We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 4

A Comparative Study on Meta Heuristic Algorithms for Solving Multilevel Lot-Sizing Problems

Ikou Kaku, Yiyong Xiao and Yi Han

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/55279

1. Introduction

Material requirements planning (MRP) is an old field but still now plays an important role in coordinating replenishment decision for materials/components of complex product structure. As the key part of MRP system, the multilevel lot-sizing (MLLS) problem concerns how to determine the lot sizes for producing/procuring multiple items at different levels with quantitative interdependences, so as to minimize the total costs of production/procurement setup and inventory holding in the planning horizon. The problem is of great practical importance for the efficient operation of modern manufacturing and assembly processes and has been widely studied both in practice and in academic research over past half century. Optimal solution algorithms exist for the problem; however, only small instances can be solved in reasonable computational time because the problem is NP-hard (Steinberg and Napier, 1980). Early dynamic programming formulations used a network representation of the problem with a series structure (Zhangwill, 1968,1969) or an assembly structure (Crowston and Wagner,1973). Other optimal approaches involve the branch and bound algorithms (Afentakis et al., 1984, Afentakis and Gavish, 1986) that used a converting approach to change the classical formulation of the general structure into a simple but expanded assembly structure. As the MLLS problem is so common in practice and plays a fundamental role in MRP system, many heuristic approaches have also been developed, consisting first of the sequential application of the single-level lot-sizing models to each component of the product structure (Yelle,1979, Veral and LaForge,1985), and later, of the application of the multilevel lot-sizing models. The multilevel models quantify item interdependencies and thus perform better than the single-level based models (Blackburn and Millen, 1985, Coleman and McKnew, 1991).

Recently, meta-heuristic algorithms have been proposed to solve the MLLS problem with a low computational load. Examples of hybrid genetic algorithms (Dellaert and Jeunet, 2000, Dellaert et al., 2000), simulated annealing (Tang, 2004, Raza and Akgunduz, 2008), particle

swarm optimization (Han et al, 2009, 2012a, 2012b), and soft optimization approach based on segmentation (Kaku and Xu, 2006, Kaku et al, 2010), ant colony optimization system (Pitakaso et al., 2007, Almeda, 2010), variable neighborhood search based approaches (Xiao et al., 2011a, 2011b, 2012), have been developed to solve the MLLS problem of large-scale. Those meta-heuristic algorithms outperform relative simplicity in solving the MLLS problems, together with their cost efficiency, make them appealing tool to industrials, however they are unable to guarantee an optimal solution. Hence those meta-heuristic algorithms that offer a reasonable trade-off between optimality and computational feasibility are highly advisable. It is very reasonable to consider the appropriateness of the algorithm, especially is which algorithm most appropriate for solving the MLLS problems?

In this chapter, We first review the meta-heuristic algorithms for solving the MLLS problems, especially focus on the implemental techniques and their effectives in those meta-heuristic algorithms. For simplicity the MLLS problems are limited with time-varying cost structures and no restrictive assumption on the product structure. Even so the solutions of the MLLS problems are not simply convex but becoming very complex with multi minimums when the cost structure is time-varying and the product structure is becoming general. Comparing those implement methods used in different algorithms we can find some essential properties of searching better solution of the MLLS problems. Using the properties therefore, we can specify the characteristics of the algorithms and indicate a direction on which more efficient algorithm will be developed.

Second, by using these properties as an example, we present a succinct approach—iterated variable neighborhood descent (IVND), a variant of variable neighborhood search (VNS), to efficiently solve the MLLS problem. To examine the performance of the new algorithm, different kinds of product structures were considered including the component commonality and multiple end-products, and 176 benchmark problems under different scales(small, medium and large) were used to test against in our computational experiments. The performance of the IVND algorithm were compared with those of three well-known algorithms in literatures—the hybrid genetic algorithm by Dellaert and Jeunet (2000a), the MMAS algorithm by Pitakaso et al. (2007), and the parallel genetic algorithm by by Homberger (2008), since they all tackled the same benchmark problems. The results show that the IVND algorithm is very competitive since it can on average find better solutions in less computing time than other three.

The rest of this chapter is organized as follows. Section 2 describes the MLLS problem. Section 3 gives an overview of related meta-heuristic algorithms firstly, and several implemental techniques used in the algorithms are discussed. Then section 4 explains the initial method and six implemental techniques used in IVND algorithm, and the scheme of the proposed IVND algorithm. In section 5, computational experiments are carried out on three 176 benchmark problems to test the new algorithm of efficiency and effectiveness and compared with existing algorithms. Finally, in section 7, we symmary the chapter.

2. The MLLS problems

The MLLS problem is considered to be a discrete-time, multilevel production/inventory system with an assembly structure and one finished item. We assume that external demand for the

http://dx.doi.org/10.5772/55279

finished item is known up to the planning horizon, backlog is not allowed for any items, and the lead time for all production items is zero. Suppose that there are M items and the planning horizon is divided into N periods. Our purpose is to find the lot sizes of all items so as to minimize the sum of setup and inventory-holding costs, while ensuring that external demands for the end item are met over the N-period planning horizon.

To formulate this problem as an integer optimization problem, we use the same notation of Dellaert and Jeunet (2000a), as follows:

i : Index of items, $i = 1, 2, \ldots, M$

t : Index of periods, $t = 1, 2, \ldots, N$

H_i: Unit inventory-holding cost for item i

S_i: Setup cost for item i

$I_{i,t}$: Inventory level of item i at the end of period t

$x_{i,t}$: Binary decision index addressed to capture the setup cost for item i

$D_{i,t}$: Requirements for item i in period t

$P_{i,t}$: Production quantity for item i in period t

$C_{i,j}$: Quantity of item i required to produce one unit of item j

$\Gamma(i)$: set of immediate successors of items i

M: A large number

The objective function is the sum of setup and inventory-holding costs for all items over the entire planning horizon, denoted by TC (total cost). Then

$$TC = \sum_{i=1}^{M}\sum_{t=1}^{N}(H_i \cdot I_{i,t} + S_i \cdot x_{i,t}). \tag{1}$$

The MLLS problem is to minimize TC under the following constraints:

$$I_{i,t} = I_{i,t-1} + P_{i,t} - D_{i,t}, \tag{2}$$

$$D_{i,t} = \sum_{j\in\Gamma_i} C_{i,j} \cdot P_{j,t+l_j} \qquad \forall i \mid \Gamma_i \neq \varphi, \tag{3}$$

$$P_{i,t} - M \cdot x_{i,t} \leq 0, \tag{4}$$

$$I_{i,t} \geq 0, \quad P_{i,t} \geq 0, \quad x_{i,t} \in \{0,1\}, \quad \forall i,t. \tag{5}$$

where Equation 2 expresses the flow conservation constraint for item i. Note that, if item i is an end item (characterized by $\Gamma(i)=\varphi$), its demand is exogenously given, whereas if it is a component (such that $\Gamma(i)\neq\varphi$), its demand is defined by the production of its successors (items belonging to $\Gamma(i)$ as stated by Equation 3). Equation 3 guarantees that the amount $P_{j,t}$ of item j available in period t results from the exact combination of its predecessors (items belonging to Γ_i in period t). Equation 4 guarantees that a setup cost is incurred when a production is produced. Equation 5 states that backlog is not allowed, production is either positive or zero, and that decision variables are 0, 1 variables.

Because $x_{i,t} \in \{0, 1\}$ is a binary decision variable for item i in period t, $X = \{x_{i,t}\}_{M \times N}$ represents the solution space of the MLLS problem. Searching for an optimal solution of the MLLS problem is equivalent to finding a binary matrix that produces a minimum sum of the setup and inventory-holding costs. Basically, there exists an optimal solution if

$$x_{i,t} \cdot I_{i,t-1} = 0 \tag{6}$$

Equation 6 indicates that any optimal lot must cover an integer number of periods of future demand. We set the first column of X to be 1 to ensure that the initial production is feasible because backlog is not allowed for any item and the lead times are zero. Since there is an inner corner property for assembly structure (see Tang (2004)), we need to have $x_{i,t} \geq x_{k,t}$ if item creates internal demand for item k. Thus we need a constraint in order to guarantee that the matrix is feasible.

3. Meta-heuristic algorithms used to solve MLLS problems

The meta-heuristic algorithms are widely used to refer to simple, hybrid and population-based stochastic local searching (Hoos and Thomas 2005). They transfer the principle of evolution through mutation, recombination and selection of the fittest, which leads to the development of solutions that are better adapted for survival in a given environment, to solving computationally hard problems. However, those algorithms often seem to lack the capability of sufficient search intensification, that is, the ability to reach high-quality candidate solutions efficiently when a good starting position is given. Hence, in many cases, the performance of the algorithms for combinatorial problems can be significantly improved by adding some implemental techniques that are used to guide an underlying problem-specific heuristic. In this chapter, our interesting is on the mechanisms of those implemental techniques used to solve a special combinatorial optimization problem, i.e. MLLS problem. Hence we first review several existing meta-heuristic algorithms for solving the MLLS problems.

http://dx.doi.org/10.5772/55279

3.1. Soft optimization approach

Soft optimization approach (SOA) for solving the MLLS problems is based on a general sampling approach (Kaku and Xu, 2006; Kaku et al, 2010). The main merit of soft optimization approach is that it does not require any structure information about the objective function, so it can be used to treat optimization problems with complicated structures. However, it was shown that random sampling (for example simple uniform sampling) method cannot produce a good solution. Several experiments had been derived to find the characteristics of an optimal solution, and as a result applying the solution structure information of the MLLS problem to the sampling method may produce a better result than that arising from the simple uniform sampling method. A heuristic algorithm to segment the solution space with percentage of number of 1s has been developed and the performance improvement of solving MLLS problem was confirmed. It should be pointed that the SOA based on segmentation still remains the essential property of random sampling but limited with the searching ranges, however the adopted new solution(s) does not remain any information of the old solution(s). Therefore the improvement of solution performance can only be achieved by increasing the numbers of samples or by narrowing the range of segment.

