Available online at www.sciencedirect.com
ScienceDirect
Applied Soft Computing 8 (2008) 1643–1653
Applied Soft Computing
www.elsevier.com/locate/asoc

Solving shortest path problem using particle swarm optimization

Ammar W. Mohemmed, Nirod Chandra Sahoo *, Tan Kim Geok

Faculty of Engineering and Technology, Multimedia University, 75450 Melaka, Malaysia

Received 22 August 2006; received in revised form 25 December 2007; accepted 15 January 2008
Available online 19 January 2008

Abstract

This paper presents the investigations on the application of particle swarm optimization (PSO) to solve shortest path (SP) routing problems. A modified priority-based encoding incorporating a heuristic operator for reducing the possibility of loop-formation in the path construction process is proposed for particle representation in PSO. Simulation experiments have been carried out on different network topologies for networks consisting of 15–70 nodes. It is noted that the proposed PSO-based approach can find the optimal path with good success rates and also can find closer sub-optimal paths with high certainty for all the tested networks. It is observed that the performance of the proposed algorithm surpasses those of recently reported genetic algorithm based approaches for this problem.

Keywords: Shortest path problem; Particle swarm optimization; Path encoding

1. Introduction

The shortest path (SP) problem concerns with finding the shortest path from a specific origin to a specified destination in a given network while minimizing the total cost associated with the path. This problem has widespread applications. Some important applications of the SP problem include vehicle routing in transportation systems [1], traffic routing in communication networks [2] and path planning in robotic systems [3]. Furthermore, the shortest path problem also has numerous variations such as the minimum weight problem, the quickest path problem, and so on.

The SP problem has been investigated extensively. The well-known algorithms for solving this problem include the Bellman's dynamic programming algorithm for directed networks, the Dijkstra labeling algorithm and Bellman–Ford successive approximation algorithm for networks with non-negative cost coefficients. The details of these algorithms can be found in [4]. These traditional algorithms have major shortcomings; firstly, they are not suitable for networks with negative weights of the edges, i.e., in communication networks, the link weights represent the transmission line capacity and negative weights correspond to links with gain rather than loss. Secondly, the algorithms search only for the shortest route, but they cannot determine any other similar/non-similar short routes (which is commonly referred to as the *k*th SP problem). Thirdly, they exhibit high computational complexity for real-time communications involving rapidly changing network topologies such as wireless ad hoc networks. Therefore, new techniques have been continuously under investigation.

Artificial neural networks (ANN) have been examined to solve the SP problem relying on their parallel architecture to provide a fast solution [5–7]. However, the ANN approach has several limitations. These include the complexity of the hardware which increases considerably with increasing number of network nodes; at the same time, the reliability of the solution decreases. Secondly, they are less adaptable to topological changes in the network graph [7], including the cost of the arcs. Thirdly, the ANNs do not consider sub-optimal paths. Among other approaches for this problem, the powerful evolutionary programming techniques have considerable potential to be investigated in the pursuit for more efficient algorithms because the SP problem is basically an optimal search problem. In this direction, genetic algorithm (GA) has shown promising results [8–11]. The most recent notable results have been reported in [10]. Their algorithm shows better performance compared to those of ANN approach and overcomes the limitations mentioned above.

It is apparent that there is always a great need for more efficient optimization algorithms for the SP problem. Among the

* Corresponding author: Present address: Department of Electrical Engineering, Indian Institute of Technology, Kharagpur 721302, India.
Tel.: +91 3222 283052.
E-mail address: ncsahoo@ee.iitkgp.ernet.in (N.C. Sahoo).

1568-4946/$ – see front matter
doi:10.1016/j.asoc.2008.01.002

notable algorithms for path finding optimization problems in network graphs, successful use of GA and Tabu Search (TS) has been reported [12–14]. The success of these evolutionary programming approaches promptly inspires investigative studies on the use of other similar (and possibly more powerful) evolutionary algorithms for this problem. Particle Swarm Optimization is one such evolutionary optimization technique [15], which can solve most of the problems solved by GA with less computation cost [16]. It is to be noted that GA and TS demand expensive computational cost. Some more comparative studies of the performances of GA and PSO have also been reported [17–20]. All these studies have firmly established similar effectiveness of PSO compared to GA. Even for some applications, it has been reported that the PSO performs better than other evolutionary optimization algorithms in terms of success rate and solution quality. The most attractive feature of PSO is that it requires less computational bookkeeping and, generally, a few lines of implementation codes. The basic philosophy and science behind PSO is based on the social behavior of a bird flock and a fish school etc. Because of the specific algorithmic structure of PSO (updating of position and velocity of particles in a continuous manner), PSO has been mainly applied to many continuous optimization problems with few attempts for combinatorial optimization problems. Some of the combinatorial optimization problems that have been successfully solved using PSO are: task assignment problem [21], traveling salesman problem [22,23], sequencing problem [24] and permutation optimization problem [25], etc.

To the best knowledge of the authors, there is no reported work on the use of PSO for solving the core shortest path problem without any use of the classical algorithms such as the Dijkstra and Bellman–Ford algorithms. The purpose of this paper is to investigate on the applicability and efficiency of PSO for this problem. In this regard, this paper reports the use of particle swarm optimization to solve the shortest path problem, where a modified indirect encoding is used to represent the particle (position). In addition, a novel heuristic operator has been used for reducing the possibility of loop formation during potential path constructions (search procedure) from an origin node to a specific destination node in the network graph. The proposed algorithm has been tested by exhaustive simulation experiments on various random network topologies. The analysis of the results indicates the superiority of the PSO-based approach over those using GA [10,11].

The paper is organized as follows. In Section 2, PSO paradigm is briefly discussed. The particle encoding mechanism is presented Section 3 followed by the overall flow of the PSO algorithm for solving the SP problem being provided in Section 4. The results from computer simulation experiments are discussed in Section 5. Section 6 concludes the paper.

