Computers and Mathematics with Applications 57 (2009) 1748–1755

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Solving uncapacitated multilevel lot-sizing problems using a particle swarm optimization with flexible inertial weight☆

Yi Han [a], Jiafu Tang [a,*], Iko Kaku [b], Lifeng Mu [a]

[a] Key Laboratory of Integrated Automation of Process Industry of Ministry of Education, Northeastern University, Shenyang, 110004, China

[b] Department of Management Science and Engineering, Akita Prefectural University, Honjo, 015-0015, Japan

ARTICLE INFO

Keywords:
Particle swarm optimization algorithm
Multilevel lot-sizing problem
Assembly structure
Uncapacitated
Genetic algorithm

ABSTRACT

The multilevel lot-sizing (MLLS) problem is a key production planning problem in materials requirements planning (MRP) system. The MLLS problem deals with determining the production lot-sizes of various items appearing in the product structure over a given finite planning horizon to minimize the production cost, the inventory carrying cost, the back ordering cost and etc. This paper proposed a particle swarm optimization (PSO) algorithm for solving the uncapacitated MLLS problem with assembly structure. All the mathematical operators in our algorithm are redefined and the inertial weight parameter can be either a negative real number or a positive one. The feasibility and effectiveness of our algorithm are investigated by comparing the experimental results with those of a genetic algorithm (GA).

1. Introduction

Material requirements planning (MRP) is an old field of study within business, but it still plays an important part in coordinating replenishment decisions for complex finished goods. The MLLS problem in MRP systems belongs to those problems that industry manufacturers daily face in organizing their overall production plans [1]. The objective of the problem is to decide the optimal production lot size and the inventory volume to minimize the production cost, the inventory carrying cost, the back ordering cost and etc [2]. The MLLS problem is a combinatorial optimization problem which can be classified into different categories according to the product structures (e.g., single level system, serial, assembly, and general systems) and the capacity structures (e.g., uncapacitated, capacitated single resource, and capacitated multiple resources) [3]. Table 1, which is based on Ref. [3], gives a brief review of some important literature for the different categories of the capacitated lot-sizing problem.

In both MRP and manufacturing resource planning (MRPII), capacity is an important factor that is always checked by a capacity requirements planning (CRP) module, but it does not mean that the uncapacitated problem is an out-of-date problem. One can justify this by the fact that, in practice, uncapacitated lot-sizing models continue to be largely used since the implementation of capacitated approaches requires much data which firms are often reluctant to collect or maintain [4]. So the uncapacitated problem still has significance.

For solving the MLLS problem, people used to adopt heuristics (e.g. Wagner–Whitin, Silver–Meal and etc.) [2]. Recently, the applications of evolutionary computing methods (ECM) were seen in some papers. Tang [14] adopted simulated annealing to solve uncapacitated serial structure problems; Dellaert an Jeunet [1,24] used genetic algorithms for solving an uncapacitated general structure problem; Xie and Dong [3] proposed a heuristic genetic algorithm for a capacitated general

☆ The project was supported by the National Natural Science Foundation of China (Grant Nos. 70625001, 70721001).

* Corresponding author.
E-mail addresses: arctic_wind@yahoo.cn (Y. Han), jftang@mail.neu.edu.cn (J. Tang), ikou_kaku@akita-pu.ac.jp (I. Kaku).

0898-1221/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2008.10.024

Table 1
Different lot-sizing problems and some important literature.

