European Journal of Operational Research, Volume 235, Issue 1, 16 May 2014, Pages 102–114

Accepted Manuscript

A Variable Neighborhood Search with an Effective Local Search for Uncapacitated Multilevel Lot-Sizing Problems

Yiyong Xiao, Renqian Zhang, Qiuhong Zhao, Ikou Kaku, Yuchun Xu

PII: S0377-2217(13)00848-5
DOI: http://dx.doi.org/10.1016/j.ejor.2013.10.025
Reference: EOR 11941

To appear in: *European Journal of Operational Research*

Received Date: 4 May 2012
Accepted Date: 8 October 2013

Please cite this article as: Xiao, Y., Zhang, R., Zhao, Q., Kaku, I., Xu, Y., A Variable Neighborhood Search with an Effective Local Search for Uncapacitated Multilevel Lot-Sizing Problems, *European Journal of Operational Research* (2013), doi: http://dx.doi.org/10.1016/j.ejor.2013.10.025

A Variable Neighborhood Search with an Effective Local Search for Uncapacitated Multilevel Lot-Sizing Problems

Yiyong Xiao[1], Renqian Zhang[2], Qiuhong Zhao[3,*], Ikou Kaku[4], Yuchun Xu[5]

[1]School of Reliability and Systems Engineering, Beihang University, Beijing, 100191, China
E-mail: xiaoyiyong@buaa.edu.cn
[2]School of Economics and Management, Beihang University, Beijing, 100191, China
E-mail: zhangrenqian@buaa.edu.cn
[3]School of Economics and Management, Beihang University, Beijing, 100191, China
E-mail: qhzhao@buaa.edu.cn
[4]Faculty of Environmental and Information studies, Tokyo City University, Tokyo Japan
E-mail: kakuikou@tcu.ac.jp
[5]School of Applied Sciences, Cranfield University, Cranfield, Bedford, MK43 0AL, UK
E-mail: yuchun.xu@cranfield.ac.uk

Abstract: In this study, we improved the variable neighborhood search (VNS) algorithm for solving uncapacitated multilevel lot-sizing (MLLS) problems. The improvement is two-fold. First, we developed an effective local search method known as the Ancestors Depth-first Traversal Search (ADTS), which can be embedded in the VNS to significantly improve the solution quality. Second, we proposed a common and efficient approach for the rapid calculation of the cost change for the VNS and other generate-and-test algorithms. The new VNS algorithm was tested against 176 benchmark problems of different scales (small, medium, and large). The experimental results show that the new VNS algorithm outperforms all of the existing algorithms in the literature for solving uncapacitated MLLS problems because it was able to find all optimal solutions (100%) for 96 small-sized problems and new best-known solutions for 5 of 40 medium-sized problems and for 30 of 40 large-sized problems.

Key Words: Metaheuristics; Multilevel lot-sizing (MLLS) problem; ADTS local search; Variable Neighborhood Search (VNS)

1. Introduction

The multilevel lot-sizing (MLLS) problem addresses how to best determine the trade-off cost in a production system with the purpose of satisfying customer demand with a minimum total cost. The MLLS problem plays an important role in modern production systems of manufacturing and assembly firms. Many planning systems, such as Material Request Planning (MRP) and Master Product Scheduling (MPS), depend heavily on the basic mathematical model and solution approaches for the MLLS problem. Nevertheless, the MLLS problem was proven to be strongly NP-hard (Arkin *et al*., 1989). Optimal solutions to large-sized problems with complex product structures are notably difficult to find. In addition, optimal algorithms exist only for small-sized MLLS problems, and these algorithms include dynamic programming formulations (Zangwill, 1968, 1969), an assembly-structure-based method (Crowston and Wagner, 1973), and branch-and-bound algorithms (Afentakis *et a*l., 1984; Afentakis and Gavish, 1986). Many heuristic approaches have been developed to solve the MLLS problem and its variants with near-optimal solutions. Early studies first applied sequential applications of single-level lot-sizing models to each component of the product structure (Yelle, 1979; Veral and LaForge, 1985),

Correspoding author.Address: 37 Xueyuan Road, Haidian District. Beijing 100191, China. Tel.: +86 10 82316181; fax: +86 10 82328037. *E-mail address:* qhzhao@buaa.edu.cn (Q.H. Zhao)

and later studies used an approximate application of multilevel lot-sizing models (Blackburn and Millen, 1982, 1985; Coleman and McKnew, 1991).

The uncapacitated MLLS acts as a fundamental problem, and its solution approach could be highly meaningful to many of its extended versions, including the capacitated MLLS, the MLLS with time-windowing, and the MLLS with order acceptance. In practice, many SME firms in China's electromechanical industry are more willing to adopt dynamic capability policies because they can improve their capacities during busy seasons with many methods, such as extra working-time, temporal employment, and rented machines. Therefore, the uncapacitated MLLS model caters to the situations of their ERP systems.

Over the past decade, several metaheuristic algorithms have been developed to solve uncapacitated MLLS problems with complex product structures. It has been reported that these algorithms are capable of providing highly cost-efficient solutions with a reasonable computing load. Dellaert and Jeunet (2000) and Dellaert *et al*. (2000) first presented a hybrid genetic algorithm (HGA) for solving uncapacitated MLLS problems with a general product structure and introduced a competitive strategy for mixing the use of five operators in the evolution of the chromosomes from one generation to the next. Homberger (2008) presented a parallel genetic algorithm (PGA) and an empirical policy for deme migration (rate, interval, and selection) for the MLLS problem. These researchers used the power of parallel calculations to decentralize the large calculation load over multiple processors (30 processors were used in their experiments). In addition to genetic algorithms, other metaheuristic algorithms, such as simulated annealing (SA) algorithms (Tang, 2004; Homberger, 2010), the particle swarm optimization (PSO) algorithm by Han *et al.* (2009), the MAX-MIN ant colony optimization (ACO) system by Pitakaso *et al.* (2006, 2007), and the soft optimization approach (SOA) based on segmentation by Kaku *et al*. (2006, 2010), have been developed for solving uncapacitated MLLS problems.

The variable neighborhood search (VNS) algorithm initiated by Mladenovic and Hansen (1997) is a top-level methodology for solving optimization problems. Because its principle is simple and easy to understand and implement, various VNS-based algorithms have been successfully applied to many optimization problems (Hansen et al., 2008a, 2008b, 2010). Mladenovic et al. (2012a) presented a new schema of the general variable neighborhood search (GVNS), which is an extended version of the basic VNS that considers multiple neighborhood structures. Labadie et al (2012) proposed a VNS procedure based on the idea of exploring, most of the time, granular instead of complete neighborhoods in order to improve the algorithms efficiency without loosing effectives. A brief summarization of recent successful VNS applications can be found in Mladenovic et al. (2012b).

Xiao et al. (2011a) first developed a VNS-based algorithm for basic schema and a *shift rule* to solve small- and medium-sized MLLS problems; this algorithm performed better than the HGA in small- and medium-sized problems. Xiao et al. (2011b) developed a reduced VNS (RVNS) combined with six SHAKING techniques to solve large-sized MLLS problems. The term "reduced" indicates a simplified version of the classical VNS algorithm because the *local search* (the most time-consuming component of VNS) was removed from the basic scheme. Although RVNS is still a generate-and-test algorithm, it differs significantly from the single-point stochastic search (SPSS) algorithm (Jeunet and Jonard, 2005) because it uses a systematic method to change multiple bits (with a maximum K_{max}) in the incumbent to generate a candidate, whereas the latter changes only a single bit. In the study conducted by Xiao et al. (2012), three indices (i.e., the *distance*, *changing range*, and *changing level*) were proposed for a neighborhood search based on which three hypotheses were verified and can be used as common rules to enhance the performance of any existing generate-and-test algorithm. Using these three hypotheses, the proposed iterated neighborhood search (INS) algorithm delivered notably good performance when tested against 176 benchmark problems.

In our previous research, the neighborhood structure is defined under a distance-based metric that measures the distance (of two solutions) using the number of different bits, and another type of neighborhood structure based on problem decomposition was also studied in the recent literature (Helber and Sahling, 2010; Lang & Shen, 2011; Seeanner et al., 2013). Several decomposition methods (i.e., product-oriented, time-oriented, and resource-oriented) combined with fix-and-optimized (or partial optimization) strategies were adapted to decompose the original problem into multiple sub-problems in order to restrain the optimization to a smaller area of the binary variables.

In this paper, we developed an effective local search procedure known as the Ancestors Depth-first Traversal Search (ADTS) for the RVNS algorithm such that the RVNS algorithm can be restored to a standard VNS. Although the ADTS procedure adds a considerable amount of computing load to the algorithm, we successfully developed an efficient method (known as trigger) using a new formulation of MLLS problems to rapidly calculate the change in the objective cost during the neighborhood search process. Thus, the new VNS algorithm is both effective and efficient in solving MLLS problems with high-quality solutions and within an acceptable computing time.

The remainder of this paper is organized as follows. In Section 2, we describe the new formulation of the MLLS problem. In Section 3, we detail a local search procedure known as ADTS which was added to our previously presented RVNS algorithm for effectively solving the MLLS problem. Section 4 outlines a highly efficient approach for rapidly calculating the cost variation of the objective function as the incumbent solution changes. In Section 5, we test the proposed algorithm on 176 benchmark MLLS problem instances of different scales (small, medium, and large) and compare its performance with that of existing methods. Finally, Section 6 presents our concluding remarks.

