0
      c11=c1×random(0,1); vp_0=c11×vp_0; vp_1=c11×vp_1;
      //Step 2: Calculate c2r2j(lij − xij)
      If lij==xij      Then vl_0=0; vl_1=0; //do not learn
      Else If lij==1  Then vl_0=0; vl_1=1; //learn from vlj1
      Else If lij==0  Then vl_0=1; vl_1=0; //learn from vlj0
      c22=c2×random(0,1); vl_0=c22×vl_0; vl_1=c22×vl_1;
      //Step 3: Determine the finial velocity
      vij0=max{ω×vij0, vp_0, vl_0}; vij1=max{ω×vij1, vp_1, vl_1};
   End For
End
```

Fig. 1. The pseudocode of the procedure for velocity update.

C. Position Update

When the PSO algorithm works in continuous domain, the position update is to add the updated velocity V_i to the current position X_i, as (3). However, the position and velocity may not be added up directly in discrete domain. In order to keep the learning concept of the original PSO, we construct the new position by using the strategy as shown in (6):

$$x_{ij} = \begin{cases} rand\{0,1\} & \text{if } (v_{ij}^0 > \alpha \text{ and } v_{ij}^1 > \alpha) \\ 0 & \text{if } (v_{ij}^0 > \alpha \text{ and } v_{ij}^1 \leq \alpha) \\ 1 & \text{if } (v_{ij}^0 \leq \alpha \text{ and } v_{ij}^1 > \alpha) \\ x_{ij} & \text{if } (v_{ij}^0 \leq \alpha \text{ and } v_{ij}^1 \leq \alpha) \end{cases} \quad (6)$$

With these strategies, the new position pays more attention to the new velocity. This can help the particle learn more from the exemplars to the learning concept of the original PSO.

D. Fitness Evaluation

An important issue in using BVDPSO for MRP optimization is how to calculate the fitness of a solution. The solution is a 0/1 string, indicating which nodes are used to construct the multicast tree. The fitness evaluation is to construct a multicast tree according to this 0/1 string and then calculate the cost of the tree. In order to construct a promising multicast tree with low cost, we firstly transform the MRP network into a cost complete graph (CCG) based on the Floyd's algorithm with special data structure. Then, based on the CCG and the 0/1 binary string of the solution, we can construct the multicast tree by using the Prim's minimum spanning tree (MST) algorithm. However, it is necessary to make some modifications to the Prim's algorithm to fit the characteristic of MRP. Moreover, the tree has to be pruned in order to delete the leaves which are not the destination nodes. The following parts describe the CCG procedure, modified Prim's algorithm, and the prune procedure.

1) CCG Procedure: In the CCG procedure, every two nodes are connected by the shortest path. It should be noted that some of the shortest paths may be 'indirect' paths, and the others are 'direct' paths. A 'direct' path means that the path exists in the original network that connects the two nodes directly. An 'indirect' path means that the path does not directly connect the two nodes but via some other nodes. Therefore, we should record the very next node of the shortest path that connects any two nodes. This can help to restore the path when necessary.

Fig. 2 gives the pseudocode of the CCG procedure based on the Floyd's algorithm with a data structure. $C(i, j)$ is the input data of the CCG procedure to give the cost between nodes i and j in the original network. $C(i, j)=-1$ means nodes i and j are not directly connected. The data structure S is used to record the shortest path (with minimal cost) that connects any two nodes. Specifically, $S(i, j).length$ is the total cost of the path that connects nodes i and j. $S(i, j).next$ is the very next node on the path from i to j. That is, if the nodes i and j are connected directly, $S(i, j).next$ is j and $S(j, i).next$ is i. Otherwise, suppose that the shortest path between nodes i and j is $(i, k_1, k_2, …, j)$, then $S(i, j).next=k_1$, $S(k_1, j).next=k_2$, and so on. With the help of $S(i, j).next$, we can reconstruct the path between nodes i and j easily. Noted that the CCG procedure is only carried out once before using BVDPSO to optimize MRP. Therefore, the CCG is the pre-process whose results can be used again and again in the fitness evaluation procedure during the entire BVDPSO process.

```
STRUCT SHORTEST{
    int length;    //The length of the shortest path between two nodes
    int next;      //The immediate next node on the shortest path
};
SHORTEST S[N][N];//S stores the shortest path between any two nodes
Procedure CCG (C[N][N]) //C gives the length between any two nodes
Begin
    For any two nodes i and j Do //Step 1: Find the connects of any two nodes
        S(i,j).length = C(i,j);
        If (i==j||C(i,j)== -1) Then S(i,j).next = -1; Else S(i,j).next = j;
    End of For
    For k=1 to N Do    //Step 2: Find out the shortest path of any two nodes
        For any two nodes i and j Do
            //whether i and j can be connected by a shorter path through node k?
            If (S(i,j).length > S(i,k).length+S(k,j).length) Then
                S(i,j).length = S(i,k).length+S(k,j).length; S(i,j).next = S(i,k).next;
            End of If
        End of For
    End of For k=1 to N
End
```

Fig. 2. The pseudocode of the CCG procedure.

2) Modified Prim's Algorithm for the MST: As there are direct and indirect paths in the CCG, the modified Prim's algorithm prefer the direct paths to the indirect paths when constructing the MST T. That is, we try to construct the tree according to the nodes with value 1 in the solution string. Therefore, when there are both direct and indirect paths connect these nodes to the tree, the algorithm selects the shortest direct path even though it may be longer than some indirect paths. When there are no direct paths, the algorithm selects the shortest indirect path.

