Int J Adv Manuf Technol (2013) 68:617–630
DOI 10.1007/s00170-013-4782-7

ORIGINAL ARTICLE

A hybrid assembly sequence planning approach based on discrete particle swarm optimization and evolutionary direction operation

Mingyu Li · Bo Wu · Youmin Hu · Chao Jin · Tielin Shi

Received: 2 July 2012 / Accepted: 21 January 2013 / Published online: 5 February 2013

Abstract Assembly sequence planning (ASP) has always been an important part of the product development process, and ASP problem can usually be understood as to determine the sequence of assembly. A good assembly sequence can reduce the time and cost of the manufacturing process. In view of the local convergence problem with basic discrete particle swarm optimization (DPSO) in ASP, this paper presents a hybrid algorithm to solve ASP problem. First, a chosen strategy of global optimal particle in DPSO is introduced, and then an improved discrete particle swarm optimization (IDPSO) is proposed for solving ASP problems. Through an example study, the results show that the IDPSO algorithm can obtain the global optimum efficiently, but it converges slowly compared with the basic DPSO. Subsequently, a modified evolutionary direction operator (MEDO) is used to accelerate the convergence rate of IDPSO. The results of the case study show that the new hybrid algorithm MEDO-IDPSO is more efficient for solving ASP problems, with excellent global convergence properties and fast convergence rate.

Keywords Assembly sequence planning · Discrete particle swarm optimization · Modified evolutionary direction operator · Hybrid algorithm

M. Li · Y. Hu (✉) · C. Jin · T. Shi
School of Mechanical Science & Engineering,
Huazhong University of Science and Technology,
Wuhan, Hubei, China 430074
e-mail: youmhwh@mail.hust.edu.cn

B. Wu
The State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology,
Wuhan, Hubei, China 430074

1 Introduction

Assembly sequence planning (ASP) is a widely researched problem, which is an important part of assembly process planning. Usually, most of the assembly sequence is determined on the basis of experience when the product design is completed. However, optimal assembly sequence planning is difficult to ensure based solely on experience. On the other hand, the performance indexes of assembly resources are hard to consider during the design process. With the number of assembly parts growing, the number of possible assembly sequences increases exponentially. Therefore, it is hard to formulate the assembly sequence with no mistakes. Along with the development of concurrent engineering, computer-integrated manufacturing systems, the ASP problem, which is directly related to the assembly quality, efficiency, and reliability in the assembly line, is one of the core issues [1, 2]. In order to solve the ASP problem, the goal will focus on saving cost and time during the assembly process [3].

To solve and optimize the ASP problem is one of the most basic problems in the assembly process, and assembly sequence affects the underlying assembly process [4]. ASP problem is crucial because it will determine many assembly aspects including number of assembly orientation changes, number of assembly tool changes, and number of assembly operation type changes. The optimization results of the above three problems can reduce the cost of the assembly process and also reduce the time required for the assembly process.

Bourjault [5] proposed an assembly precedence relation and the assembly association figure model to solve the assembly sequence planning for the first time. De Fazio and Whitney [6] optimized the method of Bourjault by improving the form of questions and reducing the number of questions to ask the operators. Hsu et al. [7] developed a knowledge-based engineering system to assist engineers predicting a near-optimal assembly sequence promptly,

and the proposed system can facilitate feasible assembly sequences in a virtual environment.

Bonneville et al. [8] first proposed the ASP approach using a genetic algorithm (GA). In this method, each sequence was treated as a chromosome in the initial population, and two simple crossover and mutation operators were used to generate offspring individuals. In order to solve ASP problems by using GA more efficiency, De et al. [9] and Marian et al. [10] improved the coding principle and the fitness function to solve ASP problems. Lu et al. [11] proposed an assembly planning approach using a multi-objective genetic algorithm and established different fitness functions through a fuzzy weight distribution algorithm. Tseng et al. [12] presented a multi-plant assembly sequence planning model with a new GA encoding scheme to solve the multi-plant assembly sequence planning problem. Zhou et al. [13] combined bacterial chemotaxis (BC) and GA, and proposed a novel BC–GA-based hybrid algorithm to decrease the probability of trapping into local optimal solutions.

Milner et al. [14] used the network model to express the assembly sequence and searched the lowest-cost assembly sequence with simulated annealing (SA) algorithm. In order to overcome the combination explosion problems of assembly sequence optimization, Chen [15] proposed an artificial neural networks algorithm method to transform the assembly constraint geometrical relationship to the assembly precedence relations and optimized the ASP problem. Wang et al. [16] proposed an ant colony optimization (ACO) algorithm to solve disassembly sequence planning problems from the view of geometric feasibility. Cheng et al. [17] investigated a genetic algorithm and ants algorithm based on GA and ACO to optimize ASP problem and assembly path feedback for complex products. Gao et al. [18] proposed a memetic algorithm (MA), which combined the parallel global search nature of evolutionary algorithm with local search to improve individual solutions. The result showed that MA achieved significant improvement for solving ASP problem.

In addition to the above algorithms, particle swarm optimization (PSO) algorithms are a new methodology proposed in recent years. PSO was first proposed by Kennedy and Eberhart [19]. Basically, the PSO algorithm has powerful local and global search abilities and can convergence efficiently. Moreover, the PSO algorithm is simple and easy to implement, and does not need many parameters adjustment. In view of the above advantages, the PSO algorithm has been widely used in assembly sequence planning and related fields [20–22]. Because the ASP solution is in discrete solution space, the traditional PSO algorithms cannot be used to solve the ASP problems directly. Kennedy and Eberhart [23] proposed a binary particle swarm optimization (BPSO) algorithm to solve the discrete combinatorial optimization problem for the first time. Hu and Eberhart [24] improved the BPSO algorithm to solve the permutation replacement problems, and the improved BPSO imports the mutation operation to prevent the optimal particles from being trapped in a local minimum. Clerc [25] redefined the position, velocity, and formulas of particles and proposed a new discrete particle swarm algorithm (DPSO) to solve the traveling salesman problem.

In recent years, more solutions based on DPSO are put forward to solve the assembly sequence planning problem. Wang and Liu [26] proposed a chaotic particle swarm optimization algorithm. The algorithm can provide a better assembly sequence by using chaos theory, but this method will increase the solution time because of the selection of the initial assembly sequence and the process. Lv and Lu [27] proposed the assembly sequence planning approach with a DPSO algorithm. The algorithm can achieve good results. But the DPSO algorithm is easily affected by the individual optimal fitness value, and the algorithm is easy to convergence in the early evolution.

Evolutionary direction operator (EDO) is a kind of optimization operator, which can produce improved individuals guided by a series of optimal evolutionary directions belonging to the current population. EDO is widely used in GA [28, 29] and many other research fields [30, 31], which can find the global optimum efficiently. All these methods are optimization problems solved in continuous solution space. Based on the excellent properties of EDO, this paper presents a new hybrid algorithm (MEDO-IDPSO) to improve the defects of DPSO. This hybrid algorithm combines the modified evolutionary direction operator (MEDO) and improved discrete particle swarm optimization (IDPSO) to solve the ASP problem in discrete solution space for the first time. This hybrid algorithm can improve the convergence rate and population diversity and also balance the global search ability and local search ability of particle swarm. Generally, the hybrid algorithm will improve the global search solution and is proved to have better performance than basic DPSO.

The rest of paper is organized as follows. “Section 2” introduces the ASP methodology. “Section 3” discusses the ASP approach with basic DPSO. In “Section 4,” a chosen strategy of global optimal particle in DPSO is introduced. Then, an IDPSO to solve the ASP problem is proposed. To overcome the slow rate of convergence problem in IDPSO, a MEDO is introduced. A new hybrid algorithm based on IDPSO and MEDO is presented in “Section 5.” Finally, this paper is concluded in “Section 6,” and the prospect of future research is given.

