Hindawi
Discrete Dynamics in Nature and Society
Volume 2019, Article ID 9085320, 11 pages
https://doi.org/10.1155/2019/9085320

*Research Article*

# Probability Mechanism Based Particle Swarm Optimization Algorithm and Its Application in Resource-Constrained Project Scheduling Problems

**Shuai Li, Zhicong Zhang, Xiaohui Yan, and Liangwei Zhang**

*Department of Industrial Engineering, Dongguan University of Technology, Songshan Lake District, Dongguan 523808, Guangdong Province, China*

Correspondence should be addressed to Shuai Li; lishuai@dgut.edu.cn

Received 1 November 2018; Accepted 15 April 2019; Published 2 May 2019

Academic Editor: Florentino Borondo



In this paper, a new probability mechanism based particle swarm optimization (PMPSO) algorithm is proposed to solve combinatorial optimization problems. Based on the idea of traditional PSO, the algorithm generates new particles based on the optimal particles in the population and the historical optimal particles in the individual changes. In our algorithm, new particles are generated by a specially designed probability selection mechanism. We adjust the probability of each child element in the new particle generation based on the difference between the best particles and the elements of each particle. To this end, we redefine the speed, position, and arithmetic symbols in the PMPSO algorithm. To test the performance of PMPSO, we used PMPSO to solve resource-constrained project scheduling problems. Experimental results validated the efficacy of the algorithm.

## 1. Introduction

Particle Swarm Optimization (PSO) is an evolutionary computational technique proposed by Kennedy and Eberhart [1] in 1995 for continuous optimization problems. Since the PSO algorithm only needs to adjust a small number of parameters and it is easy to implement, theories related to PSO algorithm has been greatly developed in the last two decades [2]. In the literature, there are many successful PSO applications for single- or multiobjective optimization [3, 4].

Although PSO was developed for continuous optimization problems initially, some work focused on its discrete version (discrete PSO and DPSO). Kennedy and Eberhart proposed a discrete binary version of particle swarm algorithm in 1997 [5], where the trajectories of the particles are changes in the probability that a coordinate will take on a zero or one value. Clerc [6] gave out a brief outline of the PSO method for solving Traveling Salesman Problems (TSP). Hendtlass used the PSO algorithm to solve small-size TSP problems and improved its performance by adding a memory capacity to each particle [7]. Pang et al. [8] used fuzzy matrixes to represent the position and velocity of the particles in the PSO and the operators in the original PSO formulas were redefined. Wang et al. [9] redefined the PSO operators by introducing the concepts of "Swap operator" and "Swap sequence". They solved the TSP problems in another way. We refer to all the DPSO algorithms above as actual DPSO algorithms, since they are from the original PSO. These studies have chosen some of the benchmark questions in the Traveling Salesman's Problems Library (TSPLIB), where the city size does not exceed 52.

Some researchers combined the DPSO with other algorithms to obtain better results or enlarge the problems scale. Zhong et al. introduced a new parameter named mutation factor to the DPSO for solving the TSP [10], and the particle is not a permutation of numbers but a set of edges. Yuan et al. [11] proposed a novel algorithm based on particle optimization algorithm (PSO) and Chaotic Optimization Algorithm (COA) to solve TSP problems. Shi et al. used an uncertain searching strategy and a crossover eliminated technique to design a novel particle swarm optimization algorithm for the traveling salesman problem [12]. Machado

et al. [13] presented a new hybrid model, based on particle swarm optimization, genetic algorithms, and fast local search, for solving traveling salesman problem. Fang et al. [14] combined particle swarm optimization with simulated annealing for TSP. Except traveling salesman problem, other problems were taken into account by many scholars. Anghinol and Paolucci [15] presented a new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem. Jia et al. [16] improved particle swarm optimization (PSO) algorithm with rank-priority-based representation employing a double justification skill for solving the resource-constrained project scheduling problem (RCPSP). Although most of the above hybrid algorithms have achieved good results, it is difficult to say that the discretization of PSO is successful because they all combine other algorithms.

For problems such as RCPSP and TSP, an excellent solution corresponds to a good arrangement consisting of good elements. In order to obtain these excellent elements, we propose a new DPSO algorithm PMPSO (probability-based particle swarm optimization algorithm). PMPSO is effective in jumping out of local optimum, computationally efficient, and easy to implement. We use this algorithm to solve the RCPSP problems, and the results proved the efficacy of the method.

## 2. Probability Mechanism Based Particle Swarm Optimization (PMPSO) Algorithm

In the PSO algorithm, a particle swarm is composed of a specific number of particles. At each iteration, all the particles move in the N-dimensional problem space with a certain velocity to find the global optima. The position of particle $i$ ($i = 1 \ldots N$) is composed of D vectors denoted $x_{id}$, where $x_{id}^k$ denotes vector at iteration $k$. The velocity of particle $i$ is denoted as of a $D$-dimensional vectors $v_{id}$, where $v_{id}^k$ denotes the velocity vector at iteration $k$. The velocity and position of each particle is adjusted according the following formulas:

$$v_{id}^{k+1} = wv_{id}^k + c_1 r_1 \left(P_{id}^k - x_{id}^k\right) + c_2 r_2 \left(P_{gd}^k - x_{id}^k\right) \quad (1)$$

$$x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1} \quad (2)$$

where $P_{id}^k$ is the local best position that the i-th particle had reached at iteration k and $P_{gd}^k$ is the global best position that all the particles had reached at iteration k. $r_1$ and $r_2$ are random numbers between 0 and 1. w is the inertia weight. $c_1$ and $c_2$ are cognitive confidence coefficients, which are all constant numbers.