2. Particle swarm optimization: a brief overview

Particle swarm optimization is a population based stochastic optimization technique inspired by the social behavior of bird flock (and fish school, etc.), as developed by Kennedy and Eberhart [15]. As a relatively new evolutionary paradigm, it has grown in the past decade and many studies related to PSO have been published [26]. The algorithmic flow in PSO starts with a population of particles whose positions, that represent the potential solutions for the studied problem, and velocities are randomly initialized in the search space. The search for optimal position (solution) is performed by updating the particle velocities, hence positions, in each iteration/generation in a specific manner as follows. In every iteration, the fitness of each particle's position is determined by some defined fitness measure and the velocity of each particle is updated by keeping track of two "*best*" positions. The first one is the best position (solution) a particle has traversed so far. This value is called *pBest*. Another "*best*" value is the best position (solution) that any neighbor of a particle has traversed so far. This best value is a neighborhood best and is called *nBest*. When a particle takes the whole population as its neighborhood, the neighborhood best becomes the global best and is accordingly called *gBest*. A particle's velocity and position are updated as follows.

$$v_{id} = v_{id} + c_1 r_1 (b_{id} - x_{id}) + c_2 r_2 (b^n_{id} - x_{id});\ i = 1, 2, \ldots, N_s \quad \text{and} \quad d = 1, 2, \ldots, D \tag{1}$$

$$x_{id} = x_{id} + v_{id} \tag{2}$$

where c_1 and c_2 are positive constants, called *acceleration coefficients*, N_s is the total number of particles in the swarm, D is the dimension of problem search space, i.e., number of parameters of the function being optimized, r_1 and r_2 are two independently generated random numbers in the range [0,1] and "n" represents the index of the best particle in the neighborhood of a particle. The other vectors are defined as: $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]$ is the position of ith particle; $\mathbf{v}_i = [v_{i1}, v_{i2}, \ldots, v_{iD}]$ is the velocity of ith particle; $\mathbf{b}_i = [b_{i1}, b_{i2}, \ldots, b_{iD}]$ is the best position of the ith particle ($pBest_i$), and $\mathbf{b}^n_i = [b^n_{i1}, b^n_{i2}, \ldots, b^n_{iD}]$ is the best position found by the neighborhood of the particle i ($nBest_i$). The pseudo-codes for general algorithmic flow of PSO are listed in Fig. 1.

Eq. (1) calculates a new velocity for each particle based on its previous velocity, the particle's position at which the best possible fitness has been achieved so far, and the neighbors' best position achieved. Eq. (2) updates each particle's position in the solution hyperspace. c_1 and c_2 are two learning factors, which control the influence of *pBest* and *nBest* on the search process. In all initial studies of PSO, both c_1 and c_2 are taken to be 2.0 yielding good results [15]. However, in most cases, the velocities quickly attain very large values, especially for particles far from their global best. As a result, particles have larger position updates with particles leaving boundary of the search space. To control the increase in velocity, velocity clamping is used in Eq. (1). Thus, if the right side of Eq. (1) exceeds a specified maximum value $V^{\max}_d$, then the velocity on that dimension is clamped to $V^{\max}_d$. Many improvements have been incorporated into this basic algorithm. A review of these modifications can be seen in [27].

The commonly used PSOs are either global version or local version of PSO. In global version, all other particles influence the velocity of a particle, while in the local version of PSO, selected

Initialize the position and velocity of each particle in the population randomly.
Calculate fitness value of each particle.
Calculate *pBest* and *nBest* for each particle.
Do
  Update velocity of each particle using Eq. (1).
  Update position of each particle using Eq. (2).
  Calculate fitness value of each particle.
  Update *pBest* for each particle if its current fitness value is better than its *pBest*.
  Update *nBest* for each particle, i.e., choose the position of the particle with the best fitness value among all its neighbors as the *nBest* for a specific neighborhood topology.
While termination criterion is not attained.

Fig. 1. Pseudo-codes for general algorithmic flow of particle swarm optimization.

number of neighbor particles affect the particle's velocity. In [28], PSO is tested with regular shaped neighborhoods, such as global version, local version, pyramid structure, ring structure, Von Neumann topology. The neighborhood topology of the particle swarm has a significant effect on its ability to find optima: the optimal pattern of connectivity among individuals depends on the problem being solved. In ring topology, parts of the population that are distant from one another are also independent of one another. Influence spreads from neighbor to neighbor in this topology, until an optima, which really is the best, is found by any part of the population and then, this optima will eventually pull all the particles into it. In contrast, the global version where every particle is connected to all other particles, every particle influences all other particles immediately. The global populations tend to converge more rapidly than the ring populations, when they converge; but they are also more susceptible to convergence towards local optima [29].

In [30], Maurice proposed the use of a constriction factor χ; the algorithm was named the constriction factor method (CFM) accordingly. Here, Eq. (1) is modified as shown in Eq. (3).

$$v_{id} = \chi[v_{id} + c_1 r_1(b_{id} - x_{id}) + c_2 r_2(b_{id}^n - x_{id})] \quad (3)$$

where

$$\chi = 2(|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|)^{-1} \quad \text{if} \quad \varphi = c_1 + c_2 > 4 \quad (4)$$

The objective behind the use of constriction factor is to prevent the velocity from growing out of bounds, thus the velocity clamping is not required. However, Eberhart and Shi [31] have reported that the best performance can be achieved with constriction factor while using velocity clamping. It is readily observed that the PSO is very simple to be implemented. But, to formulate the problem in PSO framework, the important step is to devise a suitable coding scheme for particle representation. For the SP problem, there are some potential difficulties, which are discussed in the next section.

3. Particle encoding for shortest path problem

The thorniest problem in applying PSO (and GA) to the SP problem and similar ones like the traveling salesman problem is how to encode a path in a network graph into a particle in PSO (or a chromosome in GA). This encoding in turn affects the effectiveness of a solution/search process. A brief discussion on some of the existing path encoding techniques for solving the SP problem using GA[1] is presented followed by a detailed description of the proposed encoding algorithm.

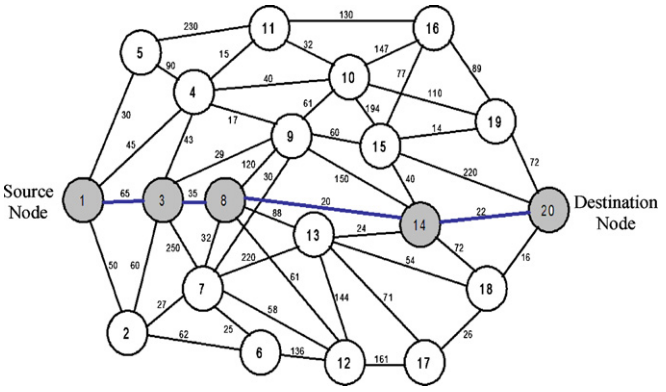


Fig. 2. A typical 20-node random network [10] where node numbers (IDs) are encircled. The weights of the connecting edges are also shown adjacent to the corresponding edges.

