Omega 56 (2015) 25–36

Contents lists available at ScienceDirect

Omega

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

Fix-and-optimize and variable neighborhood search approaches for multi-level capacitated lot sizing problems ☆

Haoxun Chen *

Industrial Systems Optimization Laboratory, Charles Delaunay Institute and UMR CNRS 6281, University of Technology of Troyes, 12 rue Marie Curie, CS 42060, 10004 Troyes, France

ARTICLE INFO

Article history:
Received 14 September 2013
Accepted 8 March 2015
Available online 16 March 2015

Keywords:
Production planning
Lot sizing
Fix-and-optimize
Variable neighbourhood search
Mixed integer programming

ABSTRACT

In this paper, a new fix-and-optimize (FO) approach is proposed for two dynamic multi-level capacitated lot sizing problems (MLCLSP), the MLCLSP without setup carryover and the MLCLSP with setup carryover. Given an MIP model of a lot sizing problem, the approach iteratively solves a series of sub-problems of the model until no better solution can be found. Each sub-problem re-optimizes a subset of binary decision variables determined based on the interrelatedness of binary variables in the constraints of the model, while fixing the values of the other binary variables. Based on the FO, a variable neighbourhood search (VNS) approach for the MLCLSP without setup carryover is also developed, which can further improve the solution obtained by the FO by diversifying the search space. Numerical experiments on benchmark instances show that both our FO and VNS approaches can obtain a better solution for most instances compared with that found by the fix-and-optimize approach proposed by Helber and Sahling (International Journal of Production Economics 2010;123:247–256).

1. Introduction

Production planning is one of the most important decisions for manufacturers. It determines how many units of each component/final product should be produced internally or procured from outside suppliers in each period over a given planning horizon, with the objective to minimize the total cost, while meeting customer demand on time. In most enterprise resource planning (ERP) systems, production plans are made using backward scheduling without considering resource capacity constraints as in material requirements planning (MRP) systems. In such planning systems, it is assumed that the lead time for each production/procurement operation is predefined without taking account of the resource capacity constraints of the operation. This might lead to either an infeasible production plan or a plan with excessive inventory. Recently, so-called advanced planning systems or advanced planning and scheduling systems (APS) have emerged as a powerful tool for supply chain planning that considers resource capacity constraints. However, to our best knowledge, most APS systems use a heuristics algorithm to generate a production plan. The performance of such an algorithm cannot be guaranteed in terms of solution quality. Therefore, there is a strong demand for the development of effective and efficient production planning methods that can be implemented in APS systems.

For a manufacturing system which produces final products from raw materials through components, its production planning problem can be formulated as a dynamic multi-level capacitated lot sizing problem (MLCLSP) if the production process is characterized by substantial setup costs and/or setup times. In the MLCLSP, it is assumed that the demand of each final product is known for each period over a planning horizon of multiple periods. The demand must be satisfied on-time. The relationships between the items (raw materials, components, and final products) in the system are given by its bills of materials (BOM). The production of each item in each period requires a manufacturing resource such as a machine, where each resource has a limited capacity. The production of each item in each period requires the setup of the corresponding resource, which incurs a setup time and a setup cost. The objective of the problem is to determine a production plan over the planning horizon for the system such that its total cost including inventory holding costs and setup costs is minimized. The MLCLSP does not consider any setup carryover between two successive periods. As an extension, the MLCLSP with linked lot sizes or the MLCLSP-L for short allows the setup state of a resource to be carried over from the current period to the next period. Both the MLCLSP and the MLCLSP-L are NP-hard. Their

☆This manuscript was processed by Associate Editor Kuhn.

* Tel.: +33 325715642; fax: +33 325715649.
E-mail address: haoxun.chen@utt.fr

http://dx.doi.org/10.1016/j.omega.2015.03.002
0305-0483/

high complexity has been recognized through the studies in the past three decades. Up to now, for many benchmark instances proposed by Tempelmeier and Derstroff [1] for the MLCLSP, their integrality gap still remains high. Trigeiro et al. [2] pointed out that for the capacitated lot sizing problem (CLSP) with setup times, its feasibility problem (the problem to check whether it has a feasible solution) is NP-complete. As the MLCLSP and the MLCLSPL can be reduced to the CLSP with setup times, they are at least as hard as the CLSP and hence also *NP*-hard.

In this paper, we study both the MLCLSP and the MLCLSP-L and propose a new fix-and-optimize (FO) approach different from those of Helber and Sahling [3] and Sahling et al. [4]. This is an iterative approach. Given an MIP model of a problem, this approach first constructs an initial feasible solution of the model by simply setting its all binary setup variables to one as in [3,4]. In each iteration of the approach, a binary setup variable of the model is randomly selected and tagged as a binary variable to be re-optimized. At the same time, all binary setup variables related to the selected variable in the model are also tagged as binary variables to be re-optimized, whereas all other binary variables are fixed at their values obtained in the last iteration. This leads to an MIP submodel (subproblem) with fixed binary variables, binary variables to be re-optimized, and real variables (also to be re-optimized). The submodel is solved by using an MIP solver and the current best solution of the original MIP model is replaced by the optimal solution of the submodel if the latter solution is better. This iterative procedure continues until no submodel has a solution better than the current best solution of the MIP model. Based on the fix-and-optimize approach, a variable neighbourhood search (VNS) approach for the MLCLSP without setup carry-over is also developed, which can further improve the solution obtained by the FO by diversifying the search space.

Compared with the fix-and-optimize approach proposed by Helber and Sahling [3] for the MLCLSP and that proposed by Sahling et al. [4] for the MLCLSP-L, our FO approach selects the binary variables to be re-optimized in an MIP model of a lot sizing problem based on the interrelatedness of binary setup variables in the constraints of the model rather than based on three problem-specific decompositions (product, resource, and process-oriented decompositions). Our FO approach is thus more general than theirs and can be applied to other 0–1 MIP models. Numerical experiments on benchmark instances show that our FO approach can obtain a better solution in a similar computation time for most instances compared with that found by the approach of Helber and Sahling. Moreover, our VNS approach can further improve the solution obtained by the FO by diversifying the search space. It outperforms the variable neighbourhood decomposition search approach proposed by Zhao et al. [5] for the MLCLSP. For the MLCLSP-L, numerical experiments on benchmark instances show that our FO approach is competitive with the fix-and-optimize approach of Sahling et al. [4] and the corridor approach of Caserta and Voß [6].