*2.1. The Representation of Particles.* When solving continuous optimization problems, the viable solution is usually directly viewed as a particle $x_{id}$ which can be updated according to (1) and (2). However, in discrete problems, the update operation of a viable solution is mostly meaningless unless we can represent a viable solution in a feasible way. Therefore, when the PSO algorithm is applied to the discrete domain, the representation of the particle becomes a key issue. In other words, the key point is to establish a link between particles and the solution of the problem. Most discrete problems can be regarded as sequencing problems. For example, the scheduling problem to a project comprised of five activities $\{1, 2, 3, 4, 5\}$ can be considered as a sequencing problem to the five elements. Usually, the solution is in a form of permutation, e.g., (2, 3, 1, 4, 5). A permutation is formed by some fundamental elements, for example, the permutation of (1, 2, 3, 4, 5) is formed by the fundamental elements of (1, 2), (2, 3), (3, 4), and (4, 5). We call each fundamental element as subpermutation. A superior permutation must be consisted of some reasonably good subpermutations. To find these good subpermutations according to (1), we describe the permutation by a two-dimensional array called adjacency matrix, in which the elements correspond to the subpermutations are set to 1, the other elements are set to 0. Here are two examples of adjacency matrixes representing permutations: (1, 2, 3, 4, 5) can be represented by

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (3)$$

and (2, 3, 1, 4, 5) can be described as

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \quad (4)$$

In the adjacency matrix, when the element $a_{i,j}$=1, it states that the number "i" is in front of the number "j" and they are adjacent in the permutation. In our algorithm, the position of a particle is a (0, 1) adjacency matrix, which corresponds to a permutation combination.

*2.2. Notation and Definitions of Operators.* The operators in (1) and (2) are redefined because the position and velocity are represented in the form of a matrix in discrete problems.

$$V_{id}^{k+1} = wV_{id}^k \oplus c_1 \otimes R_1 \dot{\times} \left(P_{id}^k \ominus X_{id}^k\right) \oplus c_2 \otimes R_2 \dot{\times} \left(P_{gd}^k \ominus X_{id}^k\right) \quad (5)$$

$$PX_{id}^k = f_p\left(X_{id}^k\right) \quad (6)$$

$$PX_{id}^{k+1} = PX_{id}^k \oplus V_{id}^{k+1} \quad (7)$$

$$X_{id}^{k+1} = f_s\left(PX_{id}^{k+1}\right) \quad (8)$$

where $X_{id}^k$ denotes the position of a particle at iteration k and $V_{id}^k$ denotes the transition probability velocity vector at iteration k. $P_{id}^k$ is the local best position that the i-th particle

had reached at iteration k, and $P_{gd}^k$ is the global best position that all the particles had reached at iteration k. $X_{id}^k$, $P_{id}^k$ and $P_{gd}^k$ are all (0,1) adjacency matrixes corresponding to their permutations. $R_1$ and $R_2$ are matrixes with the same dimensions as $X_{id}$, whose elements are random numbers between 0 and 1. w is called inertia weight and its value is between 0 and 1. $c_1$ and $c_2$ are cognitive confidence coefficients, which are constant numbers. The symbol "$\ominus$" denotes subtraction between matrixes and the negative elements in the subtraction result are set to 0. The symbol "$\oplus$" denotes addition between matrixes and all elements greater than 1 in the addition result are set to 1. The symbol "$\dot{\times}$" denotes element-wise multiplication between matrixes. Suppose A and B are two matrixes with the same dimension, then the result of A$\dot{\times}$B is the same dimension matrix, where the elements are the products of corresponding elements in A and B. The symbol "$\otimes$" is used to denote the modified multiplication. Let c be a real number, then c$\otimes$A means all the elements of the matrix A are multiplied by c, and the elements with a value greater than 1 are set to 1.

The reason why one permutation combination is distinguished from another is its subpermutation. Therefore, a good permutation combination must contain subpermutations that are good and not available in other permutations. Based on the above ideas, we have designed a probability coefficient selection method. The basic idea of the method is that new permutations are generated based on probability selection, and those good subpermutations should have high probabilities of being selected. We try to find these reasonably good subpermutations through the PSO algorithm and make them have higher probabilities of being selected when generating new permutations. According to (5), the transition probability velocity vector $V_{id}^{k+1}$ is combined by three parts: $wV_{id}^k$, $c_1 \otimes R_1 \dot{\times} (P_{id}^k \ominus X_{id}^k)$, and $c_2 \otimes R_2 \dot{\times} (P_{gd}^k \ominus X_{id}^k)$. $wV_{id}^k$ is used to govern the extent to which the old velocity $V_{id}^k$ determines the new velocity $V_{id}^{k+1}$. Conventionally, w is assigned a value of 0.9. Moreover, c1 and c2 are the learning factors used to control the effects of the local experience and global experience on the new velocity, respectively. Conventionally, they are set to 1. $P_{id}^k \ominus X_{id}^k$ is used to find the unique subpermutations in the local best position $P_{id}^k$ but not in the individual $X_{id}^k$. $P_{gd}^k \ominus X_{id}^k$ is used to find the unique subpermutations in the global best position $P_{gd}^k$ but not in the individual $X_{id}^k$.

In the following, we give an example to illustrate the operators in (2), (5), (6), (7), and (8). Let $X_{id}^k$ be the matrix of

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{9}$$

representing the permutation (1, 2, 3, 4, 5), and $P_{id}^k$ be matrix of

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{10}$$

representing the permutation (2, 3, 1, 4, 5). Then $P_{id}^k \ominus X_{id}^k$ is

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{11}$$

and this result denotes that the subpermutations (3, 1) and (1, 4) are unique for $P_{id}^k$ comparing with $x_{id}^k$. If the random matrix $R_1$ is

$$\begin{pmatrix} 0.2 & 0.5 & 0.8 & 0.7 & 0.2 \\ 0.4 & 0.6 & 0.9 & 0.3 & 0.7 \\ 0.3 & 0.1 & 0.8 & 0.9 & 0.2 \\ 0.4 & 0.5 & 0.9 & 0.4 & \mathbf{0.6} \\ 0.9 & 0.2 & 0.1 & 0.8 & 0.7 \end{pmatrix}, \tag{12}$$

then the 0-1 matrix is transformed to

$$\begin{pmatrix} 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{13}$$

by $c_1 \otimes R_1 \dot{\times} (P_{id}^k \ominus X_{id}^k)$ and in $V_{id}^{k+1}$. The elements in the matrix corresponding to the subpermutations (3,1) and (1,4) are highlighted, which increases the probabilities that they are selected in the new permutation.