3.1. Existing path encoding techniques

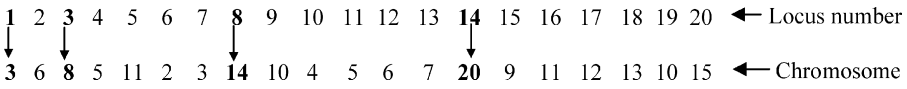
In the past, two typical encoding techniques have been used for path representations in solving the SP problem using GA. They are: direct and indirect representations.

In the direct representation scheme, the chromosome in the GA is represented by the sequence of node identification numbers, i.e., node IDs, appearing in a path starting from a source node and terminating at some destination node in the network. This encoding scheme is used in [8–10]. The GA-based approach used in [8,10] is practically feasible in wired/wireless environment. A variable-length chromosome of length equal to the number of nodes for encoding the problem has been used to list up node IDs from a source node to a destination based on a topological database of a network. To illustrate the scheme, a typical 20-node random network[2] (as used in [10]) is shown in Fig. 2. Following the representation scheme in [8] and

[1] To the best knowledge of the authors, no results using PSO have been reported for the core SP problem.

[2] Without any loss of generality, the source node is assigned the node ID equal to 1 and the destination node is assigned the node ID equal to 20 (=maximum number of nodes in the network). The IDs for the rest of the nodes are assigned in a left-to-right manner in increasing sequence starting from source to destination, as shown in Fig. 2.

[10], a typical path from node 1 and node 20 in Fig. 2 (shown in bold lines) is encoded as {1, 3, 8, 14, 20}. In [9], another similar (but slightly different) fixed-length chromosome representation has been used, i.e., each gene in a chromosome represents a node ID that is selected randomly from the set of nodes connected with the node corresponding to its locus number. For example, the path {1, 3, 8, 14, 20} in Fig. 2 is encoded as:



The disadvantage with these direct approaches is that a random sequence of node IDs may not correspond to a valid path (that terminates on destination node without any loop), increasing the number of invalid paths returned. An indirect scheme for chromosome representation scheme has been proposed by Gen et al. [11], where instead of node IDs directly appearing on the path representation, some guiding information about the nodes that constitute the path are used to represent the path. The guiding information used in that work is the priorities of various nodes in the network. During the initialization phase of GA, these priorities are assigned randomly. The path is generated by sequential node appending procedure beginning with the source node and terminating at the destination node. At each step of path construction from a chromosome, there are usually several nodes available for consideration and the one with the highest priority is added into path and the process is repeated until the destination node is reached. Fig. 3 illustrates this encoding scheme {$p1$, $p2$, ... are the priority values of nodes 1, 2, ..., respectively} and two typical chromosomes for the 20-node network of Fig. 2. Fig. 3(b) shows an example of indirect scheme for path representation from node 1 to node 20. The path construction starting from node 1 is performed as follows. From the node adjacency relations, the node with highest priority, i.e., node 3 (priority value = 60), is selected to be included in path, out of the nodes 2, 3, 4 and 5 (possible non-visited nodes to be visited from node 1). Then, out of the possible non-visited nodes that can be visited from node 3, node 8 is selected because of its highest priority and is put into the path. These steps will be repeated until a complete path {1, 3, 8, 14, 20} is obtained, as per the represented priority values shown in Fig. 3(b). It should be noted that, for the decoding scheme to

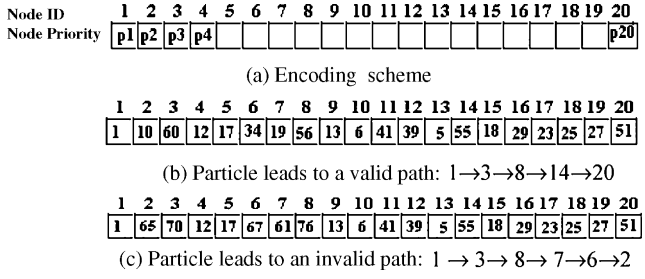


Fig. 3. Illustrations of priority-based encoding scheme [11] for the 20-node network of Fig. 2.

be effective, a dynamic node adjacency matrix is maintained in the computer implementation [11] and it is updated after selection of a node to be included in a path so that the selected node will not be a candidate for future selection. This increases memory, especially for large number of nodes, and time complexity of the algorithm. Moreover, there may be situations when the path does not terminate at the destination node leading to an invalid path. This situation is illustrated in Fig. 3(c). In this case, the partial path takes the node sequence {1, 3, 8, 7, 6, 2}. From node 2, the partial path should not be allowed to grow further as it will result in a loop (the solution to the SP problem must not include any loop). In this sense, such types of nodes (node 2) can be called as no-exit nodes. Therefore, the possibility of invalid path generation (due to such no-exit nodes) is high resulting in substantial fruitless computational effort. However, it is worth noting that this encoding scheme has obvious advantages over direct encoding schemes in terms of increased possibility of valid path generation.

3.2. Proposed modifications in the priority-based path encoding algorithm

The proposed path-encoding algorithm for PSO is essentially based on the above discussed indirect priority-based encoding. The direct encoding scheme is not preferred since a random sequence of nodes is definitely not a good choice for path construction because of the reasons mentioned earlier. But, the priority-based encoding scheme is suitably modified to address the concerns raised earlier, i.e., maintaining dynamic adjacency matrix and possibility of arriving at no-exit nodes during path construction. Thus, the position vector of a particle in PSO is represented by a priority vector of the type shown in Fig. 3(a) along with the following incorporated modifications.

(a) The nodes are allowed to take both positive and negative priority values of any magnitude. By doing so, the search space becomes wider without any constraint boundaries. Under this modification, the node that is already included in a growing path will be assigned a large negative priority value (for example, $-N_\infty = -50000$); thus that node is highly unlikely to selected again while the PSO (with velocity clamping and CFM) runs for a finite number of iterations. Even with the remotest occurrence of a node being selected again, the concerned path can be treated as an invalid path, which is computationally more efficient compared to maintaining a dynamic adjacency matrix. The specific advantage is that, at each step of path construction corresponding to a particle, there is absolutely no need to update a dynamic adjacency matrix; rather a dynamic priority vector is simply updated.