The rest of this paper is organized in seven sections. Section 2 presents the literature related to the present work. Section 3 formulates the MLCLSP and the MLCLSP-L. Our FO and VNS approaches are described in Sections 4 and 5, respectively. Sections 6 and 7 present numerical experiments of the two approaches on benchmark instances of the MLCLSP and the MLCLSP-L. The concluding section provides some remarks on future research.

2. Related literature

The MLCLSP was first introduced by Billington et al. [7], who proposed a 0–1 mixed integer programming model for the problem. Since then, various models and methods have been proposed to solve the problem. Akartunali and Miller [8] presented an extensive survey of these models and established relationships between them. Buschkühl et al. [9] gave a comprehensive review of various solution methods for the MLCLSP, whereas the review of Jans and Degraeve [10] focused on meta-heuristics for the MLCLSP and other lot sizing problems. As for the MLCLSP-L, its studies were more recent with less literature. A review of the studies before 2008 was given by Tempelmeier and Buschkühl [11]. Models and algorithms for capacitated single level lot sizing problems were reviewed by Karimi et al. [12], and those for coordinated deterministic dynamic demand lot-sizing problems were reviewed by Robinson et al. [13]. In the latter problems, a joint shared fixed setup cost is incurred each time one or more items of a product family are replenished, and a minor setup cost is charged for each item replenished. In the following, we only review previous studies on the MLCLSP and the MLCLSP-L that are related to our present work, especially the studies on mixed integer programming (MIP) based heuristics and variable neighbourhood search approaches for the two problems.

MIP-based heuristics for the MLCLSP or the MLCLSP-L solve a series of MIP subproblems (submodels) whose number of binary variables is much less than that of the original MIP model of the problem. Among these heuristics, relax-and-fix approaches and fix-and-optimize approaches are mostly used. Relax-and-fix heuristics reduce the number of binary variables to be optimized simultaneously in an MIP model of the MLCLSP by dividing it into several subproblems based on a time-oriented decomposition with moving time windows (Belvaux and Wolsey [14], Stadtler [15], Sürie and Stadtler [16]).

Pochet and Wolsey [17] may be the first ones who used an iterative fix-and-optimize approach to solve a production planning problem. This approach iteratively solves a series of MIP subproblems which optimize only a small subset of binary setup variables of the original MIP model of the problem. The MIP subproblem in each iteration is derived by fixing most binary variables of the original MIP model to their values obtained in the last iteration. This approach is easy to be implemented with a commercial MIP solver such as CPLEX or XPRESS. Later, Helber and Sahling [3] proposed a fix-and-optimize approach for the MLCLSP. Their approach considers three types of subproblems defined based on three problem-specific decompositions (i.e., product, resource, and process-oriented decompositions). Each subproblem of the first type is derived by fixing all setup variables except the setup variables related to a product. Each subproblem of the second type is derived by fixing all setup variables except the setup variables related to a resource and four periods. Each subproblem of the third type is derived by fixing all setup variables except the setup variables related to a product, one of its immediate successors, and one half of the planning horizon. Numerical experiments show that the fix-and-optimize approach outperforms the Lagrangean heuristic of Tempelmeier and Derstroff [1] and the fix-and-relax heuristic of Stadtler [15]. Sahling et al. [4] applied a similar fix-and-optimize approach to the MLCLSP-L and obtained very good results on the 1920 benchmark instances proposed by Tempelmeier and Buschkühl [11]. Stadtler and Sahling [18] presented a hybrid approach based on fix-and-relax and fix-and-optimize for a lot-sizing and scheduling problem of multi-stage flow lines with zero lead times. In addition, Chen and Chu [19] proposed a hybrid approach that combines Lagrangian relaxation with a linear programming pivot-based local search for the MLCLSP without setup time. Caserta and Voß [6] proposed an MIP-based corridor approach for the MLCLSP-L based on soft variable fixing.

Various meta-heuristics have also been used to solve the MLCLSP or other lot sizing problems (Buschkühl et al. [9] and Jans and Degraeve [10]), one of them is the variable neighbourhood

search (VNS) proposed by Hansen and Mladenović [20]. The VNS is a local search based meta-heuristic which systematically changes the neighbourhood structure of local search and which uses a random shaking routine to create a new starting solution for a local search every time when it reaches a local optimum. Contrary to other local search based meta-heuristics, the VNS does not follow a pre-specified trajectory but explores increasingly distant neighbourhoods of the current incumbent solution. Hindi et al. [21] proposed a heuristic that combines the VNS and Lagrangian relaxation for the CLSP with setup times. Xiao et al. [22] used the VNS with distance-based neighbourhood structures to solve the uncapacitated multilevel lot-sizing problem, a problem much easier than the MLCLSP. Zhao et al. [5] used the variable neighbourhood decomposition search (VNDS), a variant of the VNS, to solve the MLCLSP. In their approach, each MIP subproblem is derived based on the linear relaxation solution of the original MIP model of the problem and by adding a distance constraint. Seeanner et al. [23] proposed a hybrid approach that combines the VNDS and fix-and-optimize for a general lot sizing and scheduling problem with multiple production stages. Their VNDS decomposes the problem by fixing only a subset of variables at each local search iteration. Similar to the FO approach of Helber and Sahling [3], the subset of variables to be relaxed is determined based on product, resource, or process-oriented decomposition but the number of variables to be relaxed is controlled by the VNDS. Other meta-heuristics for the MLCLSP or similar lot-sizing problems include the ant-based approach of Almeder [24], the hybrid adaptive large neighborhood search approach of Muller et al. [25], the block-chain based tabu search algorithm of Li et al. [26], and the column generation heuristic of Tempelmeier [27].