New particles are obtained by selection based on probability coefficients. According the main idea of PSO, we need to update the selection probability coefficients considering the current position and velocity. Therefore, we must convert $X_{id}^k$ to the same form as probability coefficients matrix. In (6), $PX_{id}^k$ is defined as probability matrix corresponding to $X_{id}^k$, which stems from $X_{id}^k$ according to $f_p$ function. The function of $f_p$ can be freely constructed according to the following rules: ensure the elements in $PX_{id}^k$ corresponding to the elements of 1 in $X_{id}^k$ have big probabilities and the value of

the elements in $PX_{id}^{k}$ that cannot be used to make a selection is 0. Here we give an example. Let $X_{id}^{k}$ be

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{14}$$

Firstly we can generate a random matrix and we suppose that it is

$$\begin{pmatrix} 0.4897 & \mathbf{0.4984} & 0.8234 & 0.1456 & 0.8564 \\ 0.7567 & 0.2345 & \mathbf{0.2551} & 0.5987 & 0.3334 \\ 0.7234 & 0.3213 & 0.9876 & \mathbf{0.1386} & 0.8876 \\ 0.2223 & 0.6456 & 0.7986 & 0.1234 & \mathbf{0.2435} \\ 0.1987 & 0.2234 & 0.9876 & 0.3235 & 0.9856 \end{pmatrix}. \tag{15}$$

Secondly, we divide all the elements by 1000, except for the elements corresponding to the elements of one in $X_{id}^{k}$. Finally, we set the diagonal elements to be 0, because the diagonal elements in $X_{id}^{k}$ cannot equal one and the corresponding elements in $PX_{id}^{k}$ cannot be used for the selection operation. Then, $PX_{id}^{k}$ is

$$\begin{pmatrix} 0 & \mathbf{0.4984} & 0.0008 & 0.0001 & 0.0008 \\ 0.0007 & 0 & \mathbf{0.2551} & 0.0005 & 0.0003 \\ 0.0007 & 0.0003 & 0 & \mathbf{0.1386} & 0.0008 \\ 0.0002 & 0.0006 & 0.0007 & 0 & \mathbf{0.2435} \\ 0.0001 & 0.0002 & 0.0009 & 0.0003 & 0 \end{pmatrix}. \tag{16}$$

In (7), the selection probability coefficients matrix is updated considering the current position and the velocity. Suppose $V_{id}^{k+1}$ is

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.7 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.8 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}, \tag{17}$$

then

$$PX_{id}^{k+1} = PX_{id}^{k} \oplus V_{id}^{k+1} = \begin{pmatrix} 0 & 0.4984 & 0.0008 & \mathbf{0.7010} & 0.0008 \\ 0.0007 & 0 & 0.2551 & 0.0005 & 0.0003 \\ \mathbf{0.8007} & 0.0003 & 0 & 0.1386 & 0.0008 \\ 0.0002 & 0.0006 & 0.0007 & 0 & 0.2435 \\ 0.0001 & 0.0002 & 0.0009 & 0.0003 & 0 \end{pmatrix}. \tag{18}$$

In (8), $f_s(PX_{id}^{k+1})$ was defined as probability selecting operation, which selects the right elements and set them to one in $X_{id}^{k+1}$ according to the given probability coefficients in $PX_{id}^{k+1}$. The selection operation must obey the rule that if an element in $PX_{id}^{k+1}$ has a larger probability coefficient value, the element with the same index in $X_{id}^{k+1}$ has a greater probability of being selected and set to 1. Then we get the new adjacency matrix and the new permutation is obtained by decoding it.

*2.3. Selection Mechanism Based on Probability Coefficients.* A solution formed by means of probability coefficients selection can also break the local optimum and without losing the opportunity to find better solutions. We illustrate the selection process as follows.

*Process 1* (process of probability coefficients selection).

*Step 1.* Set all the elements of adjacency matrix $X_{id}^{k+1}$ to zero.

*Step 2.* Select a row according to the row number of the row of the maximum element in the probability matrix of $PX_{id}^{k+1}$.

*Step 3.* The corresponding element in $PX_{id}^{k+1}$ is used as the probability selection coefficient, and one element in the row is randomly selected accordingly. The selection rule is that elements with larger probability coefficients have a greater probability of being selected.

*Step 4.* Let the selected element be equal to 1, and set the corresponding so-called contradictory elements in $PX_{id}^{k+1}$ to zero. There are three types of contradictory elements: (1) elements with the same row number as the selected element; (2) elements with the same column number as the selected element; (3) elements which can lead to incompletely permutation.

*Step 5.* If the stopping criterion is not satisfied, go to Step 2.

In the above example in Section 2.2, $PX_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0.4984 & 0.0008 & 0.7010 & 0.0008 \\ 0.0007 & 0 & 0.2551 & 0.0005 & 0.0003 \\ 0.8007 & 0.0003 & 0 & 0.1386 & 0.0008 \\ 0.0002 & 0.0006 & 0.0007 & 0 & 0.2435 \\ 0.0001 & 0.0002 & 0.0009 & 0.0003 & 0 \end{pmatrix}. \tag{19}$$

We then use the same example to explain the implementation of Process 1. Firstly, $X_{id}^{k+1}$ is set to

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{20}$$

According to Process 1, we select the row of (0.8007,0.0003, 0.0000,0.1386,0.0008) as coefficients, because the biggest

value of the matrix is in this row. We randomly select a column according to the coefficients. According to the selection rule, each element could be selected except the third element, which coefficients equal to zero. However, the elements which have bigger probability coefficients have bigger probabilities to be selected. Hence, we assume that the first element was selected. Then in this example, $X_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

and $PX_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0.4984 & 0 & 0.7010 & 0.0008 \\ 0 & 0 & 0.2551 & 0.0005 & 0.0003 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0006 & 0.0007 & 0 & 0.2435 \\ 0 & 0.0002 & 0.0009 & 0.0003 & 0 \end{pmatrix}. \tag{22}$$