2 ASP methodology

2.1 Interference matrix

An interference matrix is used to determine the geometric feasibility of the product parts in each assembly direction.

An effective assembly sequence should satisfy the geometric constraint relation of assembly firstly, and the interference information can be easily obtained from 3D CAD software.

The interference matrix (IM) was first proposed by Dini and Santoch [32] in assembly sequence planning and can be derived from the geometric assembly relationship. Generally, an interference matrix can be established according to $\pm X$, $\pm Y$, and $\pm Z$ six direction spaces. Supposing an assembly consists of n parts, the interference matrix can be represented as follows:

$$\mathbf{IM} = \begin{array}{c} \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{array} \begin{array}{c} \begin{array}{cccc} p_1 & p_2 & \cdots & p_n \end{array} \\ \begin{bmatrix} I_{11x}I_{11y}I_{11z} & I_{12x}I_{12y}I_{12z} & \cdots & I_{1nx}I_{1ny}I_{1nz} \\ I_{21x}I_{21y}I_{21z} & I_{22x}I_{22y}I_{22z} & \cdots & I_{2nx}I_{2ny}I_{2nz} \\ \vdots & \vdots & \cdots & \vdots \\ I_{n1x}I_{n1y}I_{n1z} & I_{n2x}I_{n2y}I_{n2z} & \cdots & I_{nnx}I_{nny}I_{nnz} \end{bmatrix} \end{array}$$

Where, $I_{ijk}=1$ ($i\in[1\ n]$, $j\in[1\ n]$), if part P_j collides with part P_i, when P_j is moved along the direction k to the assembly position; Otherwise, $I_{ijk}=0$. Especially, the part cannot collide with itself, $I_{iik}=0$. As I_{jik} in the $-k$ direction is equal to I_{ijk} in the $+k$ direction, the three interference matrixes $\boldsymbol{M}_{+x}$, $\boldsymbol{M}_{+y}$, and $\boldsymbol{M}_{+z}$ can be used to conclude the precedence feasibility of the given assembly sequence.

The aim of ASP optimization is to reduce the assembly costs and time by applying design for assembly (DFA) approach in product design. Usually, the objective of ASP optimization can be confirmed by the assembly orientation changes, tool changes, and operation type changes. Therefore, according to the characteristics of the product assembly, the fitness function of ASP in this paper is defined by the number of assembly orientation changes, the number of assembly tool changes, and the number of assembly operation type changes.

2.2 The number of assembly orientation changes

For a feasible assembly sequence consisting of n parts, the feasible assembly directions can be obtained for part P_i to the subassembly which has been assembled before part P_i. The number of assembly orientation changes can be determined by the addition, and assembly orientation changes depends on the previous assembly operation.

Assume $\{P_1, P_2, \ldots, P_m, \ldots P_n\}$ is the assembly sequence, $D(P_i)$ ($i\in[1\ n]$) is the feasible assembly direction of each part of the assembly sequence, which is determined by the geometric feasibility method. The feasible assembly directions $D(P_i)$ can be obtained for each part P_i according to the subassembly which has been assembled before part P_i. If parts (P_1, P_2, …, P_i) exist $\cap_{i=1}^{m} D(p_i) \neq \phi$, there is no need to change the direction of the assembly, and if $\cap_{i=1}^{m} D(p_i) \neq \phi$ and $\cap_{i=1}^{m+1} D(p_i) = \phi$, the assembly of part P_{m+1} needs to change the assembly direction once.

The fewer number of assembly orientation changes, the fewer redirection operations, which can avoid complex assembly tools and fixture design and increase quality of assembly, realizing shorter time for assembly. Using the interference matrix, it is also possible to determine the number of assembly orientation changes f_1, and it can be calculated as follows:

$$f_1 = \sum_{i=1}^{n} f_i \tag{1}$$

Where $f_i=1$ ($i\in[1\ n]$), if part i changes the assembly direction relative to the part $i-1$, otherwise, $f_i=0$.

2.3 The number of assembly tool changes

In assembly process, a given assembly sequence should comply with the following basic principles: In the same assembly direction, assemble as many parts using the same assembly tools and assembly operation types as possible.

A change of the assembly tools usually increases time of assembly sequence. In complex product assembly processes, the more specialized assembly tools are used the more assembly process may be complicated. Therefore, the fewer number of assembly tool changes, the more favorable for the assembly process.

The number of assembly tool changes can be carried out with the following equation:

$$f_2 = \sum_{i=1}^{n} t_i \tag{2}$$

Where $t_i=1$ ($i\in[1\ n]$), if part i changes the assembly tool relative to the part $i-1$, otherwise, $t_i=0$.

2.4 The number of assembly operation type changes

The number of assembly operation type changes will increase the assembly time and assembly cost. Changes of the assembly operations can also require tool changes, and the number of assembly operation type changes can be determined as follows:

$$f_3 = \sum_{i=1}^{n} g_i \tag{3}$$

Where $g_i=1$ ($i\in[1\ n]$), if part i changes the assembly operation type relative to the part $i-1$, otherwise, $g_i=0$.

2.5 Formulation of the fitness function

For a given assembly product, if the assembly sequence is feasible, then the fitness function of the ASP can be given as follows based on reference [11]:

$$F = 2S - w_1 f_1 - w_2 f_2 - w_3 f_3 \tag{4}$$

Where S is represents the total number of components in an assembly sequence. f_1 is the number of assembly orientation changes; f_2 is the number of assembly tool changes, and f_3 is the number of assembly operation type changes. w_1, w_2, and w_3 are the weights of f_1, f_2, and f_3 respectively, and $w_1+w_2+w_3=1$ ($w_i \in [0\ 1]$, $i=1,2,3$). In this paper, the fitness function has the constant weight distribution, and the weights are set as $w_1=0.4$, $w_2=0.3$, $w_3=0.3$.

Considering the assembly sequence priority constraints in the assembly process, when the assembly sequence is infeasible, the fitness function is defined as follows:

$$F = \frac{2S - w_1 f_1 - w_2 f_2 - w_3 f_3}{2m} \tag{5}$$

Where m is number of the interference times in ASP. In this condition, if the number of interference times of a given assembly sequence is more than once, then the assembly sequence is infeasible. m is used to reduce the value of the sequence fitness function, in order to punish the infeasible assembly sequence. Therefore, a lower probability of the infeasible assembly sequence is given in the evolutionary process, and the infeasible sequence is set to be in a disadvantage. However, it can also cause the reduction of the diversity of the population in algorithm.

Through the above definition, it can be seen that the higher value of the fitness function is, the less time and cost of ASP is used.

2.6 Case study introduction

An assembly product example from Lu [11] is used to study and analyze the presented assembly sequence planning algorithm. The assembly product example is shown in Fig. 1.

