Int J Adv Manuf Technol (2015) 78:395–406
DOI 10.1007/s00170-014-6599-4

ORIGINAL ARTICLE

A hybrid heuristic approach for production planning in supply chain networks

Daecheol Kim · Hyun Joon Shin

Received: 26 June 2013 / Accepted: 16 November 2014 / Published online: 7 December 2014

Abstract Planning distributed manufacturing facilities is one of the most challenging tasks in the supply chain management. This paper proposes a production planning algorithm for the multi-level, multi-item capacitated lot-sizing problem (MLCLSP) in a supply chain network that takes back order into account. MLCLSP is a mixed integer linear programming (MIP) problem and is NP-hard. This paper presents an efficient, hybrid, heuristic algorithm named greedy rolling horizon search (GRHS) that combines a rolling horizon local search heuristic with an exact linear program (LP) solver. Computational experiments show that GRHS performs well in terms of total costs and computational time and is superior to existing meta-heuristics, such as tabu search, simulated annealing, and genetic algorithms.

Keywords Supply chain planning · Multi-level multi-item · Capacitated lot sizing · Back order · Hybrid heuristic · Meta-heuristics

1 Introduction

This paper presents a comprehensive solution method for multi-level, multi-item capacitated lot-sizing problems (MLCLSP) in the distributed manufacturing context. The method combines search heuristic and exact LP solver to build production schedules into the supply chain planning process. Constructing the production schedule corresponds to calculating the required quantities of raw materials, components, and end products that should be purchased or produced in each period while minimizing total costs and considering capacity and back-order constraints. In the production planning process of distributed manufacturing facilities, proper decisions on lot sizing have a significant impact on system performance and productivity and hence also on the company's ability to enhance its market competitiveness [8]. Therefore, it is crucial to develop and to improve an effective solution methodology for distributed lot-sizing problems.

The capacitated version of lot-sizing problem has been proven to be NP-hard by Maes et al. [18], meaning that this problem is also theoretically hard to solve. Real-world problems are much more complicated and computationally challenging, requiring more efficient methods [22]. To reduce these difficulties, decomposition or relaxation has been adopted as a practical method to quickly discover adequate solutions.

Typical approaches using decomposition and relaxation methods are discussed below. Belvaux and Wolsey [4] has proposed a special branch and cut system that uses relax-and-fix heuristics to solve lot-sizing problems. Stadtler [20] and Federgruen et al. [9] have adopted the idea of "time windows" in their methodologies to decompose the whole problem into several sub-problems defined by a "lot-sizing window" or a "progressive interval" then solving them in order. Akartunali and Miller [1] also have used a relax-and-fix approach with time windows. The basic idea of relax-and-fix with time windows is to relax all the binary variables continuously except for the variables in the periods of the predefined time window, solve the problem, and, using the solution obtained, fix the binary variables in the window. The next window is then processed in the same manner. Although there are many problem-specific heuristics in the mixed integer problem (MIP) literature, there are comparatively few formal MIP heuristics. In an effort to devise a tailorable heuristic, Fischetti and Lodi [10] have presented MIP heuristics that use the idea of

D. Kim
Division of Business Administration, Hanyang University,
222 Wangsimni-ro, Seongdong-gu, Seoul 133-791,
Republic of Korea
e-mail: dckim@hanyang.ac.kr

H. J. Shin (✉)
Department of Management Engineering, Sangmyung University,
Cheonan 330-720, Republic of Korea
e-mail: hjshin@smu.ac.kr

branching on the neighborhoods of the current MIP solution, and Danna et al. [7] have proposed a relaxation induced neighborhood search (RINS) that searches the neighborhood between the LP relaxation solution and the current MIP solution. Li et al. [17] have devised a simple three-stage approach that is applicable to both single-level and multi-level, multi-item capacitated dynamic lot-sizing problem and showed its simplicity and general applicability. Goren et al. [13] have developed a hybrid approach by combining genetic algorithms (GAs) and a fix-and-optimize heuristic to solve the capacitated lot-sizing problem with setup carryover and showed that the performance of the pure GAs improves when hybridized with the fix-and-optimize heuristic. Chan and Chung [5] and Chan et al. [6] applied GAs with analytic hierarchy process (AHS) and a biased random key GAs to demand driven supply chain and lot-sizing problems, respectively. Almada-Lobo and James [2] have also employed a tabu search and a variable neighborhood search meta-heuristic to solve the multi-item capacitated lot-sizing and scheduling problem with sequence-dependent setup times and costs, indicating that their approach gives an efficient performance for solving medium- to large-sized problems.

From the perspective of a problem model and solution approach, the main differences between the research above and our model are as follows: (1) single-site manufacturing facilities vs. multi-site (distributed) facilities, which increase the complexity of MIP problem; (2) overtime vs. back orders, which are realistic considerations and give manufacturing firms more flexibility when making decisions because they often accept excess demand beyond overtime capacities; and (3) efficient hybridization of the exact solver and search heuristic optimization approach, which can be applied to many similarly formulated MIP problems, such as scheduling, distribution, and portfolio optimization problems among others.

Together with increasing computing power, commercial LP/MIP solvers, such as CPLEX or XpressMP, have been capable of solving large problems. However, real-world applications are growing in size due to the complexity of production systems. Because solving the general MIP to optimality is a matter of great difficulty, commercial LP/MIP solvers are not able to solve general, real-world applications within an acceptable timeframe. Even so, the solvers perform well when solving larger LP and MIPs, provided that they have only a few binary decisions. For our approach, we use this potential and merge it with a rolling horizon local search method named greedy rolling horizon search (GRHS). We have tested GRHS on three test problem sets and compared our solutions with those of commercial solver and the existing meta-heuristics (tabu search, simulated annealing, and genetic algorithm).