We give an example in Fig. 3 to illustrate the modified Prim's algorithm. Fig. 3(a) is the original network with N_1, N_8, and N_9 as the destination nodes. Suppose that the particle position is as $X = [1, 0, 1, 0, 0, 0, 1, 1, 1]$, denoting that the nodes N_1, N_3, N_7, N_8, and N_9 are used to construct the tree. The modified Prim's algorithm is described as four steps.

Step 1): Initialize T as an empty tree. We firstly select a random destination node into T (the source node is also regarded as a destination node). Then, for any other node i with a position value 1 in the binary string, record its direct cost $D[i]$ and indirect cost $I[i]$ to the tree T. As in Fig. 3(b), we select the destination node N_1 into the T, and then record the $D[i]$ and $I[i]$ for all the other nodes i whose position values are 1. For example, $D[N_3]=3$ means that the node N_3 connects the T via a direct path with cost 3. $I[N_3]=2$ means that the shortest path for the node N_3 to the T is with cost 2, but have to use node N_2 as an intermediate node. Herein, we use the word 'record' instead of 'calculate' because the cost values have been calculated in the CCG procedure which can be used during the entire process.

Step 2): Select the next nearest node into the tree T. In this step, we prefer the direct path to the indirect path even though the direct cost may be larger than the indirect cost. For example, in Fig. 3(c), the node N_3 is selected. However, if there are no nodes connect the T via direct path, we select the shortest indirect path, e.g., the node N_7 is selected in Fig. 3(d). When an indirect path is selected, the corresponding intermediate nodes are also added into the tree, as shown in Fig. 3(d) where the nodes N_2, N_5, and N_6 are added.

Step 3): Cost update. Once a node has been added to the tree (when a direct path is selected) or some nodes have been added to the tree (when an indirect path is selected), we have to update the direct cost $D[i]$ and the indirect cost $I[i]$ to the tree T of the other node i whose position value is 1 in the binary string. For example, in Fig. 3(d), both the $D[N_8]$ and $I[N_8]$ become 4 because N_8 can connect the T directly via N_7. Also, both the $D[N_9]$ and $I[N_9]$ become 3 because the direct path connects N_5 and N_9 costs 3. Moreover, Fig. 3(e) shows the cost update of the N_8 after the adding of the N_9 into the tree.

Step 4): Repeat Step 2) and Step 3) until all the destination nodes have been added into T.

3) Prune Procedure: The prune procedure deletes all the non-destination leaf nodes in T to reduce the redundancy. The procedure is easy to implement: we find out the node i in T that has the degree of 1 (the degree of each node can be recorded during the above multicast tree constructed process), if i is not a destination node, delete it and reduce the degree of node j that connects i by 1. Repeat doing this until no non-destination node with degree 1 can be found. For example, in Fig. 3(f), the nodes N_3 and N_7 are redundant and can be deleted. After the deletion of N_7, the node N_6 can be also deleted. Therefore, we can obtain the finial multicast tree T as Fig. 3(g).
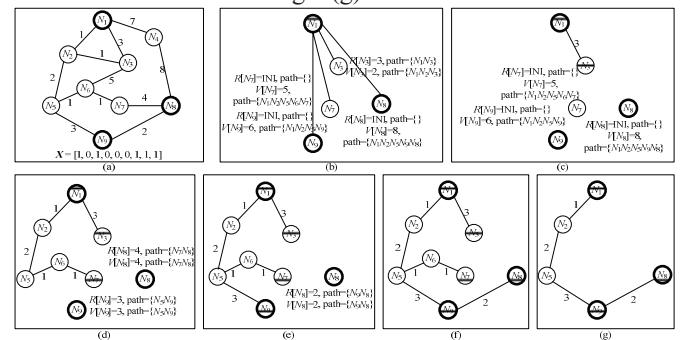

Fig. 3. Illustration of the modified Prim's algorithm to construct multicast tree.

IV. EXPERIMENTS AND COMPARISONS

A. Problem Instances and Algorithm Settings

The experiments on the problems of Categories B/C/D in the OR-library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/) [25] are carried out. The details of the problems are given in Table I. The problems are divided into three Categories B/C/D that can be regarded as small, medium, and large scale MRP in local networks. The *OPT* is the optimal solution in the library.

The population size M of BVDPSO is set to 20, ω is linearly decreasing from 0.9 to 0.4, and c_1 and c_2 are both set to 2.0. The maximum number of generation G is 1250, and therefore there are at most 20×1250=25000 fitness evaluations (FEs) each run. This is the same as the configurations in previous studies [23]. BVDPSO terminates when it finds the global optimum *OPT* or it runs out of the FEs budget. It is should be noted that the number of FEs is also the number of generated multicast tree.

B. Comparisons on Solution Accuracy

1) Small Scale Problem Instances: We first compare the solution accuracy of the results obtained by different algorithms on the problems of Category B. The comparisons are based on the relative error R that is defined as:

$$R\%=(Result–OPT)/OPT\times100 \qquad (7)$$

where the *Result* is the mean solution of 100 independent runs. The R values of the heuristic methods like SPH, DNH, ADH, and the GA methods reported in [5] are used here for comparisons. The results of ACO are based on the report in [16].

The comparisons in Table II show that the traditional heuristics are easy to be trapped into local optima and result in poor solution accuracy. The R of SPH, DNH, and ADH on some problems are even larger than 5%, indicating that the traditional heuristic algorithms may not have strong global search ability to obtain high accurate solutions. For the EC algorithm, GA and BVDPSO can obtain the *OPT* on all the problems in every run and all the relative error values are 0% but ACO can not.