2. Problem formulation and neighborhood definition

The MLLS problem under investigation is considered an uncapacitated, discrete-time, multilevel production/inventory system with a general product structure[1] and multiple-end items. We assume that external demands for the end items are known throughout the planning horizon and that backlog is not allowed. Below, we present the notations used to model the MLLS problem, which also can be found in the reports by Dellaert and Jeunet (2000) and Xiao et al. (2012).

- i: Index of items, $i = 1, 2, \ldots, m$
- t (and 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
- d_{it}: External demand for item i in period t
- D_{it}: Total demand for item i in period t
- C_{ij}: Quantity of item i required to produce one unit of item j
- Γ_i: The set of immediate successors of item i
- Γ_i^{-1}: The set of immediate predecessors of item i
- l_i: The lead time required to assemble, manufacture, or purchase item i

The decision problem focuses on how to set the production setup for all of the items in all of the planning periods such that the decision variable is an $m \times n$ matrix denoted as follows:

- Y_{it}: Binary decision variable addressed to capture the setup cost for item i in period t.

Depending on the decision variable, two other important variables are addressed to quickly capture the inventory holding costs. These can be introduced as follows:

- P_{it}: The period in which the demands of item i in period t will be set for production,
- X_{it}: The production quantity for item i in period t.

The objective function is to minimize the sum of the setup cost and the inventory holding cost for all of the items over the entire planning horizon and is denoted by TC (total cost). We extend the formulation described by Xiao et al. (2012) to cover the external demand for non-end items. Thus, the uncapacitated MLLS problem can be modeled as follows:

[1]In a pure assembly structure, each item has multiple immediate predecessors but at most only one direct successor; in a general structure, each item can have multiple immediate predecessors and multiple direct successors.

$$\text{Min.} \quad TC(Y) = \sum_{i=1}^{m} \sum_{t=1}^{n} \left[S_i \cdot Y_{it} + H_i \cdot D_{it} \cdot (t - P_{it}) \right], \tag{1}$$

$$\text{s.t.} \quad Y_{it} \in \{0,1\}, \forall i,t, \tag{2}$$

$$P_{it} = \max\left\{ t' \cdot Y_{it'} \,\middle|\, 1 \le t' \le t \right\}, \quad \forall i,t, \tag{3}$$

$$X_{it} = \sum\nolimits_{P_{it'}=t} D_{it'}, \quad \forall i,t, \tag{4}$$

$$D_{it} = d_{it} + \sum\nolimits_{j \in \Gamma_i} C_{ij} \cdot X_{j,t+l_j}, \quad \forall i,t, \tag{5}$$

$$X_{it} \ge 0, X_{it} \cdot (1 - Y_{it}) = 0, 1 \le P_{it} \le n, \quad \forall i,t. \tag{6}$$

In the above equations, Eq. (1) indicates that every item in every period will incur a setup cost, an inventory holding cost, or nothing if a zero demand is associated with it. Please note that it is no longer necessary to calculate the inventory levels of the items in the new model, which saves a substantial amount of computational effort. Eq. (2) states that the decision variables for the production setup Y_{it} are binary, and Eq. (3) states that the demands of item i in period t are always set for the production in its closest setup period (indicated by $Y_{it'}$=1) prior to period t (indicated by $t' \le t$). Apparently, the relationship $1 \le P_{it} \le t$ holds such that backlog is not allowed. Eq. (4) states that the total demand for item i in period t includes both its external demand and the sum of the lot sizes of its direct successors (items in Γ_i) multiplied by the production ratio with a lead time correction. Eq. (5) guarantees that the lot size of item i in period t will cover all of the latter periods of the demands that have been set for production in period t (indicated by $P_{it'} = t$). Eq. (6) guarantees that the lot size is non-negative and can only be positive in the periods in which production has been set up, which is useful for solving the mathematical programming model with certain commercial solvers. Fig. A1 in the Appendix section shows an example of solving this programming model using IBM ILOG CPLEX 12.2.

Based on the newly formulated model, we obtain selected properties that can be used to develop efficient heuristic rules that may aid the algorithm in finding better solutions and enhance the search efficiency.

Property 1 Feasibility validation:

$$Y_{i,P_{it}} \cdot D_{it} > 0, \ if \ D_{it} > 0 \quad \forall i,t. \tag{7}$$

This property can be read as *all positive demands have been set up for production in their corresponding periods.*

Property 2 Cost uniqueness:

$$Y_{it} \cdot (t - P_{it}) = 0, \ \forall i,t, \tag{8}$$

which indicates that the setup cost and the inventory holding cost cannot synchronously occur in the same period of time for an item.

Property 3. Backlog forbiddance:

$$1 \le P_{it} \le t, \ \forall i,t, \tag{9}$$

which indicates that all of the demands must be set for production in the period prior to or equal to the demand period. P_{it} cannot be zero or negative.

Property 4. *No demand no setup*: The optimal solution of an MLLS problem satisfies

$$Y_{it} = 0, \ if \ D_{it} = 0, \forall i,t. \tag{10}$$

Proof: Suppose that Y is the optimal solution of an MLLS problem and that there is a bit $Y_{it}\in Y$ that violates this property by Y_{it}=1 and D_{it}=0. If we can change Y into Y' by setting $Y_{it}\leftarrow 0$ and $Y_{i,t+1}\leftarrow 1$, we can state that $TC(Y')$ is lower than $TC(Y)$ because the production setup cost does not produce an increase but the inventory holding cost has been decreased by $\Delta h = H_i \cdot X_{it} - \sum_{j\in\Gamma_i^-} H_j \cdot X_{it} \cdot C_{ji}$. Because the unit inventory holding cost of an item is always greater than or equal to the sum of the inventory holding cost of all of its immediate components, we can state that $\Delta h \geq 0$. Thus, the bit $Y_{it}\in Y$ that violates this property does not exist.

Property 5. *Inner corner* property for assembly structure (Tang, 2004):

$$Y_{it} \geq Y_{j,t'-L_j}, \ \forall i, t, and \ j \in \Gamma_i^- . \tag{11}$$

Proof: For an assembly product structure without an external demand of non-end items, the demand of non-end items stems only from the lot size of its unique successor multiplied by the production ratio with a lead-time correction. In other words, if an item i is not set up for production in period t (i.e., Y_{it}=0), then all of its immediate and non-immediate predecessors will have zero demand in their corresponding periods with a lead-time correction. According to Property 4, the production setup in these periods must be zero (it cannot be 1). Thus, Property 5 holds, and if Y_{it}=1, Property 5 holds straightforwardly.

Property 6. Lot size relationship: The relationships below hold for a feasible solution of an MLLS problem:

$$\sum_{t=1}^{n} \left(X_{it} \cdot Y_{it} \right) = \sum_{t=1}^{n} D_{it}, \ \forall i , \tag{12}$$

$$\sum_{t'=1}^{t} \left(X_{it'} \cdot Y_{it'} \right) \geq \sum_{t'=1}^{t} D_{it'}, \ \forall i, t , \tag{13}$$

$$Y_{it} \cdot \sum_{t'=t}^{n} \left(X_{it'} \cdot Y_{it'} \right) = Y_{it} \cdot \sum_{t'=t}^{n} D_{it'}, \ \forall i, t . \tag{14}$$

3. VNS algorithm with an effective local search for MLLS problems

3.1 The mechanism of VNS for MLLS problems

The VNS is a top-level heuristic strategy that can be used to conduct an efficient stochastic search for better solutions using predefined and systematic changes of neighborhoods. This approach implies a general principle of *from near to far* for exploration of the neighborhoods of the incumbent solution. The underlying basis for this principle is that, in most cases, there is always a higher probability of finding better solutions in nearby neighborhoods rather than farther ones. Therefore, the definition of the neighborhood structures will be highly important for the implementation of the VNS strategy to solve an optimization problem. Xiao et al. (2011a) first introduced a distance-based definition of neighborhood structures for the MLLS problem and verified this approach with experiments in later study (Xiao et al., 2012), which showed that the *distance* and the other two indices (*changing range* and *changing level*) are highly effective factors for conducting a variable neighborhood search. Below, we introduce the definition of distance-based neighborhood structures for the MLLS problem according to Xiao et al. (2011a).

Definition 1. The *distance metric*: For a set of feasible solutions of an MLLS problem (i.e., $\{Y\}$), the distance between any two solutions in $\{Y\}$ (e.g., Y and Y') can be measured by the following:

$$\rho(Y, Y') = | Y \setminus Y' | = | Y' \setminus Y |, \qquad \forall Y, Y' \in \{Y\} ,$$

where $|\cdot\backslash\cdot|$ denotes the number of different bits between Y and Y', e.g., $|Y \backslash Y'| = \sum_{i=1}^{m}\sum_{t=1}^{n}\left|Y_{it} - Y'_{it}\right|$.