3.2. Genetic algorithm

Genetic algorithm (GA) has been developed firstly for solving the MLLS problems in (Dellaert and Jeunet, 2000a, Dellaert et al. 2000b, Han et al. 2012a, 2012b). In fact, it firstly created the way that solving MLLS problems by using meta-heuristic algorithms. Several important contributions were achieved. Firstly a very general GA approach was developed and improved by using several specific genetic operators and a roulette rule to gather those operators had been implemented to treat the two dimensional chromosomes. Secondly, comparison studies had been provided to show that better solution could be obtained than those existing heuristic algorithms, based on several benchmarks data collected from literature. Later such benchmarks provide a platform of developing meta-heuristic algorithms and evaluating their performance. Because the complexity of MLLS problem and the flexibility and implement ability of GA are matching each other so that GA seems powerful and effective for solving MLLS problem. However, even several operators as single bit mutation; cumulative mutation; inversion; period crossover and product crossover were combined in the GA algorithm but what operators were effective in better solution searching process was not presented clearly. It is the point that we try to define and solve in this chapter.

3.3. Simulated annealing

Simulated annealing (SA) has been also developed to solve the MLLS problem (Tang,2004; Raza and Akgunduz,2008). It starts from a random initial solution and changing neighbouring states of the incumbent solution by a cooling process, in which the new solution is accepted or rejected according to a possibility specified by the Metropolis algorithm. Also parameters used in the algorithm had been investigated by using the analysis of variance approach. It had been reported that the SA is appropriate for solving the MLLS problem however verified only in very small test problems. Based on our understanding for SA, different from other meta-heuristic algorithms

like GA, SA is rather like a local successive search approach from an initial solution. Then almost information of the old solution can be remained which may lead a long time to search better solution if it is far from the original solution. Also several implement methods con be used to improve the effective of SA (see Hoos and Thomas 2005). It is a general point to improve the effective of SA through shortening the cooling time with some other local searching methods.

3.4. Particle swarm optimization

Particle swarm optimization (PSO) is also a meta-heuristic algorithm formally introduced (Han et al,2009, 2011). It is a suitable and effective tool for continuous optimization problems. Recently the standard particle swarm optimization algorithm is also converted into a discrete version through redefining all the mathematical operators to solve the MLLS problems (Han et al, 2009). It 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 pBest. Another "best" value 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, which is called gBest. Gather those so-called optimal factors into current solutions then they will converge to a better solution. It has been reported that comparing experiments with GA proposed in (Dellaert and Jeunet, 2000a, Dellaert et al. 2000b) had been executed by using the same benchmarks and better performance were obtained. Consider the essential mechanism of PSO, it is clear that those optimal factors (pBest and gBest) follow the information of the particle passed and direct where the better solution being. However, it has not been explained clearly that whether those optimal factors remained the required information when the PSO is converted into a discrete form.

3.5. Ant colony optimization

A special ant colony optimization (ACO) combined with linear program has been developed recently for solving the MLLS problem (Pitakaso et al. 2007, Almeda 2010). The basic idea of ant colony optimization is that a population of agents (artificial ants) repeatedly constructs solutions to a given instance of a combinatorial optimization problem. Ant colony optimization had been used to select the principle production decisions, i.e. for which period production for an item should be schedules in the MLLS problems. It starts from the top items down to the raw materials according to the ordering given by the bill of materials. The ant's decision for production in a certain period is based on the pheromone information as well as on the heuristic information if there is an external (primary) or internal (secondary) demand. The pheromone information represents the impact of a certain production decision on the objective values of previously generated solutions, i.e. the pheromone value is high if a certain production decision has led to good solution in previous iterations. After the selection of the production decisions, a standard LP solver has been used to solve the remaining linear problem. After all ants of an iteration have constructed a solution, the pheromone information is updated by the iteration best as well as the global best solutions. This approach has been reported works well for small and medium-size MLLS problems. However for large instances the solution method leads to high-quality results, but cannot beat highly specialized algorithms.

http://dx.doi.org/10.5772/55279

3.6. Variable neighbourhood search

Variable neighborhood search (VNS) is also used to solve the MLLS problem (Xiao et al., 2011a, 2011b, 2012). The main reasoning of this searching strategy, in comparison to most local search heuristics of past where only one neighborhood is used, is based on the idea of a systematic change of predefined and ordered multi neighborhoods within a local search. By introducing a set of distance-leveled neighborhoods and correlated exploration order, the variable neighborhood search algorithm can perform a high efficient searching in nearby neighborhoods where better solutions are more likely to be found.

There may other different meta-heuristic algorithms have been proposed for solving the MLLS problems, but it can be considered that the algorithms updated above can cover almost fields of the meta-heuristic so that the knowledge obtained in this chapter may has high applicable values. All of the meta-heuristic algorithms used to solve the MLLS problems are generally constructed by the algorithm describe in Fig.1 as follows.

Repeat the following step (1), (2) and (3):
(1)Find an initial solution(s) by using the *init* function.
(2)Repeat (a) and (b) steps:
  (a)Find a new solution randomly in solution space by using the *step* function;
  (b)Decide whether the new solution should be accepted.
(3)Terminate the search process by using the *terminate* function and output the best solution found.

Figure 1. General algorithm for solving the MLLS problem

In Fig. 1, at the beginning of the search process, an initial solution(s) is generated by using the function *init.* A simple *init* method used to generate the initial solution may be the random sampling method, and often several heuristic concepts of MLLS problem are employed to initialize the solution because they can help obtaining better performance. Moreover, the *init* methods are usually classified into two categories in terms of single solution or multi solutions (usually called population). Function *step* shows the own originality of the algorithm by using different implement techniques. By comparing the total costs of old and new solutions, it accepts usually the solution with smaller one as next incumbent solution. Finally, function *terminate*, a user-defined function such as numbers of calculation, coverage rate and so on, is used to terminate the program.

All of the implement techniques used in *step* function, which are represented in all of the meta-heuristic algorithms for solving the MLLS problems, can be summarized as below.

1. Single- bit mutation

Just one position (i,t) has randomly selected to change its value (from 0 to 1 or the reverse)

2. Cumulative mutation

When a single-bit mutation is performed on a given item, it may trigger mutating the value(s) of its predecessors.

3. Inversion

One position (i,t) has been randomly selected, then compare its value to that of its neighbor in position $(i,t+1)$. If the two values are different from each other(1 and 0, or 1 and 0), then exchanges the two values. Note the last period $t=N$ should not be selected for inversion operation.

4. Period crossover

A point t in period (1, N) is randomly selected for two parents to produce off-springs by horizontally exchanging half part of their solutions behind period t.

5. Product crossover

An item i is randomly selected for two parents to produce off-springs by vertically exchanging half part of their solutions below item i.

Table 1 summaries what implement techniques and *init* function have been used in the existing meta-heuristic algorithms. From Table 1 it can be observed that all of the algorithms use the single-bit mutation method in their *step* function, but their ways of making the mutations are different. Clearly, SOA creates new solution(s) without using any previous information so that it may is recognized a non-implemental approach. Reversely SA uses single-bit and cumulative mutation based on incumbent solution therefore it reserves almost all of the previous information. Moreover, a random uphill technique is used in SA to escape from local optima so that the global optimal solution may is obtained. However a very long computing time is needed to get it. On the other hand, PSO uses the single-bit mutation method to create the new candidate in *step* function but starting with multiple initial solutions. According to the concept of original PSO, some information of solutions obtained before may be added into the construction of new solutions in order to increase the probability of generating better solutions. However, it needs a proof of such excellent property is guaranteed in the implemental process but is not appeared in the algorithm proposed before. It may is a challenge work for developing a new PSO algorithm for solving the MLLS problem. Finally, because the techniques of inversion and crossover (which were usually used in other combinational optimization problem such as a Travelling Salesman Problem) only have been used in GA, it is not able to compare them with other algorithms.

	Single- bit mutation	Cumulative mutation	Inversion	Period crossover	Product crossover	initial solution
SOA	—	—	—	—	—	many
GA	○	○	○	○	○	multi
SA	○	○	—	—	—	single
PSO	○	○	—	—	—	multi
ACO	○	—	—	—	—	single
VNS	○	○	○	—	—	single

Table 1. The implement techniques used in various algorithms

http://dx.doi.org/10.5772/55279

Note the implement techniques can change the states of the incumbent solution(s) to improve the performance with respect to total cost function, so that which implement technique could do how many contributions to the solution improvement is very important for the computing efficiency of MLLS problems. Here our interesting is on those implement techniques used in the above algorithms, which had been reported appropriate in their calculation situations. We define several criteria for evaluating the performance and effective of the implemental techniques. We firstly define distance-based neighborhood structures of the incumbent solution. The neighborhoods of incumbent solution are sets of feasible solutions associated with different distance measurements from the incumbent solutions. The distance means the exchanged number of different points of two incumbent solutions.