Production structure	Uncapacitated	Single resource	Multiple resources
Single level	[5–7]	[8–12]	
Serial	[5,13,14]	[13]	
Assembly	[5,15–17]	[18–21]	
General	[1,5,22–26,4]	[27,28]	[3,29,30]

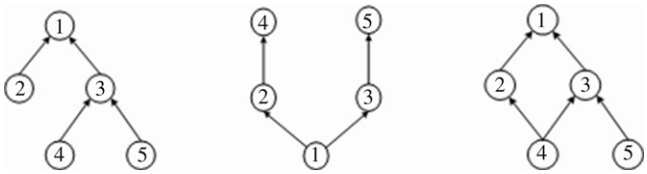

Fig. 1. Three major types of product structures.

structure problem; Dellaert and Jeunet [25] designed randomized heuristics for uncapacitated general structure problems; Jeunet and Jonard [26] developed single-point stochastic search algorithms for uncapacitated general structure problems and Pitakaso et al. [4] presented a max–min ant system for uncapacitated general structure problems. ECM may not get the optimal solution to a problem, but it takes little effort to reach a near-optimal solution. A PSO algorithm, which is one of the ECM, was developed by Kennedy and Eberhart in 1995 [31]. The original intent of PSO was to graphically simulate the graceful but unpredictable choreography of a bird flock. PSO exhibits common evolutionary computation attributes including: (1) it is initialized with a population of random solutions, (2) it searches for optima by updating generations, and (3) potential solutions, called particles, are then "flown" through the problem space by following the current optimum particles [32]. So far, few people have adopted a PSO algorithm to solve the MLLS problem. Here we proposed a PSO algorithm for solving an uncapacitated MLLS problem with assembly structure and the feasibility and effectiveness of our algorithm are investigated. Also, we plan to extend this approach to general structure problems with limited and unlimited capacities.

This paper is organized as follows: Section 2 is dedicated to the presentation and mathematical formulation of the MLLS problem. In Section 3, a brief introduction of a PSO algorithm and the framework of the proposed algorithm will be stated. The experimental frameworks and the computational results will be presented in Section 4. Finally, the conclusion and outlook can be found in Section 5.

2. Model formulation

In the MLLS problem, there are three major product structures: (1) assembly structure (2) arborescent structure (3) general structure. Fig. 1 shows three major types of product structures.

It is rather common to represent the bill of materials as a directed acyclic graph. In such a graph each node corresponds to an item and each edge (i, j) between node i and node j indicates that item i is directly required to assemble item j. Here, $\Gamma^{-1}(i)$ and $\Gamma(i)$ are used to present the sets of immediate predecessors and immediate successors of node i. The set of ancestors-immediate and non-immediate predecessors of item i is denoted by $\hat{\Gamma}^{-1}(i)$ and the set of all successors by $\hat{\Gamma}(i)$. In the last structure, product 1 is a finished good. So, we have for example $\Gamma^{-1}(1) = \{2, 3\}$; $\hat{\Gamma}^{-1}(1) = \{2, 3, 4, 5\}$; $\Gamma(4) = \{2, 3\}$; $\hat{\Gamma}(4) = \{1, 2, 3\}$ [24].

We assume that the production structure includes only one finished good (product). Variable cost parameters and variable purchase or production costs are not taken into account. In addition, we assume that no component is sold to an outside buyer, i.e. independent demands only exist for finished goods. Furthermore, no backlogging is allowed and lead-times of all items are zero. For the sake of simplicity, we assume that neither positive initial inventories ($I_{i,0} = 0, \forall i$) nor scheduled receipts are introduced. In such a context net requirements equal gross requirements for any item in the product structure [24]. The MLLS problem is a mixed integer programming problem. So we describe this problem with the following notations:

i the index of item
$C_{i,j}$ quantity of item i required to produce a unit of item j
H_i unit inventory carrying cost for item i
K_i set up cost for item i
l_i lead time to assemble, to manufacture or to purchase item i
$I_{i,t}$ inventory level of item i at the end of period t
$a_{i,t}$ a binary decision index addressed to capture the set-up cost for item i delivered in period t
$D_{i,t}$ requirements for item i in period t
$P_{i,t}$ the amounts of production/replenishment for item i in period t
M a very big number

N the total number of items
T the length of production horizon.