Next, based on Definition 1, the neighborhood structure of the incumbent solution can be defined as follows:

Definition 2. The *neighborhood structures*: A solution Y' belongs to the k^{th}-neighborhood of the incumbent solution Y (i.e., $N_k(Y)$) if and only if it satisfies

$$\rho(Y, Y') = k ,$$

which can be simply expressed as $Y' \in N_k(Y) \Leftrightarrow \rho(Y, Y') = k$, where k is a positive integer.

The application of the basic VNS scheme to solve the MLLS problem consists of two general phases. The first phase includes defining a neighborhood structure, selecting an exploring sequence, initiating a solution Y_0 as the incumbent ($Y \leftarrow Y_0$), and defining a stopping condition. The second phase involves a neighborhood search loop in which better solutions in the targeted neighborhood (from near to far) are repeatedly explored and accepted. In this loop process, the nearest neighborhood $N_1(Y)$ is explored first ($k\leftarrow 1$); if no better solution can be found in neighborhood $N_k(Y)$, then the algorithm shifts to explore a farther neighborhood by setting $k\leftarrow k+1$ until $k = K_{\max}$. While searching for a better solution, a shaking procedure is used to randomly generate a candidate Y', and a local search procedure is applied to look for the best solution Y'' near Y'. If Y'' is better than Y, then Y is moved to Y'' by $Y\leftarrow Y''$ and the algorithm continues to search $N_k(Y)$ after setting $k\leftarrow 1$; otherwise, we set $k\leftarrow k+1$. In Fig. 1, we present the basic scheme of the VNS algorithm introduced by Hansen and Mladenovic (2001).

Place Figure 1 approximately here.

3.2 An effective local search method

Previously, Xiao et al. (2011b) proposed a reduced variable neighborhood search (RVNS) algorithm for solving the uncapacitated multi-level lot-sizing problem. The term “reduced” means that the local search component of the basic VNS scheme was removed to increase the computational efficiency. However, in this work, we suggest an effective local search and add it to the RVNS algorithm because the local search method enables a significant improvement in the search for higher-quality solutions with an acceptable increase in the computational load.

The local search method is known as the Ancestors Depth-first Traversal Search (ADTS). After *shaking* out a new candidate Y' in the neighborhood $N_k(Y)$ of the incumbent Y, we use the ADTS method to find the best solution Y'' near Y' and perform a comparison between Y and Y'' (instead of Y' in RVNS) to facilitate the “move or not” decision accordingly. In this process, the incumbent Y will tentatively move to Y' and use the ADTS to find a Y''. If Y'' is better than Y', then the algorithm moves to Y''; otherwise, it returns to Y. When Y moves to Y', a total of k bits of Y must be reversed. Once one bit (i.e., Y_{it}) is reversed, the ADTS procedure is launched to seek the largest cost reduction by setting (one at a time) each of the immediate and non-immediate predecessors of item i with the same setup value in the corresponding periods of t (with a lead-time correction if the lead time is not zero). In other words, after setting $Y_{it}\leftarrow 1\text{-}Y_{it}$, we use a depth-first traversal recursion to set all of the setup values of the ancestors of item i to $1\text{-}Y_{it}$ and record the path that leads to the largest cost reduction. The recorded path is exactly the method used to change Y' to Y'', which is the best solution found from a local search around Y'.

In Fig. 2, we present a simple MLLS example involving a six-item product structure over a six-period planning horizon to illustrate the principle of ADTS. Fig. 2 (a) describes a general product structure, and Fig. 2(b) and Fig. 2 (c) present two solutions with zero lead time and one period of lead-time, respectively. Because the solution moves into its k^{th} neighborhood, it must reverse the values of the k bits that are selected randomly by the shaking procedure. The ADTS procedure will follow each change in these bits. For example, in Fig. 2(b), once bit Y_{33} is reversed from 1 to 0, the ADTS procedure acts to recursively change all of its ancestors to zero and to restore the original value through the path $Y_{43}(1\rightarrow0)\rightarrow Y_{63}$(unchanged)$\rightarrow Y_{43}(0\rightarrow1)\rightarrow Y_{53}(1\rightarrow0)\rightarrow Y_{53}(0\rightarrow1)$. In Fig. 2(c), once bit Y_{24} is reversed to 1, the ADTS procedure changes and restores the bits through the path $Y_{43}(0\rightarrow1)\rightarrow Y_{62}(0\rightarrow1)\rightarrow Y_{62}(1\rightarrow0)\rightarrow Y_{43}(1\rightarrow0)$ using the lead-time of 1 to correct the corresponding periods. Throughout the entire process, the largest cost reduction and its corresponding path will be recorded, and all of the changes that the ADTS has enacted to Y' will be restored in the end. Finally, if the largest cost reduction is positive such that the best solution Y'' found in the local search is better than Y', then the procedure moves Y' to Y'' through the recorded path. The formal description of the ADTS procedure is detailed in Fig. 3.

Place Figure 2 approximately here.

Place Figure 3 approximately here.

The concept of ADTS arose from the new formulation of the mathematical model for the uncapacitated MLLS problem. According to Eqs. (1)-(5), when an item i changes its production setup value in period t (i.e., $Y_{it}\leftarrow1\text{-}Y_{it}$), this operation will only affect the setup and the inventory holding cost of item i and its ancestors in the affected periods. Most of the items and periods remain in the unaffected zone. Therefore, a local search that focuses only on the affected items and related periods will be effective for searching for the best solution close to Y'. Our computational experiment in the latter section confirmed this conjecture and verified the effectiveness of the ADTS method.

3.3 VNS algorithm with the ADTS local search

We now describe the framework of the VNS algorithm with the ADTS local search procedure, as shown in Fig 4. This framework is based on the RVNS algorithm introduced by Xiao et al. (2011b) and was combined with the ADTS local search procedure proposed in this study. The ideas introduced by Xiao et al. (2012) to restrain the *changing ranges* and *changing level* in the shaking procedure were also adopted in this algorithm. This approach represents the latest version of the neighborhood search algorithm that we developed for the MLLS problem, and we have verified its effectiveness and efficiency through the computational experiments reported in Section 5. In Table 1, we list the interpretations of the parameters and variables that appear in the VNS algorithm.

Place Figure 4 approximately here.

Place Table 1 approximately here.

The initial method is a modified RCWW (randomized cumulative Wagner and Whitin) algorithm that was first introduced by Dellaert and Jeunet (2000) and represents a sequential use of the Wagner and Whitin (WW) algorithm from an end item to the raw materials with a randomized cumulative production setup cost. The detailed WW algorithm can be found in the study published by Wagner and Whitin (1958; republished in 2004). Xiao et al. (2011b) extended the RCWW algorithm using both the modified setup cost and the modified inventory holding cost, which are also adopted in the new VNS algorithm of this paper.

The shaking procedure in Step 7 (a) randomly produces a candidate Y from the incumbent Y', and the constraints on *distance*, *changing range of item/period*, and *changing level* are controlled by the input parameters $K_{\max}$, Δt , and α , respectively. For additional details on these constraints, please see the manuscript by Xiao et al. (2012). Other techniques, i.e., *limiting the changed bits between the first period and the last period that have positive demand* described by Dellaert and Jeunet (2000), *trigger recursive changes* described by Dellaert and Jeunet (2000) and Jeunet and Jonard (2005), *setup shifting rule* described by Xiao et al. (2011a), and the *no demand no setup* principle described by Xiao et al. (2011b), are included in the shaking procedure. All of these techniques will help improve the quality of the solution or the efficiency of the algorithm.

The stop condition in Step 9 can be either a maximum span between two consecutive improvements or a fixed computing time elapsed. The first stop condition, i.e., a maximum improvement span $P_{\max}$, can be used for a small-sized problem because finding the optimal solution requires only a few (or even less than 1) seconds in this case. For medium/large-sized problems, the second stop condition, i.e., a fixed computing time, is suggested because different problems may require considerably different computing times if using the first stop condition. When *a fixed computing time* is used as the stop condition, the maximum improvement span (i.e., max(P)) can be treated as an evaluation index that indicates the extent to which the current solution has been optimized. A small value of max(P) indicates that the current solution still has much potential for optimization, whereas a large value indicates that the current result is an already optimized solution of the algorithm obtained with great effort.

4. An efficient method for quickly calculating the cost variation

Although the ADTS local search is quite effective in searching for better solutions, it is also a notably time-consuming procedure because it must recalculate the objective cost of the incumbent solution at each step of the recursive path (in theory). To improve the computational load, we propose an efficient method to calculate the cost variation of a solution change using the advantages of the new formulation for the MLLS problem shown in Eq. (1)-Eq. (5). The new approach is a recursive accumulation of the cost reduction (or variation) (RACR in short) for recalculation of the objective function when the incumbent solution is changed. The RACR is highly efficient because it directly calculates the cost reduction from the changed point and the lot size in related periods.