3. Problem formulation

The MLCLSP and the MLCLSP-L have been explained informally in the introduction. In this section, we formulate the two problems mathematically. For the MLCLSP, we adopt the formulation of Billington et al. [7] with overtime, where inventory holding costs, setup costs, overtime costs, setup times, and unit production times are assumed to be stationary, i.e., independent of each time period. This formulation is referred to as model MLCLSP hereafter. The following notations will be used in the model.########Numbers, indices, and index sets

N set of items (including final products, intermediate products/components, and raw materials) to be produced or procured, with $N=\{1, 2, \ldots, N\}$, where N is the number of items considered;
T set of periods in the planning horizon, with $T=\{1, 2, \ldots, T\}$, where T is the number of periods considered;
K set of resources (machines) used in the production, with $K=\{1, 2, \ldots, K\}$, where K is the number of resources considered;
i, j item or operation index, $i, j=1,\ldots, N$;
t period index, $t=1,\ldots,T$;
k resource (machine) index, $k=1,\ldots, K$;
m_i resource (machine) used by item i;
S_i set of immediate successors of item i in the bill of materials;
P_i set of immediate predecessors of item i in the bill of materials;
N_k set of items produced by resource k.

Parameters

d_{it} demand of item i in period t;
a_{ij} number of units of item i required to produce one unit of its immediate successor item j;
lt_i production lead time of item i;
pt_{ki} production time required by resource k to produce each unit of item i;
st_{ki} setup time for resource k used for the production of item i;
b_{kt} available capacity of resource k in period t (in units of time);
hc_i holding cost for one unit of item i in each period;
sc_i setup cost for the resource used to produce item i;
oc_k unit overtime cost of resource k;
B_{it} large positive number.

Variables

I_{it} inventory level of item i at the end of period t;
X_{it} production quantity of item i in period t;
Y_{it} binary setup variable; $Y_{it}=1$ if item i is produced in period t and $Y_{it}=0$ otherwise;
O_{kt} overtime of resource k in period t.

With the above notations, the MLCLSP can be formulated as follows:

Model MLCLSP:

$$\text{Min} \sum_{i=1}^{N}\sum_{t=1}^{T}(hc_i I_{it}+sc_i Y_{it})+\sum_{k=1}^{K}\sum_{t=1}^{T} oc_k O_{kt} \tag{1}$$

$$I_{i,t}=I_{i,t-1}+X_{i,t-lt_i}-\sum_{j\in S_i} a_{ij}X_{jt}-d_{it},\ \forall i=1,\ldots,N, t=1,\ldots,T, \tag{2}$$

$$\sum_{i\in N_k} pt_{ki}X_{it}+\sum_{i\in N_k} st_{ki}Y_{it}\le b_{kt}+O_{kt},\ \forall k=1,\ldots,K, t=1,\ldots,T, \tag{3}$$

$$X_{it}\le B_{it}Y_{it},\ \forall i=1,\ldots,N, t=1,\ldots,T, \tag{4}$$

$$\left.\begin{array}{l} I_{it}\ge 0 \\ X_{it}\ge 0 \\ Y_{it}\in\{0,1\}\end{array}\right\},\ \forall k=1,\ldots,K, t=1,\ldots,T, \tag{5}$$

The objective function (1) to be minimized is the sum of inventory holding costs, setup costs, and overtime costs. Constraints (2) are the inventory-balance equations ensuring that the demand of each item in each period is fulfilled without backlogging. Constraints (3) give the capacity constraint of each resource in each period with overtime. Constraints (4) are the coupling constraints linking each production variable X_{it} with its corresponding setup variables Y_{it}, where the choice of each large positive number B_{it} must not limit any feasible production quantity of item i in period t. The coupling constraints imply that $X_{it}=0$ if $Y_{it}=0$ for all i and t. The nonnegative real or binary nature of each variable in the model is indicated by constraints (5).

The MLCLSP-L allows the setup state of each resource to be carried over from the current period to the next period. To formulate the MLCLSP-L, additional binary variables indicating setup carryovers and additional constraints linking the setup state variables with the setup carry-over variables are required. In addition, the inventory-balance constraints (2) in model MLCLSP cannot simply be extended to those in model MLCLSP-L, because the latter model has special boundary conditions for the inventory level of each item at the beginning of the first period and the end of the last period. For the MLCLSP-L, we adopt the formulation of Caserta and Voß [6] with overtime, which is equivalent to the formulation of Sahling et al. [4] and will be referred to as model

MLCLSP-L hereafter. In order to formulate the model, except for the notations used in model MLCLSP, we introduce additional parameters and variables as follows:Additional parameters

$\underline{I}_i$ physical initial inventory of item i
N_k number of items produced by resource k, $N_k=|N_k|$

Additional decision variables

Z_{it} binary setup carry-over variable for item i at the beginning of period t

Model MLCLSP-L

$$\text{Min} \sum_{i=1}^{N}\sum_{t=1}^{T}\left[hc_i I_{it}+sc_i(Y_{it}-Z_{it})\right]+\sum_{k=1}^{K}\sum_{t=1}^{T} oc_k \times O_{kt} \tag{6}$$

$$\text{s.t.}\quad I_{i,t-1}+X_{it}-\sum_{j\in S_i} a_{ij}\times X_{j,t+1}-I_{i,t}=d_{it},\ \forall i,t=1,\ldots,T-1 \tag{7}$$

$$I_{i,T-1}+X_{iT}-I_{i,T}=d_{iT},\ \forall i \tag{8}$$

$$\mathrm{I}_{\mathrm{i}}-\sum_{j\in S_i} a_{ij}\times X_{j,1}-I_{i,0}=0,\ \forall i \tag{9}$$

$$\sum_{i\in N_k}\left[pt_i\times X_{it}+st_i\times(Y_{it}-Z_{it})\right]\le b_{kt}+O_{kt},\ \forall k,t \tag{10}$$