As shown in Fig. 1, the assembly product consists of 22 parts, and each part has six possible assembly directions along with the $\pm X, \pm Y, \pm Z$ axes. In this study, the interference matrix $\boldsymbol{M}_{+x}$, $\boldsymbol{M}_{+y}$, and $\boldsymbol{M}_{+z}$ can be seen below, and the number of tool types and operation types of the assembly are given in Table 1.

$$\mathbf{M}_{+x} = \begin{bmatrix}
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
1&0&0&1&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
1&1&0&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&1&1&0&0&1&1&0&0&0&0&0&0&1&1&1&1&1&1\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

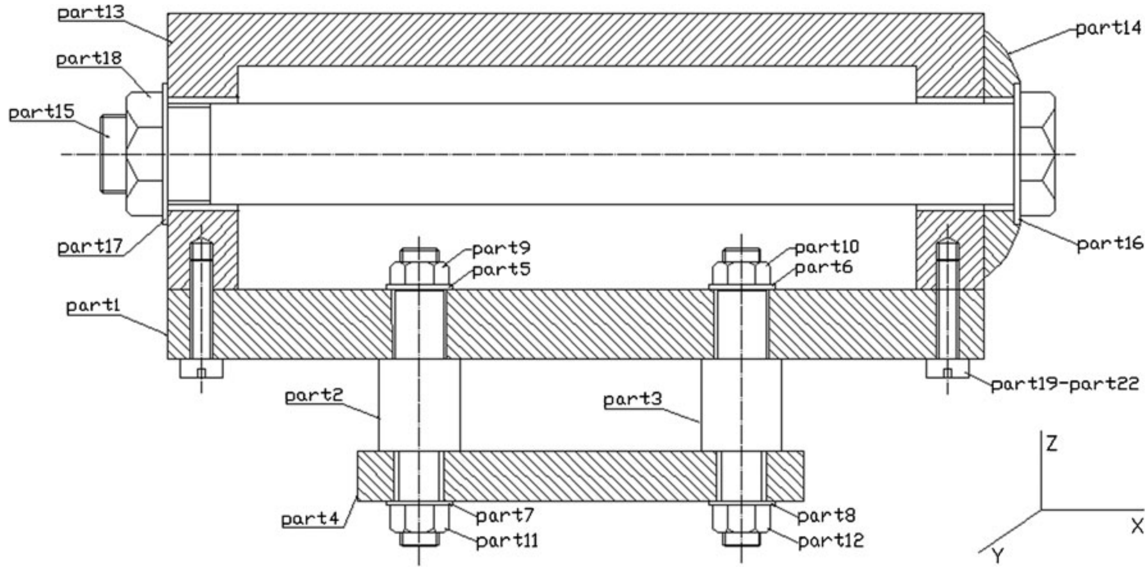

Fig. 1 An assembly product example

$$
\mathbf{M}_{+y} = \begin{bmatrix}
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
1&0&0&1&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0
\end{bmatrix}
$$

Table 1 Tool types and operation types in the assembly

Part no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Tool type	1	2	2	1	3	3	3	3	4	4	4	4	1	5	2	3	3	4	6	6	6	6
Operation type	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	2	2	2	2

$$\mathbf{M}_{+z} = \begin{bmatrix}
0&1&1&1&0&0&1&1&0&0&1&1&0&0&0&0&0&0&1&1&1&1\\
0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&1&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&1&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

3 An ASP approach with DPSO

3.1 Particle swarm optimization

In a traditional PSO algorithm method for solving continuous space optimization problems, the solution space of position and velocity belongs to the continuous real number domain. The basic concepts of PSO are a randomly initialized swarm of m particles in the n-dimensional search-space. The movements of the particles are guided by their own best known position in the search-space, as well as the entire swarm best known positions. Based on those two best known positions, the particles change their own position in the search space. In the n-dimensional search-space, the position of particle i is denoted by $X_i=(x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{in})$, and the velocity is denoted by $V_i=(v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots, v_{in})$. The personal historical best position is denoted by $P_i=(p_{i1}, p_{i2}, \ldots, p_{ij}, \ldots, p_{in})$, and the global historical best position is denoted by $P_g=(p_{g1}, p_{g2}, \ldots, p_{gj}, \ldots, p_{gn})$, in which $1\leq j\leq n$. The position and velocity of the particle can be illustrated as follows:

$$V_{ij}(t+1) = w \cdot V_{ij}(t) + c_1 r_1[P_{ij} - X_{ij}(t)] + c_2 r_2[P_{gj} - X_{ij}(t)] \tag{6}$$

$$X_{ij}(t+1) = X_{ij}(t) + V_{ij}(t+1) \tag{7}$$

Where w is the inertia weight, which determines the proportion of the inheritance of the particles on the current speed. c_1 and c_2 are learning factors, which make the particle have the ability to learn from current personal and global excellence by searching space. r_1 and r_2 are random numbers between [0 1] in the n-dimensional search-space.

3.2 Discrete particle swarm optimization

ASP problems are discrete optimization problems; the position and velocity of the ASP solution space belongs to the discrete domain. To solve ASP problems, the position and velocity of the PSO algorithm should be improved properly. The position, velocity, variable quantity, and other operation rules of the particles are redefined below, and DPSO algorithm for solving ASP problems is presented.

3.2.1 Position of particle i

The position of particle i is equivalent to an ordered arrangement of an assembly sequence in production, and the position vector of each particle i is defined as follows:

$$X_i = \left(x_{i1}, x_{i2}, \cdots, x_{ij}, \cdots x_{in}\right) \tag{8}$$

Where $x_{ij}\in\{1,2, \ldots,n\}$ are the part numbers, n is the total number of the parts of the assembly. In order to ensure the local searching behaviors and the population diversity, the initial position vector of each particle is generated randomly.

3.2.2 Velocity of particle i

The velocity of particle i is equivalent to a transformation of an assembly sequence. The velocity of the particles is

defined as the change of the particles position, which is the change of an assembly sequence. The velocity vector of each particle i is defined as follows:

$$V_i = \left(v_{i1}, v_{i2}, \cdots, v_{ij}, \cdots v_{in}\right) \tag{9}$$

And $v_{ij} \in \{0,1,2,\ldots,n\}$.

3.2.3 Addition operator (position and velocity)

The addition operator of position and velocity of particle i is equivalent to an update of an assembly sequence. Supposed that a particle position is $X_i=(x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{in})$, and a particle velocity is $V_i=(v_{i1}, v_{i2}, \ldots, v_{ij}, \ldots, v_{in})$, then the updating rule of the addition operator is as follows:

$$X_i(t+1) = X_i(t) + V_i(t+1)$$

$$x_{ij}(t+1) = \begin{cases} x_{ij}(t) & if \quad v_{ij}(t+1) = 0 \\ v_{ij}(t+1) & if \quad v_{ij}(t+1) \neq 0 \end{cases} \tag{10}$$

Where X_i $(t+1)$ is the new position of the assembly sequence in the $t+1$ times iteration, and V_i $(t+1)$ is the velocity in the $t+1$ times iteration. If velocity vector element v_{ij} $(t+1)$ is zero, the position vector x_{ij} $(t+1)$ is still $x_{ij}(t)$. Otherwise, the new particles position vector x_{ij} $(t+1)$ is changed into v_{ij} $(t+1)$, and the element of original position vector which equals v_{ij} $(t+1)$ is changed into x_{ij} (t).

3.2.4 Subtraction operator (position and position)

A new velocity vector can be obtained by the subtraction operation between two position vectors. Supposed that $X_1=(x_{11}, x_{12}, \ldots, x_{1j}, \ldots, x_{1n})$ and $X_2=(x_{21}, x_{22}, \ldots, x_{2j}, \ldots, x_{2n})$ are two different position vectors, then the subtraction operator is defined as follows:

$$V = X_2 - X_1 \tag{11}$$

Where $V=(v_1, v_2, \ldots v_n)$. If $x_{1j}=x_{2j}$, $v_j=0$, otherwise, $v_j=x_{2j}$. Especially, the addition operator and subtraction operator are irreversible.