(b) A closer look at the possibility of arriving at no-exit nodes reveals that one of the main reasons is the growth of the path in backward direction, not in the forward direction towards the destination. However, it is also emphasized that not all partial backward paths lead to such a situation. To reduce the possibility of building a backward path (hence, possible loop formation) and also, simultaneously keeping some room for any potential backward movement, a heuristic operator is incorporated as follows. A node is selected as *to-be-next node* in the growing path if its ID is larger than the present node ID by a certain specified value, i.e., if

$$\text{ID of to-be-next node} - \text{ID of present node} > -M, \text{where } M \text{ is a positive integer} \quad (5)$$

If a node is rejected based upon Eq. (5), then another node with highest priority among the rest is tested for possibility of inclusion in the growing path. This heuristic operator will try to prevent the path from growing too much backward as long as there are available alternative links. Thereby, it also reduces the possibility of loop formation.

In the following sub-section, the pseudo-codes for implementation of priority-based encoding algorithm incorporating these two modifications are provided with an illustrative example.

3.3. Pseudo-codes for path encoding algorithm

Let $N_{\max}$ be the maximum number of nodes in the network. Let V_p^k be a partial path (corresponding to the position/priority vector of a particle) under growth, which contains $k+1$ nodes with the terminal node t^k ($k=0$ corresponds to the partial path with source node only), and $(k+1)$th node is to be selected. Let $\mathbf{x}^k$ be the dynamic priority vector, which initially contains the priority values (position vector of the particle), referred to by $\mathbf{x}$. Every time a node is added to the partial path, the corresponding position in $\mathbf{x}^k$ is given a large negative value ($-N_\infty$) as explained earlier. Without loss of generality, node number 1 is taken as source node and destination node ID is $N_{\max}$. The implementations of the modified priority-based encoding along with gradual path construction process are summarized in the following steps:

Step 1: (Initialization) Let $k \leftarrow 0$, $V_p^k \leftarrow \{1\}$ and $\mathbf{x}^k \leftarrow \mathbf{x}$; $t^k \leftarrow 1$ and $\mathbf{x}^k(t^k) = -N_\infty$.

Step 2: (Termination test) If $t^k = N_{\max}$, or $k > N_{\max}$, go to *Step 4*; else $k \leftarrow k+1$ and go to *Step 3*.

Step 3: (Path extension) Select node t^k as the node with highest priority from among the nodes having direct links with node t^{k-1} if $(t^k - t^{k-1}) > -M$. Set $V_p^k \leftarrow \{V_p^{k-1}, t^k\}$ and $\mathbf{x}^k(t^k) = -N_\infty$.

Step 4: (Complete path) Return complete valid path V_p^k or return invalid path V_p^k if the terminal node is not the destination node.

In *Step 2*, if the number of iteration exceeds $N_{\max}$, it would mean either a valid path has not been found due to loops or the path does not terminate at the destination node in $N_{\max}$ steps. In that case, the objective function evaluation (discussed in the next section) of the corresponding particle is made to return a very low value as penalty. In *Step 3*, the nodes having direct links with the terminal node of the already grown partial path can be found from the network topology. The value of M should be judiciously decided based on network topology. An illustrative example of the different steps of execution of the above pseudo-codes is shown in Fig. 4. This example corresponds to the priority (particle position) vector shown in Fig. 3(c) resulting in an invalid path without proposed modifications as noted earlier. Fig. 4 shows how the proposed modifications (parameter $M=4$) to create a valid path from the same position vector and the corresponding final path is {1, 3, 8, 7, 6, 12, 17, 18, 20}, where use of Eq. (5) forces the algorithm to choose node 12 instead of node 2 (although it has the highest priority) thereby avoiding an invalid path creation. This operation reduces the number of invalid paths (thereby, the computation time).

4. Complete algorithm for shortest path problem

The quality of a particle (solution) is measured by a fitness function. Here, the fitness function is obvious as the goal is to find the minimal cost path. Thus, the fitness of ith particle is defined as:

$$f_i = \left(\sum_{j=1}^{N_i - 1} C_{yz} \right)^{-1}, \; y = \mathbf{PP}^i(j) \quad \text{and} \quad z = \mathbf{PP}^i(j+1) \quad (6)$$

where $\mathbf{PP}^i$ is the set of sequential node IDs for ith particle, $N_i = |\mathbf{PP}_i|$ = number of nodes that constitute the path represented by ith particle, and C_{yz} is the cost of the link connecting node y and node z. Thus, the fitness function takes maximum value when the shortest path is obtained. If the path represented by a particle happens to be an invalid path, then its fitness is assigned a penalty value (=0) so that the particle's attributes will not be considered by others for future search. The pseudo-codes for the complete PSO-based algorithm for shortest path search are provided in Fig. 5.

5. Simulation results and discussions

The proposed PSO-based algorithm for SP search is tested on networks with random and varying[3] topologies through computer simulations using Microsoft Visual C++ on an Intel III processor (850 MHz clock). The results are compared with two recently reported GA-based approaches, i.e., one uses direct encoding scheme [10] and the other uses the priority vector based indirect encoding scheme (but without the

[3] The random network topologies are generated by some random distribution of nodes in some pre-specified area in a two-dimensional Cartesian coordinate plane. The neighbors of a node are those nodes that appear within some pre-specified distance (transmission range) of the node. The average degree of nodes in the network depends upon the area size chosen.