$$X_{it}\le B_{it}\times Y_{it},\ \forall i,t \tag{11}$$

$$\sum_{i\in N_k} Z_{it}\le 1,\ \forall k,t \tag{12}$$

$$Z_{it}\le Y_{i,t-1},\ \forall i,t \tag{13}$$

$$Z_{it}\le Y_{i,t},\ \forall i,t \tag{14}$$

$$N_k(2-Z_{it}-Z_{i,t+1})+1\ge\sum_{i'\in N_k} Y_{i',t},\ \forall k,t,i\in N_k, \tag{15}$$

$$Z_{i1}=0,\ \forall i \tag{16}$$

$$X_{it}\ge 0, Y_{it}\in\{0,1\}, Z_{it}\in\{0,1\},\ \forall i,t \tag{17}$$

In the above model, Y_{it} is a binary setup state variable which has the same meaning as that of the setup variable Y_{it} in model MLCLSP; Z_{it} is the binary setup carry-over variable for item i at the beginning of period t. Product i can be produced in period t only if its required resource has the setup state $Y_{it}=1$. This setup state is either carried over from the preceding period $t-1$ with $Z_{it}=1$ or is resulted from a new setup operation performed in period t with $Z_{it}=0$. Note that Z_{it} is 1 if both $Y_{i,t-1}$ and Y_{it} are 1. A setup state can be carried over multiple consecutive periods. Each setup operation incurs a setup cost and a setup time.

In model MLCLSP-L, constraints (7)–(9) are inventory balance equations, where Eqs. (8) and (9) give the special boundary conditions for the inventory level of each item at the beginning of the first period and the end of the last period. The introduction of the physical initial inventory for each item i, $\underline{I}_i$, is to take account of the existence of one period production lead time of the item. $\underline{I}_i$ must cover at least the total demand from all successors of item i in the first period (period 1), i.e., $\sum_{j\in S_i} a_{ij}\times X_{j,1}$. $I_{i,0}$ is the initial inventory of item i after deducting the total demand from $\underline{I}_i$. Constraints (10) give the capacity constraints of all resources. Constraints (11) indicate that each item can be produced in a period only if the required resource is in the setup state. Constraints (12) imply that in each period, the setup carry-over of a resource is possible only for at most one item. Constraints (13) and (14) indicate that the setup carry-over of a resource for item i occurs in period t only if the resource is set up for the item in both periods $t-1$ and t. Constraints (15) indicate multi-period setup carryovers. Constraints (16) state that no resource is in its setup state at the beginning of the first period. The readers can refer to Caserta and Voß [5] and Sahling et al. [3] for a more detailed explanation of the model.

4. Fix-and-optimize

The fix-and-optimize approach proposed by Helber and Sahling [3] is based on three types of problem-specific decompositions: product, resource and process-oriented decompositions. In this section, we define MIP subproblems in a more general way, based on which a new fix-and-optimize approach for the MLCLSP and the MLCLSP-L will be proposed. In this approach, each subproblem is defined based on the interrelatedness of binary setup variables in the constraints of model MLCLSP or model MLCLSP-L given in the last section. In the following, we first present our fix-and-optimize approach for the MLCLSP and then extend it to the MLCLSP-L. For simplicity of exposition, the lead time for the production of each item in model MLCLSP is assumed to be zero, but this approach can be extended to the model with positive lead times.

4.1. Interrelatedness between binary setup variables

In model MLCLSP, binary setup variables are interrelated through inventory balance constraints (2), resource capacity constraints (3), and coupling constraints (4). If the value of setup variable Y_{it} changes, the value of its corresponding production variable X_{it} may also change due to constraints (4). The change of X_{it} may cause the change of $X_{i,t-1}$ and $X_{i,t+1}$ due to the inventory linkage between two consecutive periods imposed by constraints (2). This change may also cause the change of $X_{i't}$, $i'\in P_i$ and $i'\in S_i$ due to the linkage of $X_{i't}$ and X_{it} in the bill of materials imposed by the same constraints. Consequently, the change of the value of setup variable Y_{it} may cause the change of the values of $Y_{i,t-1}$, $Y_{i,t+1}$, and $Y_{i't}$, $i'\in P_i$ and $i'\in S_i$. Similarly, the change of the value of Y_{it} may cause the change of the values of $Y_{i't}$, $i'\in N_{m_i}$ due to constraints (3), where N_{m_i} is the set of items which are produced using the same resource as that of item i, i.e., resource m_i.

From the above observations, we define the interrelatedness between binary setup variables as follows. We say two setup variables Y_{it} and $Y_{i't'}$ are directly interrelated or 1-interrelated if one of the following three conditions holds: (1) $i'=i$ and $t'\in\{t-1, t, t+1\}$; (2) $i'\in P_i\cup S_i$ and $t'=t$; (3) $i'\in N_{m_i}$ and $t'=t$. By default, setup variable Y_{it} is 1-interrelated with itself. With this definition, the set of setup variables that are 1-interrelated with Y_{it}, denoted by $\mathrm{IR}(Y_{it})$, is given by

$$\mathrm{IR}(Y_{it})=\{Y_{it-1}, Y_{it}, Y_{it+1}\}\cup\{Y_{i't}\mid i'\in P_i\cup S_i\}\cup\{Y_{i't}\mid i'\in N_{m_i}\}.$$

For any integer $l\ge 1$, we say two setup variables Y_{it} and $Y_{i't'}$ are l-interrelated if and only if there is a series of setup variables $Y_{i_jt_j}(j=0,1,\ldots,l)$ such that

(1) $(i_0, t_0)=(i, t)$ and $(i_l, t_l)=(i',t')$,
(2) $Y_{i_jt_j}$ and $Y_{i_{j+1}t_{j+1}}$ are 1-interrelated for $0\le j\le l-1$.