The rest of this paper is organized as follows. Section 2 presents a model formulation for production planning problems in a supply chain network. Section 3 describes a hybrid heuristic approach for the model presented in Section 2. Section 4 presents experimental results to evaluate the performance of the proposed approach. Finally, Section 5 contains the concluding remarks.

2 Problem statement and model formulation

A supply chain with multiple manufacturing facilities is considered in this paper. Each facility has capacity restriction and deals with multiple component or end products. In the existing literature, overtime and overtime cost are often employed [14, 19, 3]; however, it is almost impossible to meet the excess demand by the deadline with only overtime. Therefore, this study allows back orders for every facility but imposes a penalty for it. The production of the products can share the capacities of each facility. The notations and assumptions used for problem formulation are as follows.

Assumptions:

1. A product can be used as a common part for several end products (general bill of material).
2. The planning horizon is equally divided into several planning periods.
3. The order quantity can be produced within a specified period.
4. Each resource has limited capacity.
5. Back order for both external and internal demand is allowed.
6. Each facility can have independent demand as well as dependent demand.
7. The production of an item consumes resource capacity only during that period.
8. Resource setup for an item cannot be carried over to the next period.

Notations:

$k=1,\ldots, K$	index of component and end products
$t=1,\ldots, T$	index of planning period
$j=1,\ldots, J$	index of facilities
$S(k)$	set of indices for the immediate successors of item
$F(k)$	set of indices for the facilities capable of producing item
q_{kjt}	planned output quantity for item in period t at facility j
z_{kjt}	inventory level of item k at the end of period t at facility j
bo_{kjt}	back-order quantity of item k at the end of period t at facility j

c_{jt} available capacity of facility j in period t
d_{kjt} demand for item k in period t at facility j
a_{ki} number of units of item k required to produce one unit of item i
h_{kj} inventory holding cost for item k at facility j
s_{kj} setup cost for item k at facility j
b_{kj} back-order cost for item k at facility j
v_{kjt} production time per unit of item k in period t at facility j
u_{kjt} setup time of item k in period t at facility j
M a large number
y_{kjt} binary production decision variable for item k during period t at facility j. The relationship with q_{kjt} is defined by $y_{kjt} = \begin{cases} 0 & \text{if } q_{kjt} = 0; \\ 1 & \text{otherwise.} \end{cases}$

With these assumptions and notations, the supply chain planning problem can be formulated as the following MIP.

MIP:

$$\text{Minimize} \sum_{k=1}^{K} \sum_{j=1}^{J} \sum_{t=1}^{T} \left(s_{kj} y_{kjt} + h_{kj} z_{kjt} + b_{kj} bo_{kjt} \right), \quad (1)$$

subject to

$$q_{kjt} + z_{kjt-1} - z_{kjt} + bo_{kjt} - bo_{kjt-1} - \sum_{\substack{i \in S(k) \\ j \in F(k)}} a_{ki} q_{ijt} = d_{kjt} \quad \forall\ k, j \text{ and } t, \quad (2)$$

$$\sum_{k=1}^{K} \left(v_{kjt} q_{kjt} + u_{kjt} y_{kjt} \right) \le c_{jt} \quad \forall\ j \text{ and } t, \quad (3)$$

$$bo_{kjt} - bo_{k,j,t-1} \le d_{kjt} + \sum_{\substack{i \in S(k) \\ j \in F(k)}} a_{ki} q_{ijt} \quad \forall\ k, j \text{ and } t, \quad (4)$$

$$q_{kjt} - M y_{kjt} \le 0 \quad \forall k, j \text{ and } t, \quad (5)$$

$$z_{kj0} = z_{kjT} = 0 \quad \forall k \text{ and } j, \quad (6)$$

$$z_{kjt} \ge 0 \quad \forall k, j \text{ and } t, \quad (7)$$

$$q_{kjt} \ge 0 \quad \forall k, j \text{ and } t, \quad (8)$$

$$bo_{kjt} \ge 0 \quad \forall k, j \text{ and } t, \quad (9)$$

$$y_{kjt} \in \{1, 0\} \quad \forall\ k, j \text{ and } t, \quad (10)$$

The above supply chain planning model is carried out over time horizon T. Each component or end item incurs an inventory holding cost if it is kept as inventory and a setup cost if a purchasing/production order is released. Failure to meet the demand incurs a back-order cost as well. The objective function (1) states that the sum of the setup, inventory holding, and back-order costs has to be minimized. Constraint (2) reflects the multi-level production structure and the mass-balance relationships between item inventories in the system over the planning horizon. A production quantity of product k in period t is available in period t to satisfy external demand d_{kjt} or to be used in the production of the succeeding product i. Inequality (3) states that the production quantities and setups must meet the capacity constraints for all facilities, and inequality (4) specifies that, for a particular period, the back-order level increase from the previous period should be less than the quantity of the demand during that period. Inequality (5) ensures that a facility is set up for product k in period t if the product is produced during this period. Equation (6) states that there are neither initial nor ending inventories. Inventory levels (7), production quantities (8), and back-order quantities (9) cannot be negative, and the setup variable (10) is binary. This model captures several important aspects of production planning problems: the bill of material (BOM) structure, as characterized by a_{ki}, which defines the supply structure required to produce the end item; the fundamental trade-off among setup, inventory and back-order costs; and the complicating factor of limited capacity.