The element $PX_{id}^{k+1}(1,3)$ is set to 0 because the third situation in Step 4. Next, in this example, the first row $(0, 0.4984, 0, 0.7010, 0.0008)$ is selected and it is used as coefficients to randomly select a row. Assume that the forth column is selected. Then $X_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{23}$$

and $PX_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0 & 0 & \mathbf{0} & 0 \\ 0 & 0 & 0.2551 & 0 & 0.0003 \\ \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & 0.0006 & 0 & 0 & 0.2435 \\ 0 & 0.0002 & 0.0009 & 0 & 0 \end{pmatrix}. \tag{24}$$

$PX_{id}^{k+1}(4,3)$ is set to 0, because if the corresponding element in $X_{id}^{k+1}$ is selected to 1, the circle permutation (3, 1, 4, 3) would exist, which lead to a incompletely permutation. Next, the second row $(0, 0, 0.2551, 0, 0.0003)$ is selected and it is used as weights to randomly select a column. Assume that the third column is selected. Then $X_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{25}$$

and $PX_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0 & 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2435 \\ 0 & 0.0002 & 0 & 0 & 0 \end{pmatrix}. \tag{26}$$

Next, the fourth row $(0, 0, 0, 0, 0.2435)$ is selected and it is used as coefficients to randomly select a column. The fifth column is selected. Then $X_{id}^{k+1}$ is

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{27}$$

and the whole 0-1 adjacency matrix is generated and the new permutation is (2, 3, 1, 4, 5).

*2.4. Description of the Algorithm.* PMPSO algorithm employs the probability coefficients selection mechanism to improve the exploration of the solution space and it assures the convergence through the PSO evolutionary mechanism. We summarize the procedures of the proposed algorithm as follows.

*Algorithm 2* (procedures of PMPSO algorithm).

*Step 1.* Initialization

*Step 1.1.* Set the number of the particles and the max number of iterations.

*Step 1.2.* Each particle gets a random position matrix $X_i$ corresponding to a permutation.

*Step 1.3.* Decode $X_i$ and calculate the fitness of the position.

*Step 1.4.* Initialize the local best $P_i = X_i^0$ and the global best $P_g$ is the best of all the $P_i$.

*Step 2.* Calculate the position and velocity of each particle.

*Step 2.1.* Calculate new velocities according to (5).

*Step 2.2*. Calculate $PX_{id}^{k}$ according to (6).

*Step 2.3*. Update the $PX_{id}^{k+1}$ according to (7).

*Step 2.4*. Calculate $X_{id}^{k+1}$ according to (8).

*Step 3*. Calculate the fitness of each position. If the fitness of the new position is better than that of the local best position of the particle, update the best position with the new position. If the fitness of the local best position of some particles is better than that of the global position, then update the global best with the local best position.

*Step 4*. If the stopping criterion is not satisfied, go to Step 2.

*Step 5*. Output the best solution and the fitness.

## 3. Problem Statement

Resource-constrained project scheduling problems (RCPSP) were investigated in many situations such as building construction, facility and equipment maintenance, and new product development. Due to their academic appeal and wide application, many researchers have always shown interest in these issues. The problem of RCPSP can be stated as follows. A project consists of a set $N = \{0, 1, \ldots, J+1\}$ of activities, where activities 0 and J+1 are dummy activities that represent the events of the start and finish time of the project respectively. The RCPSP is based on the following assumptions: (1) the durations of activities composing the project are deterministic and known; (2) all activities must obey their precedence constraints; (3) resources can be available in limited amounts and renewable from period to period;(4) activities cannot be interrupted when in progress; (5) the goal of optimization is to minimize project duration. The duration of activity $j$ is denoted $d_j$, and resource type $k$ requested by activity $j$ is denoted $\mathrm{r_{jk}}$. For the dummy activities, for any resource type $k$, we have $\mathrm{d_0} = \mathrm{d_{J+1}} = 0$, and $r_{0,k} = r_{J+1,k} = 0$. The availability of each unit of resource type $k$ is $R_k$ $(k = 1 \ldots K)$. A project schedule can be represented by the start time of each activity denoted $(\mathrm{s_0}, \mathrm{s_1}, \ldots, \mathrm{s_{j+1}})$ with $\mathrm{s_0} = 0$ and the finish time of each activity denoted by $(\mathrm{f_0}, \mathrm{f_1}, \ldots, \mathrm{f_{J+1}})$ with $\mathrm{f_0} = 0$. Note that $\mathrm{f_{J+1}} = \mathrm{s_{J+1}}$ and $\mathrm{f_j} = \mathrm{s_j} + \mathrm{d_j}$ for all $j$. $I_t$ is the set of tasks being executed in the time period of $t$. A schedule is feasible if it satisfies precedence relationships and all these constraints during each execution time period. The mathematical model of the problem can be formulated as follows:

$$\min \quad f_{J+1} \tag{28}$$

$$\text{s.t.} \quad f_j - f_i \geq d_j, \quad j = 1, 2, \cdots, J+1; \ \forall i \in P_j \tag{29}$$

$$\sum_{j \in I_t} r_{jk} \leq R_k, \quad k = 1, 2, \cdots, K; \ t = 0, 1, 2 \ldots \tag{30}$$

Equation (28) is the objective function that minimizes the completion time of the last activity of the project, that is, minimizes the span of the project. Equation (29) means each task's start time must be subject to the finish time of its preceding activities. It also can be expressed as activity $j$ cannot start until all of the predecessor activities have finished, where $P_j$ stands for the set of preceding activities of activity $j$. Equation (30) ensure that the total number of used resources of type $k$ cannot exceed its available maximum, denoted as $R_k$.