Definition 1. *Distance from incumbent solution*: For a set of feasible solutions of a MLLS problem, i.e., $X=\{x_{i,t}\}$, a solution X' belongs to the k^{th}-distance of incumbent solution x, i.e. $N_k(x)$, if and only if it satisfies, $x' \in N_k(x) \Leftrightarrow \rho(x, x')=k$, where k is a positive integer. Distance between any two elements in in X, e.g., x and x', is measured by

$$\rho(x,x') = \left|x \setminus x'\right| = \left|x' \setminus x\right| \qquad \forall x, x' \in X \tag{7}$$

Where $|\bullet \setminus \bullet|$ denotes the number of different points between two solutions, i.e., $|\bullet \setminus \bullet| = \sum_{i=1}^{M}\sum_{t=1}^{N} |x_{i,t} - x'_{i,t}|$

For example of a MLLS problem with 3 items and 3 periods, and three feasible solutions: x, x' and x'', which are as following,

$$x = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{vmatrix}, \quad x' = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}, \quad x'' = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{vmatrix}$$

According to **Definition 1** above, we can get their distances such that: $\rho(x, x')=1$, $\rho(x', x'')=1$, $\rho(x, x'')=2$.

Therefore, creating a solution with k^{th} distance, i.e. $N_k(x)$, to the incumbent solution can be realized by changing the values of k different points of incumbent solution. It can be considered that less changes, e.g. $N_1(x)$, can avoid too much damage to the maybe good properties of the incumbent solution, in which some points may be already in its optimal positions. While multiple changes, e.g. $N_k(x)$ (k>1), lead a new solution to be very different from the incumbent solution, so it may also destroy the goodness of the original solution. However on the other hand local optimization may be overcame by multiple changes. Hence, following hypothesis may be considered to evaluate the performance and effective of an implement technique. Secondly we define range measurements that means the changing points are how far from the incumbent solutions. Eventually, we can evaluate those implemental techniques used in the algorithms to solve the MLLS problems by measuring the distance and range when the solution has been changed.

Definition 2. *Changing Range of incumbent solution*: Any elements belong to the k^{th}-distance of incumbent solution x, i.e. $N_k(x)$, have a range in which the changed items and periods of incumbent solution have been limited.

Therefore, if the changing result of an incumbent solution by an implement technique falls into a small range, it seems be a local search and hence may give little influence on the total cost function. Otherwise a change of incumbent solution within a large range may be a global search increasing the probability of worse solutions found and resulting lower searching efficiency. Comparing the considerations of constructing mutation in the algorithms above, we can find that only short distance (for cumulative mutation, it is a very special case of long distance) has been executed in GA, SA and PSO with different implement techniques. For example, all of the algorithms use mutation to change the solution but only GA uses the probability of obtaining a good solution to increase the share rate of the operators. While SA uses a probability of accepting a worse solution to create an uphill ability of escaping from the traps of local optima, and PSO uses other two factors (*pBest* and *gBest*) to point a direction in which the optimal solution maybe exist. SOA and ACO are using basic random sampling principle to select the decisions in all positions of item and period. The difference is that SOA does not remain any information of previous solutions so large numbers of solution should be initialed and long distance (about half of items*periods) always existed, whereas ACO uses a level by level approach to do single-bit mutation in a solution so goodness of previous solution may has been remained with long distance. Finally, VNS directly use distances to search good solution so that its distance is controllable.

Next, regarding the other three implement techniques, i.e., the inversion, the item crossover, and the period crossover, which are not used in other algorithms except GA, we can find that the inversion is just a special mutation that changes the solution with two points distance. However crossover(s) may change the original solutions with a longer distance so only partial goodness of previous solution has been remained. In GA algorithm, the implement techniques with long and short distance are combined together, therefore it should be considered more effective for solving the MLLS problem. However, we have to say that those implement techniques used in above algorithms (includes GA) seem not effective in finding good solution. Because even a good solution exists near by the original solution, here is no way to guarantee that it will be found by above algorithms since the candidate solution is randomly generated. It can be considered that not only distance but also range should be used in the *step* function to find a better solution. However, two problems here need to answer. Can distance and range evaluate the effective of implement techniques using in the MLLS problem? How do the implement techniques should be applied?

For proofing our consideration above, simulation experiments were executed by using the general algorithm for solving the MLLS problems shown in Fig.1. Initial solution(s) is produced with a randomized cumulative Wagner and Whitin (RCWW) algorithm. The *step* function uses a GA algorithm, because all kinds of the implemental techniques had been included in the algorithm. We can control what the implemental technique should be used then evaluate the efficiency of the techniques. However the *terminate* functions used in different problem instances are different. In the experiments we first show the relationship among

http://dx.doi.org/10.5772/55279

distance range and searching effective. Then we investigate the performance effective of various implemental techniques. For calculating the distance, set $k \leftarrow 1$, and repeated to find randomly a better solution from $N_k(x)$ and accept it until non better solution could be found (indicated by 100 continuous tries without improvement). And then, search better solutions from next distance ($k \leftarrow k+1$) until $k > k_{max}$. The calculation is repeated for 10 runs in different rages and the averages are used. In the *step* of GA, each of the five operators (single-bit mutation, cumulative mutation, inversion, period crossover and product crossover) are used with a fixed probability 20%, to produce new generation. Initial populations are generated and maintained with a population of 50 solutions. In each generation, new 50 solutions are randomly generated and added into the populations, then genetic operation with the five operators was performed and related total cost has been calculated. In the selection, top 50 solutions starting from lowest total cost are remained as seeds for the next generation and the rest solutions are removed from the populations. We use a stop rule of maximum 50 generations in *terminate* function. The successful results are counted in terms of distance and range (maximum deviation item/period position among changed points). If there is only one point changed, then the range is zero. Finaly, simulation is excuted by using the same banch marks of Dellaert and Jeunet (2000a).

Remark 1: Distance, range and searching effective

Table 2 and 3 show an example of better solutions with different distances in different ranges. Better solution is defined as that the new solutions are better than the incumbent, so we count the number of better solutions and calculate the ratio of different distances. It can be observed from Table 2 that different distances lead to different number of better solutions found. Most of them (94.82%) are found in the nearest neighbourhood with one point distance and the ratio decreases as the distance increasing, which indicates a less probability of finding better solution among those whose distance are long from the incumbent. However, the incumbent solution did be improved by candidate solutions from longer distance. Even so the results with same tendency can be observed from Table 3 in which a limited range has been used. However very different meanings can be observed from Table 3. Firstly, limiting the search range can lead to a more efficiency for searching better solutions, which is represented by the fact that the total number of better solutions found is about 4% more than that of the non-limited search, and it also leads to obtain a solution with a better total cost (comparing with Table 2). Secondly, even the number of better solutions are almost same in distance 1 (changing just one position of the incumbent solution), but the number of better solutions in distance 2 was about three times of that of non-limited search. That is, longer time and less effect were performed in distance 1 if the range is not limited. This observation can lead a considerable result of implementing a faster searching in distance 1 and a faster changing between distances. That is the superiors of limited searching range.

Distance	Better solutions	Ratio
1	3333	94.82%
2	135	3.84%

Distance	Better solutions	Ratio
3	28	0.80%
4	14	0.40%
5	5	0.14%
Total cost=826.23	3515	100.00%

Table 2. A non limited search in distances in small MLLS problems (Parameters: k_{max}=5, $\Delta i = \pm 5$, $\Delta t = \pm 12$)

Distance	Better solutions	Ratio
1	3232	88.57%
2	332	9.10%
3	55	1.51%
4	25	0.69%
5	5	0.14%
Total cost=824.37	3649	100%

Table 3. A limited search in distances in small MLLS problems (Parameters: k_{max}=5, $\Delta i = \pm 1$, $\Delta t = \pm 3$)

That means even the implemental techniques used in algorithms such as SA, PSO, and VNS are seemly different, but the results from them may be totally similar. In addition, GA uses other implement techniques like crossover(s), so it may lead a longer distance and improve the searching performance basically. Moreover, distance and range have a very flexible controllability to produce candidate solutions. It gives a very important observation that a special range which includes some heuristic information (such as period heuristics is effective but level heuristic is not effective, and so on) can improve the performance of implemental technique, therefore they should be used as some new implemental methods into the meta-heuristic algorithms.

Remark 2: effective of various implement techniques

Here five implemental techniques, i.e., single-bit mutation, cumulative mutation, inversion, product crossover, and period crossover, were tested by using GA algorithm. These implement techniques are used in the *step* function of our GA algorithm. The RCWW algorithm is used as the initial method to produce a population of 50 solutions as the first generation. Then, each of the five implemental techniques is selected with equal probability, namely, 20%, to produce a new generation, and each generation keeps only a population of 50 solutions after the selections by total cost function. The algorithm stops after 50 generations have been evolved. A solution is said to be a better solution if its objective cost is better than those of its parent and is counted in terms of changing distance and changing range from its parent. Therefore, simulation experiment is able to show the relationships among distance, range and searching effectives (represented by the number of better solutions found) of each implemental techni-

http://dx.doi.org/10.5772/55279

que. We represent the relationships among distance, range and searching effectives in a three dimensional figure, in which the dimensions are the *distance*, the *range* and the *number of better solutions found*. Then the distribution in the figures can show how the criteria of distance and range segmenting the searching effectives.