It can be observed from Eq. (1) that the setup cost is determined only by the decision variable Y, and the inventory holding cost is determined only by the dependent variables P and D. When a bit in Y (i.e., $Y_{it} \in Y$) is changed, the objective cost will increase by one unit of the setup cost of item i if Y_{it} moves from 0 to 1 or decrease by one unit if Y_{it} moves from 1 to 0. This change also triggers subsequent changes in P_{it} and X_{it} according to Eq. (3) and Eq. (4), respectively. Therefore, the inventory holding cost for item i will be influenced: it will increase if Y_{it} moves from 1 to 0 or decrease if Y_{it} moves from 0 to 1. Subsequently, the varying lot size of item i (i.e., X_{it}) continues to recursively influence the demands for its immediate and non-immediate predecessors (components) in the corresponding periods (with a lead-time correction if L_i is non-zero). This scenario also causes an inventory holding cost variation. Note that all of these changes are occurring only in the area related to the ancestors of item i (not to all of the items) during a few periods (not in all of the periods). Therefore, the calculation of the cost reduction that results from the related area is much faster than the complete recalculation of the objective cost for all of the items throughout all of the periods. This insight helps us develop an efficient method for calculating the cost reduction in the neighborhood search and may also be applicable for any other generate-and-test algorithms to enhance their computing efficiencies. In Fig. 5, we provide the detailed steps of the RACV procedure when the incumbent Y moves to Y'.

Place Figure 5 approximately here.

We note that the efficiency of the algorithms shown in Fig. 3 and Fig. 5 can be reciprocally related to the complexity of the product structure because the algorithm must call the recursive function to traverse the entire ancestor tree of the changed item. Therefore, the number of ancestor tree nodes can be used to measure the complexity of the product structure, which will considerably influence the computing efficiency. We use the notation T_i to denote the number of ancestor tree nodes of item i and accordingly define a complexity index (CI) of the product structure using the average number of ancestor tree nodes for all of the items as follows:

$$CI = \frac{1}{n}\sum_{i=1}^{m} T_i \tag{15}$$

5. Computational experiments

5.1 Problem instances and test environment

Three classes of uncapacitated MLLS problems of different scales (small, medium, and large) were used to test the performance of the new VNS algorithm with the ADTS local search. *Class 1* consists of 96 small-sized MLLS problems involving a five-item assembly structure over a 12-period planning horizon, as developed by Coleman and McKnew (1991) based on the work of Veral and LaForge (1985) and Benton and Srivastava (1985). *Class 2* consists of 40 medium-sized MLLS problems involving 40/50-item product structures over a 12/24-period planning horizon, based on the product structures proposed by Afentakis *et al.* (1984), Afentakis and Gavish (1986), and Dellaert and Jeunet (2000). *Class 3* covers 40 large-sized problems with a size of 500 items over 36/52 periods, as synthesized by Dellaert and Jeunet (2000) and used as a benchmark by Pitakaso *et al.* (2007), Homberger (2008, 2010), and Xiao et al. (2011b, 2012). Thus, a total of 176 problem instances will be used to test the VNS algorithm. For the 136 problems classified as *Class 1* and *Class 2*, the lead times of the items were all set to 0, whereas the lead times for all of the items in the 40 large-sized problems were set to 1.

The VNS algorithm was coded in VC++6.0 and run on a PC computer equipped with an Intel(R) Core(TM) 3i-2100 3.1G CPU operating under a Windows 7 system. We used the maximum improvement span as the stop condition for the small-sized problems in *Class 1* and a fixed computing time as the stop condition for all of the medium and large-sized problems in *Class 2* and *Class 3*. We compared the performances of the VNS with those of six other existing algorithms in the literature: the hybrid genetic algorithm (HGA) developed by Dellaert and Jeunet (2000), the MAX-MIN ant system (MMAS) developed by Pitakaso *et al.*(2007), the parallel genetic algorithm (PGA) developed by Homberger (2008), the ant colony algorithm (SACO) developed by Homberger and Gehring (2009), the RVNS developed by Xiao et al. (2011b), and the ISN algorithm developed by Xiao et al. (2012).

Interested readers can download all of our experimental data, including the benchmark problems, experimental outcomes, best solution found, executable program, and source code in VC ++6.0, from the following website: http://semen.buaa.edu.cn/Teacher/Templet/DefaultDownLoadAttach.aspx?cwid=1224.

5.2 Computational experiments

First, we ran the VNS algorithm once for the *Class 1* problems using the control parameters P_{max}=100, K_{max}=5, N_{max}= 100, α= 0.5, and ΔT = 6. The parameter P_{max}=100 indicates that the algorithm stops after 100 consecutive attempts to improve Y_{best} without success. The average results are shown and compared with those of the existing algorithms in Table 2. It can be observed that the VNS algorithm is able to discover the optimal solutions for 100% of the 96 small-sized problems within a rather short computing time (much less than one second).

Place Table 2 approximately here.

Second, we ran the VNS algorithm once for the 40 problems classified as *Class 2* using the control parameters T_{max}=30 (seconds), K_{max}=5, N_{max}= 1000, α= 0.5, and ΔT = 6. The parameter T_{max}=30 s indicates that the algorithm stops after 30 seconds have elapsed. The average results are shown and compared with those of the existing algorithms in Table 3, and the detailed results are listed in Table 4 and compared with the previous best-known solutions (to date). It can be observed from Table 3 that the VNS algorithm delivers the best performances among all of the compared algorithms with respect to the *Average Cost*, *Deviation from the best-known (%)*, and the *best-known found (%)*. The *RCWW* in the second row shows the average initial solutions resulting from the RCWW method. The *VNS*$^-$ in the third row indicates the VNS algorithm without use of the ADTS procedure, which demonstrates the advantages of the local search method. The fourth row (*VNS* (P_{max}=*100*)) is the performance of the VNS algorithm with the stop condition $P_{\max}$=100 (where RACR is used) and can be compared with that of the fifth row (*VNS** ($P_{\max}$=*100*)) (where RACR is not used, and the objective value is always fully recalculated from the changing item toward top materials). A comparison of these two rows reveals that, when the same solution quality is reached, the RACR method saves a marked amount of computational time. The detailed list in Table 4 indicates that five problems were updated with new best-known solutions, and 75% of the problems (30 of 40) reached their best-known solutions in this run. The column *Max(P)* in Table 4 indicates that certain problems with lower values, such as Problems 20 and 38, could be further optimized if the maximum computing time was extended. The latest list of best-known solutions from *Class 2* problems has been updated in Table A1 in the Appendix.

Place Table 3 approximately here.

Place Table 4 approximately here.

Third, we ran the VNS algorithm once on the 40 problems classified as *Class 3* using the control parameters T_{max}=1800 (seconds), K_{max}=6, N_{max}= 2000, α= 0.5, and ΔT =8. The results are listed in Table 5. As shown in Table 5, the VNS algorithm performs quite well on large-sized MLLS problems because 75% (30 out of 40) of the tested problems were updated with new best-known solutions. We summarize the results in Table 6 and compare them with those from the existing methods in the literature. It can be observed that the VNS algorithm delivers the best performances among all of the algorithms with respect to *Deviation from Best (%)* and *Best solutions found (%)* and delivers the second-best performance in terms of *Average Cost*. It is notable that the *Average Cost* is the arithmetic mean of the solutions for the 40 tested problems and is thus primarily influenced by those problems with large objective values, e.g., Problems 15 and 35, which have the largest and second-largest objective values. If Problem 15 or 35 has a deviation from the best-known solution of merely 0.2% (increase in the objective value), this deviation will cover up all of the good performances (decrease in the objective value) from the remaining 38 problems in terms of *Average Cost* for the entire problem class. For this reason, although the VNS algorithm updated 30 of the 40 problems with new best-known solutions (Problems 15 and 35 were not updated), the *Average Cost* is not the lowest among all of the existing methods. The numbers in column *CI* of Table 5, as defined in Eq. (15) in Section 4, indicate the complexity of the product structures of the problems. The experimental results show that a notably large *CI* can greatly decrease the efficiency of the ADTS local search method and will also result in a low value for Max(P), which may indicate that the current solution is still far from the optimal solution and has potential for further optimization. For example, Problems 15 and 35 are associated with a large *CI* and therefore output a small Max(P); it is therefore assumed that better solutions would be very likely to be obtained if the computational times are extended. The latest version of the best-known solutions for *Class 3* problems has been updated in Table A2 in the Appendix. An example of the new best solution of the NO.39 instance of *Class 3* can be found at http://semen.buaa.edu.cn/Teacher/Templet/DefaultDownLoadAttach.aspx?cwid=1224.

Place Table 5 approximately here.

Place Table 6 approximately here.

In Table 7, we show a comparison of the total number of evaluated candidate solutions throughout the search process, which demonstrates the logic and computational effort defined by Alba et al. (2002) that a generate-and-test algorithm uses to reach the best-found solution. The penultimate row in Table 7 (*VNS's shaking solutions*) is the total number of candidate solutions generated by the shaking procedure, i.e., the number of Y'. The last row (*VNS's bit changes*) in Table 7 shows the total number of times that one bit change (including 1→0 and 0→1) occurred in the incumbent solution, which are the shifts that primarily conducted by the shaking procedure of VNS ($Y \rightarrow Y'$) and the ADTS procedure ($Y' \rightarrow Y''$). The comparison in Table 7 shows that, although the VNS algorithm delivered the best performance in terms of solution quality, it required much more logic and computational effort than other methods. This result demonstrates the benefits of the proposed RACR method for the rapid calculation of the objective function when a bit change occurs. In other words, RACR enables the VNS algorithm (and other generate-and-test algorithms, in our opinion) to test a larger number of candidate solutions within the same computational time.

Place Table 7 approximately here.