3 GRHS algorithm

The goal of our study is to find a near-optimal solution because solving the MIP problem for realistic scenarios using an optimal procedure, such as branch and bound, is impractical. To incorporate these points in our heuristic, we propose a hybrid heuristic algorithm named GRHS. Based on the rolling horizon local search procedure, GRHS makes the principle production decisions (y_{kjt}), that is to determine when and in which facility should production take place. Then the production quantity (q_{kjt}), inventory level (z_{kjt}), and back-order quantity (bo_{kjt}) are subsequently determined by solving the

Fig. 1 GRHS algorithm procedures

```
GRHS ( maxitr , RSize , TFence )

begin
H* := ∅
for itr :=1 to maxitr do
    S* := ∅
    STBase := 0
    construction procedure (S*, H*)
    improvement procedure (S*, H*, STBase, TFence, RSize)
end for
end
```

remaining LP problem with a general LP solver. And these happen in a hybrid manner. As an additional improvement, the reduced LP is repeatedly solved only within the specified time window with production decisions (y_{kjt}) beyond the window fixed. The time window switches the search scope by moving back and forth along the time horizon. A decision-maker can trade in solution time for solution quality by deciding on the time window parameters: rolling speed and width and the number of iterations in which the MIP solution is constructed, relaxed, and optimized. The details of GRHS are described as follows.

Notations:

S^*	is the current set of y_{kjt}'s for all k, j, t
$opp(y_{kjt})$	is the opposite value of the current value of y_{kjt}, i.e., if $y_{kjt}=1$, then $opp(y_{kjt})=0$, and vice versa
H	is the set of decision variables in an MIP solution
H^*	is the set of decision variables in the best-so-far MIP solution
L	is the set of decision variables in an LP solution
TFence	is the width of the rolling horizon
RSize	is the speed of the rolling forward
STBase	is the starting period of the rolling horizon
maxitr	is the maximum number of iterations allowed in the algorithm

3.1 GRHS

GRHS is an iterative algorithm and has *maxitr* iterations. In each iteration, the algorithm goes through two major procedures. Firstly, termed the "construction procedure," generates an initial feasible solution. The next "improvement procedure" enhances the initial solution using the local search with rolling horizon method. GRHS can be characterized as a global search because the algorithm explores unvisited solution space through *maxitr* numbers of multi-start, starting from the new initial solution. A general overview of GRHS is presented in pseudo codes in Fig. 1.

3.2 Construction procedure

The construction procedure creates a feasible solution, as shown in Fig. 2. To obtain an initial feasible solution for starting GRHS, the MIP model can be reduced to an LP model by randomly assigning an integer value of 0 or 1 to y_{kjt}. The LP model can then be solved with a known optimal LP solver. The optimal LP solution is applied to the original MIP model. The resulting optimal LP solutions might be infeasible for the original MIP problem. This infeasibility can be eliminated by finding all unreasonable production lots with $q_{kjt}=0$ and $y_{kjt}=1$ then changing the corresponding y_{kjt} to 0. The construction

Fig. 2 Construction procedure

```
construction (S*, H*)

begin
    for all k , j and t do                                  // convert a MIP into a LP
      r := a random number drawn from [0,1]
      if r > 0.5 then S* := S* U { y_kjt := 1 }              // randomly decide the binary integer variable y_kjt
      else S* := S* U { y_kjt := 0 }
    end for
    L := Solution{LP_Solver(S*)}
    H := L U S*
    if H is feasible then H* := H
    else
      for all k , j and t do                                // feasibility process
        if q_kjt = 0 and y_kjt = 1 then S* := S*-{ y_kjt }, S* := S* U { y_kjt := 0 }
      end for
      H* := L U S*
    end if
end
```

Fig. 3 Improvement procedure

```
improvement procedure (S*, H*, STBase, TFence, RSize)

begin
  while STBase < T do
    for all k , j and STBase ≤ t < min(STBase + TFence, T) do    // search only for time window
      S*:= S* - { y_kjt }
      S*:= S* U { opp(y_kjt) }                                   //Change S* by substituting opp(y_kjt) for y_kjt
    end for
    L :=solution{LP_Solver(S*)}
    H := L U S*
    if H is infeasible then
      for all k , j and t do                                     // feasibility process
        if q_kjt = 0 and y_kjt = 1 then S* := S* - { y_kjt }, S* := S* U { y_kjt := 0 }
      end for
    end if
    H := L U S*
    if value(H) < value(H*) then H* := H
    STBase := STBase + Rsize
  end while
end
```

procedure finally generates a feasible solution. As illustrated in Fig. 3, the solution can be used as an initial solution for further enhancement.

3.3 Improvement procedure

The output of the construction procedure is an initial solution for the next improvement procedure where it can be enhanced through the local search with rolling horizon procedures. Note that the number of alternative solutions generated in the local search process grows very large as the size of the MIP problem increases. Although the computational time for solving a single LP problem is trivial, the entire local search process results in a computational burden for obtaining an efficient solution. To lessen the burden, this study adopts time windows and utilizes y_{kjt} variables only in the time window for neighborhood generation. There are $K \times J \times T$ numbers of y_{kjt} variables in the original MIP formulation. In each time window, however, there are at most $K \times J \times TFence$ numbers of y_{kjt} variables, which are the target of neighborhood generation.