Firstly, it can be considered that the single-bit mutation and the inversion are very similar since the candidate solutions are always in distance 1 and range ±0 by single-bit mutation and always in distance 2 and range ±1 by inversion, so the better solutions found are always in distance 1 and range ±0 for the former and distance 2 and range ±1 for the latter, which have been verified in Fig.2 and Fig.3. Comparing Fig.2 and Fig.3, we also can find that the searching effective of reversion is little better than that of single-bit mutation because more better solutions can be found by reversion.

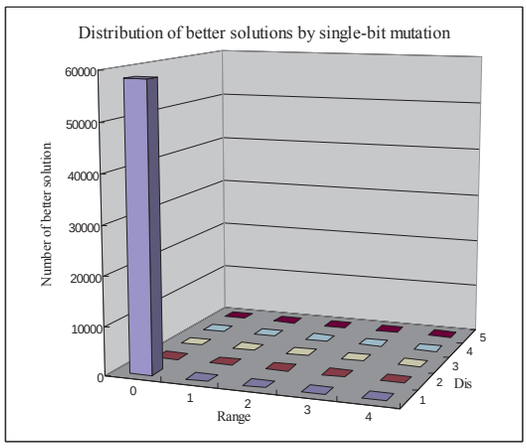

Figure 2. The result of single-bit mutation

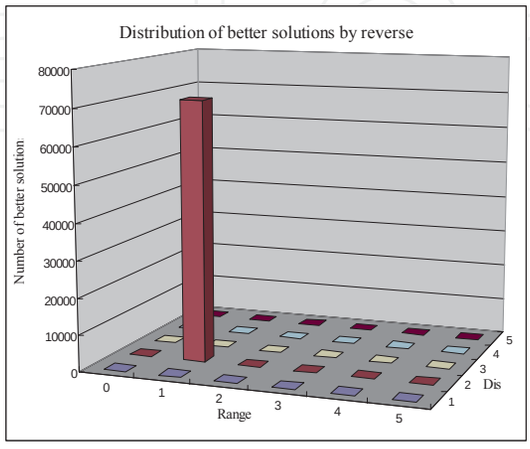

Figure 3. The result of reverse

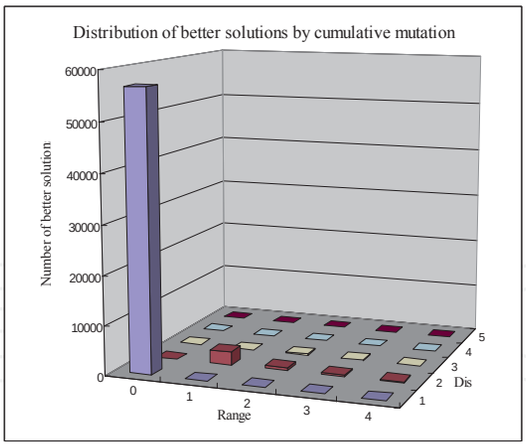

Figure 4. The result of cumulative mutation

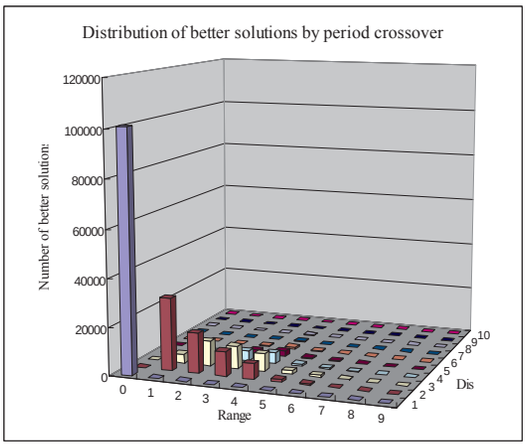

Figure 5. The result of period crossover

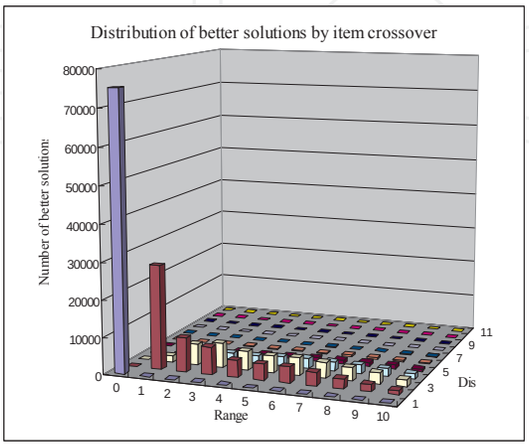

Figure 6. The result of item crossover

http://dx.doi.org/10.5772/55279

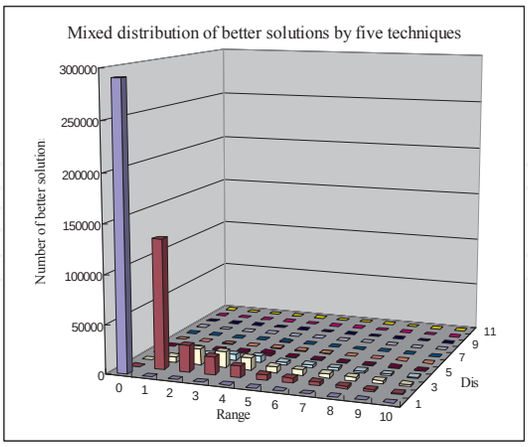

Figure 7. The mixed result of five implemental techniques

Secondly, it seems the cumulative mutation may trigger longer distance and larger range (than the single-bit mutation) to the incumbent solutions, the results of the experiment shown in Fig.4 can illustrate the correctness of this observation. Also it can be observed that only those with small distance and range are more effective in improving the performance of cost function.

Thirdly, the implemental techniques of item crossover and period crossover are more complex than cumulative mutation and reversion, and the offspring consequently may be associated with very large range and very long distance in comparison to their parents. Still, it can be observed from our experimental results in Fig.5 and Fig.6 that only those offspring with small distance and range are more effective on improving the fitness function. Moreover, comparing with simple implemental techniques (single-bit mutation, reversion and cummulative mutation), crossover techniques can achieve significent performance improvements in searching effective.

Finally in Fig.7, we show the mixed distribution of all the five implemental techniques. The results of simulation experiments show that a total signifecent effective of searching better solution can be obtained by using the mixed implemental techniques. Also repeatly, almost achievements of searching effective are in smaller distance and range. This phenominon may be used to conduct a more effective implemental technique in GA algorithms than pure free crossover.

However, for candidate solutions with larger range and longer distance, they still provided improvements on fitness function with probability so should not be ignored by any meta-heuristic algorithms for optimality purpose. Nevertheless, we can develop more effective and more efficient implemental techniques by matching to the ratio of better solution can be found in different ranges and distances. For example, those crossover operations resulting in offspring with too long distance are non-effective and should be avoided as possible.

4. An example: A IVNS algorithm for solving MLLS problems

The VNS/VND algorithm initiated by Mladenovic and Hansen (1997), Hansen and Mladenovic (2001) and Hansen et al. (2001, 2008), is a top-level methodology for solving the combinatorial optimization problems. Because its principle is simple and easily understood and implemented, it has been successfully applied to several optimization problems in different fields. The success of VNS is largely due to its enjoying many of the 11 desirable properties of meta-heuristic generalized by Hansen et al., (2008), such as simplicity, user-friendliness, efficiency, effectiveness, and so on. Since the MLLS problem is observed to share common characteristics, e.g., a binary decision variable, with those problems successfully solved by VNS/VND based algorithm, it is promising to develop an efficient algorithm for this problem (Xiao et al., 2011a, 2011b, 2012). Here an iterated variable neighborhood descent (IVND) algorithm, which is a variant of VNS, is proposed as an example to show the utility and performance improvement of considerations descripted above.

4.1. Initial method

We use a randomized cumulative Wagner and Whitin (RCWW) based approach to initialize the solution for our proposed IVNS algorithm. The RCWW method was introduced by Dellaert and Jeunet(2000a), of which the main idea is based on the fact that lot-sizing a product in the one period will trigger demands for its components in previous periods with leading time corrections(or in same period for the case of zero leading time). Therefore, the real setup cost of an item is in fact greater than its own setup cost and should be modified when using the wellknown sequentiail WW algorithm to generate a initial solution. The time-varying modified setup cost is a improved concept introduced by Dellaert et al(2000b, 2003) and used by Pitakaso et al(2007) which disposes of using a constant modified setup cost for the whole planning horizon; it suggested the costs might vary from one time period to the next, and reported good in getting better initial solutions.

In the IVND algorithm, we use the sequential WW algorithm based on randomly initialized constant modified setup cost to generate initial solutions. For each item i, its modified setup cost $S_i^{'}$ can be calculated recursively by

$$S_i^{'} = S_i + r\left[\sum_{j\in\Gamma_i^{-1}}\left(S_j + \frac{S_j^{'}}{\left|\Gamma_j^{-1}\right|}\right)\right] \tag{8}$$

where S_i is the original setup cost of item i, r is a random value uniformly distributed between [0,1], $|\Gamma_i^{-1}|$ is the set of immediate predecessors(components) of product i and $|\Gamma_i^{-1}|$ is its cardinality.

In addition to the modified setup cost, we also use the modified unit inventory holding cost to construct initial solutions. It is simply based on the consideration that a positive inventory balance of one product in one period causes not only its own inventory holding cost but also

http://dx.doi.org/10.5772/55279

additional inventory holding cost from its predecessors because not all the demands for predecessors are met by timely production; some of them may also be satisfied by inventory. Therefore and similarly, the modified unit inventory holding cost of product i, i.e., $H_i^{'}$ can be calculated recursively by

$$H_i^{'} = H_i + q\left[\sum_{j\in\Gamma_i^{-1}}\left(H_j + \frac{H_j^{'}}{\left|\Gamma_j^{-1}\right|}\right)\right] \tag{9}$$

where H_i is the original unit inventory holding cost of product i and q is a random value uniformly distributed between [0,1].