Finally, a parameter $N_{\max}$ was added to our VNS algorithm to indicate the extent to which a neighborhood should be explored before moving to the next neighborhood by $k \leftarrow k+1$; in contrast, in the basic scheme of the classical

VNS algorithm, the parameter N_{max} is fixed to 1. We then performed the following experiment to verify the role of N_{max} in the VNS algorithm. We used the 40 *Class 2* medium-sized problems as the benchmark problem set to test the parameter N_{max}, which was varied from 1, 10, 50, 100, 200, 500, 1000, to 2000. The other parameters were fixed at T_{max}=30 s, K_{max}=5, ΔT = 6, and α= 0.2, the stop condition was fixed at 30 seconds of computing time for each problem. The *Average Cost*, *Deviation from best (%)*, and *best solutions found (%)* are recorded in the experiments with respect to every value of N_{max}. The experimental results are shown in Table 8. It can be observed that, with the same computing time, the VNS algorithm tends to obtain a better quality of solutions as the parameter N_{max} increases. The best value for N_{max} is approximately 1000, and a continued increase of N_{max} to values higher than 1000 will not improve the solutions. It is notable that the solution quality is quite poor when N_{max}=1, which is exactly the case for the basic VNS scheme shown in Fig. 1 without the N_{max} parameter. In Table 9, we performed similar experiments for parameter K_{max}, which was increased from 1 to 10 at intervals of 1, and the other parameters were fixed at T_{max}=30 s, N_{max}=1000, ΔT = 6, and α = 0.2. The experimental results are shown in Table 9. It can also be observed that the VNS algorithm tends to obtain better solutions as the parameter K_{max} increases, and an optimal value of K_{max}=8, which gave even better outcomes than those observed in the previous experiments with K_{max}=5 shown in Table 3, was found. We believe that a proper K_{max} will also improve the solutions shown in Table 5 for the large-sized problems, because the K_{max}=6 used is relatively small. Interested readers can download our program to perform their own experiments (http://semen.buaa.edu.cn/Teacher/Templet/DefaultDownLoadAttach.aspx?cwid=1224). Note that when K_{max}=1, the VNS algorithm will degrade to the single-point stochastic search (SPSS) studied by Jeunet and Jonard (2005). Several new best-known solutions were observed in the experiments, but we did not add them to the best-known list of Table A1 because they were found using multiple runs.

Place Table 8 approximately here.

Place Table 9 approximately here.

6. Conclusions

The uncapacitated MLLS model represents a basic form of many extended versions of the MLLS problem under various constraints. The solution approach is foundational and plays a basic role in the solution of many other extended problems. We recently conducted studies on this problem using a variable neighborhood search and presented certain effective and efficient techniques and algorithms. In this paper, we continue our contributions to this topic and focus primarily on two points. First, we suggest an effective local search method that can be added to our previously presented RVNS algorithm; as a result, the solution quality can be markedly improved because many of the well-known benchmark problems were updated with new best-known solutions obtained from our computational experiments. Second, a highly efficient approach for the rapid calculation of the cost variation of the objective function when the incumbent solution is changed was presented. This approach can significantly improve the efficiency of the VNS algorithm and, more importantly, may be helpful in improving the efficiency of many other generate-and-test algorithms for MLLS problems that have been historically presented in the literature.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Projects No. 71271009, 71271010, and 71071007 and the Japan Society of Promotion of Science (JSPS) under Grant No. 24510192.

Appendix A

Place Figure A1 approximately here.

Place Table A1 approximately here.

Place Table A2 approximately here.

References

Alba, E., Nebro,A.J., Troya, J.M. (2002). Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel Distributed Computing*, 62, 1362-1385.

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.

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

Arkin, E., Joneja, D., Roundy R. (1989). Computational complexity of uncapacitated multi-echelon production planning problems. *Operation Research Letters*. 8, 61–66

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

Blackburn, J.D., Millen, R.A. (1982). Improved heuristics for multi-stage requirements planning system. *Management Science*, 28, 44-56.

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.

Coleman, B.J., McKnew, M.A. (1991). An improved heuristic for multilevel lot sizing in material requirements planning. D*ecision Sciences*, 22, 136-156.

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

Dongarra, J.J. (2005). Performance of various computers using standard linear equations software (Linpack Benchmark Report). University of Tennessee Computer Science Technical Report, CS-89-85,

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

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 Economic*s, 68, 241-257.

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

Hansen, P., Mladenovic, N., Moreno-Perez, J.A. (2008a). Variable neighborhood search. *European Journal of Operational Research*, 191, 593-595.

Hansen, P., Mladenovic, N., Moreno-Perez, J.A. (2008b). Variable neighbourhood search: Methods and applications. *4OR-A Quarterly Journal of Operations Research*, 6 (4), 319-360.

Hansen, P., Mladenovic, N., Moreno-Perez, J.A. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175 (1), 367-407.

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

Helber, S., Sahling F. (2010). A fix-and-optimize approach for the multi-level capacitated ot sizing problem. International Journal of Production Economics, 123(2):247–56.

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

Homberger, J., Gehring, H. (2009). An ant colony optimization approach for the multi-level unconstrained lot-sizing problem. *Proceedings of the 42nd Hawaii international conference on system sciences*, Hawaii; CD-ROM. p.7.

Homberger, J. (2010). Decentralized multi-level uncapacitated lot-sizing by automated negotiation. *4OR - A Quarterly Journal of Operations Research*, 8, 155-180.

Kaku, I., Xu, C.H. (2006). A soft optimization approach for solving a complicated multilevel lot-sizing problem. *Proceedings of the 8th Conference of Industrial Management*, pp.3-8.

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 Contro*l, 6, 817-827.

Lang, J. C., Shen, Z. J. M. (2011). Fix-and-optimize heuristics for capacitated lot-sizing with sequence-dependent setups and substitutions. *European Journal of Operational Research*, 214, 595–605.

Labadie, N., Mansini, R., Melechovsky, J., & Calvo, W., R. (2012). The Team Orienteering Problem with Time Windows: An LP-Based Granular Variable Neighborhood Search. European Journal of Operational Research, 220, 15-27.

Mladenovic, N., Urosevic, D., Hanafi, S., Ilic A. (2012a). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220, 270–285.

Mladenovic, N., Kratica, J., Kovacevic-Vujcic, V., & Cangalovic, M. (2012b). Variable neighborhood search for metric dimension and minimal doubly resolving set problems. *European Journal of Operational Research*, 220, 328–337.

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

Pitakaso, R., Almeder, C., Doerner, K.F., Hartl, R.F. (2006). Combining population-based and exact methods for multi-level capacitated lot-sizing problems, *International Journal of Production Research*, 44(22), 4755–4771.

Pitakaso, R., Almeder, C., Doerner, K.F., Hartlb, R.F. (2007). A MAX-MIN ant system for unconstrained multi-level lot-sizing problems. *Computer and Operations Research*, 34, 2533-2552.

Seeanner, F., Almada-Lobo, B., Meyr, H. (2013). Combining the principles of variable neighborhood decomposition search and the fix&optimize heuristic to solve multi-level lot-sizing and scheduling problems. *Computers & Operations Research*, 40, 303–317.

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

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

Wagner, H.M., Whitin, T.M. (2004). Dynamic Version of the Economic Lot Size Model. *Management Science*, 50, 1770-1774, Supplement.

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

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

Xiao, Y.Y., Kaku, I., Zhao, Q.H., Zhang, R.Q. (2012). Neighborhood search techniques for solving uncapacitated multilevel lot-sizing problems. *Computers & Operations Research*, 39, 647-658.

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

Zangwill, W.I. (1968). Minimum concave cost flows in certain network. *Management Science*, 14, 429-450.

Zangwill, 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.

Captions of the Figures.

Fig 1. The basic VNS scheme

Fig. 2 An example for illustrating the PDTS local search

Fig 3. Algorithm for the PDTS local search procedure

Fig 4. The VNS algorithm with a PDTS local search for the MLLS problem

Fig 5. An efficient way (RACR) to quickly calculate the cost variation

Fig A1. An example of solving the model by using IBM ILOG CPLEX 12.2.

Table 1. Interpretation of the parameters and variables that appear in Algorithm 2.

Pass NO.	Notation	Physical means
1	K_{max}	The largest neighbourhood, i.e., the maximum number of changed bits between Y and Y'.
1	N_{max}	The maximum consecutive unsuccessful attempts allowed to search a neighbourhood.
1	α	A parameter in [0.05, 1] used to control the item level of changed bits. See Xiao et al.(2012)
1	Δt	A parameter used to restrict the period scope of the changed bits. See Xiao et al.(2012)
2	P_{max}	The maximum span between any two abutting improvements on Y_{best}.
2	T_{max}	The maximum computing time in seconds.
3	Y_{best}	The best solution found.
5	Y	The incumbent solution.
6	k	The current neighbourhood.
6	N	The number of continuous unsuccessful attempts in searching the current neighbourhood.
6	P	The number of unsuccessful attempts at improving Y_{best}.
7(a)	Y'	The current candidate solution.
7(a)	$\rho(Y, Y')$	The distance between Y and Y'. See Definition 1 in Xiao et al. (2012).
7(a)	$\delta_i(Y, Y')$	The changing range of items. See Definition 2 in Xiao et al. (2012).
7(a)	$\delta_t(Y, Y')$	The changing range of periods. See Definition 2 in Xiao et al. (2012).
7(a)	$L(Y,Y')\sim P(\alpha)$	The changing level between Y and Y' must obey the line probability distribution of α.