It was proved that the optimization of the RCPSP is NP-hard in the strong sense [17]. Mingozzi et al. [18] developed an exact method for solving the problems with sizes up to 30 activities. Donrndorf et al. [19] also reported satisfactory results using a branch-and- bound algorithm for the problems with 30 activities. For large-scale RCPSP instances, most researchers are employing metaheuristics algorithm to solve them, including tabu algorithms [20], genetic algorithms [21, 22], ant colony optimization [23], simulated annealing [24], and variable neighborhood search [25]. However, the heuristic methods based on heuristic rules are often variable effectiveness on different cases and may be trapped within local optima [21–23]. Since the advent of the PSO algorithm, there have been many attempts to apply PSO to solve RCPSP. In contrast to other heuristic methods, PSO has the advantages of computational efficiency, great capability of escaping local optima, and easy implementation [26, 27]. Zhang et al. [28] took the priorities of the scheduling activities as particles to develop a DPSO algorithm for solving the RCPSP and the size of the problem investigated was 25. Xie et al. [29] combined the chaos algorithm with PSO to solve the typical multiple resources-constrained project scheduling problem. Ni et al. [30] proposed the hybrid particle swarm optimization based on differential evolution for solving the RCPSP with fifteen activities. In the following, we use the proposed PMPSO algorithm to solve the RCPSP problems.

## 4. RCPSP Solution Based on PMPSO Algorithm

To solve a RCPSP problem using PMPSO, we must have the information of the project, such as the duration of each activity, each resource required for each activity, the availability of each resource, and precedence relationships of the project activities. Then we can use PMPSO algorithm to solve the problem. The flow chart of the solution is given as Figure 1.

*4.1. Encoding and Decoding of the Particles.* We directly take the set of activity number sequence as the source of particles, which has the advantage of simplicity and effectiveness. Then the permutation is transformed to a 0-1 adjacency matrix, which is the Encoding of particle. The 0-1 adjacency matrix is considered to be the position of the particle. According to the 0-1 adjacency matrix, we can derive the permutation, which called decoding of the particle.

*4.2. Generation of Feasible Sequences.* A feasible sequence $\mathrm{S} = \{s_1, s_2, \ldots, s_n\}$ is a sequence in which the predecessor activity of any activity in the sequence must be arranged to the left of the activity. A feasible sequence is the basis of solving the RCPSP, regardless the algorithm. Researchers used various methods to get the feasible sequence. Montoya-Torres et al.

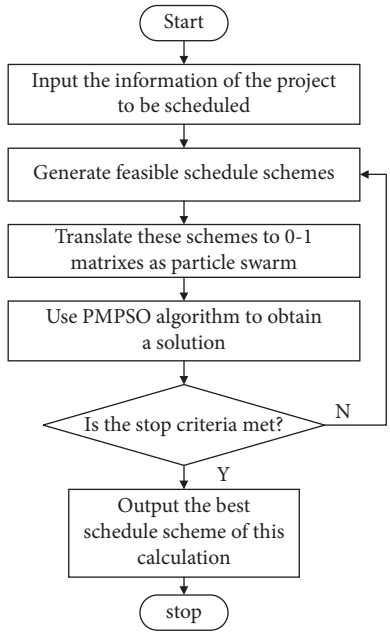


Figure 1: Flow chart of the solution.

[31] transformed sequences to feasible sequences using SSGS (Serial Schedule Generation Schemes) when they solved the RCPSP problem with genetic algorithm. Peng and Wang [32] adopted the priority list in the genetic representation combing the SGS to generate the feasible sequence. Xie et al. [33] allotted the priorities to the activities and then applied the bubble method to generate the feasible sequence. Zhou et al. [33] eliminated the improper sequence with a logic judgment procedure to get the feasible sequence. In all these articles, the feasible scheduling sequences of the activities were acquired indirectly. In order to improve the validity of the calculation, we have designed a method called Modulation Weighting Mechanism for Generating feasible scheduling sequences (MWMG), which can directly obtain a feasible solution. First, the precedence relationships of the activities must be given as a 0-1 matrix R, in which the element $r_{ij}$=1 means that the activity $i$ is a predecessor of activity $j$. Secondly, an activity must be scheduled after all his predecessor activities. If the activity $i$ as the predecessor of the activity $j$ is scheduled, we can deactivate the coupling relationship between the activities $i$ and $j$ by setting the element $r_{ij}$=0. Finally, when we generate the scheduling scheme according the probability weight matrix, the weight matrix should be modulated by the relation 0-1 matrix $R$ to ensure that all the predecessor relations are obeyed. Because the activity "0" is dummy activity which represents the event of the project start, we begin to schedule from the activity "0". The generation process of the scheduling sequence is illustrated as follows.

*Process 3* (process of generating feasible sequence).

*Step 1.* A 0-1 matrix representing the scheduling scheme is initialized by setting all elements to zero. Set the first column of the weight matrix to zero to let the project begin from activity "0". Set the first row of the relationship matrix to zero to release subsequent activities as activity "0" so that these activities can be scheduled.

*Step 2.* The value of the element in the modulation vector depends on the value of the above row vector. When the element value of the row vector is zero, the corresponding element value in the modulation vector is 1; otherwise, the element value is 0, where the element in the modulation vector equals to 1 means that the activity denoted by the column number can be scheduled, because all of its predecessor activities have been scheduled.

*Step 3.* A row is selected from the weight matrix as a weight vector and is modulated by multiplying each element in the weight vector by its corresponding element in the modulation vector. At the beginning, the selected row is the first row.

*Step 4.* Select the column number according the modulated weight vector randomly. Then, the corresponding element value in the 0-1 matrix of the scheduling scheme is set to 1 according to the row number and the column number.

*Step 5.* Renew the weight matrix according the rules in Section 2.3.

*Step 6.* Update the row number by the column number.

*Step 7.* Set the row of the relationship matrix to zero to release the subsequent activities of the activity represented as the column number so that they can be scheduled.

*Step 8.* If the stopping criterion is not satisfied, go to Step 3.

*Step 9.* Decode the 0-1 matrix of the scheduling scheme to get the feasible sequence.