Let decision variable $a_{i,t} \in \{0, 1\}$ denotes whether or not the item i is produced in period t, then the decision matrix is expressed as:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,T} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,T} \\ \vdots & \vdots & \vdots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,T} \end{pmatrix}. \tag{2.1}$$

We combined the mathematical model of reference [4] and that of reference [24] and presented the mathematical model of the uncapacitated MLLS problem with assembly structure as follows:

$$\text{Min.} \sum_{i=1}^{N} \sum_{t=1}^{T} (H_i \times I_{i,t} + K_i \times a_{i,t}). \tag{2.2}$$

Subject to (each constraint must hold $\forall i, t$)

$$I_{i,t} = I_{i,t-1} + P_{i,t} - D_{i,t} \tag{2.3}$$

$$D_{i,t} = \begin{cases} D_{i,t} & \text{if } \Gamma(i) = \Phi \\ \sum_{j \in \Gamma(i)} C_{i,j} \times P_{j,t+l_j} & \text{otherwise} \end{cases} \tag{2.4}$$

$$P_{i,t} = a_{i,t} D_{i,t} + \sum_{m=t+1}^{T} \left(a_{i,m+1} D_{i,m} \prod_{q=i+1}^{m} (1 - a_{i,q}) \right) \tag{2.5}$$

$$P_{i,t} - M a_{i,t} \leq 0, \quad a_{i,t} \in \{0, 1\} \tag{2.6}$$

$$P_{i,t} \geq 0, \; I_{i,t} \geq 0. \tag{2.7}$$

The objective function (2.2) is the sum of set-up and inventory holding costs for all items over the entire planning horizon. Eq. (2.3) stands for the balance equation of production/replenishment inventory and demand. The second constraint (2.4) provides a formula to calculate the internal demand. This is a typical format of modeling the multilevel lot-sizing problem. Constraint equation (2.5) guarantees that the replenishment size of a certain period depends on the set-up decision for the latter periods. Constraint (2.6) guarantees that a set-up cost will be incurred when a batch is purchased or produced. Finally, Constraint (2.7) states that a backlog is not allowed and that production is either positive or zero.

Some attributes of the optimal solution can be found in reference [1].

3. Design scheme of a PSO algorithm for the MLLS problem

A PSO algorithm is an evolutionary computation technique formally introduced in 1995 [32]. It has been applied to many scientific research fields. The convergence and parameterizations aspects of the PSO algorithm have been discussed thoroughly [33]. The formulas have been developed for applications in artificial life and articulated five basic principles of swarm intelligence. The concept of particle swarm originated as a substitution of a simplified social system. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. PSO starts its search procedure with a particle swarm. Each particle in the swarm keeps track of its coordinates in the problem space, which are associated with the best solution (fitness) it has achieved so far. This value is called p_{Best}. Another "best" value that is tracked by the global version of the particle swarm optimization is the overall best value, and its location, obtained so far by any particle in the population is called g_{Best}. The updates of the particles are accomplished according to the following equations:

$$v_{i+1} = \omega * v_i + C_1 * rand() * (p_{best} - x_i) + C_2 * Rand() * (g_{best} - x_i). \tag{3.1}$$

$$x_{i+1} = x_i + v_{i+1}. \tag{3.2}$$

The acceleration constants C1 and C2 in Eq. (3.1) represent the weighting of the stochastic acceleration terms that pull each particle towards p_{Best} and g_{Best} positions. Thus, adjustment of these constants changes the amount of "tension" in the system. Low values of them allow particles to roam far from target regions before being tugged back, while high value results in abrupt movement toward, or past through target regions. Particle's velocities on each dimension are confined to a maximum velocity $v_{\max}$ which is a parameter specified by the user. If the sum of accelerations would cause the velocity on that dimension to exceed $v_{\max}$, then the velocity on that dimension is limited to $v_{\max}$ [32]. For more information on the principle of the PSO algorithm see Ref. [32,33].