Note that in the above definition, multiple identical variables $Y_{i_jt_j}$ may appear in the series. Since each setup variable Y_{it} is 1-interrelated with itself, we can easily derive that if two binary variables Y_{it} and $Y_{i't'}$ are l-interrelated, then they are also l'-interrelated for any integer $l'>l$. Two setup variables Y_{it} and $Y_{i't'}$ are called interrelated if there is a finite integer $l\ge 1$ such that they are

l-interrelated. The interrelatedness defined above is reflexive, transitive and symmetric.

The definition of such interrelatedness between two setup variables is motivated by the fact that the change of the value of one setup variable (production variable) may lead to the change of the value of another setup variable (production variable) if they are coupled with each other through a constraint.

4.2. Definition of subproblems

Let $\Psi=\{Y_{it} \mid i\in N, t\in T\}$ denote the set of all setup variables. For any subset $S\subseteq\Psi$, we define $IR^l(S)$ as the set of setup variables that are l-interrelated with a binary variable in S. We then have $IR^{l+1}(S)=IR(IR^l(S))$. Particularly, when $S=\{Y_{it}\}$, i.e., S is a singleton, we write $IR^l(S)$ as $IR^l(Y_{it})$. $IR^l(Y_{it})$ is the set of setup variables that are l-interrelated with Y_{it}, where Y_{it} is called the seed binary variable of $IR^l(Y_{it})$. Define $FB^l(Y_{it})=\Psi\backslash IR^l(Y_{it})$, which is the set of binary setup variables that are not l-interrelated with Y_{it}.

With the above definition of interrelatedness between setup variables, we can now define the subproblems for our fix-and-optimize approach for the MLCLSP. For a given positive integer $l\geq 1$ and a setup variable Y_{it}, the subproblem of level l associated with Y_{it}, denoted by subproblem $SP^l_{i,t}$, is derived by fixing all setup variables in $FB^l(Y_{it})$, while re-optimizing the remaining setup variables, i.e., the setup variables in $IR^l(Y_{it})$. In the following, $FB^l(Y_{it})$ and $IR^l(Y_{it})$ are called the set of binary variables to be fixed and the set of binary variables to be re-optimized respectively for the given seed binary variable Y_{it} and interrelatedness level l. Obviously, $IR^l(Y_{it})\subseteq IR^{l+1}(Y_{it})$, $k=1, 2, \ldots$. This implies that the bigger the parameter l, the larger the set $IR^l(Y_{it})$. That is, a bigger l implies that more binary setup variables are re-optimized in the corresponding subproblem.

The subproblems defined above are for the MLCLSP with lead time $lt_i=0$ for each item i. When lead time lt_i is positive for some items, $IR^l(Y_{it})$ for $SP^l_{i,t}$ can be defined similarly. The only change is that the second condition for defining the 1-interlatedness between two setup variables Y_{it} and $Y_{i't'}$ must be replaced by "$i'\in P_i$ and $t'=t-lt_{i'}$ or $i'\in S_i$ and $t'=t+lt_i$" to take account of the production lead time $lt_{i'}$ of each item $i'\in P_i$ required for the production of item i and the production lead time lt_i of item i required for the production of each item $i'\in S_i$.

To effectively and efficiently implement the fix-and-optimize approach, a trade-off must be made between its computation time and solution quality when we choose parameter l for each subproblem $SP^l_{i,t}$. In our numerical experiments, we take $l=1$, 2, or 3.

One major difference between our subproblem definition and the subproblem definition of Helber and Sahling [3] is that we consider simultaneously product, resource and process-induced interrelatedness between setup variables. Our subproblem definition is more general and flexible. It provides a new way to make a good trade-off between computation time and solution quality for a fix-and-optimize approach.

4.3. Fix-and-optimize algorithm for the MLCLSP

Similar to that of Helber and Sahling's [3], our fix-and-optimize algorithm also solves iteratively a series of subproblems. However, our algorithm differs from theirs in the subproblem definition and the number of MIP subproblems solved in each iteration. In our algorithm, the number of possible subproblems is $N\times T$, where N is the number of items and T is the number of periods. Initially, a setup is planned for each product in each period, i.e., we set $Y_{it}=1$ for all i and t. In each iteration, a setup variable Y_{it} or a pair (i, t) is randomly selected from $N\times T$ with the same probability for each element in $N\times T$, and the corresponding subproblem $SP^l_{i,t}$ is solved, where the level of interrelatedness l is a control parameter of the algorithm. The subproblem's solution is accepted if and only if it is better than the current best solution of the original problem. Here, the first solution is better than the second solution if one of the two conditions holds: (1) the first solution is capacity-feasible (i.e., without over time), and its cost is lower than the cost of the second solution; (2) the first solution yields a cost lower than that of the second solution which is not capacity-feasible. In other words, a capacity-infeasible solution is never considered a candidate for the best solution if there is already a known capacity-feasible solution. The algorithm will terminate if no improvement is observed after a certain number of iterations. This number, denoted by n, is another control parameter of the algorithm. The bigger the value of n, the larger the computation time of this algorithm. When $N\times T$ is big, it may be too time consuming for the algorithm to terminate after performing more than $N\times T$ iterations without improvement. For this reason, we take n equal to or smaller than $N\times T$ in our numerical experiments.

Let $\overline{Y}=\{\overline{Y}_{it}\}$ denote the values of all setup variables at a solution of model MLCLSP. For ease of presentation, $\overline{Y}$ is also called a setup plan or a solution of the model. This will not cause any ambiguity, because the values of the production variables $\{X_{it}\}$ and the overtime variables $\{O_{kt}\}$ of model MLCLSP can be determined according to $\overline{Y}$ by solving a corresponding linear programming model. Let *Iter* denote the number of iterations performed since the last improvement, our fix-and-optimize algorithm is presented in pseudo-code in Algorithm 1.