4.2. Implemental techniques

Here six implemental techniques are used in the IVND algorithm which are integrated together to deliver good performance in the computing experiments.

1. *Limit the changing range of incumbent solution within one item*. Limit the changing range of incumbent solution within one item, i.e., $N_1(x)$, when exploring a neighborhood farther than $N_1(x)$. That is, when multiple mutations are done on incumbent solution, they must occur on same item but different periods.

2. *All mutations are limited between the first period and the last period that have positive demand*. This technique makes the search algorithm to avoid invalid mutations. Nevertheless, the first period with positive demand should be fixed with 1.

3. *No demand, no setup*. Mutation must not try of arranging setups for products in periods without positive demand, which is obviously non-optimal operation and should be banned in the whole searching process.

4. *Triggerrecursive mutations*. A mutation of canceling a setup for a product in a period will trigger recursive mutations on all of its ancestors. While a mutation of canceling a setup occurs, e.g. changing the value of bit x_{it} from 1 to 0, it withdraws demands for the immediate predecessors in previous periods of leading time. As a consequence, some of these predecessors their demands may drop to zero such that their setups (if they have) in these periods should be canceled at the same time; other predecessors who remain non-zero demands due to product commonality should remain unchanged. The recursive mutations are only triggered for cancellation of a setup for production; they will not occur when arranging a setup.

5. *Shift a setup rather than cancel a setup*. When the setup for product i at period t need to be canceled, try to shift the setup to the first period with positive demand behind t, rather than simply cancel it. For example, when x_{it} is to be canceled, find the first period t^* behind t of product i that satisfies $D_{it^*}>0$ and arrange a setup by setting $x_{it^*}\leftarrow 1$ if x_{it^*} is 0. Notably, this arrangement of this setup will also trigger recursive mutations.

6. *Only affected products and periods are recalculated of their inventory holding costs and setup costs.* Different to other evolutionary algorithms like GA and PSO where a group of incumbent solutions have to be maintained, the IVND algorithm has only one incumbent solution. In fact, when a mutation occurs, most area of the solution states including setup matrix Y_{it}, lot-sizing matrix X_{it} and demand matrix D_{it} are remain unchanged. Thus, it just needs to recalculate the affected products and the affected periods of the setup cost and inventory holding cost after a mutation operation. By taking this advantage, the computing efficiency of IVND algorithm can be significantly improved since the recalculation of the objective function--the most time-consuming part of IVND algorithm, are avoided.

The above six implemental techniques are all used in our proposed IVND algorithm to mutate the incumbent solution into its neighborhood. Steps of implementing these techniques on neighborhood search, e.g., neighborhood $N_k(x)$, can be explained by Fig.8.

To generate a candidate solution from neighborhood $N_k(x)$ of the incumbent solution x:

(1)Select randomly k bits of x, e.g., $x_{i,t_1}, x_{i,t_2}, ..., x_{i,t_k}$, and sequentially mutate them. The first three implement techniques mentioned above must follow in the selection: the first implemental technique that the to-be bits must be within an identical item; the second implemental technique that the to-be mutated periods should between the first period and last period with positive demand; the third implemental technique that those periods without demand should not be selected for mutation.

(2)For each mutation from 1 to 0(noticeably not including those from 0 to 1), the forth implemental technique must be followed to trigger recursive mutations toward its predecessors with zero demand. In the recursive mutation process, the fifth implemental technique must be implemented to try of shifting the setup to the first sequential period with positive demand, rather than simply removed it.

(3)Whenever a bit of the incumbent solution is changed, the sixth implemental technique is implemented to recalculate the objective function just by recalculating the affected items and their periods.

Figure 8. The implementation of implemental techniques

Although the new solutions from $N_k(x)$ may has a greater than k unit distance from the incumbent solution x after implemented with these six implemental techniques, it is still considered as a member of $N_k(x)$. These implemental techniques are only deemed as additional actions implemented on the new solution toward better solution. Moreover, benefiting from these actions, higher efficiency of VNS algorithm could be consequently anticipated, which has been confirmed in the experiments of Section 4.

4.3. The IVND algorithm

The algorithm IVND is a variant of the basic VND algorithm. It starts from initiating a solution as the incumbent solution, and then launches a VND search. The VND search repeatedly tries of finding a better solution in the nearby neighborhood of the incumbent solution and moves to the better solution found; if a better solution cannot be found in current neighborhood, then go to explore a father neighborhood until the farthest neighborhood is reached. Once the VND process is stopped(characterized by the farthest neighborhood been explored), another initial solution is randomly generated and restarts the VND search again. This simply iterated search

http://dx.doi.org/10.5772/55279

process loops until the stop condition is met. The stop condition can be a user-specified computing time or a maximum span between two restarts without improvement on the best solution found. In our experiments of the next section, we use the later one, i.e., a fixed times of continuous restarts without improvement, as the stop condition. The basic scheme the proposed IVND algorithm is illustrated in Fig. 9.

Define the set of neighborhood structures $\mathbf{N}_k$, k=1,…,k_{max}, that will be used in the search; choose a stop condition.
Repeat the following step (1), (2) and (3) until the stop condition is met:
(1)Find an initial solution x_0 by using RCWW algorithm
(2)Set $k \leftarrow 1$, $n \leftarrow 0$;
(3)Until $k=k_{max}$ repeat (a), (b), and (c) steps:
  (a)Find at random a solution x' in $N_k(x)$;
  (b)Move or not: if x' is better than x, then $x \leftarrow x'$, $k \leftarrow 1$ and $n \leftarrow 0$, go to step (a) ; otherwise, $n \leftarrow n+1$;
  (c) If $n=N$, then shift to search a farther neighborhood by $k \leftarrow k+1$ and reset $n \leftarrow 0$;
(4)Output the best solution found.

Figure 9. The IVND algorithm for MLLS problem

There are three parameters, i.e., P, N, and K_{max}, in the IVND algorithm for determining the tradeoff between searching efficiency and the quality of final solution. The first parameter P is a positive number which serves as a stop condition indicating the maximum span between two restarts without improvement on best solution found. The second parameter N is the maximum number of explorations between two improvements within a neighborhood. If a better solution cannot be found after N times of explorations in the neighborhood $N_k(x)$, it is then deemed as explored and the algorithm goes to explore a farther neighborhood by $k \leftarrow k+1$. The third parameter K_{max} is the traditional parameter for VND search indicating the farthest neighborhood that the algorithm will go.

5. Computational experiments and the results

5.1. Problem instances

Three sets of MLLS problem instances under different scales(small, medium and large) are used to test the performance of the proposed IVND algorithm. The first set consists of 96 small-sized MLLS problems involving 5-item assembly structure over a 12-period planning horizon, which was developed by Coleman and McKnew (1991) on the basis of work by Veral and LaForge (1985) and Benton and Srivastava (1985), and also used by Dellaert and Jeunet (2000a). In the 96 small-sized problems, four typical product structures with an one-to-one production ratio are considered, and the lead times of all items are zero. For each product structure, four cost combinations are considered, which assign each individual item with different setup costs and different unit holding costs. Six independent demand patterns with variations to reflect low, medium and high demand are considered over a 12-period planning horizon. Therefore, these combinations produce $4 \times 4 \times 6 = 96$ problems for testing. The optimal

solutions of 96 benchmark problem are previously known so that can serve as benchmark for testing against the optimality of the new algorithm.

The second set consists of 40 medium-sized MLLS problems involving 40/50-item product structure over a 12/24-period planning horizon, which are based on the product structures published by Afentakis et al. (1984), Afentakis and Gavish (1986), and Dellaert and Jeunet (2000). In the 40 medium-sized problems, four product structures with an one-to-one production ratio are constructed. Two of them are 50-item assembly structures with 5 and 9 levels, respectively. The other two are 40-item general structure with 8 and 6 levels, respectively. All lead times were set to zero. Two planning horizons were used: 12 and 24 periods. For each product structure and planning horizon, five test problems were generated, such that a total number of $4\times2\times5=40$ problems could be used for testing.

The third set covers the 40 problem instances with a problem size of 500 products and 36/52 periods synthesized by Dellaert and Jeunet (2000). There are 20 different product structures with one-to-one production ratio and different commonality indices[1]. The first 5 instances are pure assembly structures with one end-product. The instances from 6 to 20 are all general structure with five end-products and different communality indices ranges from 1.5 to 2.4. The first 20 instances are all over a 36-period planning horizon; the second 20 instances are of the same product structures of the first 20 instances but over a 52-period planning horizon. The demands are different for each instances and only on end-products.

Since the hybrid GA algorithm developed by Dellaert and Jeunet (2000a) is the first meta-heuristic algorithm for solving the MLLS problem, it was always selected as a benchmark algorithm for comparison with newly proposed algorithm. Therefore, we compared the performance of our IVND algorithm with the hybrid GA algorithm on the all instances of three different scales. We also compared our IVND algorithm with the MMAS algorithm developed by Pitakaso et al.(2007), and the parallel GA algorithm developed by Homberger (2008) since both of them used the same three set of instances used in this paper.