Now, we put the MLLS problem into the framework of a PSO algorithm. Our algorithm can be viewed as a discrete one. All the mathematical operators in Eqs. (3.1) and (3.2) are redefined and the inertial weight parameter can be either

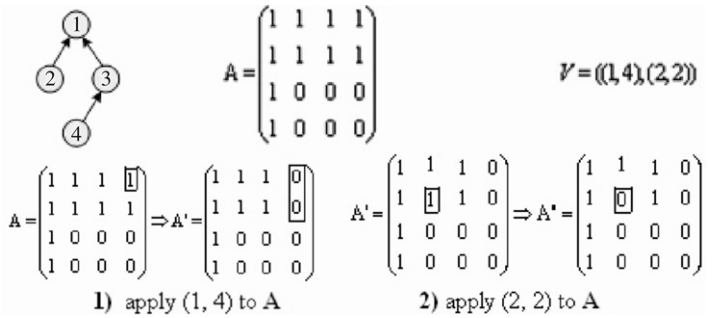

Fig. 2. Illustration of the plus operation between a position and a velocity.

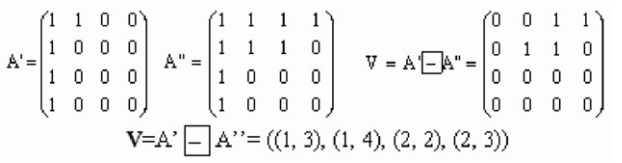

Fig. 3. Illustration of the minus operation between two positions.

a negative real number or a positive one. The thought of adopting a discrete PSO algorithm for solving the MLLS problem comes from Ref. [34]. During the execution of our algorithm, only feasible solutions are taken into consideration [1]. Any infeasible solution is immediately revised into a feasible solution after its appearance. In this algorithm, the state space (S) is composed of a collection (A) of decision matrixes (candidate solutions) like $S = \{A_i\}$ and a position in S is denoted by a matrix in collection (A). Before using a PSO algorithm, some symbols and the velocity (V) in Eqs. (3.1) and (3.2) need to be redefined as follows:

3.1. Velocity (V)

The velocity of a PSO algorithm is defined as a collection of numerical pairs; the length of a velocity V, denoted by $\|V\|$, equals to the number of numerical pairs. For example, a velocity V is given as $V = ((i_k, j_k)), i_k \in \{1 \ldots N\}, j_k \in \{1 \ldots T\}, k\uparrow_1^{\|V\|}$, the number of pairs (i_k, j_k) equals $\|V\|$. N is the number of items and T is the number of planning periods.

3.2. Velocity plus velocity

The plus of two velocities is simply considered as a combinational operation on two velocities. In order to shorten the length of the resulting velocity, the repetitive numerical pairs will be eliminated. For illustration, let $V_1 = ((1, 4), (2, 3), (5, 7))$ and $V_2 = ((1, 3), (2, 3))$, then $V = V_1 \oplus V_2 = ((1, 3), (1, 4), (2, 3), (5, 7))$.

3.3. Velocity minus velocity

The subtraction of two velocities is simply considered as an elimination-and-recombination operation on two velocities. First, the same numerical pairs are eliminated from both velocities. Then, the two velocities are combined together. For illustration, let $V_1 = ((1, 4), (2, 3), (5, 7))$ and $V_2 = ((1, 3), (2, 3))$, then $V = V_1 \ominus V_2 = ((1, 3), (1, 4), (5, 7))$.

3.4. Position plus velocity

The plus of a position and a velocity is a continuous mutation on bits in a position according to corresponding numerical pairs of a velocity. It can be seen from Fig. 2 that a production structure is presented, a position (matrix) is given out and a velocity V is shown. The matrix A is firstly transformed into A' by mutating '1' at position (1, 4). Then, a cumulative mutation operation [24] at the position (2, 4) is performed to satisfy the feasibility constraint by changing '1' into '0'. After that, A' is changed further into A'' by substituting the value '1' at (2, 2) with '0'.

3.5. Position minus position

The result of a subtraction between a position and another position is a velocity. The resulting velocity is obtained through recording those different points of the two positions. For illustration, Fig. 3 showed the subtraction operation between A' and A''.