Algorithm 1. FO-Algorithm(l, n)

Find an initial feasible solution of model MLCLSP by setting $Y_{it}=1$ for all i and t;
Set the current best solution $\overline{Y}=\{\overline{Y}_{it}\}$ of the model to the feasible solution;
  Iter←0;
  Repeat
  Iter←*Iter*+1;
  Randomly choose a pair (i, t) from $N\times T$ with the same probability for each element in $N\times T$;
  Solve subproblem $SP^l_{i,t}$;
  If the solution of $SP^l_{i,t}$ is better than the current best solution $\overline{Y}$ of model MLCLSP, then
    Set $\overline{Y}$ to the solution of $SP^l_{i,t}$;
    Iter←0;
  End if
Until *Iter*≥n;

When parameter l is big, it may be too time-consuming for the FO-algorithm to solve each subproblem to optimality. In this case, we may limit the time used for solving each subproblem by a given time. Moreover, to speed up the resolution of each subproblem in the algorithm, model MLCLSP is strengthened (tightened) by adding some valid constraints.

For model MLCLSP, according to [28], for any $1\leq l\leq T$, $L=\{1, 2\ldots, l\}$, and $S\subseteq L$, the following (l, S) inequalities are valid.

$$\sum_{t\in S} x_{it}+\sum_{t\in L\backslash S} d_{i,t,l}y_{it}\geq d_{i,1,l},\ \forall i\in N, \tag{18}$$

where $d_{i,t,l}$ is the accumulative demand of item i from period t to period l, i.e., $d_{i,t,l}=\sum_{\tau=t}^{l} d_{i,\tau}$. Note that in (18), the demand of each item i in each period t is the echelon demand that includes the item's own demand and all demand from its successors in the bill of materials in the same period. The number of (l, S) inequalities is

exponential with respect to T. To limit the size of model MLCLSP, we only add to it the inequalities with $S=\{1, 2, \ldots, l-1\}$ and $1 \le l \le T$, as given by (19):

$$\sum_{t=1}^{l-1} x_{it} + d_{il} y_{il} \ge \sum_{t=1}^{l} d_{it}, \forall i \in N, 1 \le l \le T, \quad (19)$$

Simple experimentation shows that even just adding the $N \times T$ inequalities typically produces an important improvement in the lower bound value of the LP relaxation of the underlying model, as pointed out by Belvaux and Wolsey [29]. Van Vyve and Wolsey [30] said that these inequalities are most effective. Note that the inequalities (19) are not considered in the definition of $IR^l(Y_{it})$.

4.4. Fix-and-optimize algorithm for the MLCLSP-L

The fix-and-optimize approach presented above is not limited to the MLCLSP, it can also be applied to the MLCLSP-L after a slight adoption to take account of more binary variables (setup carry-over variables) appeared the MIP model of the MLCLSP-L presented in Section 3. In order to do so, we first define the interrelatedness of binary variables in model MLCLSP-L.

First, similar to that in model MLCLSP, each setup variable Y_{it} in model MLCLSP-L is 1-interrelated with Y_{it-1}, Y_{it}, and Y_{it+1}, 1-interrelated with Y_{jt-1} for all $j \in P_i$, and 1-interrelated with Y_{jt+1}, for all $j \in S_i$, through inventory balance Eqs. (7)–(9). Here, the second subscript in Y_{jt-1} is taken as $t-1$ for $j \in P_i$ and the second subscript in Y_{jt+1} is taken as $t+1$ for $j \in S_i$ are due to the existence of one period production lead time for each item in the model. Second, Y_{it} is 1-interrelated with Y_{jt} and Z_{jt} for all $j \in N_{m_i}$ through capacity constraints (10), where m_i denotes the resource (machine) used to produce item i. Third, Y_{it-1} is 1-interrelated with Z_{it} (or equivalently, Y_{it} is 1-interrelated with Z_{it+1}) through constraints (13), and Y_{it} is 1-interrelated with Z_{it} through constraints (14). Finally, Y_{it} is 1-interrelated with Z_{jt} and Z_{jt+1} for all $j \in N_{m_i}$ through constraints (15).

For each Z_{it}, it is 1-interrelated with Y_{jt} and Z_{jt} for all $j \in N_{m_i}$ through constraints (10), 1-interrelated with Z_{jt} for all $j \in N_{m_i}$ through constraints (12), 1-interrelated with Y_{it-1} and Y_{it} through constraints (13) and (14), and 1-interrelated with Z_{it-1}, Z_{it+1}, Y_{jt} and Y_{jt-1} for all $j \in N_{m_i}$ through constraints (15).

Formally, the set of binary variables that are 1-interrelated with Y_{it} is given by

$IR(Y_{it})=\{Y_{it-1}, Y_{it}, Y_{it+1}\} \cup \{Y_{i't-1} \mid i' \in P_i\} \cup \{Y_{i't+1} \mid i' \in S_i\} \cup \{Y_{i't} \mid i' \in N_{m_i}\} \cup \{Z_{i't} \mid i' \in N_{m_i}\} \cup \{Z_{it}, Z_{it+1}\} \cup \{Z_{i't}, Z_{i't+1} \mid i' \in N_{m_i}\}$,

and the set of binary variables that are 1-interrelated with Z_{it} is given by

$IR(Z_{it})=\{Y_{i't} \mid i' \in N_{m_i}\} \cup \{Z_{i't} \mid i' \in N_{m_i}\} \cup \{Y_{it-1}, Y_{it}\} \cup \{Z_{it-1}, Z_{it+1}\} \cup \{Y_{i't-1} \mid i' \in N_{m_i}\}$.

With the interrelatedness of binary variables defined above, we can similarly define the set of binary variables to be re-optimized $IR^l(Y_{it})$ and the set of binary variables to be fixed $FB^l(Y_{it})$ for the MLCLSP-L for any setup variable Y_{it} or pair (i, t), $i=1, \ldots, N$, $t=1, \ldots, T$. The fix-and-optimize algorithm for the MLCLSP-L is the same as that for the MLCLSP except that their definitions of $IR(Y_{it})$ are different.