The improvement procedure is iterative, and each iteration is composed of six major steps. In the first step, the time window is defined by *STBase*, *TFence* and *RSize*. *STBase* represents the starting point of the time window, and the width of it can be confined by *TFence*. *RSize* determines the rolling speed of the time window. The time window can be used as the rolling horizon. In the second step, y_{kjt} variables within the time window get new values, and the remaining variables stay the same. That is, in the procedure, only each y_{kjt} variable for $STBase \le t < \min(STBase + TFence, T)$ in the current solution gets the new value, which is the opposite of the current value. For example, if y_{kjt}, which is within the time window, has 0 value in the current solution, the value is changed to 1. The new value can be represented as $opp(y_{kjt})$. Then the resulting problem becomes an LP problem that is solved by the known optimal solver in the third step. The optimal LP solution, the

Table 1 Problem types

Parameters	Type I	Type II	Type III
Planning periods in (T)	18	18	18
Number of facilities (J)	5(A,B,C,D,E)	5(A,B,C,D,E)	7(A,B,C,D,E,F,G)
Number of items (K)	11(2)	20(4)	40(6)
BOM coefficient (a_{kj})	1	1	1
Processing time for a product (v_{kjt})	U [0.5, 2.0]	U [0.5, 2.0]	U [0.5, 2.0]
Setup time (u_{kjt})	U [1, 5] / U [10, 15]	U [1, 5] / U [10, 15]	U [1, 5] / U [10, 15]
Setup cost (s_{kj})	U [1, 10] / U [20, 30]	U [1, 10] / U [20, 30]	U [1, 10] / U [20, 30]
Inventory holding cost (h_{kj})	U [1, 10]	U [1, 10]	U [1, 10]
Back-order cost (b_{kj})	U [20, 30]	U [20, 30]	U [20, 30]
Demand per period (d_{kjt})	U [5, 30]	U [5, 30]	U [5, 30]

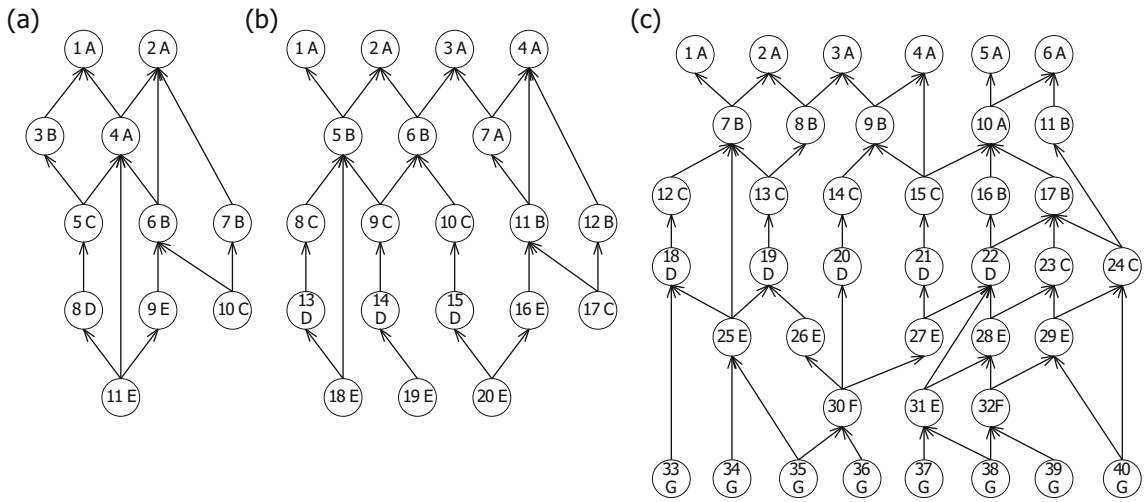

Fig. 4 BOM structures for each problem type: **a** I, **b** II, and **c** III (The alphabets in the circles indicate the facilities indices)

result of the third step, is applied to the original MIP problem in the fourth step. If the MIP solutions are infeasible, then the fifth step is to follow the same process performed in the construction procedure to make the solution feasible. In the last step, the resulting new feasible MIP solution is compared with the best-so-far solution (H^*). The better solution is saved as H^*. The current window is moved forward by *RSize*(-*STBase*←*STBase*+*RSize*) for the next iteration of the local search, and the above processes are continued until *STBase* becomes equal to or greater than *T*.

When a time window is rolling, *RSize* and *TFence* control the rolling speed and the scope of the local search space. The effects of these decision parameters on algorithm performance will be examined through computational experiments in Section 4.

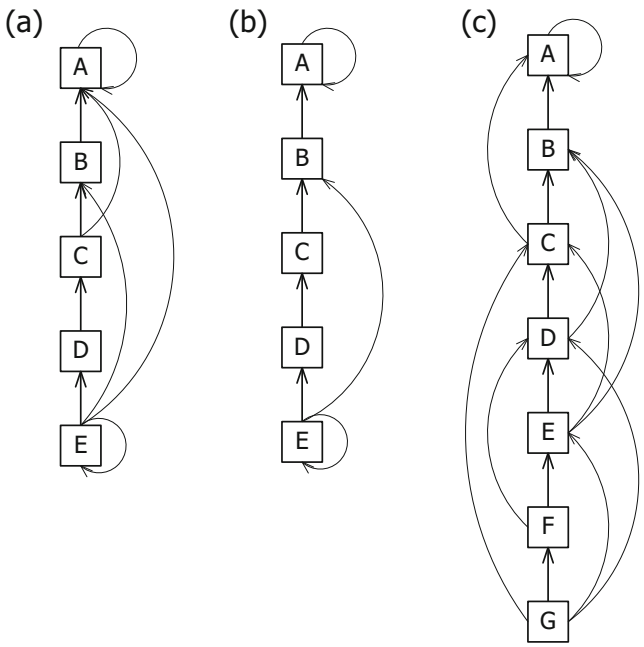

Fig. 5 Facility precedence structures of problem type: **a** I, **b** II, and **c** III

4 Computational experiments

4.1 Test problems

Three types of problems (types I, II, and III) are used to verify the performance of the proposed algorithm. As in Table 1, the planning horizon of all problem types consists of 18 periods (T=18). While problem type III consists of seven facilities (*A*, *B*, *C*, *D*, *E*, *F*, *G*) with 40 end products, subassemblies, and component parts, types I and II consist of five facilities (*A*, *B*, *C*, *D*, *E*) with 11 and 20 items, respectively. In addition, the numbers in parentheses represent the number of end products for each problem type.