5.2. Test environment

The IVND algorithm under examination were coded in VC++6.0 and ran on a notebook computer equipped with a 1.6G CPU operating under Windows XP system. We fixed the parameter $K_{\max}$ to be 5 for all experiments, and let the parameter P and N changeable to fit for the different size of problem. The symbol '$\mathrm{IVND}_{N}^{P,}$' specifies the IVND algorithm with the parameter P and N, e.g., IVND_{200}^{50} indicates P=50, N=200, and K_{max}=5 by default. The effect of individual parameter on the quality of solution was tested in section 5.6.

5.3. Small-sized MLLS problems

We repeatedly ran IVND_{200}^{50} on the 96 small-sized MLLS problems for 10 times and got 960 results among which 956 were the optimal results so the optimality was 99.58% indicated by

1 Commonality index is the average number of successors of all items in a product structure

http://dx.doi.org/10.5772/55279

the column *Best solutions found(%)*. The column *average time(s)* indicates the average computing time in second of one run for each problem. The average result of 10 and the minimum result of 10 are both shown in Table 4 and compared to the HGA algorithm of Dellaert and Jeunet (2000a), the MMAS algorithm of Pitakaso et al.(2007) and the PGA algorithm of Homberger (2008). It can be observed from Table 4 that $IVND_{200}^{50}$ uses 0.7 second to find 100% optimal solutions of 96 benchmark problems. Although the PGAC and the GA3000 can also find 100% optimal solutions, they take longer computing time and also take the advantage of hardware (for PGAC 30 processors were used to make a parallel calculation). After that, we adjust the parameter *P* from 50 to 200 and repeatedly ran $IVND_{200}^{200}$ on the 96 problems for 10 times again. Surprisingly, we got 960 optimal solutions (100% optimality) with computing time of 0.27 second.

Method	Avg. cost	Best solutions found (%)	Mean dev. if best solution not found	Average time(s)	CPU type	Number of processors	Sources
HGA_{50}	810.74	96..88	0.26	5.14s	--	1	Dellaert et al.(2000)
MMAS	810.79	92.71	0.26	<1s	P4 1.5G	1	Pitakaso et al.(2007)
GA_{100}	811.98	75.00	0.68	5s	P4 2.4G	1	Homberger(2008)
GA_{3000}	**810.67**	**100.00**	**0.00**	**5s**	**P4 2.4G**	**1**	**Homberger(2008)**
PGAI	810.81	94.79	0.28	5s	P4 2.4G	30	Homberger(2008)
PGAC	**810.67**	**100.00**	**0.00**	**5s**	**P4 2.4G**	**30**	**Homberger(2008)**
$IVND_{100}^{50}$ (Avg. of 10)	810.69	99.58	0.02	0.07s	NB 1.6G	1	IVND
$IVND_{100}^{50}$ (Min. Of 10)	**810.67**	**100.00**	**0.00**	**0.7s**	**NB 1.6G**	**1**	**IVND**
$IVND_{200}^{200}$ (Avg. Of 10)	**810.67**	**100.00**	**0.00**	**0.27s**	**NB 1.6G**	**1**	**IVND**
OPT.	**810.67**	**100.00**	**0.00**	**--**	**--**	**-**	**--**

Table 4. Comparing IVND with existing algorithms on 96 small-sized problems

5.4. Medium-sized MLLS problems

Secondly, we ran $IVND_{600}^{100}$ algorithm on 40 medium-sized MLLS benchmark problems and repeated 10 runs. We summarize the 400 results and compare them with the existing algorithms in Table 5. More detailed results of 40 problems are listed in Table 6 where the previous best known solutions summarized by Homberger(2008) are also listed for comparison. After that, we repeatedly ran $IVND_{600}^{100}$ algorithm for several times and updated the best solutions for these 40 medium-sized problems which are listed in column *new best solution* in Table 6.

Method	Avg. cost	Optimality on previous best solutions (%)	Comp. Time(s)	CPU type	Number of processors	Sources
HGA_{250*}	263,931.8	17.50	<60s	--	1	Dellaert et all(2000)
MMAS	263,796.3	22.50	<20s	P4 1.5G	1	Pitakaso et al.(2007)
GA_{100}	271,268.2	0.00	60s	P4 2.4G	1	Homberger(2008)
GA_{3000}	266,019.8	15.00	60s	P4 2.4G	1	Homberger(2008)
PGAI	267,881.4	0.00	60s	P4 2.4G	30	Homberger(2008)
PGAC	263,359.6	65.00	60s	P4 2.4G	30	Homberger(2008)
$IVND_{600}^{100}$(Avg. of 10)	263,528.5	30.00	2.67	NB 1.6G	1	IVND
$IVND_{600}^{100}$ (Min. of 10)	263,398.8	60.00	26.7	NB 1.6G	1	IVND
Prev. best solution	263,199.8	--	--	--	--	--
New best solution	260,678.3	--	--	--	-	--

Table 5. Comparing IVND with existing algorithms on 40 medium-sized problems

It can be observed from Table 5 that the PGAC and $IVND_{600}^{100}$ (Min. of 10) algorithm are among the best and very close to each other. Although the optimality of PGAC (65%) is better than that of $IVND_{600}^{100}$ (Min. of 10) (60%) in terms of the baseline of previous best known solutions, it may drop at least 17% if in terms of new best solutions since 12 of 40 problems had been updated their best known solutions by IVND algorithm(see the column *new best solution* in Table 6) and 7 of the 12 updated problems' previous best known solution were previously obtained by PGAC. Furthermore, by taking account into consideration of hardware advantage of the PGAC algorithm(multiple processors and higher CPU speed), we can say that the IVND algorithm performances at least as best as the PGAC algorithm on medium-sized problems, if not better than.

	Instance				$IVND_{400}$		Prev. best Solution	New best solution	New method
	S	D	I	P	Avg. of 10	Min. of 10			
0	1	1	50	12	196,084	196,084	**194,571**	**194,571**	B&B
1	1	2	50	12	165,682	165,632	**165,110**	**165,110**	B&B
2	1	3	50	12	**201,226**	**201,226**	**201,226**	**201,226**	B&B
3	1	4	50	12	188,010	188,010	**187,790**	**187,790**	B&B
4	1	5	50	12	161,561	161,561	**161,304**	**161,304**	B&B
5	2	1	50	12	**179,761**	**179,761**	**179,762**	**179,761**	B&B

http://dx.doi.org/10.5772/55279

	Instance				$IVND_{400}$		Prev. best Solution	New best solution	New method
	S	D	I	P	Avg. of 10	Min. of 10			
6	2	2	50	12	**155,938**	**155,938**	**155,938**	**155,938**	B&B
7	2	3	50	12	**183,219**	**183,219**	**183,219**	**183,219**	B&B
8	2	4	50	12	136,474	**136,462**	**136,462**	**136,462**	B&B
9	2	5	50	12	186,645	**186,597**	**186,597**	**186,597**	B&B
10	3	1	40	12	**148,004**	**148,004**	**148,004**	**148,004**	PGAC
11	3	2	40	12	197,727	**197,695**	**197,695**	**197,695**	PGAC
12	3	3	40	12	**160,693**	**160,693**	**160,693**	**160,693**	MMAS
13	3	4	40	12	**184,358**	**184,358**	**184,358**	**184,358**	PGAC
14	3	5	40	12	**161,457**	**161,457**	**161,457**	**161,457**	PGAC
15	4	1	40	12	185,507	**185,170**	**185,170**	**185,161**	PGAC→IVND
16	4	2	40	12	**185,542**	**185,542**	**185,542**	**185,542**	PGAC
17	4	3	40	12	192,794	192,794	**192,157**	**192,157**	MMAS
18	4	4	40	12	136,884	**136,757**	**136,764**	**136,757**	PGAC→IVND
19	4	5	40	12	166,180	166,122	**166,041**	**166,041**	PGAC
20	1	6	50	24	344,173	343,855	**343,207**	**343,207**	PGAC
21	1	7	50	24	293,692	293,373	**292,908**	**292,908**	HGA
22	1	8	50	24	356,224	355,823	**355,111**	**355,111**	HGA
23	1	9	50	24	325,701	**325,278**	**325,304**	**325,278**	PGAC
24	1	10	50	24	386,322	**386,059**	**386,082**	**385,954**	HGA→IVND
25	2	6	50	24	341,087	341,033	**340,686**	**340,686**	HGA
26	2	7	50	24	378,876	**378,845**	**378,845**	**378,845**	HGA
27	2	8	50	24	346,615	346,371	**346,563**	**346,358**	HGA→IVND
28	2	9	50	24	413,120	412,511	**411,997**	**411,997**	HGA
29	2	10	50	24	390,385	**390,233**	**390,410**	**390,233**	PGCA→IVND
30	3	6	40	24	**344,970**	**344,970**	**344,970**	**344,970**	HGA
31	3	7	40	24	352,641	**352,634**	**352,634**	**352,634**	PGAC
32	3	8	40	24	356,626	356,456	**356,427**	**356,323**	PGAC→IVND
33	3	9	40	24	411,565	**411,438**	**411,509**	**411,438**	MMAS→IVND
34	3	10	40	24	**401,732**	**401,732**	**401,732**	**401,732**	HGA
35	4	6	40	24	289,935	**289,846**	**289,883**	**289,846**	PGAC→IVND
36	4	7	40	24	339,548	339,299	**337,913**	**337,913**	MMAS
37	4	8	40	24	320,920	320,426	**319,905**	**319,905**	PGCA

Instance					$IVND_{400}$		Prev. best Solution	New best solution	New method
	S	D	I	P	Avg. of 10	Min. of 10			
38	4	9	40	24	367,531	367,326	**366,872**	**366,848**	PGCA→IVND
39	4	10	40	24	305,729	305,363	**305,172**	**305,053**	PGCA→IVND
Average					263,529	263,399	263,199.8	260,677.1	
Avg. computing time					2.67s	26.7s			

Note. Boldface type denotes previous best solution; underline type denotes better solution; Boldface&underline denotes the new best solution.