Similarly, for model MLCLSP-L, we add to it part of its (l, S) inequalities given by (20) in order to speed up the resolution of each subproblem

$$I_{\_i} + \sum_{j \in S_i} a_{ij} \times I_{j,0} + \sum_{t=1}^{l-1} x_{it} + d_{il} y_{il} \ge \sum_{t=1}^{l} d_{it}, \forall i \in N, 1 \le l \le T, \quad (20)$$

In (20), d_{it} is the echelon demand of item i in period t with the consideration of one period production lead time for each item, and the first two terms on the left-hand side are introduced because of the existence of the physical initial inventory for each item. Similarly, inequalities (20) are not considered in the definition of $IR^l(Y_{it})$ for the MLCLSP-L.

5. Variable neighbourhood search for the MLCLSP

Despite its relatively large neighbourhood structure, the fix-and-optimize based local search for the MLCLSP proposed in the last section can only find a local optimum of the problem in most cases. In order to find a global optimum or a solution close to the global optimum, we must diversify the search space so that more promising regions can be explored. One way to realize such diversification is the systematic change of the neighbourhood structure of a local search adopted by the variable neighbourhood search (VNS) proposed by Hansen and Mladenović [20]. In this section, we will propose a VNS approach based on fix-and-optimize for the MLCLSP. In principle, this VNS approach can be extended to the MLCLSP-L. However, effective extension of the approach to the problem with setup carryover needs to take account of the particularities of the MLCLSP-L, which requires further study in future.

5.1. Variable neighbourhood search

As most VNS approaches, our VNS for the MLCLSP pre-selects (pre-defines) a finite set of neighborhood structures N_k, $k=1, 2, \ldots, k_{max}$, with $N_1(\overline{Y}) \subseteq N_2(\overline{Y}) \subseteq \ldots \subseteq N_{kmax}(\overline{Y})$, where k_{max} is the number of neighborhood structures considered and $N_k(\overline{Y})$ denotes the set of neighbouring solutions in the kth neighbourhood of a solution $\overline{Y}$ of the problem. For each neighbourhood structure given, a local optimum is found by a local search starting from the current solution. The diversification is realized by applying a shaking routine that creates a new starting solution for the local search by randomly perturbing the current solution.

To describe the VNS approach, let us define:

$\overline{Y}$	setup plan of the current solution,
$\overline{Y}^*$	setup plan of the incumbent (the current best solution),
CT	computation time so far,
CT_{max}	maximum computation time allowed,
k_{max}	number of neighbourhood structures considered,
k	index of neighbourhood structure N_k.

Because each solution of the MLCLSP is uniquely determined by its setup plan, in the following, the terms “setup plan” and “solution” are interchangeable without risk of ambiguity. For instance, $\overline{Y}$ and $\overline{Y}^*$ are also called the current solution and the incumbent, respectively. With the above notations, our VNS approach for the MLCLSP is presented in pseudo-code in Algorithm 2.

Algorithm 2. Variable neighbourhood search approach for the MLCLSP

Find a feasible initial solution $\overline{Y}_0$ of model MLCLSP;
Set $\overline{Y} \leftarrow \overline{Y}_0$, $\overline{Y}^* \leftarrow \overline{Y}_0$, and k$\leftarrow$1;
Repeat
  (*Local Search*) Apply a fix-and-optimize local search algorithm (to be described later) with $\overline{Y}$ as the initial solution to obtain a local optimum $\overline{Y}'$ of model MLCLSP with the neighbourhood structure N_k;
  If $\overline{Y}'$ is better than the incumbent $\overline{Y}^*$, then
    set $\overline{Y}^* \leftarrow \overline{Y}'$, $\overline{Y} \leftarrow \overline{Y}'$, and $k \leftarrow 1$;
  Else

set $\overline{Y} \leftarrow \overline{Y}^*$ and $k \leftarrow k+1$;
If $k > k_{max}$, then $k \leftarrow 1$;
End if
End if
(*Shaking*) Generate a new starting solution $\overline{Y}''$ from the current solution $\overline{Y}$ by a random swap-fix-and-optimize routine (to be described later) and set $\overline{Y} \leftarrow \overline{Y}''$;
Determine the current computation time *CT*;
Until $CT \geq CT_{max}$;

5.2. Local search by fix-and-optimize

In the VNS approach described above, a fix-and-optimize algorithm is used to find a local optimum of model MLCLSP starting from the current solution for a given neighbourhood structure N_k (N_k will be defined implicitly later). Since the local search algorithm is called many times in the VNS, a direct adoption of the fix-and-optimize algorithm presented in the last section (Section 4) may be too time consuming. In order to make the VNS more efficiently, we must reduce the computation time of each local search loop. This can be realized by limiting the number of subproblems to be solved in each iteration of the fix-and-optimize algorithm.

Suppose that the current neighbourhood structure is N_k and $\overline{Y}=\{\overline{Y}_{it}\}$ is the current solution of model MLCLSP, where $\overline{Y}_{it} \in \{0, 1\}$ for any $i=1, \ldots, N$, $t=1, \ldots, T$. For any integer $r_k \geq 1$, define MLCLSP($\overline{Y}$, r_k) as the MIP model derived from model MLCLSP by adding to it the following constraint.

$$\sum_{i=1}^{N}\sum_{t=1}^{T}(Y_{it}(1-\overline{Y}_{it})+\overline{Y}_{it}(1-Y_{it})) \leq r_k, \tag{21}$$

All feasible solutions of model MLCLSP($\overline{Y}$, r_k) define a neighbourhood of the current solution $\overline{Y}$ of model MLCLSP such that the Hamming distance between the setup plan of any neighbouring solution and the setup plan of the current solution is no larger than r_k.

If r_k is large, it may be still time consuming to explore all feasible solutions of model MLCLSP($\overline{Y}$, r_k). In this case, further reduction of the size of the neighbourhood is necessary.