Figures 4 and 5 illustrate the BOM structure and the process plans among facilities for the different parts. To simplify the experiment without any loss of generality, BOM coefficients are set to 1. Figure 5 presents the precedence relation between the facilities or within a facility. Note that while the product structure is non-cyclic (as no operation can be its own predecessor), the facility precedent structure may be cyclic in general.

When it comes to test instance, we consulted the exiting literature (e.g., [15, 23]) to generate data. No agreed-upon standard test bed has been established and accepted in the field, and it is doubtful whether such a data set can be established which covers all the different distributed lot-sizing problems arising in reality. Therefore, to simplify our experiment and without loss of generality, we employed uniform distribution referring to the existing literature. As given in Table 1, two different setup costs and times are used. In other words, if the

Table 2 Decision parameters

Parameters	Values
maxitr	1, 3, 5
RSize	2, 4, 6
TFence	3, 6, 12, 18

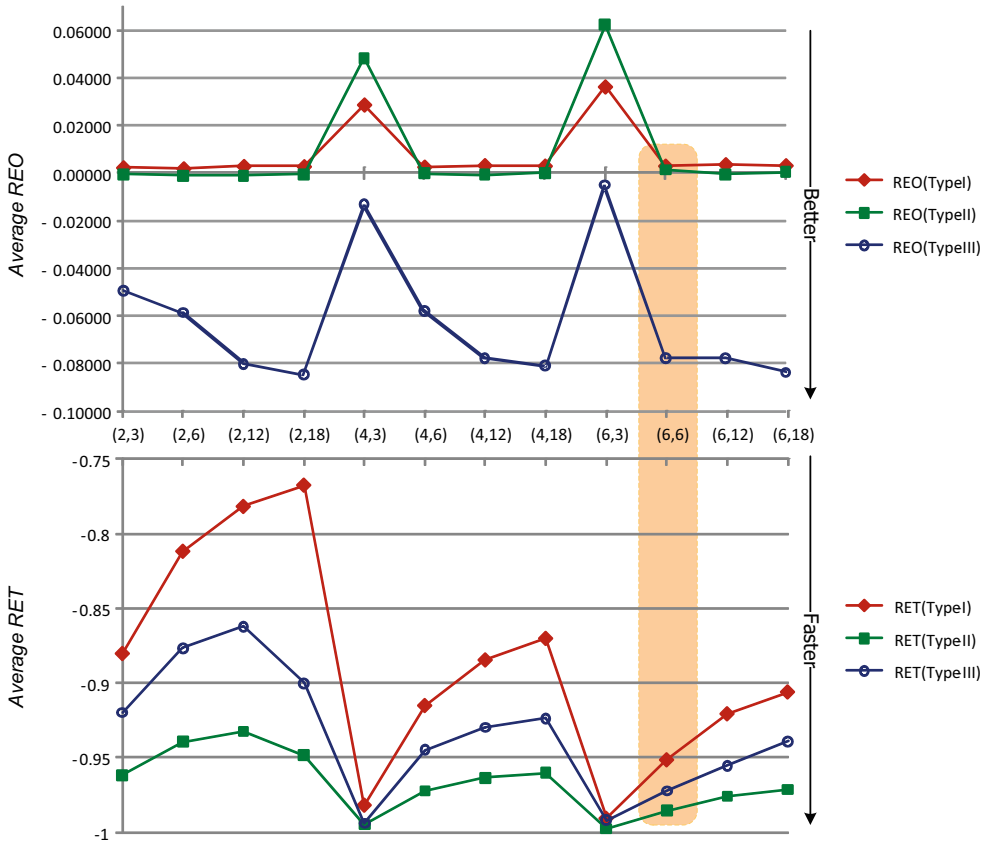

Fig. 6 Selections of efficient parameter pair (*RSize*, *TFence*) based on *REO* and *RET* analysis

next operation occurs in the same facility, the setup cost and time are created from uniform distribution with smaller parameters than the occasion that the next operation occurs in a different facility. The same inventory holding and back-order costs are applied to all problem types. Demands per period are the same for all problem types. The available capacity of each facility per period can be calculated by multiplying the sum of the averages of processing time and setup time by the average of demand:

$$C_{jt} = Avg(d_{kjt}) \times \left(Avg(v_{kjt}) + Avg(u_{kjt})\right) \quad (11)$$

4.2 Preliminary experiments

The decision parameters are determined during the preliminary experiments. The decision parameters that are expected to affect the performance of GRHS algorithm are (1) *maxitr*, (2) *RSize*, and (3) *TFence*. Whereas *maxitr* decides the total number of iterations, *RSize* and *TFence* control the rolling speed and the breadth of the time window, respectively. The various levels of parameters used in this experiment are presented in Table 2.