3.2.5 Multiplication operator of velocity vectors

Multiplication operator of velocity vectors is used to make adjustment of the velocity. Suppose that $V_1=(v_{11}, v_{12}, \ldots, v_{1j}, \ldots, v_{1n})$ is velocity vector of particles, c is a control parameter, $c \in [0\ 1]$, then the multiplication operator is defined as follows:

$$V_2 = c \cdot V_1 \tag{12}$$

Where $V_2=(v_{21}, v_{22}, \ldots, v_{2j}, \ldots, v_{2n})$, and the multiplication operation of velocity and control parameters can be ensured as follows:

$$v_{2j} = \begin{cases} v_{1j}, & r \geq c \\ 0, & \text{else} \end{cases} \tag{13}$$

Where r $(r \in [0\ 1])$ is a randomly uniform distribution number. It can be seen that, the greater control parameter c is, the more elements V_2 inherits from the vector V_1.

3.2.6 Addition operator of velocity vectors

The addition operator of velocities is used to obtain the effect of the superposed velocities. The sum of two velocities is a new velocity. Supposed that the particles velocity vectors are $V_1=(v_{11}, v_{12}, \ldots, v_{1j}, \ldots, v_{1n})$ and $V_2=(v_{21}, v_{22}, \ldots, v_{2j}, \ldots, v_{2n})$, then

$$V_3 = V_1 + V_2 \tag{14}$$

Where $V_3=(v_{31}, v_{32}, \ldots, v_{3j}, \ldots, v_{3n})$, and the addition operation of velocity can be ensured as follows:

$$v_{3j} = \begin{cases} v_{1j}, & v_{1j} \neq 0, \quad v_{2j} = 0 \\ v_{1j}, & v_{1j} \neq 0, \quad v_{2j} \neq 0, \quad r < 0.5 \\ v_{2j}, & \text{else} \end{cases} \tag{15}$$

Based on special definitions of the position and velocity, the mutative equations of the position and velocity of particle i is as follows:

$$V_i(t+1) = w \cdot V_i(t) + c_1[P_i - X_i(t)] + c_2[P_g - X_i(t)] \tag{16}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{17}$$

3.2.7 Weight coefficient and learning factors

The weight coefficient and learning factors should be redefined in DPSO to allow the particles to have the ability of self-learning and individual learning in groups. The weight coefficient and learning factors are variables in DPSO; the value of variables are determined according to the following equations:

$$w = w_{\min} + \frac{w_{\max} - w_{\min}}{G_{\max}} \times n \tag{18}$$

$$c_1 = c_{1s} + \frac{c_{1l} - c_{1s}}{G_{\max}} \times n \tag{19}$$

$$c_2 = c_{2s} + \frac{c_{2l} - c_{2s}}{G_{\max}} \times n \tag{20}$$

Table 2 The convergence performance of DPSO

N_{size}	30	60	90	120	150	180	210	240	270	300
n_t	20	20	20	20	20	20	20	20	20	20
n_{nd}	1	2	3	3	5	6	6	7	8	7
t_{nd}	53	47.5	45	39	46	56	47.5	42.14	38.25	37.43
F_{nd}	36.5	37.62	38.13	38.25	38.5	38.45	38.63	38.75	38.73	38.82

Where w_{max} and w_{min} are the initial weight and final weight, and w_{max}=0.9, w_{min}=0.1. G_{max} is the maximum iteration number, while n denotes current iteration number. c_{1l}, c_{1s}, c_{2l}, and c_{2s} are constant parameters defined in advance, c_{1l}=0.8, c_{1s}=0.2, c_{2l}=0.2, and c_{2s}=0.8. The weight coefficient w and learning factors c_1, c_2 greatly impact the performance of the DPSO. The weight strategies described have following advantages: (1) In the beginning of DPSO, a smaller value of w is helpful to escape from the local minimum, while a bigger value of w is beneficial to algorithm convergence at the end; (2) by changing c_1 from 0.8 to 0.2 and changing c_2 from 0.2 to 0.8, the method can reserve more diversity of the swarm at the beginning period of the algorithm and thus have more ability to escape from local minimum.

3.3 The case study of DPSO

Applying the DPSO algorithm to the case study, set the initial parameters as follows: The maximum number of iterations G_{max}=200, and the initial iteration number t=1.

To study the convergence performance of DPSO, the population size N_{size} is respectively set as 30, 60, 90, 120, 150, 180, 210, 240, 270, and 300. The DPSO algorithm program runs 20 times to search out the optimal solution or near-optimal solution for the different swarm size and the average number of iterations. The results are shown in Table 2.

As indicated in Table 2, n_t is the number of algorithm program runs, n_{nd} is the number of the best fitness value obtained from DPSO, t_{nd} is the average number of iterations, and F_{nd} is the average fitness value of the DPSO algorithm.

As seen in Table 2, after the DPSO algorithm runs 20 times, there are only a few global optimal solutions n_{nd} found, which means the premature convergence phenomena is a serious problem with the DPSO algorithm. That is to say, the DPSO algorithm is easily falls into the local optimal solution, as seen in Fig. 2. It can be seen that the solution falls into local optimum of 9.325 with 21 iterations. Although the average number of the global optimal solutions obtained is few in DPSO, the convergence rate of algorithm is very fast, as shown in Fig. 3. It obtains the global optimum of 38.7 with 34 iterations.

In the DPSO algorithm, the position and velocity of each particle will be updated according to the individual best fitness value and global best fitness value. Therefore, DPSO algorithm is subjected to these two values, which is useful for fast search and convergence of the DPSO algorithm ostensibly. However, the DPSO algorithm in the early evolution can quickly converge to the local optimum along with the increase of the iterative numbers, and the evolution process of the DPSO algorithm remains unchanged. In this situation, the obtained global best fitness value is the local optimum, and DPSO algorithm is easy to fall into local convergence.

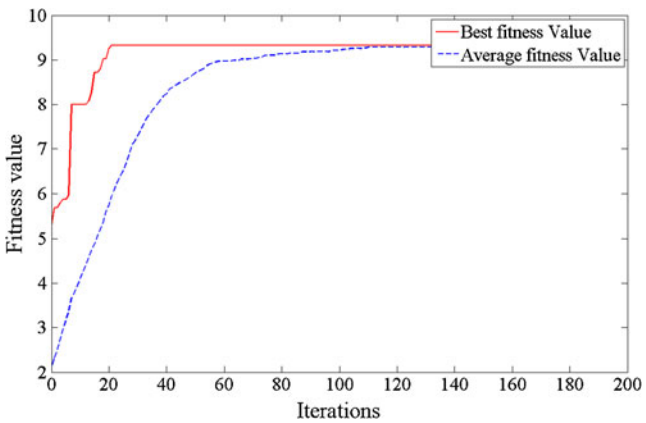

Fig. 2 The premature convergence phenomena in DPSO algorithm

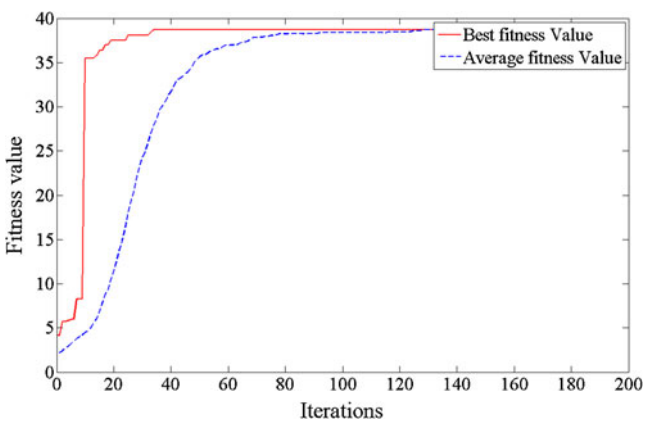

Fig. 3 The fast convergence of DPSO algorithm

4 An ASP approach with IDPSO

4.1 Chosen strategy of global best particle

From above section, the basic DPSO algorithm needs to be improved, to solve the local convergence problem. In this paper, the particles are trained by the historical best fitness value of other particles with the larger probability in the early evolutionary stage and by the global best fitness value with the larger probability in the later evolutionary stage. This method will increase the ability of DPSO algorithm to search the solution space and increase the opportunity for the DPSO to reach the global optimum.