Through the above generation mechanism, not only the feasible scheme can be directly obtained, but also the excellent subpermutations have better chance to enhance their probabilities.

*4.3. Generation of a Scheduling Scheme.* We adopt the method of Serial Scheduling Generation Scheme (SSGS) to obtain a feasible scheduling scheme. According to the sequence, we schedule the activities successively. The following gives the generation process of the scheduling scheme.

*Process 4* (process of generating a scheduling scheme).

*Step 1.* Select an activity $i$ from the feasible sequence in order, which duration is $d_i$, and calculate the early start time of the activity.

*Step 2.* Schedule the start time of the activity as early as possible, denoted $t_{is}$, and calculate its finish time denoted $t_{if}$, where $t_{if} = t_{if} + d_i$.

*Step 3.* Calculate the availability of the resources required for the activity $i$ during the time period in which the activity is performed.

*Step 4*. If the availability of the resources is satisfied, go to Step 5; otherwise, let $t_{is} = t_{is} + 1$ and go to Step 2.

*Step 5*. If there are still unscheduled activities, go to Step 1; otherwise go to Step 6.

*Step 6*. Output the result.

*4.4. Total Process of Solving a RCPSP Problem Using PMPSO Algorithm.* According to Processes 1–4 we summarize the general process of solving a RCPSP problem using PMPSO algorithm as follows.

*Process 5* (process of solving the RCPSP problem).

*Step 1*. Input the information of the project to be scheduled and initialize the parameters of the PMPSO algorithm.

*Step 2*. Initialize the probability coefficients matrixes according to the rules mentioned in Process 1.

*Step 3*. Generate 0-1 matrixes of the scheduling scheme using the method in Process 3.

*Step 4*. Decode each 0-1 matrix to get a scheduling scheme and calculate the fitness function value according to Process 4.

*Step 5*. Update the best position of the swarm and the best positions of each particle experienced.

*Step 6*. Update the probability coefficients matrixes according to (8).

*Step 7*. If the stopping criterion of the PMPSO algorithm is not satisfied, go to Step 3; otherwise go to Step 8.

*Step 8*. Output the result.

## 5. Computational Experiments

We conduct the experiments for PMPSO in this section. The test set comes from J30, J60, J90 and J120 of the PSPLIB, provided on the website (http://www.om-db.wi.tum.de/psplib/) [34]. J30, J60 and J90 all contain 480 instances, while J120 contains 600 instances. Each instance set has four resource types. The currently best known solution in PSPLIB as reported on June 17, 2015. For the J30 set, these solutions are all optimal. The test environment is as follows: CPU: INTEL–2.4G, RAM: 4GB DDR 333, HD: 250G 7200 rpm, OS: Win 10.

We set $w$ to 0.98, both $c_1$ and $c_2$ to 1, and the maximal number of iteration to 5000. Then we randomly select 10 instances from each test set and run the scheduler 100 times on each problem instance. Table 1 shows the statistical results of these solutions.

From Table 1, we observed that PMPSO can get the best known solution for all of the instances randomly selected from J30, and the maximum deviation of the mean of all solutions from the optimal value is only 0.17%. For other test sets, PMPSO can get most of the best known solutions, and the max deviation from the best known solution is 1.69%. As we can see from Table 1, the best known solutions of these instances were found by various researchers using various methods. In this paper, we use PMPSO to get some of the best known solutions and some solutions deviating from the best known solutions slightly. We also execute the scheduling program for each instance of each problem set. Table 2 shows the results of relative deviations from the currently best known solution.

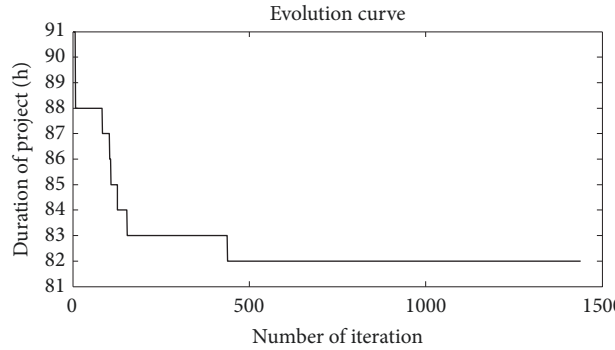


Figure 2: Evolution curve of the instance of J305-2.

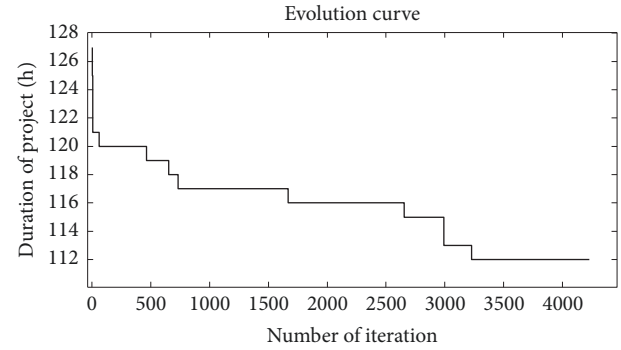


Figure 3: Evolution curve of the instance of J1201-5.

Tables 1 and 2 summarize the experimental results of PMPSO. We observed that the larger the problem, the greater the deviation from the best value or the current best known solution, and as the size of the problem increases, the average number of best solutions obtained decreases. For most instances, we can get the best known solutions. Moreover, as it is shown in the Table 2, using the algorithm we can get a very high percentage of hits for the problem size within 30. We note that the best known solutions in PSPLIB are found by different people using kinds of algorithms. Therefore, PMPSO is an algorithm that adapts to problems of various scales and has unique advantages especially for small-scale problems.

To observe the performance of PMPSO, we also illustrate the evolution curves of the instances of J305-2 and J1201-5, which are showed in Figures 2 and 3, respectively.