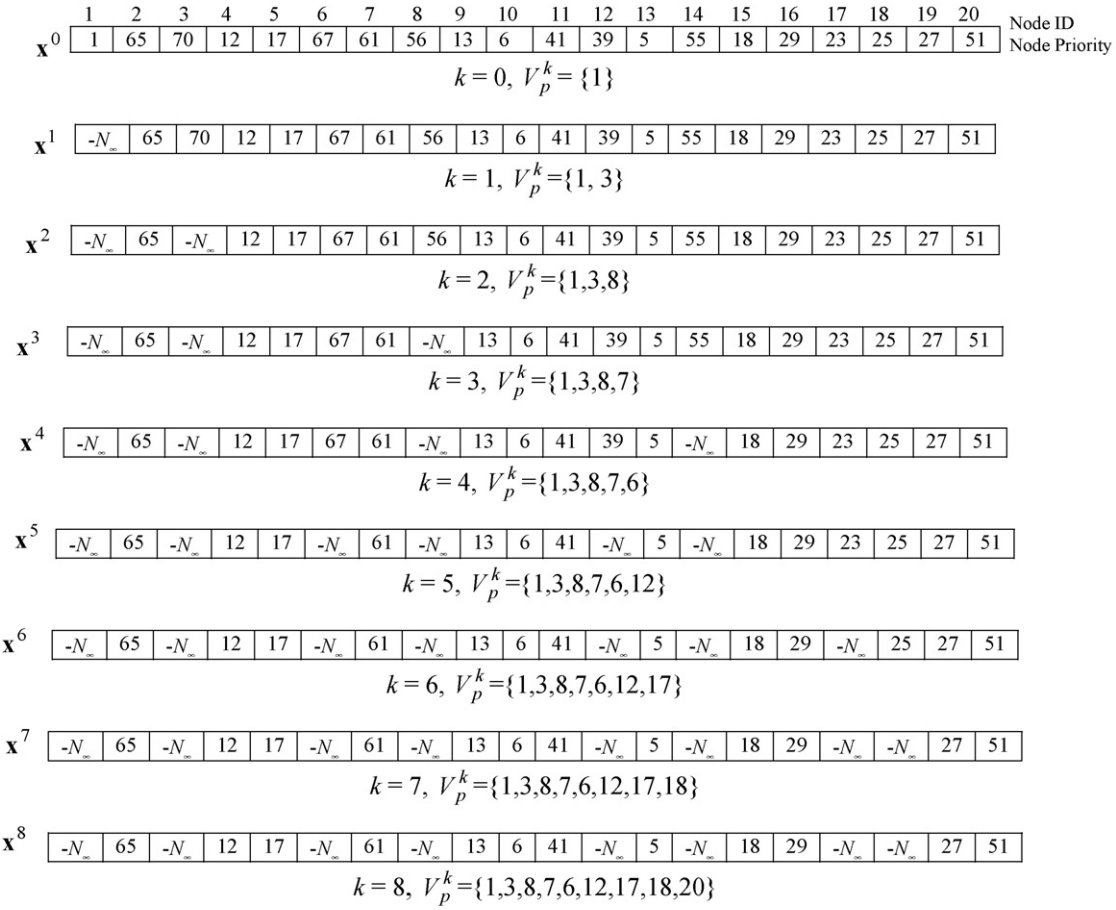


Fig. 4. Particle priority vector updating to construct the path {1, 3, 8, 7, 6, 12, 17, 18, 20} illustrating the use of proposed modifications in path encoding.

modifications proposed in this work) [11]. The selection of parameter settings of PSO is discussed now:

(a) *Population size*: In general, any evolutionary search algorithm shows improved performance with relatively larger population. However, very large population size means greater cost in terms of fitness function evaluations. In [32,33], it is stated that a population size of 30 is a reasonably good choice (it is small enough to be efficient, yet large enough to produce reliable results). In this study, the effects of different population size have been investigated.

(b) *Particle initialization*: The particle's position vector which represents the node priorities is initialized with random integer values in the range [−100, 100] and the velocities in the range [−10, 10]. Choosing integer values rather than real numbers reduces the memory space requirement. Also,

```
Initialize the particle population (priority and velocity vector for each particle) randomly.
Evaluate fitness of each the particle {
    Construct a path from particle priority vector as discussed in Section 3.
    If it is a valid path,
        return its fitness value calculated by Eq. (6).
    else
        return a penalty value(= 0) as its fitness.}
Calculate pBest and nBest for each particle.
Iteration_Count = 1;
Do
    Calculate velocity of each particle using Eqs. (3) and (4).
    Apply velocity clamping on each dimension if necessary (as discussed in Section 2).
    Update position of each particle using Eq. (2).
    Evaluate fitness value of each particle.
    Update pBest for each particle if its current fitness value is better than pBest.
    Update nBest for each particle, i.e., choose the particle with the best fitness value among all
    the neighbors as the nBest for a specific neighborhood topology.
    Iteration_Count = Iteration_Count + 1;
While maximum number of iterations is not attained {Iteration_Count < Maximum_Iterations}.
```

Fig. 5. Pseudo-codes for solving SP problem using PSO.

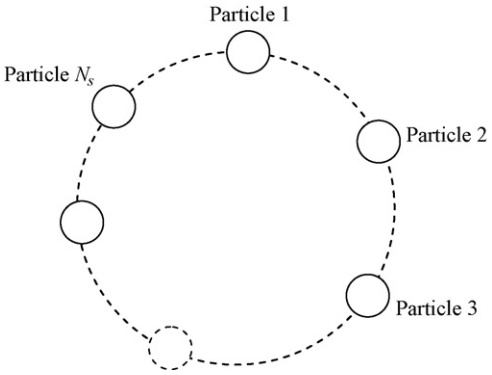


Fig. 6. Ring topology for particle neighborhood.

while performing comparison of priorities of various nodes during path extension, the integer number comparison is generally efficient compared to real number comparison. The initial velocities are chosen to be smaller so as to mimic the social behavior of birds at the beginning of search process. It should also be noted that the velocity updating by Eq. (3) gives non-integer velocity values, which are again converted to the nearest integer in the implementation. The maximum velocity for all the search dimensions is set as ±3000.

(c) *Neighborhood topology*: Ring neighborhood topology [29] is used to avoid premature convergence to local optima. In this topology, each particle in the population is connected to its two immediate neighbors as shown in Fig. 6.

(d) *Constriction factor* χ: In [34], it is shown that the CFM has almost linear convergence for $\varphi > 4$. Here, both c_1 and c_2 are chosen to be 2.05; thus $\varphi = 4.1$. From Eq. (4), $\chi = 0.729$.

(e) During the path construction process, the value of M in Eq. (5), as discussed in Section 3, is set to 4. The algorithm is set to run for a maximum of 500 iterations unless stated otherwise.

In the following, the simulation results are compared with those obtained from GA-based search using direct path encoding scheme [10] and indirect (priority-based) path encoding scheme [11]. In all the simulation tests, the optimal solution obtained using Dijkstra's algorithm [4] is used as reference for comparison purposes.