3.6. Coefficient multiple velocity

Let c be a real coefficient and V be a velocity. The result of multiplication between c and V is obtained as a velocity depending based on the following two cases of c.

- If $|c| < 1$, denote $\|cV\|$ be the greatest integer number smaller than or equal to $c\|V\|$, then we can get $c \otimes V = ((i_k, j_k))$, $k\uparrow_1^{\|cV\|}$. For example, let $c = 0.6$ and $V = ((2, 3), (2, 5), (3, 7))$, then $\|V\| = 3$, $c\|V\|$=1.8, $\|cV\| = 1$, $c \otimes V = (2, 3)$.
- If $|c| \geq 1$, then V and the length of V is kept unchanged. Hence, the appearance of repetitive numerical pairs is avoided and the multiplication of c and V is simplified.

When using our algorithm, the inertial weight parameter ω should not be ignored. For one hand, if ω is a real number between 0 and 1 then the formulas to execute the PSO algorithm are as Eqs. (3.3) and (3.4); if ω is a real number between -1 and 0 then the formulas for running the PSO algorithm are as Eqs. (3.5) and (3.4). The priority relation between those redefined mathematical operators is the same as before.

$$v_{i+1} = \omega \otimes v_i \oplus (C_1 * rand()) \otimes (p_{best} \boxminus x_i) \oplus (C_2 * Rand()) \otimes (g_{best} \boxminus x_i) \tag{3.3}$$

$$x_{i+1} = x_i \boxplus v_{i+1} \tag{3.4}$$

$$v_{i+1} = (\mid \omega \mid \otimes v_i) \ominus ((C_1 * rand()) \otimes (p_{best} \boxminus x_i) \ominus (C_2 * Rand()) \otimes (g_{best} \boxminus x_i)). \tag{3.5}$$

The pseudo code of PSO algorithm is shown in Table 2.

Table 2
The pseudo code of PSO algorithm

```
for i = 1 to PopSize
        initialize Pop(i)
        revise Pop(i) to a feasible candidate solution
        assess Pop(i)
        copy Pop(i) to P_local(i) and copy Pop(i)_value to P_local (i)_value
    next i
        record the best Pop(i) as Pbest
    do while Iter_ID <= Max_Iteration
                for j = 1 to PopSize
            w = Rnd() * 2 − 1
            If w >= 0 Then
                call Multiple_wv()
                call Plocal_sub_Pop()
                call Pbest_sub_Pop()
                call Multiple_C1_rand_V1()
                call Multiple_C2_Rand_V2()
                call V_add_V1_V2()
            else
                w = −w
                call Multiple_WV()
                call BL_sub_Pos()
                call Multiple_CV()
                call Multiple_CV()
                call V1_add_V2()
                call add_V()
            end if
                    call Plus_Pos_V()
    next j
for k = 1 to PopSize
  update Plocal(k) and Plocal(k)_value
next k
    update Pbest
    check the improvement of Pbest
    if Yes then
            number = 0
            else
                number = number + 1
            end if
                Iter_ID = Iter_ID + 1
                The stopping rule is reached?
                Yes, stop and return Pbest.
                No, loop
```

4. Experimental framework and simulation results

In this section, we present three computational experiments to test our algorithm (two small-sized problems and a medium-sized problem). The platform of the experiments is a PC with a 2.8 GHz CPU and 1G RAM. This algorithm is coded

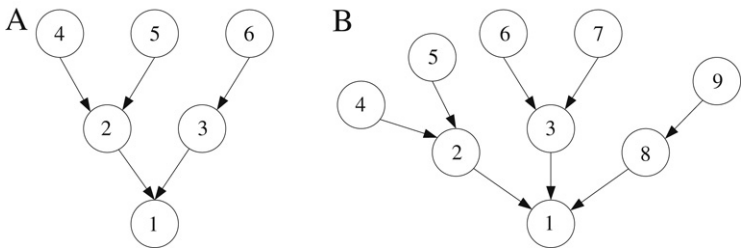

Fig. 4. Two production structures of small-sized MLLS problems.