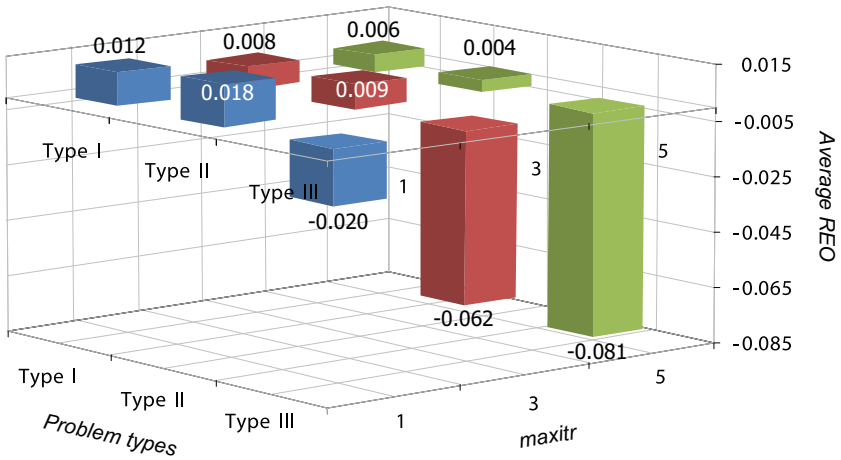

Fig. 7 Average REO comparison according to *maxitr*

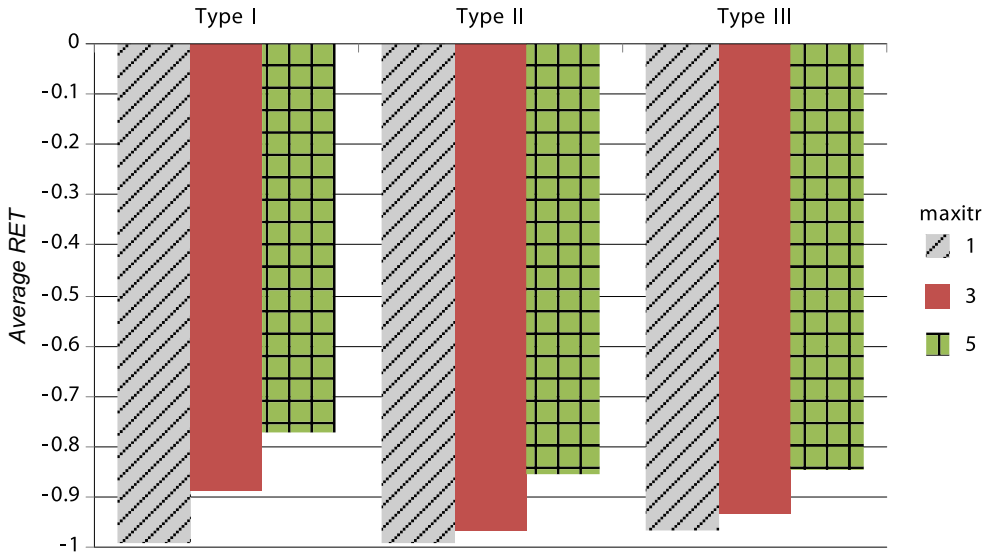

Fig. 8 Average RET comparison according to *maxitr*

To analyze the effects of the decision parameters, the performance of GRHS against a known MIP solver, XpressMP (©Dash Optimizations), for three problem types is evaluated. For each problem type, 30 problems are randomly generated. A relative error on objective value (REO) and a relative error on computational time (RET) are used as performance measures:

$$\mathrm{REO} = \frac{S_{\mathrm{GRHS}} - S_{\mathrm{MIP}}}{S_{\mathrm{MIP}}} \tag{12}$$

where

S_{GRHS} is the total cost of the solution obtained by GRHS.

S_{MIP} total cost of the solution obtained by solving the original MIP problem using XpressMP solver. Note that while problem types I and II solutions are optimal, the type III solution may not be optimal because we stop XpressMP after 12 h of running time.

$$\mathrm{RET} = \frac{T_{\mathrm{GRHS}} - T_{\mathrm{MIP}}}{T_{\mathrm{MIP}}} \tag{13}$$

where

T_{GRHS} and T_{MIP} are the computational times required to obtain S_{GRHS} and S_{MIP} of equation (12), respectively.

All the algorithms proposed in this paper were implemented in XpressMP and calculated on a Pentium 3.0-GHz machine with a 4-GB main memory.

4.2.1 Results and findings

Figure 6 shows the GRHS performance results compared to the MIP solver on total costs and computational times for various decision parameters. Results show that for problem types I and II, *RSize* and *TFence* do not affect the performances unless *RSize* is bigger than *TFence*. For type III, as *TFence* increases, GRHS performs better for all *RSize* values. However, when the *RSize* value is 6, *TFence* does not impact the REO except when the value of *TFence* is 3. Regarding RET, in most cases, a larger *RSize* and a smaller *TFence* work better. From these results, the values for *RSize* and *TFence* are chosen to be 6 and 6 and are used for the comparison with meta-heuristics in the next section.

Figures 7 and 8 represent the performance results of GRHS with respect to the MIP solver on REO and RET for three levels of *maxitr*. As seen in the figures, a larger *maxitr* gives better GRHS results for all problem types. However, for problem type III, the average REO improvement is relatively small when the *maxitr* value is changed from 3 to 5. Similarly, for problem type III, the average RET is not increased substantially when the *maxitr* value is increased from 1 to 3.

Table 3 Multiple range tests (Method 99.0 % Duncan)

Algorithms	Count	Mean	Homogeneous groups
GRHS	100	0.000158	X
SA	100	0.001299	X
TS	100	0.001547	X
GA	100	0.023259	X

However, that is not the case when the *maxitr* value is increased from 3 to 5. Therefore, the appropriate *maxitr* value is selected as 3 in this study.

4.3 Comparison with the existing meta-heuristics

Using problem type III, this section compares GRHS with the three existing meta-heuristic algorithms: tabu search (TS), simulated annealing (SA), and genetic algorithms (GA). Detailed procedures for TS, SA, and GA employed for the comparison are described in Appendix. In addition, comprehensive overviews of TS, SA, and GA can be found in Glover [11, 12], Kirkpatrick et al. [16], and Vergana et al. [21].