2) Medium Scale Problem Instances: The experimental results of the problems in Category C are compared in Table III. These problems are regarded to be more difficult than those in Category B because they have more nodes, edges, and destination nodes. Herein, 30 independent runs are carried out for each problem because of the large computational burden. As the results for the Category C problems are not available in [5] or relevant literature for the heuristics like SPH, DNH, or ADH, they are not used in the comparisons. Instead, we compare BVDPSO with recent heuristics DCH [6] and GRASP [7], and JPSO for multicast routing (JPSOMR) [20]. The results reported in [6] for DCH and the results reported in [20] for GRASP and

Table I The Details of Steiner Tree Problems

No.	$|N|$	$|E|$	$|D|$	OPT	No.	$|N|$	$|E|$	$|D|$	OPT	No.	$|N|$	$|E|$	$|D|$	OPT
B01	50	63	9	82	C01	500	625	5	85	D01	1000	1250	5	106
B02	50	63	13	83	C02	500	625	10	144	D02	1000	1250	10	220
B03	50	63	25	138	C03	500	625	83	754	D03	1000	1250	167	1565
B04	50	100	9	59	C04	500	625	125	1079	D04	1000	1250	250	1935
B05	50	100	13	61	C05	500	625	250	1579	D05	1000	1250	500	3250
B06	50	100	25	122	C06	500	1000	5	55	D06	1000	2000	5	67
B07	75	94	13	111	C07	500	1000	10	102	D07	1000	2000	10	103
B08	75	94	19	104	C08	500	1000	83	509	D08	1000	2000	167	1072
B09	75	94	38	220	C09	500	1000	125	707	D09	1000	2000	250	1448
B10	75	150	13	86	C10	500	1000	250	1093	D10	1000	2000	500	2110
B11	75	150	19	88	C11	500	2500	5	32	D11	1000	5000	5	29
B12	75	150	38	174	C12	500	2500	10	46	D12	1000	5000	10	42
B13	100	125	17	165	C13	500	2500	83	258	D13	1000	5000	167	500
B14	100	125	25	235	C14	500	2500	125	323	D14	1000	5000	250	667
B15	100	125	50	318	C15	500	2500	250	556	D15	1000	5000	500	1116
B16	100	200	17	127	C16	500	12500	5	11	D16	1000	25000	5	13
B17	100	200	25	131	C17	500	12500	10	18	D17	1000	25000	10	23
B18	100	200	50	218	C18	500	12500	83	113	D18	1000	25000	167	223
					C19	500	12500	125	146	D19	1000	25000	250	310
					C20	500	12500	250	267	D20	1000	25000	500	537

Table II Comparisons on Solution Accuracy (R%) of Different Algorithms on the Problems of Category B

Methods	B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12	B13	B14	B15	B16	B17	B18	Average R
SPH	**0**	**0**	**0**	5.08	**0**	**0**	**0**	**0**	**0**	4.65	2.27	**0**	7.88	2.55	**0**	3.15	3.82	1.83	3.95%
DNH	**0**	8.43	1.45	8.47	4.92	4.92	**0**	**0**	2.27	13.95	2.27	**0**	6.06	1.28	2.2	7.87	5.34	4.59	2.04%
ADH	**0**	**0**	**0**	5.08	**0**	**0**	**0**	**0**	**0**	4.65	2.27	**0**	4.24	0.43	**0**	**0**	3.05	**0**	0.96%
GA	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0%**
ACO	**0**	**0**	**0**	**0**	**0**	NA	NA	**0**	**0**	1.63	1.14	NA	1.39	0.13	**0**	4.72	NA	3.44	NA
BVDPSO	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0%**

Table III Comparisons on Solution Accuracy (R%) of Different Algorithms on the Problems of Category C

Methods	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20
DCH	3.5	23.6	7.16	6.02	1.25	14.06	18.62	9.03	7.21	2.92	15.92	13.04	11.24	10.21	4.31	18.18	27.77	10.31	15.06	7.11
GRASP	**0**	**0**	0.9	1.8	0.5	**0**	**0**	1.7	1	0.3	1.9	0.9	1.7	1.5	0.1	3.6	4.4	4.4	3.7	0.9
GA	**0**	**0**	0.5	0.44	0.1	**0**	**0**	1.14	1.27	0.38	**0**	0.43	1.36	1.42	1.04	**2.73**	1.11	4.78	5	2.02
DPSO	**0**	**0**	0.05	0.02	**0**	**0**	**0**	0.18	**0.3**	0.14	0.31	**0**	0.97	0.56	0.16	3.64	2.22	2.65	0.96	**0**
JPSOMR	1.2	3.5	0.9	0.4	**0**	**0**	0.7	**0.1**	0.5	**0.1**	0.9	**0**	**0.2**	**0.3**	**0**	9.1	**1.1**	2.4	1.4	**0**
BVDPSO	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.20	0.42	0.13	**0**	1.23	2.02	0.78	**0**	8.79	1.67	**1.30**	**0.87**	0.06

Average R over all the 20 instances: DCH: 11.33%; GRASP: 1.47%; GA: 1.45%; DPSO: 0.61%; JPSOMR: 1.14%; BVDPSO: 0.87%

Table IV Comparisons on Solution Accuracy (R%) of Different Algorithms on the Problems of Category D

Methods	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20
DCH	9.43	12.72	7.92	5.58	1.78	29.85	29.12	9.7	6.76	2.79	27.58	14.28	10.8	7.64	4.03	15.38	26.08	18.83	17.41	6.89
GA	**0**	**0**	0.86	0.87	0.4	**0**	**0**	2.2	1.86	0.77	**0**	**0**	2.26	2.5	1.33	**0**	**0**	6.68	6.94	2.42
BVDPSO	**0**	**0**	**0.03**	**0.06**	**0.05**	**0**	**0**	**0.92**	**0.94**	**0.29**	**0**	**0**	**1.78**	**0.87**	**0.36**	**0**	**0**	3	**1.55**	**0.04**

Average R over all the 20 instances: DCH: 13.23%; GA: 1.45%; BVDPSO: 0.49%

JPSOMR are directly used. Moreover, the results of DPSO using mutation and the results of GA that are available in the literature [23] are adopted for comparison. It should be noted that JPSOMR and DPSO are both discrete PSO variants. Comparing BVDPSO with these discrete PSOs is interesting in evaluating the BVDPSO performance in discrete optimization.