Table 2. Comparison of the VNS algorithm with existing algorithms on *Class 1* problems

(P_{max}=100, K_{max}=5, N_{max}= 100, α= 0.5, ΔT = 6)

Method	Avg. Cost	Optimality (%)	Avg. Time (s)	CPU	Source
RCWW	861.05	4.17	<0.01	3i 3.1G	This study
VNS	**810.67**	**100.00**	**0.1**	3i 3.1G	This study
HGA50	810.74	96.88	5.1	PIII 450 M	Dellaert and Jeunet (2000)
MMAS	810.79	92.71	1.0	PIV 1.5G	Pitakaso *et al.*(2007)
GA100	811.98	75.00	5.0	PIV 2.4G	Homberger (2008)
GA3000	**810.67**	**100.00**	5.0	PIV 2.4G	Homberger (2008)
PGA	**810.67**	**100.00**	5.0	PIV 2.4G	Homberger (2008)
SA-NM(1)	**810.67**	**100.00**	4.0	I.C.2D 1.83G	Homberger (2010)
VNS[illegible]	**810.67**	**100.00**	1.8	NB 1.6G	Xiao *et al.* (2011a)
RVND	**810.67**	**100.00**	0.3	I.C.2D 1.6G	Xiao *et al.* (2011b)
INS	**810.67**	**100.00**	3.0	I.C.2D 1.6G	Xiao et al.(2012)
OPT.	**810.67**	--	--	--	-

Note: The best values are indicated in bold face. MMAS's CPU time had been adapted by the factor of Dongarra (2005).

Table 3. Comparing VNS with existing methods on 40 medium-sized problems

Method	Avg. Cost	Dev. from best known (%)	Best known found (%)	Avg. time(s)	CPU type	Sources
RCWW	279,258	5.71%	0	<1	Int.3i 3.1G	This study
VNS	**263,254**	**0.05**	**80**	30	Int.3i 3.1G	This study
VNS⁻	263,316	0.07	57.5	30	Int.3i 3.1G	This study, without ADTS
VNS (P_{max}=100)	263,275	0.06	70	2.4	Int.3i 3.1G	This study
VNS* (P_{max}=100)	263,275	0.06	70	135	Int.3i 3.1G	This study, without RACR
HGA_{250}	263,932	0.76	10	<60	PIII 450M	Dellaert and Jeunet (2000)
MMAS	263,796	0.24	15	**<20**	PIV 1.5G	Pitakaso *et al.* (2007)
PGA	263,360	≥0.07	≤55	60	PIV 2.4G	Homberger (2008)
SACO	263,347	--	--	32	PIV 2.4G	Homberger and Gehring (2009)
SA-NM(1)	263,292	--	--	40	I.C.2D 1.8G	Homberger (2010)
	263,571	0.16	--	26.7	NB 1.6G	Xiao *et al.* (2011a)
RVND	263,399	0.10	45	26.7	I.C.2D 1.6G	Xiao *et al.* (2011b)
INS	263,323	0.07	55	34.2	I.C.2D 1.6G	Xiao *et al.*(2012)
Best known	263,140	0	--	--	--	--

Note: The best values are indicated in bold face; Int.3i 3.1G is abbreviated for Intel(R) Core(TM) 3i-2100 CPU 3.1G; *VNS⁻* indicates the VNS algorithm without the PDTS local search; MMAS's CPU time had been adapted by the factor of Dongarra (2005).

Table 4. Results of the experiment on *Class 2* problems (T_{max}=30s, k_{max}=5, N_{max}=1000, ΔT = 6, α= 0.2)

Prob.	S	D	I	P	Prev. Best known	This study	Dev. from best known(%)	MAX(P)
0	1	1	50	12	**194,571**	196,084	0.78	4,548
1	1	2	50	12	**165,110**	165,632	0.32	3,190
2	1	3	50	12	**201,226**	**201,226**	0.00	4,085
3	1	4	50	12	**187,790**	188,010	0.12	4,576
4	1	5	50	12	**161,304**	161,561	0.16	5,148
5	2	1	50	12	**179,761**	**179,761**	0	2,378
6	2	2	50	12	**155,938**	**155,938**	0	2,619
7	2	3	50	12	**183,219**	**183,219**	0	2,579
8	2	4	50	12	**136,462**	**136,462**	0	2,222
9	2	5	50	12	**186,597**	**186,597**	0	1,841
10	3	1	40	12	**148,004**	**148,004**	0	1,969
11	3	2	40	12	**197,695**	**197,695**	0	1,772
12	3	3	40	12	**160,693**	**160,693**	0	2,115
13	3	4	40	12	**184,358**	**184,358**	0	1,848
14	3	5	40	12	**161,457**	**161,457**	0	2,075
15	4	1	40	12	**185,161**	**185,161**	0	2,578
16	4	2	40	12	**185,542**	**185,542**	0	2,999
17	4	3	40	12	**192,157**	192,794	0.33	2,845
18	4	4	40	12	**136,757**	**136,757**	0	2,594
19	4	5	40	12	**166,041**	**166,041**	0	1,655
20	1	6	50	24	**342,916**	343,138	0.06	786
21	1	7	50	24	292,908	***292,820***	0	1,375
22	1	8	50	24	354,919	***354,834***	0	1,057
23	1	9	50	24	325,212	***325,174***	0	2,472
24	1	10	50	24	**385,936**	**385,936**	0	1,654
25	2	6	50	24	**340,686**	340,705	0.01	1,274
26	2	7	50	24	**378,845**	**378,845**	0	1,153
27	2	8	50	24	**346,313**	**346,313**	0	1,363
28	2	9	50	24	**411,997**	**411,997**	0	1,046
29	2	10	50	24	**390,194**	**390,194**	0	1,238
30	3	6	40	24	**344,970**	**344,970**	0	1,644
31	3	7	40	24	**352,634**	**352,634**	0	1,398
32	3	8	40	24	**356,323**	**356,323**	0	1,203
33	3	9	40	24	**411,438**	**411,438**	0	1,391
34	3	10	40	24	**401,732**	**401,732**	0	1,347
35	4	6	40	24	**289,846**	**289,846**	0	2,014
36	4	7	40	24	**337,913**	339,053	0.34	1,423
37	4	8	40	24	319,905	***319,781***	0	1,671
38	4	9	40	24	366,848	***366,409***	0	699
39	4	10	40	24	**305,011**	**305,011**	0	1,775
				Avg.	263,160	263,254	0.05	2,090

Note: *S* denotes the product structure type; *D* represents the external demand type; *I* is the number of items; *P* is the number of periods; boldface type denotes the best-known; and italic type denotes the new best-known solution.

Table 5. Results of experiments on *Class 3* problems ($T_{\max}$=30 m, $K_{\max}$=6, N_{max}=2000, ΔT = 8, α= 0.2)

	I	P	CI	Prev. best known	This study	Dev. from best known(%)	MAX(P)
0	500	36	13	**591,585**	597,560	1.01	5151
1	500	36	9	816,043	***815,716***	0	5478
2	500	36	6	**908,616**	927,716	2.10	5873
3	500	36	4	**929,929**	943,403	1.45	4717
4	500	36	3	1,145,749	***1,144,209***	0	3436
5	500	36	177	7,639,920	***7,595,372***	0	87
6	500	36	50	3,921,407	***3,873,241***	0	222
7	500	36	21	2,694,924	***2,628,883***	0	571
8	500	36	9	1,880,021	***1,833,488***	0	737
9	500	36	5	1,502,194	***1,487,855***	0	1173
10	500	36	1966	59,121,845	***59,152,662***	0	25
11	500	36	239	13,422,827	***13,349,944***	0	61
12	500	36	56	4,718,146	***4,617,206***	0	186
13	500	36	18	2,908,634	***2,856,389***	0	408
14	500	36	8	1,737,525	***1,722,205***	0	657
15	500	36	13228	**468,463,630**	470,266,674	0.38	2
16	500	36	500	18,677,678	***18,666,341***	0	56
17	500	36	103	7,292,340	***7,277,613***	0	88
18	500	36	25	3,519,932	***3,455,399***	0	276
19	500	36	11	2,278,214	***2,253,609***	0	721
20	500	52	13	**1,187,090**	1,187,246	0.01	2107
21	500	52	9	1,341,412	***1,340,695***	0	1558
22	500	52	6	**1,400,480**	1,401,630	0.08	3362
23	500	52	4	1,382,150	***1,379,391***	0	2835
24	500	52	3	1,657,248	***1,656,209***	0	3245
25	500	52	177	12,671,808	***12,640,011***	0	29
26	500	52	50	7,159,416	***7,158,729***	0	176
27	500	52	21	4,148,783	***4,057,733***	0	407
28	500	52	9	2,889,151	***2,829,847***	0	703
29	500	52	5	2,183,815	***2,162,356***	0	1594
30	500	52	1966	**101,497,679**	101,619,428	0.12	10
31	500	52	239	18,028,225	***18,003,896***	0	30
32	500	52	56	6,780,986	***6,747,357***	0	101
33	500	52	18	4,055,536	***3,944,879***	0	310
34	500	52	8	2,559,885	***2,529,417***	0	586
35	500	52	13228	**755,506,278**	757,276,966	0.23	3
36	500	52	500	**33,309,777**	33,332,938	0.07	22
37	500	52	103	10,464,662	***10,409,592***	0	22
38	500	52	25	5,116,338	***4,989,034***	0	177
39	500	52	11	3,391,440	***3,331,201***	0	304
	Avg.		823	39,522,583	39,586,601	0.14%	1,188

Note: *I* is the number of products; *P* is the number of periods; *CI* is the Complexity Index of the product structure defined in Eq.(15); boldface type denotes the best known solution; and italic type denotes the new best solution found.