Table 6. Results of 40 medium-sized problems and the new best solutions

5.5. Large-sized MLLS problems

Next, we ran $IVND^{50}_{1000}$ algorithm on 40 large-sized MLLS benchmark problems and repeated 10 runs. We summarize the 400 results and compare them with the existing algorithms in Table 7, and show detailed results of 40 problems in Table 8.

Method	Avg. Cost	Optimality on prev. best solutions (%)	Time (m)	CPU type	Number of processors	Sources
HGA_{1000*}	40,817,600	10.00		--	1	Dellaert et all(2000)
MMAS	40,371,702	47.50		P4 1.5G	1	Pitakaso et al.(2007)
GA_{100}	41,483,590	0.00	60	P4 2.4G	1	Homberger(2008)
GA_{3000}	--	--	60	P4 2.4G	1	Homberger(2008)
PGAI	41,002,743	0.00	60	P4 2.4G	30	Homberger(2008)
PGAC	39,809,739	52.50	60	P4 2.4G	30	Homberger(2008)
$IVND^{50}_{1000}$(Avg. of 10)	40,051,638	65.00	4.44	NB 1.6G	1	IVND
$IVND^{50}_{1000}$(Min. of 10)	39,869,210	70.00	44.4	NB 1.6G	1	IVND
Prev. best solution	39,792,241	--	--	--	--	**--**
New best solution	39,689,769	**--**	**--**	**--**	**-**	**--**

Table 7. Comparing IVND with existing algorithms on 40 large-sized problems

It can be observed from Table 7 and Table 8 that the IVND algorithm shows its best optimality among all existing algorithms since 70% of these 40 problems were found new best solution by IVND algorithm. The average computing time for each problem used by IVND algorithm was relatively low. However, four problems, i.e., problem 19, 15, 25, and 50, used much long

http://dx.doi.org/10.5772/55279

time to finish their calculation because the interdependencies among items are relatively high for these four problems. The column *Inter D.* in Table 8 is the maximum number of affected items when the lot-size of end-product is changed.

	Instance			$IVND_{1000}$			Prev. best known	New best known	New source
	I	P	Inter D.	Avg. of 10	Min. of 10	Time (m)			
0	500	36	500	597,940	597,560	6.3	**595,792**	**595,792**	PGAC
1	500	36	500	817,615	816,507	6.8	**816,058**	**816,058**	HGA
2	500	36	500	929,097	927,860	6.4	**911,036**	**911,036**	PGAC
3	500	36	500	945,317	944,626	6.2	**942,650**	**942,650**	MMAS
4	500	36	500	1,146,946	**1,145,980**	6.5	**1,149,005**	**1,145,980**	MMAS→IVND
5	500	36	11036	7,725,323	**7,689,434**	71.9	**7,812,794**	**7,689,434**	PGAC→IVND
6	500	36	3547	3,928,108	**3,923,336**	22.5	**4,063,248**	**3,923,336**	MMAS→IVND
7	500	36	1034	2,724,472	2,713,496	17.2	**2,704,332**	**2,703,004**	HGA→IVND
8	500	36	559	1,898,263	1,886,812	10.3	**1,943,809**	**1,865,141**	PGAC→IVND
9	500	36	341	1,511,600	1,505,392	6.0	**1,560,030**	**1,502,371**	MMAS→IVND
10	500	36	193607	59,911,520	**59,842,858**	179.9	**59,866,085**	**59,842,858**	PGAC→IVND
11	500	36	22973	13,498,853	**13,441,692**	58.6	**13,511,901**	**13,441,692**	PGAC→IVND
12	500	36	3247	4,751,464	**4,731,818**	34.4	**4,828,331**	**4,731,818**	PGAC→IVND
13	500	36	914	2,951,232	2,937,914	18.6	**2,910,203**	**2,910,203**	HGA
14	500	36	708	1,759,976	1,750,611	8.9	**1,791,700**	**1,740,397**	MMAS→IVND
15	500	36	1099608	472,625,159	472,088,128	106.1	**471,325,517**	**471,325,517**	PGAC
16	500	36	24234	18,719,243	**18,703,573**	80.5	**18,750,600**	**18,703,573**	MMAS→IVND
17	500	36	7312	7,321,985	**7,292,340**	33.7	**7,602,730**	**7,292,340**	MMAS→IVND
18	500	36	1158	3,592,086	**3,550,994**	23.4	**3,616,968**	**3,550,994**	PGAC→IVND
19	500	36	982	2,326,390	2,293,131	13.8	**2,358,460**	**2,291,093**	MMAS→IVND
20	500	52	500	1,189,599	1,188,210	22.4	**1,187,090**	**1,187,090**	MMAS
21	500	52	500	1,343,567	**1,341,412**	14.9	**1,341,584**	**1,341,412**	HGA→IVND
22	500	52	500	1,403,822	1,402,818	8.7	**1,400,480**	**1,384,263**	MMAS→IVND
23	500	52	500	1,386,667	1,384,263	9.1	**1,382,150**	**1,382,150**	MMAS
24	500	52	500	1,660,879	**1,658,156**	8.8	**1,660,860**	**1,658,156**	MMAS→IVND
25	500	52	11036	12,845,438	12,777,577	117.7	**13,234,362**	**12,776,833**	PGAC→IVND
26	500	52	3547	7,292,728	**7,246,237**	27.7	**7,625,325**	**7,246,237**	PGAC→IVND

	Instance			$IVND_{1000}$			Prev. best known	New best known	New source
	I	P	Inter D.	Avg. of 10	Min. of 10	Time (m)			
27	500	52	1034	4,253,400	4,231,896	21.9	**4,320,868**	**4,199,064**	PGAC→IVND
28	500	52	559	2,905,006	2,889,328	10.7	**2,996,500**	**2,864,526**	MMAS→IVND
29	500	52	341	2,198,534	**2,186,429**	7.8	**2,277,630**	**2,186,429**	MMAS→IVND
30	500	52	193607	103,535,103	103,237,091	297.4	**102,457,238**	**102,457,238**	PGAC
31	500	52	22973	18,160,129	18,104,424	49.4	**18,519,760**	**18,097,215**	PGAC→IVND
32	500	52	3247	6,932,353	**6,905,070**	44.3	**7,361,610**	**6,905,070**	MMAS→IVND
33	500	52	914	4,109,712	4,095,109	41.1	**4,256,361**	**4,080,792**	PGAC→IVND
34	500	52	708	2,602,841	2,573,491	20.5	**2,672,210**	**2,568,339**	MMAS→IVND
35	500	52	1099608	768,067,039	762,331,081	121.4	**756,980,807**	**756,980,807**	PGAC
36	500	52	24234	33,393,240	33,377,419	137.7	**33,524,300**	**33,356,750**	MMAS→IVND
37	500	52	7312	10,506,439	**10,491,324**	52.1	**10,745,900**	**10,491,324**	MMAS→IVND
38	500	52	1158	5,189,651	5,168,547	29.5	**5,198,011**	**5,120,701**	PGAC→IVND
39	500	52	982	3,406,764	3,394,470	14.3	**3,485,360**	**3,381,090**	MMAS→IVND
Average				40,051,638	39,869,210	44.4	39,792,241	39,689,769	--
Avg. computing time				4.44m	44.4m				

Note. Boldface type denotes previous best solution; underline type denotes better solution; Boldface&underline denotes the new best solution.

Table 8. Results of 40 large-sized problems and the new best solutions

5.6. The effectiveness of individual parameter of VIND

Finally, we used the 40 medium-sized problems to test parameters, i.e., P, N and K_{max}, of IVND algorithm their relation between effectiveness and computing load (using medium-sized problems is just for saving computing time). We did three experiments by varying one parameter while fixing other two parameters. First, we fixed N=200 and K_{max}=5, and increased P from 10 to 200, and repeated 10 runs for each P. Second, we fixed P=50 and K_{max}=5, and increase N from 50 to 500. thirdly, P, N were fixed to 50 and 200, and K_{max} was increased from 1 to 10. The average costs gotten by the three experiments against varied parameters are shown in Table 9. A general trend can be observed that increases parameter P, N or K_{max} will all lead to better solutions been found but at the price of more computing time. However, all the parameter may contribute less to the quality of solution when they are increased large enough. Obviously, the best effectiveness-cost combination of these parameters exists for the IVND algorithm which is a worthwhile work to do in future works, but we just set these parameters manually in our experiments.

http://dx.doi.org/10.5772/55279

P	N	K_{max}	Avg. Cost of 10 runs	Comp. time of 10 runs (s)
5	200	5	264,111	30
10	200	5	263,914	57
20	200	5	263,816	104
30	200	5	263,744	144
50	200	5	263,712	233
70	200	5	263,671	308
100	200	5	263,640	433
130	200	5	263,614	553
160	200	5	263,620	674
200	200	5	263,579	832
50	10	5	266,552	40
50	50	5	264,395	87
50	100	5	263,920	135
50	150	5	263,775	186
50	200	5	263,702	220
50	250	5	263,672	274
50	300	5	263,634	309
50	400	5	263,613	376
50	500	5	263,603	463
50	600	5	263,585	559
50	200	1	263,868	74
50	200	2	263,775	100
50	200	3	263,715	136
50	200	4	263,713	178
50	200	5	263,709	225
50	200	6	263,704	276
50	200	7	263,693	341