In the IDPSO algorithm, suppose that n is the number of iterations, $G_{\max}$ is the maximum number of iterations, and G_t is a variable as follows:

$$G_t = \frac{n}{G_{\max}} \tag{21}$$

Suppose r is an equally distributed random number between [0 1]. If $r>G_t$, then randomly select a particle from the whole population except the global best particle, and use the optimal location position of the particle P_r instead of global optimal position P_g. The velocity of particle i is renewed as follows:

$$V_i(t+1) = w \cdot V_i(t) + c_1[P_i - X_i(t)] + c_2[P_r - X_i(t)] \tag{22}$$

If $r \le G_t$, the position and velocity of particle i is still renewed using Eqs. 16 and 17.

4.2 Proposed IDPSO algorithm for ASP

The inputs of the proposed IDPSO algorithm are the number of assembly parts, interference matrix $\boldsymbol{M}_{+x}$, $\boldsymbol{M}_{+y}$, and $\boldsymbol{M}_{+z}$, and the weight coefficient w_1=0.4, w_2=0.3, w_3=0.3. The main procedure of the proposed IDPSO algorithm is shown in Fig. 4:

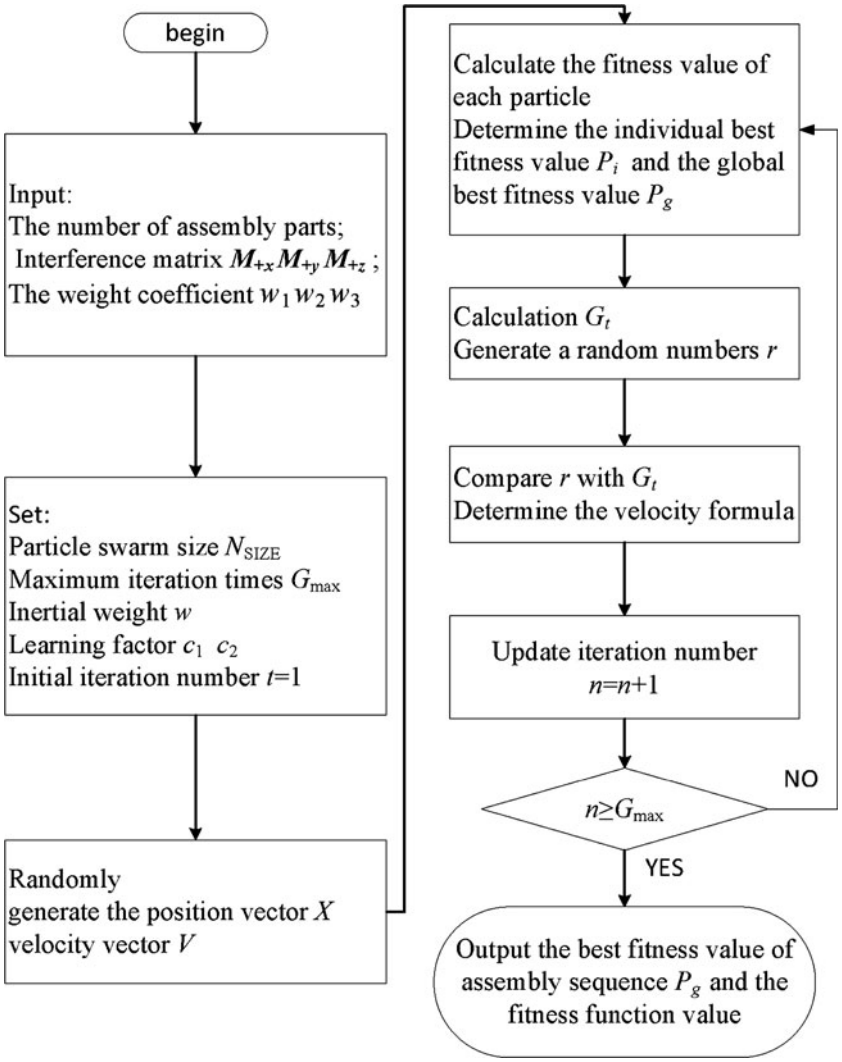

Fig. 4 Flow chart of improved discrete particle swarm optimization

Table 3 The convergence performance of IDPSO

N_{size}	30	60	90	120	150	180	210	240	270	300
n_t	20	20	20	20	20	20	20	20	20	20
n_{ni}	6	9	10	11	13	15	16	18	18	19
t_{ni}	125.83	123.22	129.6	126.82	122.92	119.87	115.5	125.39	116.17	121.16
F_{ni}	38.55	38.37	38.5	38.82	38.57	38.77	38.81	38.83	38.84	38.81

Step 1 Set the population size N_{size}, the maximum number of iterations G_{max}, the inertial weight w, the learning factors c_1, c_2, and the initial iteration number $t=1$.

Step 2 Randomly generate the position vector X and velocity vector V in the initial population.

Step 3 Calculate the fitness value of each particle. If the current fitness value is better than the individual best fitness value P_i in an iterative process, then substitute the individual best fitness value P_i with the current fitness value.

Step 4 According to the fitness value of each particle, determine the individual best fitness value P_i and the global best fitness value P_g.

Step 5 Calculate G_t and generate a uniform distribution random numbers r in the range [0 1].

Step 6 If $r>G_t$, generate the new position vector X and velocity vector V according to Eqs. 22 and 17; if $r\leq G_t$, generate the new position vector X and velocity vector V according to Eqs. 16 and 17.

Step 7 Determine the iteration numbers. If $n\geq G_{max}$, go to step 8. Otherwise, go to step 3.

Step 8 Output the best fitness value of assembly sequence P_g and the fitness function value.

4.3 The case study of IDPSO

Apply the IDPSO algorithm to the previous case study and set the initial parameters as follows: the maximum number of iterations $G_{max}=200$, and the initial iteration number $t=1$.

To study the convergence performance of IDPSO, population size N_{size} is respectively set as 30, 60, 90, 120, 150, 180, 210, 240, 270, and 300. The IDPSO algorithm program runs 20 times to search the optimal solution or near-optimal solution for the different population size and the average number of iterations. The results are shown in Table 3.

As indicated in Table 3, n_t is the number of algorithm program runs, n_{ni} is the number of the best fitness value of IDPSO algorithm obtained, t_{ni} is the average number of iterations, and F_{ni} is the average fitness value of IDPSO algorithm. The convergence performance of IDPSO is depicted in Fig. 5, and the best fitness value is 39.3 with 118 iterations.

Comparing Table 3 with Table 2, it can be seen that the local convergence problem of basic DPSO algorithm has been greatly improved by IDPSO, which shows the IDPSO algorithm has better searching efficiency in solution space. As shown in Table 3, when the population size N_{size} is 300, the average iterations of IDPSO t_{ni} is 121.16, which increase obviously compared with the average iterations of basic DPSO $t_{nd}=37.43$ as shown in Table 2.

As shown in Fig. 6, comparing the convergence performance between DPSO and IDPSO, the iteration number of IDPSO is 120, while the iteration number of DPSO is 33. Through the comparison, IDPSO has a significantly lower convergence rate than DPSO.