Figure 2 shows that evolution curve drops sharply and then converges at the best value. This shows that the PMPSO algorithm has the advantage of fast convergence, which is

Table 1: Solutions of the selected instances.

| Instance | | Results in PSPLIB | Results by PMPSO | | | |
|---|---|---|---|---|---|---|
| | | Best known Solution | Best solution | Worst solution | Average | Deviation (%) |
| J30 | J301-1 | 43 | 43 | / | 43 | 0 |
| | J302-4 | 43 | 43 | / | 43 | 0 |
| | J303-9 | 65 | 65 | / | 65 | 0 |
| | J305-10 | 70 | 70 | 72 | 70.04 | 0.06 |
| | J308-1 | 44 | 44 | / | 44 | 0 |
| | J3010-4 | 58 | 58 | / | 58 | 0 |
| | J3015-2 | 47 | 47 | 48 | 47 | 0 |
| | J3026-6 | 53 | 53 | / | 53 | 0 |
| | J3030-1 | 47 | 47 | 49 | 47.08 | 0.17 |
| | J3040-8 | 57 | 57 | / | 57 | 0 |
| J60 | J602-5 | 54 | 54 | 56 | 54.15 | 0.27 |
| | J603-9 | 67 | 67 | 68 | 67.05 | 0.07 |
| | J605-1 | 76 | 76 | 77 | 76.51 | 0.67 |
| | J608-8 | 66 | 66 | 69 | 66.44 | 0.66 |
| | J6020-4 | 86 | 86 | 88 | 86.13 | 0.15 |
| | J6025-2 | 98 | 98 | 100 | 98.02 | 0.02 |
| | J6026-9 | 65 | 65 | 67 | 65.07 | 0.1 |
| | J6030-2 | 70 | 70 | 71 | 70.05 | 0.07 |
| | J6040-3 | 70 | 70 | 71 | 70.42 | 0.6 |
| | J6048-8 | 88 | 88 | 89 | 88.01 | 0.01 |
| J90 | J903-5 | 75 | 75 | 76 | 75.51 | 0.68 |
| | J904-4 | 92 | 92 | 94 | 93.03 | 1.12 |
| | J908-9 | 97 | 97 | 99 | 98.04 | 1.07 |
| | J9010-1 | 77 | 77 | 78 | 77.11 | 0.14 |
| | J9018-8 | 92 | 92 | 93 | 92.34 | 0.37 |
| | J9021-4 | 106 | 106 | 107 | 106.5 | 0.47 |
| | J9022-2 | 85 | 85 | 86 | 85.23 | 0.27 |
| | J9036-9 | 102 | 103 | 105 | 103.12 | 1.09 |
| | J9038-1 | 85 | 85 | 87 | 85.82 | 0.91 |
| | J9041-2 | 168 | 168 | 170 | 169.12 | 0.67 |
| J120 | J1203-5 | 84 | 85 | 87 | 85.21 | 1.44 |
| | J1204-6 | 90 | 91 | 92 | 91.5 | 1.67 |
| | J1208-9 | 94 | 94 | 95 | 95.55 | 1.65 |
| | J12010-1 | 111 | 112 | 114 | 112.88 | 1.69 |
| | J12020-8 | 107 | 107 | 109 | 108.56 | 1.46 |
| | J12021-4 | 135 | 135 | 137 | 136.24 | 0.92 |
| | J12032-2 | 131 | 131 | 131 | 132.23 | 0.94 |
| | J12046-9 | 166 | 167 | 168 | 167.88 | 1.13 |
| | J12048-2 | 113 | 113 | 114 | 114.55 | 1.37 |
| | J12058-5 | 120 | 120 | 122 | 121.98 | 1.65 |

Table 2: Experimental results by PMPSO on each instance.

| | Average deviation (%) | Best deviation (%) | Worst deviation (%) | no. best/no. instance | Percentage of hits (%) |
|---|---|---|---|---|---|
| J30 | 0.01 | 0 | 0.20 | 465/480∗ | 96.88 |
| J60 | 0.22 | 0 | 0.67 | 385/480∗ | 80.21 |
| J90 | 0.31 | 0 | 1.1 | 370/480 | 77.08 |
| J120 | 0.98 | 0 | 2.5 | 251/600 | 41.83 |

∗hit means finding the best known solution

the most important advantage of the original PSO algorithm. It also demonstrates the outstanding performance of solving small-scale problems.

For large-scale problems, PMPSO is doing well too, which can be observed from Figure 3. For the instance of J1201-5, which has 120 activities, PMPSO can get its best known solution at 3200 iterations.

In Figure 3, before the curve levels off to the minimal value, it experiences several platforms. This shows that the PMPSO algorithm can effectively avoid converging on local optimum and reduce solution time greatly.

## 6. Conclusions

This paper proposes a probability mechanism based particle swarm optimization algorithm. The design of the algorithm includes the following features: a special decoding scheme, a mechanism for finding good subpermutations, a new particle swarm iteration formula, and a selection mechanism based on probability coefficients. When applying the PMPSO algorithm to discrete optimization problems, we only need to convert the solution to a 0-1 matrix form, and then use our mutated PSO formula. The experimental results showed that state-of-the-art solutions can be obtained using the PMPSO algorithm to solve some benchmark problems. In our future work, we will establish a better probability selection mechanism to improve the selection process. We will implement dynamic adjustment of parameters to improve the performance of PMPSO. In addition, we can combine PMPSO with other algorithms, such as local search, metaheuristic, to get better results. In the experiments of this paper, PMPSO is capable of solving the RCPSP efficiently. Although these experiments are made based on PSPLIB, PMPSO should perform equally well for the other types of sequencing problems, such as production scheduling, route planning, and the assignment problem.

## Data Availability

The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

## Conflicts of Interest

The authors declare that there are no conflicts of interest in the article.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 71201026; Grant no. 71801045). This work was partially supported by Research Start-Up Funds of DGUT (GC300502-46) and the Youth Innovative Talent Project (2017KQNCX191) from the Department of Education of Guangdong Province, China.

## References

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, Perth, Australia, 1995.