P	N	K_{max}	Avg. Cost of 10 runs	Comp. time of 10 runs (s)
50	200	8	263,684	420
50	200	9	263,688	527
50	200	10	263,678	567

Table 9. Experimental results of different parameters for medium-sized problem

6. Summarization

The consideration of meta-heuristic is widly used in a lot of fields. Deffirent meta-heuristic algorithms are developed for solving deffirent problems, especially combinational optimization problems. In this chapter, we discussed a special case of MLLS problem. First, the general definition of MLLS problem was described. We shown its solution structure and explained its NP completeness. Second, we reviewed the meta-heuristic algorithms which have been use to solve the MLLS problem and pointed their merits and demerits. Based on the recognition, third, we investigated those implement techniques used in the meta-heuristic algorithms for solving the MLLS problems. And two criteria of distance and range were firstly defined to evaluate the effective of those techniques. We brifly discussed the mechanisms and characteristics of the techniques by using these two criteria, and provided simulation experiments to prove the correctness of the two criteria and to explain the performance and utility of them. This is one of our contributions. Fourth, we presented a succinct and easily implemented IVND algorithm and six implemental techniques for solving the MLLS problem. The IVND algorithm was evaluated by using 176 benchmark problems of different scales (small, medium and large) from literatures. The results on 96 small-sized benchmark problems showed the IVND algorithm of good optimality; it could find 100% optimal solutions in repeated 10 runs using a very low computing time(less than 1s for each problem). Experiments on other two sets of benchmark problems (40 medium-sized problems and 40 large-sized problems) showed it good efficiency and effectiveness on solving MLLS problem with product structure complex. For the medium-sized problems, the IVND can use 10 repeated runs to reach 40% of the 40 problems of their previous known best solutions and find another 20% of new best known solutions. By more repeated runs, our IVND algorithm actually had updated 30% (12 problems) of the best known solutions, and computing efficiency was also very acceptable because the longest computing time for each problem was less than one minute. For the 40 large-sized problems, the IVND algorithm delivered even more exciting results on the quality of solution. Comparison of the best solutions achieved with the new method and those established by previous methods including HGA, MMAS, and PGA shows that the IVND algorithm with the six implemental techniques are till now among the best methods for solving MLLS problem with product structure complexity considered, not only because it is easier to be understood and implemented in practice, but more importantly, it also provides quite good solutions in very acceptable time.

http://dx.doi.org/10.5772/55279

Acknowledgements

This work is supported by the Japan Society for the Promotion of Science (JSPS) under the grant No. 24510192.

Author details

Ikou Kaku[1*], Yiyong Xiao[2] and Yi Han[3]

*Address all correspondence to: kakuikou@tcu.ac.jp

1 Department of Environmental and Information studies, Tokyo City University, Japan

2 School of Reliability and System Engineering, Beihang University, Beijing, China

3 College of Business Administration, Zhejiang University of Technology, Hangzhou, China

References

[1] Afentakis, P, Gavish, B, & Kamarkar, U. (1984). Computationally efficient optimal solutions to the lot-sizing problem in multistage assembly systems, *Management Science*, , 30, 223-239.

[2] Afentakis, P, & Gavish, B. (1986). Optimal lot-sizing algorithms for complex product structures, *Operations Research*, , 34, 237-249.

[3] Almeder, C. (2010). A hybrid optimization approach for multi-level capacitated lot-sizing problems, *European Journal of Operational Research*, , 200, 599-606.

[4] Benton, W. C, & Srivastava, R. (1985). Product structure complexity and multilevel lot sizing using alternative costing policies, *Decision Sciences*, , 16, 357-369.

[5] Blackburn, J. D, & Millen, R. A. (1985). An evaluation of heuristic performance in multi-stage lot-sizing systems, *International Journal of Production Research*, , 23, 857-866.

[6] Coleman, B. J, & Mcknew, M. A. (1991). An improved heuristic for multilevel lot sizing in material requirements planning. *Decision Sciences*, , 22, 136-156.

[7] Crowston, W. B, & Wagner, H. M. (1973). Dynamic lot size models for multi-stage assembly system, *Management Science*, , 20, 14-21.

[8] Dellaert, N, & Jeunet, J. (2000). Solving large unconstrained multilevel lot-sizing problems using a hybrid genetic algorithm, *International Journal of Production Research*, 38(5), 1083-1099.

[9] Dellaert, N, Jeunet, J, & Jonard, N. (2000). A genetic algorithm to solve the general multi-level lot-sizing problem with time-varying costs, *International Journal of Production Economics*, , 68, 241-257.

[10] Han, Y, Tang, J. F, Kaku, I, & Mu, L. F. (2009). Solving incapacitated multilevel lot-sizing problem using a particle swarm optimization with flexible inertial weight, *Computers and Mathematics with Applications*, , 57, 1748-1755.

[11] Han, Y, Kaku, I, Tang, J. F, Dellaert, N, Cai, J. H, Li, Y. L, & Zhou, G. G. (2011). A scatter search approach for uncapacitated multilevel lot-sizing problems, *International Journal of Innovative Computing, Information and Control*, 7(7), 4833-4848.

[12] Han, Y, Cai, J. H, Kaku, I, Lin, H. Z, & Guo, H. D. (2012a). A note on "a genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem", *Applied Mechanics and Materials*, , 127, 527-530.

[13] Han, Y, Cai, J. H, Kaku, I, Li, Y. L, Chen, Y. Z, & Tang, J. F. (2012b). Evolutionary algorithms for solving unconstrained multilevel lot-sizing problem with series structure, *Journal of Shanghai Jiaotong University*, 17(1), 39-44.

[14] Hansen, P, & Mladenovic, N. (2001a). Variable neighborhood search: principles and applications, *European Journal of Operational Research*, 130(3), 449-467.

[15] Hansen, P, Mladenovic, N, & Perez, D. (2001b). Variable neighborhood decomposition search, *Journal of Heuristics*, , 7, 335-350.

[16] Hansen, P, & Mladenovic, N. and Pe´rez J. A. M., (2008). Variable neighborhood search, *European Journal of Operational Research*, Editorial., 191, 593-595.

[17] Homberger, J. (2008). A parallel genetic algorithm for the multilevel unconstrained lot-sizing problem, *INFORMS Journal on Computing*, 20(1), 124-132

[18] Hoos, H. H, & Thomas, S. (2005). Stochastic Local Search-Foundations and Applications, Morgan Kaufmann Publishers.

[19] Kaku, I, & Xu, C. H. (2006). A soft optimization approach for solving a complicated multilevel lot-sizing problem, *in Proc. 8th Conf. Industrial Management, ICIM'200638*

[20] Kaku, I, Li, Z. S, & Xu, C. H. (2010). Solving Large Multilevel Lot-Sizing Problems with a Simple Heuristic Algorithm Based on Segmentation, *International Journal of Innovative Computing, Information and Control*, 6(3), 817-827.

[21] Mladenovic, N, & Hansen, P. (1997). Variable neighborhood search, *Computers & Operations Research*, , 24, 1097-1100.

[22] Pitakaso, R, Almeder, C, Doerner, K. F, & Hartlb, R. F. ant system for unconstrained multi-level lot-sizing problems, *Computer & Operations Research*, , 34, 2533-2552.

[23] Raza, A. S, & Akgunduz, A. (2008). A comparative study of heuristic algorithms on Economic Lot Scheduling Problem, *Computers and Industrial Engineering.* 55(1), 94-109.

[24] Steinberg, E, & Napier, H. A. (1980). Optimal multilevel lot sizing for requirements planning systems, *Management Science*, 26(12), 1258-1271.

[25] Tang, O. (2004). Simulated annealing in lot sizing problems, *International Journal of Production Economics*, , 88, 173-181.

[26] Veral, E. A. and LaForge R. L., (1985). The performance of a simple incremental lot-sizing rule in a multilevel inventory environment, *Decision Sciences*, , 16, 57-72.

[27] Xiao, Y. Y, Kaku, I, Zhao, X. H, & Zhang, R. Q. (2011a). A variable neighborhood search based approach for uncapacitated multilevel lot-sizing problems, *Computers & Industrial Engineering*, , 60, 218-227.

[28] Xiao, Y. Y, Zhao, X. H, Kaku, I, & Zhang, R. Q. (2011b). A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems, *European Journal of Operational Research, 214*, 223-231.

[29] Xiao, Y. Y, Zhao, X. H, Kaku, I, & Zhang, R. Q. (2012). Neighborhood search techniques for solving uncapacitated multilevel lot-sizing problems, *Computers & Operations Research*, 57((3)

[30] Yelle, L. E. (1979). Materials requirements lot sizing: a multilevel approach, *International Journal of Production Research*, , 17, 223-232.

[31] Zhangwill, W. I. (1968). Minimum concave cost flows in certain network, *Management Scienc*e, , 14, 429-450.

[32] Zhangwill, W. I. (1969). A backlogging model and a multi-echelon model of a dynamic economic lot size production system-a network approach, *Management Science*, , 15, 506-527.

IntechOpen