Table 6. Comparing VNS with existing algorithms on 40 large-sized problems

Method	Avg. Cost	Dev. from Best known (%)	Best solutions found (%)	Time (m)	CPU type	Source
RCWW	42,263,980	12.57	**0**	0.2s	Int.3i 3.1G	This study
VNS	39,586,601	**0.14**	**75%**	**30**	Int.3i 3.1G	This study
HGA_{1000}	40,817,600	5.31	0	63.5	PIII 450M	Dellaert and Jeunet (2000)
MMAS	40,371,702	4.03	5	40	PIV 1.5G	Pitakaso *et al.* (2007)
PGA	39,809,739	3.02	≤10	60	PIV 2.4G	Homberger (2008)
SACO	39,790,184	--	≤25	46.7	PIV 2.4G	Homberger and Gehring (2009)
SA-NM(1)	40,178,548	<1.24	<15	33.0	I.C.2D 1.8G	Homberger (2010)
RVNS	39,869,210	1.44	0	44.4	I.C.2D 1.6G	Xiao *et al.* (2011b)
INS	**39,539,091**	1.19	10	46.6	I.C.2D 1.6G	Xiao *et al.*(2012)
Best known	39,491,868	0	--	--	--	--

Note: The best values are indicated in bold face; *Int.3i 3.1G* is abbreviated for Intel(R) Core(TM) 3i-2100 CPU 3.1G; MMAS's CPU time had been adapted by the factor of Dongarra (2005).

Table 7. Comparison of logic computational efforts

	AVG of Class 1		AVG of Class 2		AVG of Class 3		
Method	Logic efforts	time	Logic efforts	time	Logic efforts	time	CPU type
HGA	12,500	5s	>62,500	60s	250,000	3810s	PIII 450M
MMAS	520	1s	1520	20s	5,020	2400s	PIV 1.5G
PGA	930,781	5s	1,950,322	60s	2,578,855	3600s	PIV 2.4G
VNS's shaking solutions	81,069	1s	23,274,630	30s	426,623,946	1800s	Int.3i 3.1G
VNS's bit changes	336,894	1s	219,813,273	30s	--	1800s	Int.3i 3.1G

Note: the data for PGA and MMAS comes from Homberger (2008), and the data of HGA is estimated; MMAS's CPU time had been adapted by the factor of Dongarra (2005).

Table 8. Performances of VNS on 40 medium-sized problems with N_{max} varying
(T_{max}=30s, k_{max}=5,ΔT = 6, and α= 0.2)

N_{max}	1	5	10	30	50	100	200	500	1000	2000
Avg. Cost	264,930	264,411	263,771	263,362	263,299	263,264	263,259	263,252	**263,250**	263,252
Dev. from best (%)	0.68	0.48	0.24	0.084	0.060	0.047	0.045	**0.042**	**0.042**	**0.042**
Best known found (%)	22.5	27.5	32.5	47.5	57.5	67.5	70.0	77.5	**80.0**	77.5

Note: boldface type denotes the best values.

Table 9. Performances of VNS on 40 medium-sized problems with K_{max} varying
(T_{max}=30s, N_{max}=1000, ΔT = 6, and α= 0.2)

K_{max}	1	2	3	4	5	6	7	8	9	10
Avg. Cost	263,372	263,287	263,278	263,254	263,254	263,248	263,248	**263,247**	263,248	263,248
Dev. from best (%)	0.088	0.056	0.052	0.043	0.043	**0.041**	**0.041**	**0.041**	**0.041**	**0.041**
Best known found (%)	47.5	60.0	67.5	72.5	77.5	77.5	80.0	**82.5**	77.5	**82.5**

Note: boldface type denotes the best values.

Table A1. New best known solutions for *Class 2*

Prob. ID	S	D	I	P	Best solution	Method
0	1	1	50	12	194,571	B&B
1	1	2	50	12	165,110	B&B
2	1	3	50	12	201,226	B&B
3	1	4	50	12	187,790	B&B
4	1	5	50	12	161,304	B&B
5	2	1	50	12	179,761	RVND
6	2	2	50	12	155,938	B&B
7	2	3	50	12	183,219	B&B
8	2	4	50	12	136,462	B&B
9	2	5	50	12	186,597	B&B
10	3	1	40	12	148,004	PGAC
11	3	2	40	12	197,695	PGAC
12	3	3	40	12	160,693	MMAS
13	3	4	40	12	184,358	PGAC
14	3	5	40	12	161,457	PGAC
15	4	1	40	12	185,161	SA-NM(1)
16	4	2	40	12	185,542	PGAC
17	4	3	40	12	192,157	MMAS
18	4	4	40	12	136,757	SA-NM(1)
19	4	5	40	12	166,041	PGAC
20	1	6	50	24	342,916	SA-NM(1)
21	1	7	50	24	**292,820**	**VNS**
22	1	8	50	24	**354,834**	**VNS**
23	1	9	50	24	**325,174**	**VNS**
24	1	10	50	24	385,936	INS
25	2	6	50	24	340,686	HGA
26	2	7	50	24	378,845	HGA
27	2	8	50	24	346,313	INS
28	2	9	50	24	411,997	HGA
29	2	10	50	24	390,194	INS
30	3	6	40	24	344,970	HGA
31	3	7	40	24	352,634	PGAC
32	3	8	40	24	356,323	SA-NM(1)
33	3	9	40	24	411,438	RVND
34	3	10	40	24	401,732	HGA
35	4	6	40	24	289,846	RVND
36	4	7	40	24	337,913	MMAS
37	4	8	40	24	**319,781**	**VNS**
38	4	9	40	24	**366,409**	**VNS**
39	4	10	40	24	305,011	SA-NM(1)
		Average			263,140	

Note: bold face indicates the new best values found in this paper.

Table A2. New best known solutions for *Class 3*

Prob. ID	I	P	I	Best solution	Method
0	500	36	500	591,585	SA-NM(1)
1	500	36	500	**815,716**	**VNS**
2	500	36	500	908,616	SA-NM(1)
3	500	36	500	929,929	SACO
4	500	36	500	**1,144,209**	**VNS**
5	500	36	500	**7,595,372**	**VNS**
6	500	36	500	**3,873,241**	**VNS**
7	500	36	500	**2,628,883**	**VNS**
8	500	36	500	**1,833,488**	**VNS**
9	500	36	500	**1,487,855**	**VNS**
10	500	36	500	59,121,845	INS
11	500	36	500	**13,349,944**	**VNS**
12	500	36	500	**4,617,206**	**VNS**
13	500	36	500	**2,856,389**	**VNS**
14	500	36	500	**1,722,205**	**VNS**
15	500	36	500	468,463,630	INS
16	500	36	500	**18,666,341**	**VNS**
17	500	36	500	**7,277,613**	**VNS**
18	500	36	500	**3,455,399**	**VNS**
19	500	36	500	**2,253,609**	**VNS**
20	500	52	500	1,187,090	MMAS
21	500	52	500	**1,340,695**	**VNS**
22	500	52	500	1,400,480	MMAS
23	500	52	500	**1,379,391**	**VNS**
24	500	52	500	**1,656,209**	**VNS**
25	500	52	500	**12,640,011**	**VNS**
26	500	52	500	**7,158,729**	**VNS**
27	500	52	500	**4,057,733**	**VNS**
28	500	52	500	**2,829,847**	**VNS**
29	500	52	500	**2,162,356**	**VNS**
30	500	52	500	101,497,679	INS
31	500	52	500	**18,003,896**	**VNS**
32	500	52	500	**6,747,357**	**VNS**
33	500	52	500	**3,944,879**	**VNS**
34	500	52	500	**2,529,417**	**VNS**
35	500	52	500	755,506,278	INS
36	500	52	500	33,309,777	SA-NM(1)
37	500	52	500	**10,409,592**	**VNS**
38	500	52	500	**4,989,034**	**VNS**
39	500	52	500	**3,331,201**	**VNS**
	Average			39,491,868	

Note: bold face indicates the new best values found in this paper.

Initialisation. Define the set of neighbourhood structures N_k, k=1, …, K_{max}, that will be used in the search; find an initial solution Y_0; set $Y \leftarrow Y_0$; and choose a stop condition.