It can be observed from Table III that BVDPSO yields the best performance on 11 (C01-C07, C11, C15, C18, and C19) out of the 20 problems. DCH, GRASP, and GA often fail to produce good results on these hard problems. Even though DPSO can obtain the best results on some of the problems, it works well by using a mutation-like operation and by introducing a new parameter into the PSO paradigm. This makes DPSO a little difficult to use [23]. For JPSOMR, the average R over all the 20 problems is 1.14%, beaten by BVDPSO with average R=0.87%. By observing the results in Category C, BVDPSO offers very good performance and outperforms other algorithms in general.

3) Large Scale Problem Instances: The problems in Category D are also solved by BVDPSO, and the results are compared with those obtained by DCH and GA in Table IV. The problems in Category D are all very difficult because the networks are large with thousands of nodes and edges. The results of DCH are reported in [6]. As no EC results on these problems are available in the literature, e.g., the JPSOMR only tested on Category B/C problems, herein we implement GA and compare with BVDPSO. The data for GA and BVDPSO are the mean results of 10 independent runs. Herein the parameter configurations for GA are set with a population size of 50, the maximal generation number of 500, using a one-point crossover and a bit mutation with the crossover and mutation probabilities 0.7 and 0.04 respectively, as recommended in [5][23].

The results show that the DCH heuristic fails to obtain the *OPT* on all the 20 problems. For the EC algorithms, both GA and BVDPSO can obtain the *OPT* on problems D01, D02, D06, D07, D11, D12, D16, and D17. These problems, however, are relative simple because they are all with small number of destination nodes, e.g., 5 or 10 destination nodes. However, when the problems become more complex with larger number of destination nodes, e.g., more than 100, or even as many as 500, GA is always trapped into local optima and results in larger R values. On contrast, BVDPSO can still obtain very good results or near-optimal solutions, resulting in smaller R values. BVDPSO performs best over all the 20 problems, with average R=0.49%, while GA is with average R=1.45%. Moreover, for BVDPSO, 8 results are 0, 4 results are smaller than 0.1%, none of the results is larger than 2%. On contrast, 6 results obtained by GA are larger than 2%, and 2 results are even larger than 6%. The results demonstrate the strong global search ability and good performance of BVDPSO in solving large scale problems.

C. Comparisons on Convergence Speed

In order to verify the advantages of the proposed BVDPSO in convergence speed when solving MRP, we compared the mean FEs to obtain the *OPT* by BVDPSO and some other EC algorithms like GA [5] and DPSO [23] in Table V, Table VI, and Table VII for the problems of Category B, C, and D, respectively. The FEs values for GA and DPSO on the problems

of Category B and C are reported in [23] and are directly used here for comparisons. In these tables, the mean FEs value is considered only on the runs that can obtain the *OPT*. However, if an algorithm fails totally to obtain the *OPT* on the problem, the result is indicated by the symbol '-' in the table. The mean FEs can indicate the convergence speed. Generally, we can say that an algorithm has faster convergence speed if it can obtain the *OPT* with fewer FEs. That is, obtain the optimal solution by generating fewer multicast tree.

The results in Tables V&VI show that BVDPSO is faster than GA to obtain the *OPT* of all problems in Categories B/C, and are faster than DPSO on most of these problems. In average, BVDPSO used only 111.72 average FEs to obtained the *OPT* over all the 18 Category B problems, fewer than that of DPSO (141) and much fewer than that of GA (725). As some algorithms totally fail to obtain *OPT* on some problems in Category C, we calculated the average FEs over the problem instances (C01-C03, C05-C07, C10-C12, C16, and C17) that all the algorithms successfully solve. Results show that BVDPSO is faster than both DPSO and GA nearly 10 times.

The results in Table VII for the problems of Category D show that BVDPSO uses fewer FEs than GA to obtain the *OPT* if they both can, i.e., on the problems D01, D02, D06, D07, D11, D12, D16, and D17. However, on some of the problems, none of the algorithms can obtain the *OPT*, which are not presented in the table. This may be caused by the difficulty of the Category D problems whose networks are with lots of nodes and a large amount of edges. Nevertheless, BVDPSO can still obtain the *OPT* solution on the problems D03, D04, D05, and D20 sometimes where GA totally fails. From the comparisons, BVDPSO is not only robust to obtain the global optimum, but also is with much faster convergence speed.