Because problem types I and II are not practical problems, only problem type III cases are tested for comparison. One hundred random type III problems are generated. Note that it is meaningless to perform computational time analysis because GRHS with *maxitr*=3 and (*RSize*, *TFence*=(6,6)) for type III, is approximately 33 % (i.e., planning horizon / *TFence*×100) faster than TS, SA, and GA that have *maxitr*×*K*×*J*×*T* iterations in total. Therefore, we only take into account the comparison of the algorithms' objective function values.

We use the STATGRAPHICS Centurion XV software package to perform the analysis of variance (ANOVA). If a factor is significant, the Duncan multiple range test is then performed. All of the statistical analyses use a significance level of α=0.01. In addition, we measure a relative deviation error (RDE) as follows:

$$\mathrm{RDE} = \frac{S_{\mathrm{Algorithm}} - S_{\mathrm{Best}}}{S_{\mathrm{Best}}} \tag{13}$$

where

$S_{\mathrm{Algorithm}}$ is the total cost of the solution obtained by GRHS, TS, SA, or GA.

S_{Best} is the best total cost found during execution.

We use "$A>B$" to represent that algorithm A is significantly better than algorithm B and "$A=B$" to represent that algorithms A and B do not show a statistically significant difference. Duncan's test, which is presented in Table 3, shows GRHS>SA=TS>GA.

5 Conclusion

In this paper, we have considered a multi-level, multi-item capacitated lot-sizing problem in the supply chain network that takes back order into account. To solve the problem, we have proposed an efficient hybrid heuristic algorithm named GRHS, which is a hybrid rolling horizon heuristic algorithm. Computational results have shown that the proposed algorithm not only finds excellent solutions for problems of realistic size in a reasonable computation time but also is superior to existing meta-heuristic algorithms, such as tabu search, simulated annealing, and genetic algorithms. Extending the GRHS algorithm to supply chain planning with demand uncertainty and developing a collaborative coordination methodology under a partial information sharing environment would be interesting to pursue in further research.

Appendix

Procedures for comparative meta-heuristics

Evaluate procedures used in comparative meta-heuristics

```
evaluate ( S , f_LP , V , improved , H )
begin
 improved := false
 f_LP := solution{LP_Solver( S )}                    // solve LP by simplex method (XPRESS-MP)
if f_LP = infeasible then return                     // infeasible
if f_LP < V then                                     // improved solution
    improved := true                                 // set improved
    V := f_LP                                        // update V
    H := H ∪ S                                       // add S to H
end if
end
```

Tabu search (TS)

TS ($maxitr$)

S^* is the current setup pattern and consists of:

y_{kjt} binary integer variable about whether or not to produce for all k, j and t

S^{**} is the best neighbor of S^*

V^{**} the objective function value for S^{**}

V is the best objective function value found yet

L_{kjt} the tabu value for setup variable y_{kjt}, where L_{kjt} = 0 if the move is not tabu

L^* is the tabu tenure value for a move that has just been made tabu

begin

$H := \varnothing$

$V := \infty$ // initialize V

$L_{kjt} := 0 \ \forall k, j, t$ // initialize tabu values

$L^* := 7$ // set tabu tenure for new move to 7

initialize S^* // initialize starting solution by randomly setting y_{kjt} to 0 or 1

evaluate (S^*, f_{LP}, V, $improved$, H) // evaluate the initial solution

for itr :=1 to $maxitr$ **do** // iterate for $maxitr$

  $V^{**} := \infty$ // initialize best neighbor(V^{**}) value

  for all k, j and t **do** // examine neighbors of S^*

    if y_{kjt} = 0 **then** // switch setup pattern y_{kjt} 0→1 or 1→0 for particular k, j and t

      y_{kjt} := 1 **and** flag := 1

    else

      y_{kjt} := 0 **and** flag := 0

    end if

    evaluate (S^*, f_{LP}, V, $improved$, H) // evaluate the changed setup pattern

    if $improved$ **then** L_{kjt} := 0 // aspiration – make the move non-tabu

    if L_{kjt} = 0 and $f_{LP} < V^{**}$ **then** // if improved non-tabu neighbor

      $V^{**} := f_{LP}$ // update V^{**} with f_{LP}

      $S^{**} := S^*$ // update S^{**} with S^*

      $k' := k$ // record move

      $j' := j$

      $t' := t$

    else // else if does not get improved or the move is tabu

      if flag = 1 **then** // then restore the changed setup pattern to the previous value

        y_{kjt} := 0

      else

        y_{kjt} := 1

      end if

    end if

  end for

  if $V^{**} = \infty$ **then**

    finished with current solution S^* // no improved non-tabu moves

  else

    $S^* := S^{**}$ // take the best neighbor found

    L_{kjt} := max[0, $L_{kjt} - 1$] $\forall k, j, t$ // reduce all tabu tenures by one