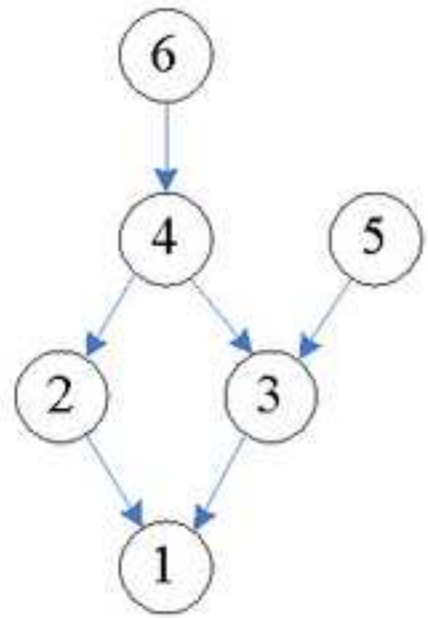
Repeat the following until the stop condition is met:

(1) Set; $k \leftarrow 1$; (2) Until k=K_{max} repeat the following steps:

(a) Shaking. Generate a point Y' at random from the k^{th} neighbourhood of Y ($Y \in N_k(Y)$);

(b) Local search. Apply some local search method with Y' as the initial solution; denote with Y'' the local optimum that is obtained;

(c) Move or not. If Y'' is better than Y, move there ($Y \leftarrow Y''$), and continue the search from $N_1(x)$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k+1$;



(a) Product structure

	$t=1$	2	3	4	5	6
$i=6$	1	0	0	0	1	0
$i=5$	1	0	1	0	1	0
$i=4$	1	0	1	0	1	0
$i=3$	1	0	1	0	1	1
$i=2$	1	0	1	1	1	1
$i=1$	1	0	1	1	1	1

(b) A solution Y with zero lead time

	$t=1$	2	3	4	5	6
$i=6$	1	0	1	0	0	0
$i=5$	0	1	0	1	0	0
$i=4$	0	1	0	1	0	0
$i=3$	0	0	1	1	1	0
$i=2$	0	0	1	0	1	0
$i=1$	0	0	0	1	1	1

(c) A solution Y with one lead time

When Y moves into Y' by setting $Y_{it} \leftarrow 1 - Y_{it}$, the PDTS procedure is called with parameter i and t as follows:

1) For each $j \in \Gamma_i^-$ do (a), (b), (c), and (d)

   (a) Let $t' \leftarrow t - l_j$. //correct the period with the lead time

   (b) If $Y_{jt'} = Y_{it}$ then $Y_{jt'} \leftarrow 1 - Y_{it}$ and calculate the cost reduction according to Eq. (1).

   (c) Recursively call the PDTS procedure with parameters j and t'.

   (d) Rollback the change performed to Y in this procedure.

2) If the largest cost reduction searched is positive, then move Y' to Y'' along the corresponding change path.

1. Set the parameters $K_{\max}$, $N_{\max}$, α, and Δt.
2. Choose a stop condition: 1— a maximum improvement span $P_{\max}$ or 2 — a fixed time $T_{\max}$.
3. $Y_{best} \leftarrow$ a large number.
4. Repeat the following steps, 5, 6, 7, 8, and 9.
5. Initialise a solution Y as the incumbent by using the modified RCWW algorithm in Xiao et al. (2011)
6. Set $k \leftarrow 1$, $N \leftarrow 0$, $P \leftarrow 0$;
7. Repeat the following (a), (b), (c), and (d) until $k = K_{\max}$:

   (a) Shaking: produce a new solution Y' from Y, satisfying $\rho(Y, Y') = k$, $\delta_s(Y, Y') = 0$, $\delta_t(Y, Y') = \Delta t$, and $L(Y, Y') \sim P(\alpha)$.

   (b) Move Y into Y'

   (b) Start a PDTS procedure (local search) to find the best solution Y'' around Y'.

   (d) If Y'' is better than Y, then $Y \leftarrow Y''$, $k \leftarrow 1$, and $N \leftarrow 0$; otherwise, rollback all of the changes done to Y and $N \leftarrow N + 1$.

   (e) If N=$N_{\max}$, then $k \leftarrow k + 1$ and $N \leftarrow 0$;
8. If Y is better than Y_{best}, then $Y_{best} \leftarrow Y$, $P \leftarrow 0$; otherwise, $P \leftarrow P + 1$.
9. If *stop condition is met* (P>$P_{\max}$ for condition 1 and *computing time*>$T_{\max}$ for condition 2), then output Y_{best} and terminate the program.

For the incumbent Y and its known cost $TC(Y)$, if Y moves to Y' by changing a bit, i.e., $Y_{it} \leftarrow 1-Y_{it}$, then the cost reduction $CR=TC(Y')-TC(Y)$ can be calculated by following function $Cal\_CR(i, t)$.

Function $Cal\_CR\ (i, t)$

(1) **If** Y_{it} changes from 1 to 0 **Then**

Let $P_{it'} \leftarrow P_{i,t-1}$, $\forall t' \geq t, P_{it'} = t$.

Let $\Delta X \leftarrow X_{it}$, $X_{it'} \leftarrow X_{it'} + X_{it}$, and $X_{it} \leftarrow 0$, where $t' = P_{i,t-1}$.

Let $CR \leftarrow -S_i + H_i \cdot (t - P_{i,t-1}) \cdot X_{it} + Sub\_Func(i, P_{i,t-1}, t, \Delta X)$.

End If

(2) **If** Y_{it} changes from 0 to 1 **Then**

Let $P_{it'} \leftarrow t$, $\forall t' \geq t, P_{it'} = P_{i,t-1}$.

Let $\Delta X \leftarrow \sum D_{it'}$, where $t' \geq t, P_{it'} = t$.

Let $X_{it'} \leftarrow X_{it'} - \Delta X$, where $t' = P_{i,t-1}$.

Let $X_{it} \leftarrow \Delta X$.

Let $CR \leftarrow S_i - H_i \cdot (t - P_{i,t-1}) \cdot \Delta X + Sub\_Func(i, P_{i,t-1}, t, -\Delta X)$.

End If

Return CR

End Function

Function $Sub\_Func\ (i, t_1, t_2, \Delta X)$

Let $CR' \leftarrow 0$

For each $j \in \Gamma_i^j$ do

Let $\Delta D \leftarrow \Delta X \cdot C_{ji}$, $t_1 \leftarrow t_1 - L_j$, and $t_2 \leftarrow t_2 - L_j$.

Let $D_{j,t_1} \leftarrow D_{j,t_1} + \Delta D$ and $D_{j,t_2} \leftarrow D_{j,t_2} + \Delta D$.

Let $CR' \leftarrow CR' + \Delta D \cdot H_j \cdot (t_1 - P_{j,t_1}) - \Delta D \cdot H_j \cdot (t_2 - P_{j,t_2})$.

Let $X_{j,t'} \leftarrow X_{j,t'} + \Delta D$ and $X_{j,t''} \leftarrow X_{j,t''} - \Delta D$, where $t' = P_{j,t_1}$ and $t'' = P_{j,t_2}$.

If $P_{j,t_1} < P_{j,t_2}$ then $CR' \leftarrow CR' + Sub\_func(j, t_1, t_2, \Delta D)$.

End For

Return CR'

End Function

```
//Data File: MyMLLS.dat,   No.1 instance for small-sized problems *************************
ItemNum=5;
PeriodNum=12;
//Items' unit inventory holding costs and setup costs
Hi=[0.073205,0.066550,0.060500,0.055000,0.050000];
Si=[1.280000,3.200000,8.000000,20.000000,50.000000];
//BOM
Cij= [[0,0,0,0,0], [1,0,0,0,0], [0,1,0,0,0], [0,0,1,0,0], [0,0,0,1,0]];
//External demands
dDit=
[[80,100,125,100,50,50,100,125,125,100,50,100],
[0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0]];
//*****************************************************************************

//Model File: MyMLLS.mod********************************************************
using CP;
int ItemNum=...;
int PeriodNum=...;
range Items = 1..ItemNum;
range Periods = 1..PeriodNum;
float Si[Items]=...;
float Hi[Items]=...;
int Cij[Items][Items]=...;
int dDit[Items][Periods]=...;
//Variables
dvar boolean Yit[Items][Periods];
dvar int Dit[Items][Periods];
dvar int Pit[Items][Periods] in 1..PeriodNum;
dvar int Xit[Items][Periods];
//Objective
minimize
  sum( i in Items, t in Periods )
    (Si[i]*Yit[i][t]+Hi[i]*(t-Pit[i][t])*Dit[i][t]);
 //Constraints
subject to {
  //Dit:
  forall( i in Items, t in Periods )
    Dit[i][t] == dDit[i][t] + sum(j in Items)Cij[i][j]*Xit[j][t]; //leading time is zero
  //Pit :
  forall( i in Items, t in Periods )
    Pit[i][t]==max(tt in Periods:tt<=t)(tt*Yit[i][tt]);
  //Xit:
  forall( i in Items, t in Periods )
   {
      Xit[i][t]==sum(tt in Periods:tt>=t)((Pit[i][tt]==t)*Dit[i][tt]);
      Xit[i][t]>=0;
      Xit[i][t] *(1-Yit[i][t])==0;
   }
//*****************************************************************************
```

Research Highlights

- A new VNS algorithm for solving uncapacitated multilevel lot-sizing problems is developed.
- A local search called the Predecessors Depth-first Traversal Search is embedded in VNS.
- An approach for fast calculation of the cost change for the VNS algorithm is proposed.
- The experimental results show that the new VNS algorithm outperforms all of the existing algorithms.