Table V Comparisons on Convergence Speed on the Problems of Category B

No.	GA	DPSO	BVDPSO	No.	GA	DPSO	BVDPSO
B01	105	42	**41.6**	B10	780	**72**	97
B02	160	**54**	56.6	B11	585	142	**104.2**
B03	100	52	**43.8**	B12	260	**144**	159.4
B04	120	**82**	85.8	B13	1100	468	**101.8**
B05	130	**50**	67	B14	4020	342	**99.2**
B06	515	258	**227.4**	B15	1380	94	**88**
B07	275	**42**	43.4	B16	885	**110**	143
B08	185	48	**47.6**	B17	925	**144**	238.2
B09	275	**56**	61.8	B18	1250	338	**305.2**
Average FEs over B01-B18: GA=725, DPSO=141, BVDPSO=**111.72**							

Table VI Comparisons on Convergence Speed on the Problems of Category C

No.	GA	DPSO	BVDPSO	No.	GA	DPSO	BVDPSO
C01	380	**66**	72.67	C11	1410	12090	**268.67**
C02	815	204	**183.33**	C12	3535	4156	**781.11**
C03	15430	4706	**644**	C13	-	-	-
C04	-	5902	**733.33**	C14	-	-	**4000**
C05	17970	806	**725.33**	C15	-	21280	**2654.67**
C06	1615	3160	**138.67**	C16	2940	12780	**360**
C07	3255	3774	**253.33**	C17	2450	11944	**491.30**
C08	-	20502	**1820**	C18	-	-	**3440**
C09	-	-	**7513.33**	C19	-	24826	**2555**
C10	18980	20648	**4505**	C20	-	4388	**2100**
Average FEs over C01-C03, C05-C07, C10-C12, C16, and C17: GA=6252.73, DPSO=6757.64, BVDPSO=**765.77**							

Table VII Comparisons on Convergence Speed on the Problems of Category D

No.	GA	BVD-PSO	No.	GA	BVD-PSO	No.	GA	BVD-PSO
D01	1180	**102**	D05	-	**2006.67**	D12	595	**360**
D02	1260	**138**	D06	960	**200**	D16	555	**356**
D03	-	**1437.78**	D07	2780	**310**	D17	1230	**554**
D04	-	**1553.33**	D11	3170	**356**	D20	-	**2426.67**
Average FEs over D01, D02, C06, D07, D11, D12, D16, and C17: GA=1466.25, BVDPSO=**297**								

D. Analysis of BVDPSO Parameters

In this subsection, the performance related parameters ω, c_1, and c_2 are investigated because they can affect the algorithm performance but have light influences on the computational burden. Moreover, the computational burden related parameters M and G are investigated because they affect both algorithm performance and computational burden.

1) Performance Related Parameters: Herein we undertake investigations based on the Category C problems. For investigating the inertia weight ω, we fix c_1 and c_2 as 2.0. For investigating the acceleration coefficients c_1 and c_2, the ω is set linearly decreasing from 0.9 to 0.4. The results are presented in Fig. 4. As the R values of different problems have different ranges, we only give the results of some problems that are in similar ranges. The results of ω show that the R values get smaller as ω increases for most of the problems. This may be due to that larger ω can increase the diversity to maintain better global search. However, when ω is set to be 1.0 or larger, the results are much poorer, which are not plotted in the figure due to the out of range. Moreover, the common setting for ω (i.e., linearly decreasing from 0.9 to 0.4) performs best on most of the problems. The results of c_1 and c_2 show that the standard value 2.0 helps BVDPSO work best on most of the problems. Therefore, the results demonstrate the advantages that the standard settings for ω and c_1/c_2 in the continuous PSO are still effective and optimal in our proposed discrete BVDPSO.
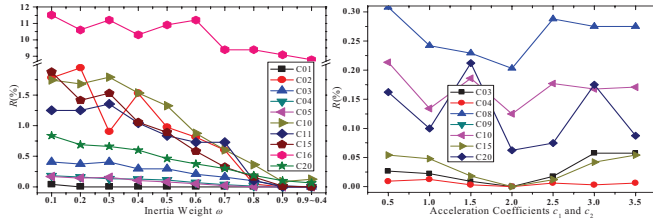

Fig. 4. Parameters analysis on ω and c_1/c_2.

2) Computational Burden Related Parameters: Herein, we make further investigation on the influences of M and G based on the problems of C09 and C14. All the experiments are implemented in the VC++ 6.0, and run on a PC with Pentium IV 2.8 GHz CPU and 256MB memory.

The experimental results are plotted in Fig. 5. The investigation includes the population size M of 20, 50, and 100; and the maximal generations G of 500, 1250, and 2500, making up 9 combinations. In the figure, the R value and the CPU time are the average values of 30 independent runs for each combination. It can be observed that the solution accuracy gets better and better as the M or the G increase. However, the mean CPU time also increases as the M increases or the G increases. Therefore, the trade-off should be determined between the solution accuracy and the computational burden.
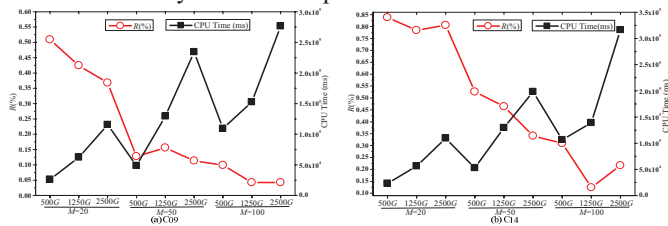

Fig. 5. Parameter analysis on G and M.

The figure also reveals an interesting observation that BVDPSO with a larger M always outperforms the one with a smaller M because of the larger population diversity, whilst BVDPSO with larger a G may sometimes does not perform better than the one with small a G because of random noise. From the experimental results in Fig. 5 and Section IV-B, it can be observed that setting the number of particles to a small value as M=20 is sufficient for most of the problems (e.g., all the problems in Category B and many problems in Categories C/D). Setting M=50 should be promising for very difficult problems, e.g., with node number up to 500 or higher. If higher solution accuracy is required, setting M=100 would be a good choice.