In IDPSO algorithm, the global best particle may not be the global optimum in the early iteration process, and the

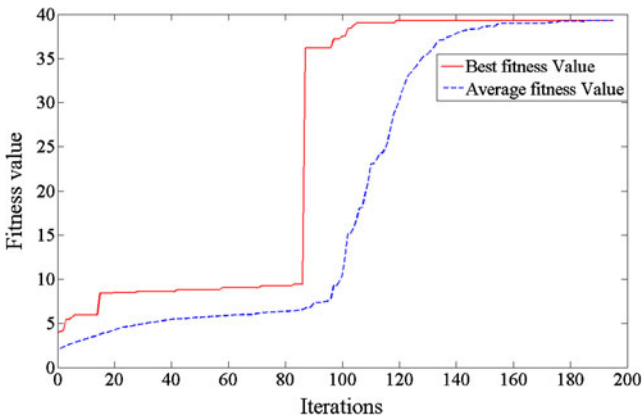

Fig. 5 The convergence performance of IDPSO

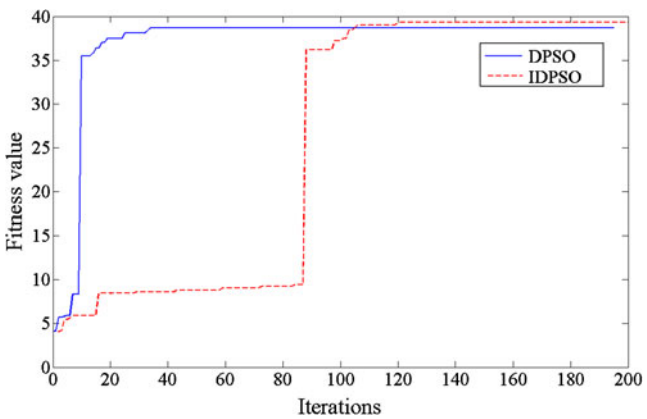

Fig. 6 Compare the convergence performance of DPSO and IDPSO

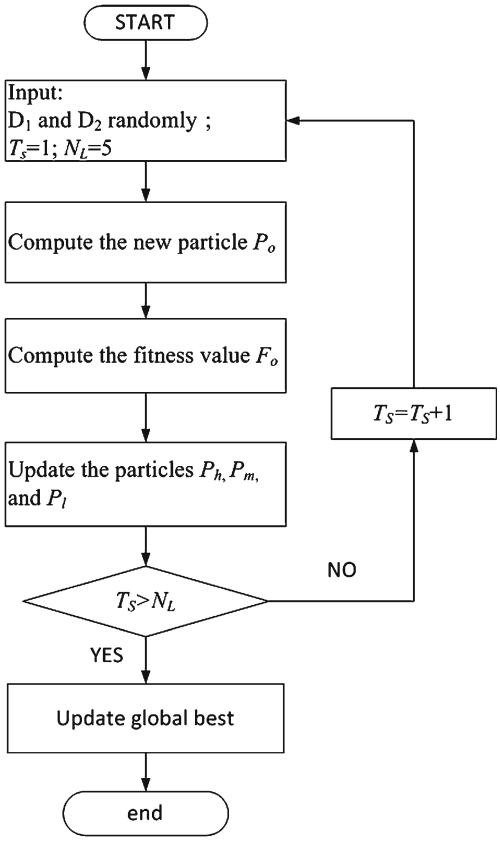

Fig. 7 The flowchart of MEDO

diversity of the whole population has been kept, which is obviously the reason why the average iteration number increases. In order to reduce the iteration number and improve the efficiency of the algorithm, the IDPSO algorithm will be improved in the next section.

5 An ASP approach with hybrid algorithm

The influencing factors on the global convergence of the IDPSO algorithm mainly depend on two aspects: One is the initial distribution of the particle swarm, and the other is the search strategy. The search strategy of the IDPSO algorithm is reflected mainly in the performance of the particle velocity mainly. Firstly, the updating of the personal best particle location reflects its local search ability, which will improve the precision of local position and speed up the local convergence rate. Secondly, the updating of the global best particle location reflects its guidance of the global optimal solution. Both above-mentioned updating strategies will help to improve the convergence rate of the algorithm.

In order to reduce the average iteration number of the IDPSO algorithm, a hybrid algorithm MEDO-IDPSO based on IDPSO and the MEDO is proposed below, which is proven to improve the local and global search capability of IDPSO.

5.1 MEDO

Yamamoto and Inoue [28] used the EDO in GAs for the first time, which can be adopted to determine a better direction for solution searching. The modified EDO presented in this paper does not require gradient evaluation, and thus, it is much faster than the gradient-based optimizers. If numerous local optimum solutions exist in the search space, then EDO cannot easily exploit its searching ability.

To improve the local optimum solution and reduce the average iteration number of the IDPSO algorithm more efficiently, the MEDO is proposed in discrete solution space. In the MEDO process, three best particles in each generation are chosen to perform the evolutionary direction

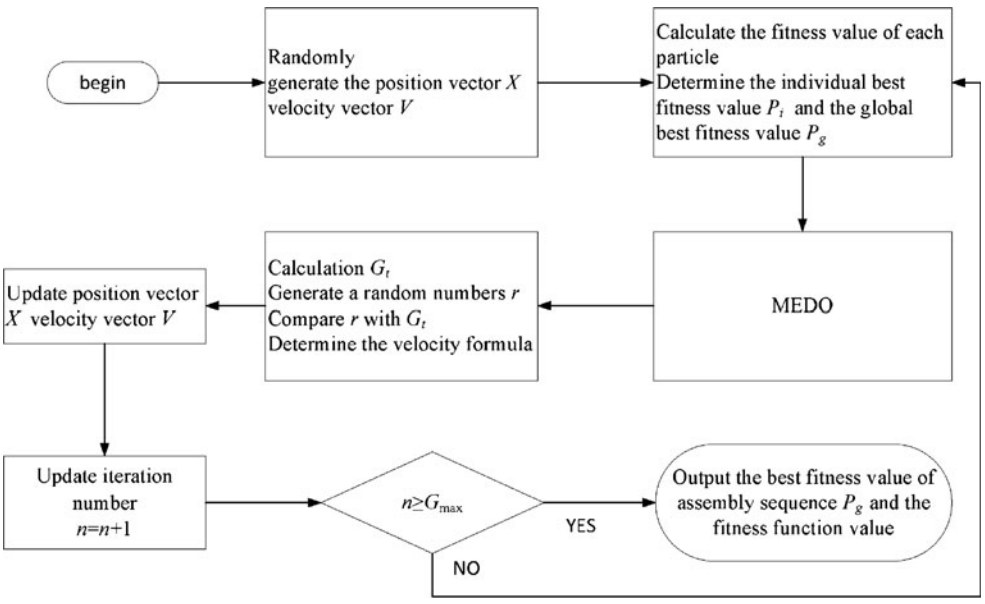

Fig. 8 The flowchart of the hybrid algorithm

Table 4 The convergence performance of hybrid algorithm

N_{size}	30	60	90	120	150	180	210	240	270	300
n_t	20	20	20	20	20	20	20	20	20	20
n_{nh}	5	8	10	13	14	15	15	16	17	17
t_{nh}	75.40	76.63	78.90	78.15	80.14	78.33	76.47	75.31	77.06	72.47
F_{nh}	37.60	38.28	38.51	38.65	38.56	38.68	38.79	38.88	38.85	38.91

operator algorithm, and then a new particle superior to the original best particle will be obtained. The three best particles are obtained after a generation of learning. These three best particles are ordered according to their fitness value, and called the "high", "medium" and "low" particles. Three input and the output particles of the MEDO process are denoted as follows:

Input particles:

"high" particle, $P_h=(p_{h1}, p_{h2},\ldots, p_{hj}, \ldots, p_{hn})$, and the fitness value is F_h.

"medium" particle, $P_m=(p_{m1}, p_{m2},\ldots, p_{mj}, \ldots, p_{mn})$, and the fitness value is F_m.