5.1. Comparison of results with those of GA-based search using direct path encoding [10]

The main objective of these simulation experiments is to investigate the quality of solution and convergence speed for different network topologies and number of nodes. First, the quality of solution (route optimality) is investigated. The route optimality is defined as the percentage of time the PSO finds the global optimum (i.e., the shortest path) [10]. The route failure ratio is the inverse of route optimality. It is asymptotically the probability that the computed route is not optimal, because it is the relative frequency of route failure [10]. The 20-node fixed topology network [10] shown in Fig. 2 is considered first. For comparison of quality of solution, the shortest path solution computed by Dijkstra's algorithm is taken as reference and this path is shown in bold lines in Fig. 2. Using the proposed PSO algorithm with a population size of 25, this optimum path is found after two iterations only. However, it is observed that the average number of iterations required to obtain the optimal path with different random topologies for 20-node network is 13.

To verify the results for different network topologies, randomly generated networks with 15–50 nodes with randomly assigned link costs are investigated for shortest path solution. A total of 1000 random network topologies were considered in each case (number of nodes). A comparison the quality of solution in terms of route failure ratio between the proposed PSO-based search and GA-based search reported in [10] (where the number of chromosomes in each case is same as the number of nodes in the network) is shown in Fig. 7. In Fig. 7, the PSO results for a fixed population size of 25 as well as the population size being equal to number of nodes are shown for fair comparison. It clearly illustrates that the quality of solution obtained with PSO is higher than that of GA-based search. For example, in case of 45 node-networks, the route failure ratio is 0.14 (86% route optimality) with a population size of 25; but the GA search has route failure ratio 0.36 (64% route optimality) with population size equal to the number of nodes, i.e., 45. The overall statistics of these results are collected in Table 1. The PSO search attains an average route failure ratio 0.0615 (94% route optimality) with a population size equal to 25 compared to 0.1712 for the GA search [10]. The standard deviation of route failure ratio for the PSO search amounts 0.0197 compared to 0.1067 for GA search. Moreover, as seen from Fig. 7, when the population size in PSO is equal to the number of nodes in each case, the performance of proposed algorithm is clearly superb compared to the GA-based approach. In the following results presented in this sub-section, the population size in PSO is fixed to 25 irrespective of network topology.

Fig. 8 shows the comparison of the average CPU time required to achieve the results shown in Fig. 7, where the population size in PSO is fixed to 25. For less than 40 nodes, the

Table 1
Comparison of statistics of the quality of solution (population size in PSO = 25)

Performance measure	Algorithms	
	GA search [10]	PSO search (proposed)
Route failure ratio		
Average	0.1712	0.061548
Standard Deviation	0.1067	0.0917

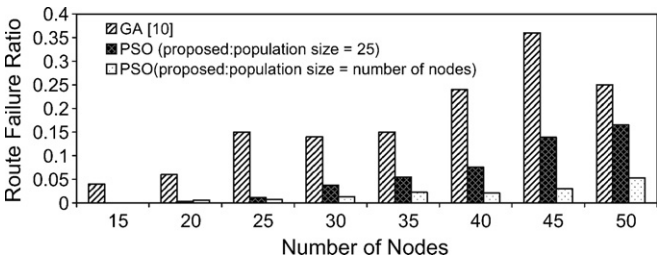


Fig. 7. Comparison of route failure ratio between PSO and GA [10] for networks of varying topologies.

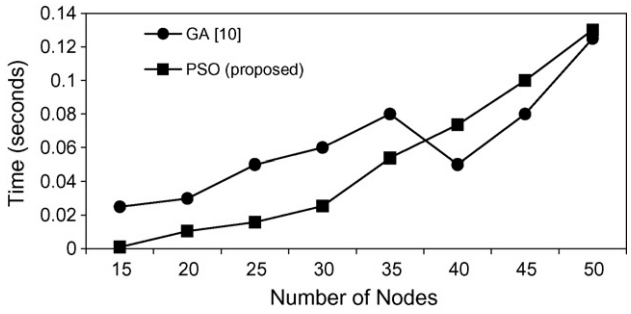


Fig. 8. Comparison of convergence time between PSO (population size = 25) and GA [10].

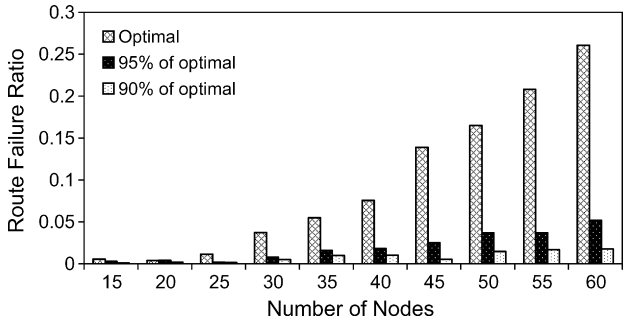


Fig. 9. Route failure ratio to achieve optimal, 95% and 90% of optimal path with PSO search (population size = 25).

average time required for PSO is less than those with GA search [10]. For subsequent higher node networks, the two algorithms almost have the same time efficiency. However, it must be noted that, in the results reported for GA search [10], the population size is taken equal to number of nodes. Thus, from this perspective, the PSO with fixed population size of 25 performs better in comparison.

In addition to seeking solution for optimal path, it is also equally important to look for closer sub-optimal paths. This is quite useful in real-time computer networking. Fig. 9 shows the route failure ratio to achieve the optimal path, 95% and 90% of the optimal path {path cost = 95% (and 90%) of the optimal path cost} with the proposed algorithm. Although the failure ratio increases with the increasing number of nodes, it is still lower to get the 95% of optimal path and 90% of the optimal path. For example, 60-node network has route optimality of 95% for near optimum path with the proposed PSO search (population size = 25). Finally, Figs. 10 and 11 show the average number of iterations and the number of times the objective function evaluation is performed, respectively, with PSO search (population size = 25). As expected, the iteration number increases with node number. Thus, the proposed algorithm is very much suitable for small and medium size networks in real time. For networks with large number of nodes, it would be better to consider parallel implementation of particle swarm optimization.

5.2. Comparison of results with those of GA-based search using indirect path encoding [11]

For performance comparison of PSO-based search using modified indirect encoding scheme with those reported in

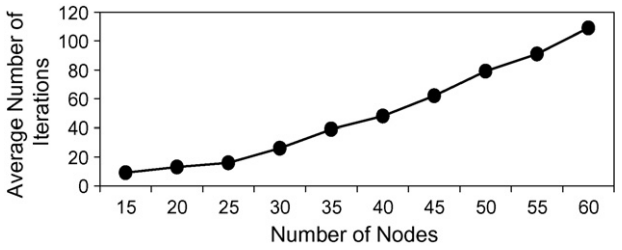


Fig. 10. Average number of iterations to obtain optimal path with PSO search (population size = 25).