E. Network Simulation Results

We use the networks simulator 2 (NS-2) to further evaluate the multicast tree optimized by BVDPSO in this subsection. NS-2 is a simulator that can test the performance of any given network topology by simulating the network traffic [30]. In the simulation, we adopt two network topology cases. In Case 1, we artificially conduct a network topology whose links are with 10Mb bandwidth and with a random delay within [1ms, 10ms]. When we configure this network into graph for BVDPSO optimization, the cost of each edge is set to the same as the delay of the corresponding link. In Case 2, the C15 problem is used as the network topology. As the cost of each edge is a random value in [1, 10], when we configure the C15 network for NS-2, we set the delay of each link with the value the same as the cost, while the bandwidth of each link with the value as the result of (11–cost). The network topology of Case 1 is given in Fig. 6.
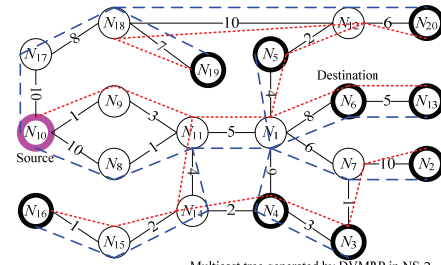

Fig. 6. Network topology of Case 1 and the different multicast trees.

In the simulation, we use distance vector multicast routing protocol (DVMRP) provided by NS-2 to enable the multicast function. For performance evaluation and comparison, we also input the multicast trees optimized by GA and BVDPSO into NS-2 for simulation. Therefore, three multicast approaches DVMRP, GA, and BVDPSO are evaluated and compared based on Case 1 and Case 2 in the NS-2 environments.

We use constant bit rate (CBR) traffic with 512 bytes packet size for the multicast traffic. The packet rate is set as 1Mb, 2Mb, 4Mb, 8Mb, 16Mb, and 32Mb to emulate different traffic loads. We compare the network performance of the three multicast routing approaches by the packet loss metric and the average delay metric. For a CBR packet sent from the source, if it can reach all the destination nodes, it is regarded as a successful packet. This way, the packet loss ratio can be calculated. For any successful CBR packet, we can also obtain the time from the source to any destination node. We consider the maximal time to one of the destination node as the delay of the packet. This way,

we can obtain the average delay of all the successful packets. The packet loss ratio and average delay of different multicast routing approaches under different traffic loads are compared in Fig. 7. Note that in Case 1, the GA approach can obtain the same multicast tree as BVDPSO does, as indicated in Fig. 6. In this sense, their network performance is the same in this case and is therefore presented by the same curves in Figs. 7.

Fig. 7 shows that none of the three approaches lose packets when the traffic load is light (e.g., no more than 4Mb) except that DVMRP has slight packet loss ratio on Case 1. With the increasing of the traffic load, all the approaches are affected and more packets are lost. However, GA and BVDPSO can always do better than DVMRP while BVDPSO can always does better than GA, as indicating in the Fig. 7 that BVDPSO losses the least packets while DVMRP losses the most packets. Fig. 7 further shows that the average delay of BVDPSO is always less than those of DVMRP and GA. These results demonstrate that BVDPSO has advantages in obtaining promising multicast tree that transfer information from the source to all the destinations faster and more reliable than both DVMRP and GA do.
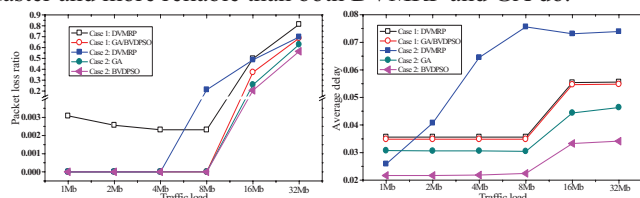

Fig. 7. Packet loss ratio and average delay under different traffic loads.

V. CONCLUSION

A bi-velocity discrete particle swarm optimization has been developed in this paper to optimize MRP in communication networks. This proposed BVDPSO algorithm is motivated by the considerations of providing a simple and yet efficient method for solving MRP with higher solution accuracy than traditional heuristics and also with faster convergence speed than existing EC methods, providing the researchers a new and practical method for multicast in communication network.

The effectiveness and efficiency of BVDPSO have been demonstrated by comparing it with 9 other algorithms on all the Categories B/C/D problems in the OR-library. The results show that BVDPSO can obtain better solutions with higher accuracy than the heuristic methods and with faster convergence speed than the GA and previous PSO based methods. This makes contributions to the communication and networking community by providing a new multicast design method that can obtain optimal or near-optimal solutions to MRP very rapidly by generating very few multicast trees.

For future research, we plan to investigate the performance of BVDPSO in solving the MRP with practical QoS constraints and multiple objectives like other industrial problems [31][32]. For example, we can also take into account the average delay of all destination nodes as another optimization objective. We can also take the delay and the link bandwidth as QoS constraints. As PSO has potentials in solving constrained problems [33] and multi-objective optimization problems [34], the proposed BVDPSO is promising to be extended to solve the constrained and multi-objective MRP problems. The dynamic routing

characteristics of MRP can also be considered [35]. Moreover, the proposed novel bi-velocity strategy offers a simple and general technique for BVDPSO to solve a class of binary optimization in our future work.

REFERENCES

[1] A. Sabbah, A. EI-Mougy, and M. Ibnkahla "A survey of networking challenges and routing protocols in smart grids," *IEEE Trans. Ind. Informt.*, vol. 10, no. 1, pp. 210-221, Feb. 2014.