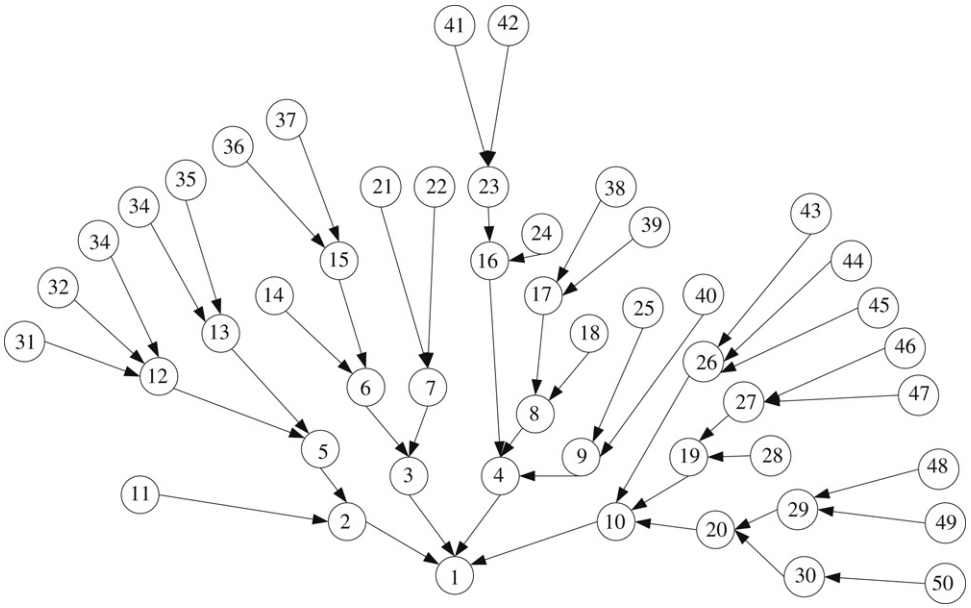

Fig. 5. The production structure used in experiment 3.

in Visual Basic 6.0. Two stopping rules are adopted. One is max-iteration rule and the other is no-prominent-improvement rule. In all the experiments, we use a swarm of 60 particles, acceleration constants $C_1 = C_2 = 2$ and the maximal velocity $v_{\max} = 0.1 \times N \times T$. The max iteration is 500. If in 200 iterations the best achieved solution is not improved, then the algorithm is stopped. We run 100 times for each experiment. Fig. 4A is a product structure used for the first experiment, Fig. 4B is a product structure used for the second experiment, and Fig. 5 is a product structure [15] for the last experiment.

The constant parameters used for small-sized problems are listed in Table 3. Those parameters for medium-sized MLLS problem used in experiment 3 are randomly generated. In experiment 3, the length of planning horizon is 10. The results of the small-sized problem are listed in Table 4 and the results of the medium-sized problem are listed in Table 5.

Through Tables 4 and 5, we can see that both the PSO algorithm and GA outperform the WW algorithm in terms of the best achieved solution. In most cases, our algorithm is more stable than GA with comparatively less calculation time. Only in the 6×15 case our algorithm is somewhat less stable. In all cases, our algorithm obtained better results than those of GA as we observe the best solution, the worst solution and the mean cost. All these experiments showed that our algorithm is a feasible and effective method for solving the uncapacitated MLLS problem with assembly structure.

5. Conclusions and future study

In this paper, a PSO algorithm is proposed to solve the uncapacitated MLLS problems with assembly structure. Our contribution is twofold: (1) all the mathematical operators in PSO formulas are redefined; (2) we provide a new way for using a PSO algorithm to those researchers who work on MLLS problems. In our future study, a PSO algorithm suitable for solving more complicated MLLS problems should be developed and the application scope should be further extended to other similar research fields.

Table 3
The constant parameters used for small-sized problems.