[11] using GA-based search and indirect encoding scheme, same testing conditions are simulated. However, [11] only reports the number of nodes and edges in all the used networks and no information on the cost of the edges is provided. Thus, the closest possible network is generated in this study where the number of nodes of each network is exactly the same used in [11] and the number of edges is as close as possible to those of [11]. The results are summarized as follows.

(a) Test #1

Three random networks of different sizes are generated. The testing conditions are given in Table 2. The statistical results for frequency of obtaining optimal solution over 400 independent runs (for each network) are compared in Table 3. Clearly, the proposed PSO-based search performs better.

(b) Test #2

In this test, the effects of population size on convergence characteristics are compared. The number of generations/iterations in every run is fixed. The testing conditions are: number of iterations/generations = 200, the chosen network is that mentioned for case study III (only) in Table 2 for respective algorithms, population sizes for both are varied from 10 to 100. The comparisons of frequency for obtaining optimal path obtained from 200 random runs (for each population size) are summarized in Table 4. As anticipated, the frequency for obtaining optimal solution increases with population size for both approaches. The superior performance of the proposed PSO-based search is again highlighted in the results.

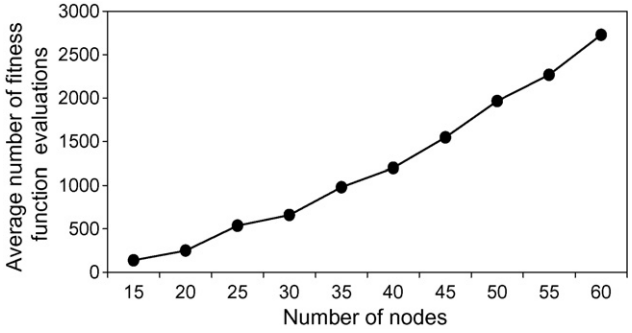


Fig. 11. Average number of fitness function evaluations to obtain the optimum path with PSO search (population size = 25).

Table 2
Different testing conditions for Test #1

Case Study	Network used in [11]: number of (nodes, edges)	Network used here: number of (nodes, edges)	Population size used in search	Number of generations/iterations
I	(6,10)	(6, 10)	10	50
II	(32,66)	(32, 66)	20	100
III	(70,211)	(70,216)	40	200

Table 3
Comparison of statistical results between GA-based search [11] and proposed algorithm for Test #1

Case study	Frequency for obtaining the optimal path	
	GA-based search (%) [11]	Proposed PSO based search (%)
I	100	100
II	98	100
III	64	80

Table 4
Comparison of statistical results between GA-based search [11] and proposed algorithm for Test #2

Population size	Frequency for obtaining the optimal path	
	GA-based search (%) [11]	Proposed PSO based search (%)
10	21	33
40	64	81
60	83	90
100	92	99.5

(c) Test #3

In this test, the convergence characteristics of search algorithms are compared when the population size is fixed and the number of generations/iterations is gradually increased from 100 to 3000. The chosen networks for both the search algorithms are again the respective networks given in case study III in Table 2. The results over 200 runs for each case are summarized in Table 5. For population size of 10 in both GA- and PSO-based searches, it is observed that the success rates for PSO are better when number of iterations is less than 1200. But, as the searches are performed for more iterations, GA performs better than PSO (population size = 10). These results indicate that the gradual (iterative) improvements in the performance of PSO greatly depend upon the population size. It is expected since the very basic concept behind this PSO search is the interaction between the particles in the population. More particles means more interactive search and the search space can be explored more effectively with more particles. In order to verify this fact, this test is repeated for PSO with two additional cases for different population sizes (15 and 20). The statistical results are also reported in Table 5. Clearly, the frequency for obtaining the optimal path increases as more particles are allowed in the PSO population.

5.3. Overall remarks on performance of the proposed algorithm

The simulation experiments performed in this study consistently show superior performance of PSO over that of GA This can be attributed to the following characteristics of the proposed PSO-based algorithm for this particular problem.

- In PSO, the determination of new search directions based on a sort of heuristic combination of present velocity and position of the particle, best position of the particle attained so far, and position of the global/neighborhood best particle offers advantages in terms efficient exploration of search space resulting in attractive convergence characteristics for optimization problems.
- It is widely known that an effective encoding of the problem is very much essential for efficient search of problem/solution space as well as convergence characteristics of any evolutionary programming based search algorithm. For example, it is already seen that an indirect coding has better attributes compared to direct coding for this shortest path problem, when used in GA-based search algorithm. In indirect coding of potential solutions, the search domain

Table 5
Comparison of statistical results between GA-based search [11] and proposed algorithm for Test #3

Number of generations/iterations	Frequency for obtaining optimal path			
	GA-based search [11] (population size = 10) (%)	Proposed PSO based search		
		Population size = 10 (%)	Population size = 15 (%)	Population size = 20 (%)
100	10	57	68	81
400	42	76	86	96.8
800	66	78	88	97
1200	76	80	88	97.5
2000	92	81	89	97.6
3000	94	82	90	97.8

generally becomes wider than the original problem domain and, hence, this offers more freedom for the search to be performed effectively.

- Further, the two modifications incorporated to this priority-based encoding further enhance the performance of the PSO search, i.e.,
  (1) The provisions for both positive and negative node-priority values offer more freedom for particle movement (wider search space).
  (2) The incorporation of the heuristic operator for reducing the possibility of invalid path generation further improves the overall performance of the algorithm.
- Finally, the PSO requires less computational bookkeeping and few lines of implementation codes compared to GA.