[2] J. F. Chang, S. C. Chu, J. F. Roddick, and J. S. Pan, "A parallel particle swarm optimization algorithm with communication strategies," *Journal of Information Science and Engineering*, vol. 4, no. 21, pp. 809–818, 2005.

[3] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, pp. 84–88, IEEE, La Jolla, Calif, USA, July 2000.

[4] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004.

[5] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 4104–4109, IEEE, Orlando, Fla, USA, October 1997.

[6] M. Clerc, "Discrete particle swarm optimization, illustrated by the traveling salesman problem," in *New Optimization Techniques in Engineering*, B. V. Babu and G. C. Onwubolu, Eds., vol. 141 of *Studies in Fuzziness and Soft Computing*, pp. 219–239, Springer, Berlin, Germany, 2004.

[7] T. Hendtlass, "Preserving diversity in particle swarm optimization," in *Lecture Notes in Computer Science*, vol. 2718, pp. 4104–4108, Springer, 2003.

[8] W. Pang, K. Wang, C. Zhou, and L. Dong, "Fuzzy discrete particle swarm optimization for solving traveling salesman problem," in *Proceedings of the Fourth International Conference on Computer and Information Technology*, pp. 796–800, Wuhan, China, September 2004.

[9] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proceedings of the International Conference on Machine Learning and Cybernetics*, pp. 1583–1585, IEEE, November 2003.

[10] W. Zhong, J. Zhang, and W. Che, "A novel discrete particle swarm optimization to solve traveling salesman problem," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pp. 3283–3287, Singapore, September 2007.

[11] Z. Yuan, L. Yang, Y. Wu, L. Liao, and G. Li, "Chaotic particle swarm optimization algorithm for traveling salesman problem," in *Proceedings of the IEEE International Conference on Automation and Logistics*, pp. 1121–1124, Jinan, China, 2007.

[12] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. X. Wang, "Particle swarm optimization based algorithms for TSP and generalized TSP," *Information Processing Letters*, vol. 103, pp. 169–176, 2007.

[13] T. R. Machado and H. S. Lopes, "A hybrid particle swarm optimization model for the traveling salesman problem," in *Natural Computing Algorithms*, H. Ribeiro, R. F. Albrecht, and A. Dobnikar, Eds., pp. 255–258, Springer, 2005.

[14] L. Fang, P. Chen, and S. Liu, "Particle swarm optimization with simulated annealing for TSP," in *Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, C. A. Long, V. M. Mladenov, and Z. Bojkovic, Eds., pp. 206–210, Corfu Island, Greece, February 2007.

[15] D. Anghinolfi and M. Paolucci, "A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup

times," *European Journal of Operational Research*, vol. 193, no. 1, pp. 73–85, 2009.

[16] Q. Jia and Y. Seo, "An improved particle swarm optimization for the resource-constrained project scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 9, pp. 2627–2638, 2013.

[17] J. Blazewicz, J. K. Lenstra, and A. H. Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983.

[18] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco, "An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation," *Management Science*, vol. 44, no. 5, pp. 714–729, 1998.

[19] U. Dorndorf, E. Pesch, and T. Phan-Huy, "A branch-and- bound algorithm for the resource constrained project scheduling problem," *Mathematical Methods of Operations Research*, vol. 52, no. 3, pp. 413–439, 2000.

[20] T. Baar, P. Brucker, and S. Knust, "Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman, and C. Roucarion, Eds., pp. 1–18, Kluwer, Boston, Mass, USA, 1998.

[21] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics (NRL)*, vol. 45, no. 7, pp. 733–750, 1998.

[22] S. Hartmann, "A self-adapting genetic algorithm for project scheduling under resource constraints," *Naval Research Logistics (NRL)*, vol. 49, no. 5, pp. 433–448, 2002.

[23] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 333–346, 2002.

[24] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268–281, 2003.

[25] K. Fleszar and K. S. Hindi, "Solving the resource-constrained project scheduling problem by a variable neighbourhood search," *European Journal of Operational Research*, vol. 155, no. 2, pp. 402–413, 2004.

[26] R. C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *Evolutionary Programming VII*, vol. 1447 of *Lecture Notes in Computer Science*, pp. 611–616, Springer, Berlin, Germany, 1998.

[27] J. Robinson, S. Sinton, and Y. R. Samii, "Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna," in *Proceedings of the IEEE International Symposium in Antennas and Propagation Society International symposium and URSI*, vol. 1, pp. 168–175, IEEE, San Antonio, Tex, USA, 2002.

[28] H. Zhang, H. Li, and C. M. Tam, "Particle swarm optimization for resource-constrained project scheduling," *International Journal of Project Management*, vol. 24, no. 1, pp. 83–92, 2006.

[29] X. Yang, Y. Chun-ming, C. Jun-lan, and Z. Rong, "Resource-constrained project scheduling based on chaos particle swam optimization," *Industrial Engineering Journal*, vol. 15, no. 3, pp. 57–61, 2012.

[30] L. Ni, C Duan, and C.-l. Jia, "Hybrid particle swarm optimization algorithm based on differential evolution for project scheduling problems," *Application Research of Computers*, vol. 28, no. 4, 2011.

[31] J. R. Montoya-Torres, E. Gutierrez-Franco, and C. Pirachicán-Mayorga, "Project scheduling with limited resources using a genetic algorithm," *International Journal of Project Management*, vol. 28, no. 6, pp. 619–628, 2010.

[32] P. Wuliang and W. Chengen, "A multi-mode resource-constrained discrete time-cost tradeoff problem and its genetic algorithm based solution," *International Journal of Project Management*, vol. 27, no. 6, pp. 600–609, 2009.

[33] X. Yang, Y. Chun-ming, C. Jun-lan, and Z. Rong, "Resource-constrained project scheduling based on chaos particle swam optimization," *Industrial Engineering Journal*, vol. 15, no. 3, 2012.

[34] D. Debels, B. de Reyck, R. Leus, and M. Vanhoucke, "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling," *European Journal of Operational Research*, vol. 169, no. 2, pp. 638–653, 2006.