Item number	Relation between items		Fees per unit	
	$\Gamma(i)$	$C(i, \Gamma(i))$	Holding cost	Setup cost
1	0	0	1	130
2	1	1	2	120
3	1	3	1	25
4	2	2	2	30
5	2	4	3	30
6	3	2	1	40
7	3	1	1	130
8	1	2	2	120
9	8	1	1	25

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$D_{1,t}$	32	41	148	36	120	28	32	12	30	10	32	41	148	36	120

Table 4
The simulation results of small-sized MLLS problems.

Problem	Algorithm	Best	Worst	Mean cost	Std of cost	Mean time (s)
6×10	PSO	1493	1917	1560.7	84.4	4.2
	GA	1493	2656	1904.89	248.63	5.6
	WW	1707	1707	1707	0	<0.1
6×12	PSO	1895	5565	2143.94	578.4	5.3
	GA	1895	6446	2526.9	601.79	8.0
	WW	2123	2123	2123	0	<0.1
6×15	PSO	2546	8214	3664.73	1503.7	6.3
	GA	2623	9170	3982.04	1305.2	10.7
	WW	2909	2909	2909	0	<0.1
9×10	PSO	2043	2877	2158.9	136.2	6.5
	GA	2043	5813	2581.79	767.47	10.1
	WW	2807	2807	2807	0	<0.1
9×12	PSO	2522	9525	3057.8	935.7	7.2
	GA	2522	9951	3887.2	1457.55	12.9
	WW	3498	3498	3498	0	<0.1
9×15	PSO	3448	12457	5951.9	2771.6	9.2
	GA	3714	12966	8302.23	2906.44	16.3
	WW	4834	4834	4834	0	<0.1

Table 5
The simulation results of medium-sized MLLS problem.

Algorithm	Best	Worst	Mean cost	Std of cost	Mean time (s)
PSO	4921	13 339	5938.28	2106.2	30.9
GA	4921	25 037	9241.86	5012.88	33
WW	13207	13 207	13207	0	0.1

References

[1] N. Dellart, J. Jeunet, N. Jonard, A genetic algorithm to solve the general multi-level lot-sizing problem with time-varying costs, International Journal of Production Research 68 (2000) 241–257.
[2] A. Silver, F. Pyke, R. Peterson, Inventory Management and Production Planning and Scheduling, 2000.
[3] J. Xie, J.F. Dong, Heuristic genetic algorithm for general capacitated lot-sizing problems, Computer and Mathematics with Applications 44 (2002) 263–276.
[4] R. Pitakaso, C. Almeder, K.F. Doerner, R.F. Hartl, A MAX–MIN ant system for unconstrained multi-level lot-sizing problems, Computers and Operations Research 34 (2007) 2533–2552.
[5] Y.P. Gupta, Y. Keung, A review of multi-stage lot-sizing models, International Journal of Operations and Production Management 10 (1990) 57–73.
[6] B.J. Coleman, A further analysis of variable demand lot-sizing techniques, Production and Inventory Management Journal 33 (1992) 19–24.
[7] A. Aggarwal, J.K. Park, Improved algorithms for economic lot size problems, Operations Research 41 (1993) 549–571.
[8] I. Barang, T.J. VanRoy, L.A. Wolsey, Strong formulations for multi-item capacitated lot-sizing, Management Science 30 (1984) 1255–1261.
[9] G.D. Eppen, R.K. Martin, Solving multi-item capacitated lot-sizing problems using variable redefinition, Operations Research 35 (1987) 832–848.
[10] H.O. Guenther, Planning of lot sizes and capacity requirements in a single stage production system, European Journal of Operational Research 31 (1987) 223–231.
[11] J. Maes, L.N.V. Wassenhove, Multi-item single-level capacitated dynamic lot-sizing heuristics: A general review, Journal of the Operational Research Society 39 (1988) 991–1004.
[12] W.W. Trigeiro, L.J. Thomas, J.O. McClain, Capacitated lot sizing with setup times, Management Science 35 (1989) 353–366.
[13] M.R. Lambrecht, J. VanderEechen, H. Vanderveken, Review of optimal and heuristic models for a class of facilities in series dynamic lot size problems, in: Multi-level Production/Inventory Control Systems: Theory and Practice, North-Holland, Amsterdam, 1981, pp. 69–94.
[14] O. Tang, Simulated annealing in lot sizing problems, International Journal of Production Economics 88 (2004) 173–181.
[15] P. Afentakis, B. Gavish, V. Karmarkar, Computationally efficient optimal solutions to the lot-sizing problem in multi-stage assembly systems, Management Science 30 (1984) 222–239.