6. Conclusions

In this paper, investigative results on using particle swarm optimization to solve the shortest path problem have been reported. The proposed PSO-based search uses a modified (priority-based) indirect path-encoding scheme so as to widen the scope of search space and to reduce the probability of invalid path/loop creation during the path construction procedure using a heuristic operator. The performance of the proposed approach has been compared with those reported in two recent works using GA-based search (one of these uses direct encoding while the other uses indirect encoding) by carrying out exhaustive simulation tests on different random networks with varying number of nodes (from 15 to 70 nodes). The results are highly encouraging with much superior performance exhibited by the proposed PSO-based search. Moreover, in addition to obtaining the shortest path, the proposed algorithm also successfully finds near-optimal paths, i.e., path with costs of 95% (or 90%) of the optimal path cost, with good success rates. This feature is highly beneficial in real-time computer networking. It should also be noted that the performances of the PSO-based approach for SP problem can be further improved by incorporating adaptive tuning features to fine tune the parameters of PSO and investigations are underway for this aspect.

References

[1] F.B. Zahn, C.E. Noon, Shortest path algorithms: An evaluation using real road networks, Transport. Sci. 32 (1998) 65–73.
[2] Moy, J., 1994. Open Shortest Path First Version 2. RFQ 1583, Internet Engineering Task Force. http://www.ietf.org.
[3] G. Desaulniers, F. Soumis, An efficient algorithm to find a shortest path for a car-like robot, IEEE Trans. Robot. Automat. 11 (6) (1995) 819–828.
[4] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart, and Winston, New York, 1976, pp. 59–108.
[5] M.K. Ali, F. Kamoun, Neural networks for shortest path computation and routing in computer networks, IEEE Trans. Neural Netw. 4 (1993) 941–954.
[6] J. Wang, A recurrent neural network for solving the shortest path problem, in: Proceedings of the IEEE International Symposium on Circuits and Systems, 1994, pp. 319–322.
[7] F. Araujo, B. Ribeiro, L. Rodrigues, A neural network for shortest path computation, IEEE Trans. Neural Netw. 12 (5) (2001) 1067–1073.
[8] M. Munemoto, Y. Takai, Y. Sato, A migration scheme for the genetic adaptive routing algorithm, in: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1998, pp. 2774–2779.
[9] J. Inagaki, M. Haseyama, H. Kitajima, A genetic algorithm for determining multiple routes and its applications, in: Proceedings of the IEEE International Symposium on Circuits and Systems, 1999, pp. 137–140.
[10] C.W. Ahn, R.S. Ramakrishna, A genetic algorithm for shortest path routing problem and the sizing of populations, IEEE Trans. Evol. Comput. 6 (6) (2002) 566–579.
[11] M. Gen, R. Cheng, D. Wang, Genetic Algorithms for solving shortest path problems, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 1997, pp. 401–406.
[12] G. Raidl, A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem., in: Proceedings of the ACM Symposium on Applied Computing, 2000, pp. 440–445.
[13] Z. Fu, A. Kurnia, A. Lim, B. Rodrigues, Shortest path problem with cache dependent path lengths, in: Proceedings of the Congress on Evolutionary Computation, 2003, pp. 2756–2761.
[14] J. Kuri, N. Puech, M. Gagnaire, E. Dotaro, Routing foreseeable light path demands using a tabu search meta-heuristic, in: Proceedings of the IEEE Global Telecommunication Conference, 2003, pp. 2803–2807.
[15] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[16] R. Hassan, B. Cohanim, O.L. DeWeck, G. Venter, A comparison of particle swarm optimization and the genetic algorithm, in: Proceeedings of the First AIAA Multidisciplinary Design Optimization Specialist Conference, 2005, 18–21.
[17] E. Elbeltagi, T. Hegazy, D. Grierson, Comparison among five evolutionary-based optimization algorithms, Adv. Eng. Inform. 19 (1) (2005) 43–53.
[18] C.R. Mouser, S.A. Dunn, Comparing genetic algorithms and particle swarm optimization for an inverse problem exercise., Aust. N. Z. Ind. Appl. Math. (ANZIAM) J. 46(E) (2005) C89–C101.
[19] R.C. Eberhart, Y. Shi, Comparison between genetic algorithms and particle swarm optimization., in: Proceedings of the seventh annual conference on Evolutionary Programming (Springer-Verlag), 1998, pp. 611–616.
[20] D.W. Boeringer, D.H. Werner, Particle swarm optimization versus genetic algorithms for phased array synthesis, IEEE Trans. Antennas Propagat. 52 (3) (2004) 771–779.
[21] A. Salman, A. Imtiaz, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocess. Microsyst. 26 (8) (2002) 363–371.
[22] K.-P. Wang, L. Huang, C.-G. Zhou, W. Pang, Particle swarm optimization for traveling salesman problem, in: Proceedings of International Conference on Machine Learning and Cybernetics, 2003, pp. 1583–1585.
[23] Clerc, Maurice, 2000. Discrete particle swarm optimization illustrated by the traveling salesman problem. http://www.mauriceclerc.net.
[24] L. Cagnina, S. Esquivel, R. Gallard, Particle swarm optimization for sequencing problem: a case study., in: Proceedings of the IEEE Conference on Evolutionary Computation, 2004, pp. 536–541.
[25] X. Hu, R.C. Eberhart, Swarm intelligence for permutation optimization: a case study of n-queens problem., in: Proceedings of the IEEE Swarm Intelligence Symposium, 2003, pp. 243–246.
[26] H. Xiaohui, S. Yuhui, R. Eberhart, Recent advances in particle swarm, in: Proceedings of the CEC Congress on Evolutionary Computation, vol. 1, 2004, pp. 90–97.
[27] S. Yuhui. Particle swarm optimization. Feature article, IEEE Neural Networks Society. February, 8–13, 2004.
[28] J. Kennedy, R. Mendes, Population structure and particle swarm performance., in: Proceeding of the Congress on Evolutionary Computation, vol. 2, 2002, 1671–1676.
[29] J. Kennedy, Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, in: Proceedings of the Congress on Evolutionary Computation, 1999, pp. 1931–1938.
[30] C. Maurice, The swarm and queen: Towards a deterministic and adaptive particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 1999, pp. 1951–1957.

[31] R.C. Eberhart, Y. Shi, Comparing inertia weight and constriction factors in particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2000, pp. 84–88.
[32] Y. Shi, R.C. Eberhart, Empirical study of particle swarm optimization, in: Proceedings of the Congress on Evolutionary Computing, vol. III, 1999, pp. 1945–1950.
[33] A. Carlisle, G. Dozier, An off-the-shelf PSO, in: Proceedings of the Workshop on Particle Swarm Optimization, 2001, pp. 1–6.
[34] M. Clerc, J. Kennedy, The particle swarm explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (1) (2002) 58–73.