[2] G. Kandavanam, D. Botvich, S. Balasubramaniam, and C. Kulatunga, "PaCRAm: Path aware content replication approach with multicast for IPTV networks," in *Proc. IEEE Globecom*, 2010, pp. 1-6.

[3] G. Kandavanam, D. Botvich, S. Balasubramaniam, and B. Jennings, "A hybrid genetic algorithm/variable neighborhood search approach to maximizing residual bandwidth of links for route planning," in *Proc. 9th Int. Conf. Artificial Evolution*, 2009, pp. 49-60.

[4] K. Han, Y. Liu, and J. Luo, "Duty-cycle-aware minimum-energy multicasting in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 910-923, Jun. 2013.

[5] Y. Leung, G. Li, and Z. B. Xu, "A genetic algorithm for the multiple destination routing problems," *IEEE Trans. Evol. Comput.*, vol. 2, no. 4, pp. 150-161, Nov. 1998.

[6] C. Shampa, B. Arvind, and R. Aman, "Directed convergence heuristic: A fast & novel approach to Steiner tree construction," *Int. Conf. Very Large Scale Integration*, 2006, pp. 255-260.

[7] N. Skorin-Kapov and M. Kos, "A GRASP heuristic for the delay-constrained multicast routing problem," *Telecommun. Syst.*, vol. 32, no. 1, pp. 55-69, 2006.

[8] Y. H. Du, J. Fang, and C. Miao, "Frequency-domain system identification of an unmanned helicopter based on an adaptive genetic algorithm," *IEEE Trans. Ind. Electron.*, vol. 61, no. 2, pp. 870-881, Feb. 2014.

[9] S. H. Chung and H. K. Chan, "A two-level genetic algorithm to determine production frequencies for economic lot scheduling problem," *IEEE Trans. Ind. Electron.*, vol.59, no.1, pp. 611-619, Jan. 2012.

[10] Z. H. Zhan, J. Zhang, Y. Li, O. Liu, S. K. Kwok, W. H. Ip, and O. Kaynak, "An efficient ant colony system based on receding horizon control for the aircraft arrival sequencing and scheduling problem," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 2, pp. 399-412, Jun. 2010.

[11] Y. J. Gong, M. Shen, J. Zhang, O. Kaynak, W. N. Chen, and Z. H. Zhan, "Optimizing RFID network planning by using a particle swarm optimization algorithm with redundant reader elimination," *IEEE Trans. Ind. Informat.*, vol. 8, no. 4, pp. 900-912, Nov. 2012.

[12] K. Ishaque, and Z. Salam, "A deterministic particle swarm optimization maximum power point tracker for photovoltaic system under partial shading condition," *IEEE Trans. Ind. Electron.*, vol. 60, no. 8, pp. 3195-3206, Aug. 2013.

[13] K. Shen, D. Zhao, J. Mei, L. Tolbert, W. Jianze, M. Ban, J. Yanchao, and X. Cai, "Elimination of harmonics in a modular multilevel converter using particle swarm optimization based staircase modulation strategy," *IEEE Trans. Ind. Electron.*, 2014, in press.

[14] R. B. Godoy, J. Pinto, C. A. Canesin, E. Alves Coelho, and A. Pinto, "Differential-evolution-based optimization of the dynamic response for parallel operation of inverters with no controller interconnection," *IEEE Trans. Ind. Electron.*, vol. 59, no. 7, pp. 2859-2866, Jul. 2012.

[15] Y. D. Hong, C. S. Park, and J. H. Kim, "Stable bipedal walking with a vertical center-of-mass motion by an evolutionary optimized central pattern generator," *IEEE Trans. Ind. Electron.*, vol. 61, no. 5, pp. 2346-2355, May 2014.

[16] G. Singh, S. Das, S. Gosavi, and S. Pujar, "Ant colony algorithms for Steiner trees: An application to routing in sensor networks," *Recent Development in Biologically Inspired Computing*, pp. 181-206, 2005.

[17] R. J. Wai, J. D. Lee, and K. L. Chuang, "Real-time PID control strategy for maglev transportation system via particle swarm optimization," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 629-646, Feb. 2011.

[18] Z. H. Zhan, J. Zhang, Y. Li, and H. Chung, "Adaptive particle swarm optimization," *IEEE Trans. Syst., Man, and Cybern. B*, vol. 39, no. 6, pp. 1362-1381, Dec. 2009.

[19] H. Wang, X. X. Meng, S. Li, and H. Xu, "A tree-based particle swarm optimization for multicast routing," *Computer Networks*, vol. 54, pp. 2775-2786, 2010.

[20] R. Qu, Y. Xu, J. P. Castro, and D. Landa-Silva, "Particle swarm optimization for the Steiner tree in graph and delay-constrained multicast

routing problems," *J. Heuristics*, vol. 19, no. 2, pp. 317-342, April. 2013.

[21] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst., Man, and Cybern.*, 1997, pp 4104-4109.

[22] W. N. Chen, J. Zhang, S. H. Chung, W. L. Zhong, W. G. Wu, and Y. H. Shi, "A novel set-based particle swarm optimization method for discrete optimization problems," *IEEE Trans. Evol. Comput.*, vol. 14, no. 2, pp. 278-300, Apr. 2010.

[23] W. L. Zhong, J. Huang, and J. Zhang, "A novel particle swarm optimization for the Steiner tree problem in graphs," in *Proc. IEEE Congr. Evol. Comput.*, 2008, pp. 2465-2472.

[24] Z. H. Zhan and J. Zhang, "Discrete particle swarm optimization for multiple destination routing problems," in *Proc. EvoWorkshops*, 2009, pp. 117–122.