"low" particle, $P_l=(p_{l1}, p_{l2},\ldots, p_{lj}, \ldots, p_{ln})$, and the fitness value is F_l.

Output particle:

$P_o=(p_{o1}, p_{o2},\ldots, p_{oj}, \ldots, p_{on})$, and the fitness value is F_o.

Where n represents the dimension of particles in IDPSO. The MEDO process proposed in this paper is described as follows:

Step 1 Set the evolution direction parameters $D_1$and D_2 randomly, the MEDO initial index $T_s=1$, and the total number of the MEDO loop $N_L=5$. Find the three particles with the best fitness values from a generation of IDPSO and mark them as P_h, P_m, and P_l.

Step 2 Generate the new particle P_o as follows

$$P_{oj} = P_{lj} + D_1 \times (P_{lj} - P_{mj}) + D_2 \times (P_{lj} - P_{hj}) \tag{23}$$

Step 3 Calculate the fitness value F_o of the new particle.

Step 4 Update the particles P_h, P_m, and P_l, and the updating principle is as follows:

1. If $F_o>F_h$, then $P_h=P_o$, $P_m=P_h$, and $P_l=P_m$;
2. Else if $F_m<F_o<F_h$, the $P_h=P_h$, $P_m=P_o$, and $P_l=P_m$;
3. Else if $F_l<F_o<F_m$, the $P_h=P_h$, $P_m=P_m$, and $P_l=P_o$;
4. Else if $F_l=F_o=F_m$, then $F_o=F_o+N_r$, N_r is a random permutation of the number between [1 N], where N is the number of parts in the assembly.
5. Else if $F_o<F_l$, then initialize D_1 and D_2 randomly, D_1, $D_2\in$[0 1].

Step 5 If $T_s>N_L$, then go to next step. Otherwise, $T_s=T_s+1$, go to step 2.

Step 6 If the fitness value of the new particle is higher than the global best fitness value, then the global best particle will be replaced by the new particle. Figure 7 shows the flowchart of the proposed MEDO.

The flowchart of the hybrid algorithm MEDO-IDPSO is shown in Fig. 8, and the whole learning process is described step-by-step as follows. In the iteration process of hybrid algorithm, the role of the MEDO is to accelerate the velocity of all the iterations and meanwhile reduce the time of algorithm operation. On the other hand, the role of G_t is to prevent the hybrid algorithm from falling into local optimum. These are two key points in the hybrid algorithm.

5.2 The case study of hybrid algorithm

Apply the hybrid algorithm MEDO-IDPSO to the previous case study and set the initial parameters as follows: the maximum number of iterations $G_{max}=200$ and the initial iteration number $t=1$.

To study the convergence performance of the hybrid algorithm, population size N_{size} is set as 30, 60, 90, 120, 150, 180, 210, 240, 270, and 300. The hybrid algorithm program runs 20 times to search out the optimal solution or near-optimal solution for the different population size and the average number of iterations. The results are shown in Table 4.

As shown in Table 4, n_t is the time of algorithm program runs, n_{nh} is the number of the best fitness value obtained from the hybrid algorithm, t_{nh} is the average number of iterations, and F_{nh} is the average fitness value of the hybrid algorithm.

The fitness function value of an optimal assembly sequence obtained is 39.9 with 68 iterations, as shown in Fig. 9, and the corresponding assembly sequence is given in Table 6. The optimal assembly sequence has two orientation changes, eight

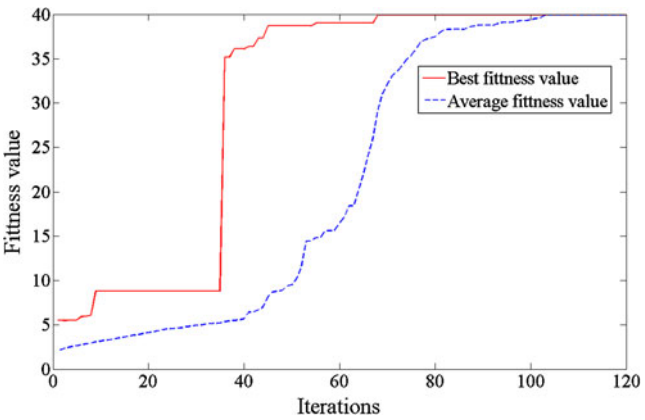

Fig 9 An optimal assembly sequence value from hybrid algorithm

tool changes, and three operation type changes. Comparison about the results of ASP optimization performance between GA, DPSO, IDPSO, DPSO-SA, and MEDO-IDPSO is shown in Table 5 using the same n_{size} and G_{max} (except GA and DPSO-SA). The computation time of each algorithm is obtained using Dell Laptop XPS L412Z with Intel (R) Core (TM) i5-2430 M CPU 2.4 GHz (except GA).

As shown in Table 5, all different optimization algorithm population size is 300 (expect GA). The maximum iteration number of GA, DPSO, IDPSO, and MEDO-IDPSO is G_{max}=200, while GA is G_{max}=500 and DPSO-SA are G_{max}=400. n_{tc} is the number of algorithm program runs, n_{nc} is the number of the best fitness value obtained from different algorithms, t_{nc} is the average number of iterations, F_{nc} is the average fitness value, and F_{op} is the best fitness value. From the GA approach of Lu's [11], the crossover probability p_c=0.8 and mutation probability p_m=0.2.

Comparing different algorithm results in Table 5, the number of the optimal solutions obtained with the hybrid algorithm increases remarkably compared with DPSO. On the other hand, the average iteration number of the hybrid algorithm reduced significantly compared with IDPSO. The proposed MEDO-IDPSO method obtained the best fitness value of 39.9 while keeping the algorithm stability (n_{nc}=17) and searching efficiency (t_{nc}=72.47).

From the above optimization approaches, the best assembly sequences with different algorithms are shown in Table 6. And the best assembly sequence of MEDO-IDPSO has two orientation changes, eight tool changes, and three operation type changes.

The case study proves the correctness, advantage, and effectiveness of the hybrid algorithm to solve ASP problems. The hybrid algorithm not only has the stability but can also overcome the weakness of the slow convergence of IDPSO.

In summary, the hybrid algorithm has stronger convergence properties than basic DPSO and solved the local convergence problem of the basic DPSO algorithm, which is proven to be more suitable to solve the ASP problems. Moreover, the proposed hybrid algorithm is simple and easy to realize.

Table 5 Performance evaluation between different optimization approaches

Algorithm/item	GA [11]	DPSO	IDPSO	DPSO-SA	MEDO-IDPSO
n_{size}	80	300	300	300	300
G_{max}	500	200	200	400	200
n_{tc}	20	20	20	20	20
n_{nc}	18	7	19	17	17
t_{nc}	–	37.43	121.16	367	72.47
F_{nc}	–	38.82	38.81	38.6	38.91
F_{op}	38.82	39	39.3	39.3	39.9

Table 6 The best assembly sequence among different optimization approaches

Algorithm	The best assembly sequence
GA	4-2-3-1-5-6-8-7-16-17-10-9-13-14-15-18-11-12-21-20-19-22
DPSO	14-5-6-9-10-1-13-3-2-4-7-8-17-16-18-11-12-21-20-22-19-15
IDPSO	10-9-5-6-13-1-2-3-4-14-17-7-16-8-18-12-11-20-19-22-21-15
DPSO-SA	10-9-5-6-13-1-2-3-4-14-17-7-16-8-18-12-11-20-19-22-21-15
MEDO-IDPSO	10-9-18-13-1-5-6-17-16-14-15-2-3-4-8-7-12-11-22-20-21-19

6 Conclusions

To solve ASP problem, the interference matrix has been used to determine the directions of the geometric feasibility in this paper. The fitness function reflects the number of assembly orientation changes, the number of assembly tool changes, and the number of assembly operation type changes, whose object is to reduce the time and cost of the manufacturing process. In view of the local convergence problem of basic DPSO in ASP, this paper presented a hybrid algorithm MEDO-IDPSO based on DPSO and MEDO to overcome this problem. Initially, an IDPSO was proposed to improve the stability of basic DPSO with careful choice of global optimal particles, whereafter, a MEDO was brought in to accelerate the convergence rate of IDPSO, and a hybrid algorithm MEDO-IDPSO was presented.