    $L_{k'j't'} := L^*$ // set tabu move

  end if

end for

end

Simulated annealing (SA)

```
SA ( maxitr )
begin
H := ∅
V := ∞                                                    // initialize V
initialize S*                                             // initialize starting solution by randomly setting y_kjt to 0 or 1
evaluate ( S*, f_LP, V, improved, H )                     // evaluate the initial solution
Temp := |V/10|                                            // initialize SA parameters
β := 0.95
for itr :=1 to maxitr do                                  // iterate for maxitr
   V** := ∞                                               // initialize best neighbor(V**) value
  for all k, j and t do                                   // examine neighbors of S*
     if y_kjt = 0 then                                    // switch setup pattern y_kjt 0→1 or 1→0 for particular k, j and t
          y_kjt := 1 and flag := 1
     else
          y_kjt := 0 and flag := 0
     end if
     evaluate ( S*, f_LP, V, improved, H )                // evaluate the changed setup pattern
     if f_LP < V** then                                   // if better than current solution S*
          V** := f_LP                                     // update V** with f_LP
          S** := S*                                       // update S** with S*
     else                                                 // else if does not get improved
          r := a random number drawn from [0,1]
          if r < exp[−(f_LP − V**)/Temp] then
               V** := f_LP                                // update V** with f_LP
               S** := S*                                  // update S** with S*
          else
               if flag = 1 then                           // then restore the changed setup pattern to the previous value
                    y_kjt := 0
               else
                    y_kjt := 1
               end if
          end if
     end if
  end for
  if V** = ∞ then
     finished with current solution S*                    // no improved moves
  else
     Temp := β*Temp                                       // reduce temperature
  end if
end for
end
```

Genetic algorithm

```
GA ( maxitr )
P            is the population
S*, S**      are two solutions (parents) selected from the population to mate
C            is the offspring (child) of S* and S** consists of:
             y_kjt  binary integer variable about whether or not to produce for all k, j and t
begin
H := ∅
V := ∞                                                        // initialize V
initialize P := { S_1,...,S_100 }          // random initialization, 100 individuals in the population
set
evaluate ( S_p, f^p_LP, V, improved, H ) p = 1,...,100        // evaluate the solutions
for itr := 1 to maxitr * K * J * T  do                        // maxitr * K * J * T iterations in all
   select S*, S** ∈ P  by binary tournament method
   crossover C := ( S*, S** ) by uniform method
   randomly choose k, j and t
   if y_kjt = 1 then y_kjt := 0   else y_kjt := 1             // mutation
   evaluate ( C, f^C_LP, V, improved, H )                     // evaluate child
   find i such that f^Si_LP = max[ f^Sp_LP | p = 1,...,100]   // find the worst population member
   S_i := C                                                   // replace the worst member with child
end for
end
```

References

1. Akartunali K, Miller AJ (2009) A heuristic approach for big bucket multi-level production planning problems. Eur J Oper Res 193:396–411
2. Almada-Lobo B, James RJW (2010) Neighbourhood search meta-heuristics for capacitated lot-sizing with sequence-dependent setups. Int J Prod Res 48:861–878
3. Almeder C (2010) A hybrid optimization approach for multi-level capacitated lot-sizing problems. Eur J Oper Res 200: 599–606
4. Belvaux G, Wolsey LA (2001) Modeling practical lot-sizing problems as mixed-integer programs. Manag Sci 47:993–1007
5. Chan FTS, Chung SH (2004) A multi-criterion genetic algorithm for order distribution in a demand driven supply chain. Int J Comput Integr Manuf 17(4):339–351
6. Chan FTS, Tibrewal RK, Prakash A, Tiwari MK (2014) A biased random key genetic algorithm approach for inventory-based multi-item lot-sizing problem. J Eng Manuf. doi:10.1177/0954405414523594
7. Danna E, Rothberg E, Pape CE (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. Math Program 102:71–90
8. Ertogral K, Wu SD (2000) Auction-theoretic coordination of production planning in the supply chain. IIE Trans 32:931–940
9. Federgruen A, Meissner J, Michal T (2007) Progressive interval heuristics for multi-item capacitated lot-sizing problems. Oper Res 55:490–502
10. Fischetti M, Lodi A (2008) Repairing MIP infeasibility through local branching. Comput Oper Res 35:1436–1445
11. Glover F (1989) Tabu search—part I. ORSA J Comput 1:190–206
12. Glover F (1990) Tabu search—part II. ORSA J Comput 2:4–32
13. Goren HG, Tunali S, Jans R (2012) A hybrid approach for the capacitated lot sizing problem with setup carryover. Int J Prod Res 50:1582–1597
14. Helber S, Sahling F (2010) A fix-and-optimize approach for the multi-level capacitated lot sizing problem. Int J Prod Econ 123: 247–256
15. Hung YF, Chien KL (2000) A multi-class multi-level capacitated lot sizing model. J Oper Res Soc 51:1309–1318
16. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
17. Li Y, Tao Y, Wang F (2012) An effective approach to multi-item capacitated dynamic lot-sizing problems. Int J Prod Res 50:5348–5362
18. Maes J, McClain JO, Van Wassenhove LN (1991) Muti-level capacitated lotsizing complexity and LP-based heuristics. Eur J Oper Res 53:131–148
19. Sahling F, Buschkuhl L, Tempelmeier H, Helber S (2009) Solving a multi-level capacitated lot sizing problem with multi-period setup carry-over via a fix-and-optimize heuristic. Comput Oper Res 36: 2546–2553
20. Stadtler H (2003) Multilevel lot sizing with setup times and multiple constrained resources: internally rolling schedules with lot-sizing windows. Oper Res 51:487–502
21. Vergana FE, Khouja M, Michalewicz Z (2002) An evolutionary algorithm for optimizing material flow in supply chain. Comput Ind Eng 43:407–421
22. Wu T, Shi L (2011) Mathematical models for capacitated multi-level production planning problems with linked lot sizes. Int J Prod Res 49: 6227–6247
23. Wu CH, Lin JT, Wu HH (2010) Robust production and transportation planning in thin film transistor-liquid crystal display (TFT-LCD) industry under demand and price uncertainties. Int J Prod Res 48: 6037–6060