[16] P. Afentakis, A parallel heuristic algorithm for lot sizing in multistage production systems, IIE Transactions 19 (1987) 34–42.
[17] K. Rosling, Optimal lot sizing for dynamic assembly systems, in: Multistage Production Planning and Inventory Control, Springger, Berlin, 1986, pp. 119–131.
[18] J. Maes, J.O. McClain, L.N.V. Wassenhove, Multilevel capacitated lot-sizing complexity and LP-based heuristic, European Journal of Operational Research 53 (1991) 131–148.
[19] P.J. Billington, J.O. McClain, L.J. Thomas, Heuristic for multi-level lot-sizing with a bottleneck, Management Science 32 (1986) 989–1006.
[20] Y. Roll, R. Karni, Multi-item multi-level lot sizing with an aggregate capacity constraint, European Journal of Operational Research 51 (1991) 73–87.
[21] R. Kuik, M. Salomon, L.N.V. Wassenhove, Linear programming, simulated annealing and tabu search heuristic for lot sizing in bottleneck assembly systems, IIE Transactions 25 (1993) 62–72.
[22] C.E. Heinrich, C. Schneeweiss, Multi-stage lot-sizing for general production systems, in: Multistage Production Planning and Inventory Control, Springer, Berlin, 1986, pp. 150–181.
[23] P. Afentakis, B. Gavish, Optimal lot-sizing algorithms for complex product structures, Operations Research 34 (1986) 237–249.
[24] N. Dellaert, J. Jeunet, Solving large unconstrained multilevel lot-sizing problems using a hybrid genetic algorithm, International Journal of Production Research 38 (2000) 1083–1099.
[25] N. Dellaert, J. Jeunet, Randomized multi-level lot-sizing heuristics for general product structures, European Journal of Operational Research 148 (2003) 211–228.
[26] J. Jeunet, N. Jonard, Single-point stochastic search algorithms for the multi-level lot-sizing problem, Computers and Operations Research 32 (2005) 985–1006.
[27] S. Helber, Lot-sizing in capacitated production planning and control system, OR Specktrum 17 (1995) 5–18.
[28] H. Tempelmeier, M. Derstroff, A lagrangean-based heuristic for dynamic multilevel multiitem constrained lotsizing with setup times, Management Science 42 (1996) 738–757.
[29] A.R. Clark, V.A. Armentano, A heuristic for a resource-capacitated multi-stage lot-sizing problem with lead-time, Journal of the Operational Research Society 46 (1995) 1208–1222.
[30] G. Belvaux, L.A. Wolsey, A specialized branch-and-cut system for lot-sizing problems, Management Science 46 (2000) 724–738.
[31] J. Kennedy, R Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, 1995, pp. 1942–1948.
[32] Y. Dong, J.F. Tang, B.D. Xu, D.W. Wang, Application of particle swarm optimization to nonlinear programming problem, Computers and Mathematics with Applications 49 (2005) 1655–1668.
[33] P.Y. Yin, A discrete particle swarm algorithm for optimal polygonal approximation of digital curves, Journal of Visual communication and Image Representation 15 (2004) 241–260.
[34] M. Clerc, Discrete particle swarm optimization illustrated by the traveling salesman problem. Available on: http://www.mauriceclerc.net, 2000.