[25] J. E. Beasley, "OR-library: Distributing test problems by electronic mail," *J. Opl. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, 1990.

[26] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc. IEEE Congr. Evol. Comput.*, 2002, pp. 1671-1676.

[27] Z. H. Zhan, J. Zhang, Y. Li, and Y. H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 15, no. 6, pp. 832-847, Dec. 2011.

[28] W. N. Chen, J. Zhang, Y. Lin, N. Chen, Z. H. Zhan, H. Chung, Y. Li, and Y. H. Shi, "Particle swarm optimization with an aging leader and challengers," *IEEE Trans. Evol. Comput.*, vol. 17, no. 2, pp. 241-258, 2013.

[29] K. Chan, T. Dillon, and E. Chang, "An intelligent particle swarm optimization for short-term traffic flow forecasting using on-road sensor systems," *IEEE Trans. Ind. Electron.*, vol. 60, no. 10, pp. 4714-4725, 2013.

[30] L. Han, J. Wang, X. Wang, and C. Wang, "Bypass flow-splitting forwarding in FISH networks," *IEEE Trans. Ind. Electron.*, vol. 58, no. 6, pp. 2197-2204, Jun. 2011.

[31] H. P. Li and Y. Shi, "Network-based predictive control for constrained nonlinear systems with two-channel packet dropouts," *IEEE Trans. Ind. Electron.*, vol. 61, no. 3, pp. 1574-1582, Mar. 2014.

[32] A. F. Zobaa, "Optimal multiobjective design of hybrid active power filters considering a distorted environment," *IEEE Trans. Ind. Electron.*, vol. 61, no. 1, pp. 107-114, Jan. 2014.

[33] Y. Gong, J. Zhang, H. Chung, W. Chen, Z. H. Zham, Y. Li, and Y. Shi, "An efficient resource allocation scheme using particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 6, pp. 801-816, Dec. 2012.

[34] Z. H. Zhan, J. Li, J. Cao, J. Zhang, H. Chung, and Y. H. Shi, "Multiple populations for multiple objectives: A coevolutionary technique for solving multiobjective optimization problems," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 445-463, 2013.

[35] M. Kodialam and T. Lakshman, "Dynamic routing of restorable bandwidth guaranteed tunnels using aggregated network resource usage information," *IEEE/ACM Trans. Networking*, vol. 11, no. 3, pp. 399-410, Jun. 2003.

Meie Shen received the B.S. degree in industrial automation from Huazhong University of Science and Technology, Wuhan, China, in 1986 and M.S. degree in automatic control from Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, China, in 1989.

Currently, she is an Associate Professor at the School of Computer Science in Beijing Information Science and Technology University. Her research interests include intelligent algorithms, automatic control theory and application.

Zhi-Hui Zhan (SM'09-M'13) received the Bachelor's degree and the Ph. D degree in 2007 and 2013, respectively, from the Department of Computer Science of Sun Yat-Sen University, Guangzhou, China.

He is currently a lecturer in Sun Yat-sen University. His current research interests include particle swarm optimization, differential evolution, ant colony optimization, genetic algorithms, and their applications in real-world problems.

Dr. Zhan's doctoral dissertation was awarded the China Computer Federation Outstanding Dissertation in 2013.

Wei-Neng Chen (SM'07-M'12) received the Bachelor's degree and the Ph.D. degree from the Department of Computer Science of Sun Yat-sen University, Guangzhou, China, in 2006 and 2012, respectively.

His doctoral dissertation was awarded the China Computer Federation outstanding dissertation in 2012. He is currently an associate professor with the School of Advanced Computing, Sun Yat-sen University. He has published more than 30 papers in international journals and conferences. His current research interests include swarm intelligence algorithms and their applications in real-world applications.

Yue-Jiao Gong (SM'10) received the B.S. degree in computer science in 2010, from Sun Yat-Sen University, Guangzhou, China, where she is currently working toward the Ph.D. degree.

Her current research interests include artificial intelligence, evolutionary computation, swarm intelligence, and their applications in design and optimization of intelligent transportation systems, wireless sensor networks, and RFID systems.

Jun Zhang (M'02–SM'08) received the Ph.D. degree from the City University of Hong Kong, Kowloon, Hong Kong, in 2002.

He is currently a Changjiang Chair Professor with the School of Advanced Computing, Sun Yat-sen University. He has published over 100 technical papers in his research areas. His current research interests include computational intelligence, cloud computing, high performance computing, data mining, wireless sensor networks, operations research, and power electronic circuits.

Dr. Zhang was a recipient of the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the *IEEE Transactions on Evolutionary Computation*, the *IEEE Transactions on Industrial Electronics*, and the *IEEE Transactions on Cybernetics*.

Yun Li (SM'87–M'90) received his B.S. in electronics science, M. Eng. in electronic engineering, and Ph.D. in computing and control in 1984, 1987 and 1990, respectively.

During 1989-1990, he was with U.K. National Engineering Laboratory and Industrial Systems and Control Ltd. He joined University of Glasgow as Lecturer in 1991, was Founding Director of University of Glasgow Singapore during 2011-2013 and Founding Director of the University's international joint programme with University of Electronic Science and Technology of China (UESTC) in 2013. He established IEEE Computer-Aided Control System Design Evolutionary Computation Working Group and European Network of Excellence in Evolutionary Computing Workgroup on Systems, Control, and Drives in 1998. He was invited to Kumamoto University, Japan, as Visiting Professor in 2002 and has been Visiting Professor to UESTC since 2004. Professor Li is a Chartered Engineer, has supervised over 20 Ph.D. students and has over 180 publications.