The case study results show that the hybrid algorithm MEDO-IDPSO not only had better global convergence properties than basic DPSO but also had a faster convergence rate than IDPSO in solving the ASP problem. Through the case studies of an assembly product example from Lu [11], the best fitness value obtained in this paper is 39.9 with 68 iterations. This result is much better than Lu's [11] GA (genetic algorithm) method, whose best fitness value is 38.82 with around 400 iterations. And it is also better than Lv's [27] DPSO method, whose best value is 39.4 with around 100 iterations. The results of the case study showed that the hybrid algorithm based on IDPSO and MEDO is efficient for solving ASP problem, which can obtain more satisfactory optimization results than GA and DPSO.

Future works will be mainly focused on the improvement of the evolutionary direction in MEDO, such as proper choice of direction parameters D_1 and D_2 with consideration of DPSO and to make the hybrid algorithm MEDO-IDPSO more efficient and stable to solve the ASP problem.

Acknowledgments This work is sponsored by the National Key Technology R&D program of China (No. 2009BAG12A01-G01-3), the National Natural Science Foundation of China (No. 51175208, No. 51075161), and the State Key Basic Research Program of China (NO.2011CB706803). The CSR Sifang Locomotive & Rolling Stock Co., Ltd, very gratefully acknowledged.

References

1. Demoly F, Yan XT, Eynard B, Rivest L, Gomes S (2011) An assembly oriented design framework for product structure engineering and assembly sequence planning. Robot Comput Integr Manuf 27(1):33–46
2. Lai HY, Huang CT (2004) A systematic approach for automatic assembly sequence plan generation. Int J Adv Manuf Technol 24 (9–10):752–763
3. Rashid MFF, Hutabarat W, Tiwari A (2012) A review on assembly sequence planning and assembly line balancing optimization using soft computing approaches. Int J Adv Manuf Technol 59(1–4):335–349
4. Wang LH, Keshavarzmanesh S, Feng HY, Buchal RO (2009) Assembly process planning and its future in collaborative manufacturing: a review. Int J Adv Manuf Technol 2009(1–2):132–144
5. Bourjault A (1984) Contribution á une approche méthodologique de l'assemblage automatisé: Elaboration automatique des sequences opératoires. Ph.D. thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté, France
6. De Fazio TL, Whitney DE (1987) Simplified generation of all mechanical assembly sequences. IEEE Trans Robot Autom 3 (6):640–658
7. Hsu YY, Tai PH, Wang MW, Chen WC (2011) A knowledge-based engineering system for assembly sequence planning. Int J Adv Manuf Technol 55(5–8):763–782
8. Bonneville F, Perrard C, Henrioud JM (1995) A genetic algorithm to generate and evaluate assembly plans. IEEE Symposium on Emerging Technology and Factory Automation, Paris, France 2:231–239
9. De LP, Latinne P, Rekiek B (2001) Assembly planning with an ordering genetic algorithm. Int J Prod Res 39(16):3623–3640
10. Marian RM, Luong LHS, Abhary K (2006) A genetic algorithm for the optimisation of assembly sequences. Comp Ind Eng 50 (4):503–527
11. Lu C, Wong YS, Fuh JYH (2006) An enhanced assembly planning approach using a multi-objective genetic algorithm. Proc Inst Mech Eng Part B J Eng Manuf 220(2):255–272
12. Tseng YJ, Yu FY, Huang FY (2010) A multi-plant assembly sequence planning model with integrated assembly sequence planning and plant assignment using GA. Int J Adv Manuf Technol 48 (1–4):333–345
13. Milner JM, Graves SC, Whitney DE (1994) Using simulated annealing to select least-cost assembly sequences. Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, USA, pp 2058–2063
14. Zhou W, Zheng JR, Yan JJ, Wang JF (2011) A novel hybrid algorithm for assembly sequence planning combining bacterial chemotaxis with genetic algorithm. Int J Adv Manuf Technol 52 (5–8):715–724
15. Chen W, Tai P, Deng W, Hsieh L (2008) A three-stage integrated approach for assembly sequence planning using neural networks. Expert Syst Appl 34(3):1777–1786
16. Wang JF, Liu JH, Zhong YF (2005) A novel ant colony algorithm for assembly sequence planning. Int J Adv Manuf Technol 25 (11):1137–1143
17. Cheng H, Li Y, Zhang KF (2009) Efficient method of assembly sequence planning based on GAAA and optimizing by assembly path feedback for complex product. Int J Adv Manuf Technol 42 (11–12):1187–1204
18. Gao L, Qian WR, Li XY, Wang JF (2010) Application of memetic algorithm in assembly sequence planning. Int J Adv Manuf Technol 49(9–12):1175–1184
19. Kennedy J, Eberhart RC (1995) Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, pp 1942–1948
20. Liao C, Tseng C, Luarn P (2007) A discrete version of particle swarm optimization for flowshop scheduling problems. Comput Oper Res 34(10):3099–3111
21. Tseng YJ, Yu FY, Huang FY (2011) A green assembly sequence planning model with a closed-loop assembly and disassembly sequence planning using a particle swarm optimization method. Int J Adv Manuf Technol 57(9–12):1183–1197
22. Guo YW, Li WD, Mileham AR, Owen GW (2009) Applications of particle swarm optimization in integrated process planning and scheduling. Robot Comput Integr Manuf 25(2):280–288
23. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. Proceedings of the IEEE international conference on Systems, Man and Cybernetics, Piscataway, NJ, USA, pp 4104–4108
24. Hu XH, Eberhart RC, Shi YH (2003) Swarm intelligence for permutation optimization: a case study of n-queens problem. Proceedings of the IEEE Swarm Intelligence Symposium, Indianapolis, IN, USA, pp 243–246
25. Clerc M (2004) Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: Onwubolu GC, Babu BV (eds) New optimization techniques in engineering. Studies in fuzziness and soft computing. Springer, Heidelberg, pp 219–239
26. Wang Y, Liu JH (2010) Chaotic particle swarm optimization for assembly sequence planning. Robot Comput Integr Manuf 26 (2):212–222
27. Lv HG, Lu C (2010) An assembly sequence planning approach with a discrete particle swarm optimization algorithm. Int J Adv Manuf Technol 50(5–8):761–770
28. Yamamoto K, Inoue O (1995) New evolutionary direction operator for genetic algorithms. AIAA J 33(10):1990–1993
29. Chiang CL (2005) Improved genetic algorithm for power economic dispatch of units with vale-point effects and multiple fuels. IEEE Trans Power Syst 20(4):1690–1699
30. Su CT, Lii GR, Hwung HR (2000) Position control employing fuzzy-sliding mode and genetic algorithms with modified evolutionary direction operator. Cybern Syst 31(8):873–891
31. Lin CJ, Wang JG, Lee CY (2009) Pattern recognition using neural-fuzzy networks based on improved particle swam optimization. Expert Syst Appl 36:5402–5410
32. Dini G, Santochi M (1992) Automated sequencing and subassembly detection in assembly planning. CIRP Ann Manuf Technol 41 (1):1–4