When an MIP model is solved by a branch-and-bound or branch-and-cut algorithm, two solutions are typically available: the incumbent solution that is feasible with respect to the integrality constraints and the linear relaxation solution of the model at the current node in the search tree, where the second solution is usually fractional. The Relaxation Induced Neighbourhood Search (RINS) proposed by Danna et al. [31] and implemented in the MIP solver of CPLEX tries to improve the incumbent solution by re-optimizing the integer variables whose values in the incumbent solution are different from those in the linear relaxation solution. The RINS is based on a belief that the optimal solution and the linear relaxation solution of an MIP model have many integer variables with the same values in both solutions.

Motivated by the RINS, we propose a local search algorithm for our VNS by combining the fix-and-optimize algorithm presented in the last section and the RINS for model MLCLSP($\overline{Y}$, r_k). Let $\hat{Y}=\{\hat{Y}_{it}\}$ denote the values of all setup variables at the (optimal) linear relaxation solution of model MLCLSP($\overline{Y}$, r_k), with $0 \leq \hat{Y}_{it} \leq 1$, and $\Omega=\{Y_{it} \mid \hat{Y}_{it} \neq \overline{Y}_{it}\}$ denote the set of setup variables whose values at the current (feasible) solution of model MLCLSP are different from those at the linear relaxation solution of model MLCLSP($\overline{Y}$, r_k).

In each local search loop of the VNS with neighbourhood structure N_k, the fix-and-optimize algorithm only considers the subproblems of level *l* with $Y_{it} \in \Omega$, where *l* is a control parameter of positive integer. That is, in each iteration of the algorithm, a setup variable Y_{it} is randomly chosen from Ω with the same probability for each element in Ω, and the corresponding subproblem is solved. Different from subproblem SP^l_{it} defined in the last section, this subproblem also re-optimizes the setup variables in Ω.

Let us define $\mathrm{IR}^l(Y_{it}, +\Omega)=\mathrm{IR}^l(Y_{it}) \cup \Omega$, $\mathrm{FB}^l(Y_{it}, -\Omega)=\Psi \setminus (\mathrm{IR}^l(Y_{it}) \cup \Omega)$. The subproblem associated with Y_{it}, denoted by $SP^l_{it}(+\Omega)$, re-optimizes all setup variables in $\mathrm{IR}^l(Y_{it}, +\Omega)$, while fixing all setup variables in $\mathrm{FB}^l(Y_{it}, -\Omega)$.

In each iteration of the fix-and-optimize algorithm, when a better solution $\overline{Y}'$ of model MLCLSP is found, model MLCLSP($\overline{Y}$, r_k) will be solved again with $\overline{Y}$ replaced by $\overline{Y}'$ in constraint (21), and Ω will be updated accordingly. The fix-and-optimize algorithm terminates when no improvement is obtained in last $|\Omega|$ iterations, where $|\Omega|$ is the number of elements in set Ω. This algorithm, referred to as Fix-and-Optimize ($\overline{Y}$, *l*, r_k), is presented in pseudo-code in Algorithm 3.

Because the neighbourhood structures of our VNS approach satisfy the condition $N_1(\overline{Y}) \subseteq N_2(\overline{Y}) \subseteq \ldots \subseteq N_{kmax}(\overline{Y})$, we take $1 < r_1 < r_2 \ldots < r_{k_{\max}}$, where r_k, $k=1,\ldots,k_{max}$ are all positive integers. For instances, we take $r_k=2(k+1)$ in our numerical experiments to be presented later.

Algorithm 3. Fix-and-optimize($\overline{Y}$, *l*, r_k)

Solve the linear relaxation of model MLCLSP($\overline{Y}$, r_k) to get $\hat{Y}=\{\hat{Y}_{it}\}$ and $\Omega=\{Y_{it} \mid \hat{Y}_{it} \neq \overline{Y}_{it}\}$;
Iter $\leftarrow 0$;
Repeat
Iter $\leftarrow$ *Iter* $+1$;
Randomly choose a setup variable Y_{it} from Ω with the same probability for each element in Ω;
Solve subproblem $SP^l_{i,t}(+\Omega)$;
If the solution of $SP^l_{i,t}(+\Omega)$ is better than the current solution of model MLCLSP, then
Set $\overline{Y}$ to the setup plan of the solution of $SP^l_{i,t}(+\Omega)$;
Solve the linear relaxation of model MLCLSP($\overline{Y}$, r_k) to get $\hat{Y}=\{\hat{Y}_{it}\}$ and $\Omega=\{Y_{it} \mid \hat{Y}_{it} \neq \overline{Y}_{it}\}$;
Iter $\leftarrow 0$;
End if
Until *Iter* $\geq |\Omega|$;

5.3. Shaking by random swap-fix-and-optimize

In our VNS approach, the starting solution $\overline{Y}''$ for each local search loop is generated by a shaking routine that performs a series of random swap-fix-and-optimize operations on the current solution $\overline{Y}$. Here, the swap of a setup variable Y_{it} means that its value is changed from $Y_{it}=\overline{Y}_{it}$ to $Y_{it}=1-\overline{Y}_{it}$, where $\overline{Y}_{it} \in \{0, 1\}$ is the value of Y_{it} at the current solution. Since the values of all production variables X_{it} and overtime variables O_{it} are determined by the values of all setup variables Y_{it} by solving a linear programming model derived from model MLCLSP, the shaking (perturbation) of a solution of model MLCLSP is realized by swapping some setup variables.

Since our ultimate goal is to find a high quality capacity-feasible solution of model MLCLSP, a perturbed solution is acceptable only if one of the following two conditions is satisfied: (1) the current solution is capacity-infeasible, (2) both the current solution and the perturbed solution are capacity-feasible. For condition (2), we want to ensure that the new starting solution $\overline{Y}''$ is also capacity-feasible if the current solution $\overline{Y}$ is capacity-feasible such that the capacity feasibility of solution can be kept after the shaking. However, if the capacity constraints of model MLCLSP are tight, a capacity-infeasible solution may be generated if we swap (change the value of) a binary setup variable. To avoid the capacity infeasibility and to ensure that a diversified solution of good quality can be obtained after the shaking, we propose a random swap-fix-and-optimize procedure to generate

a new starting solution $\overline{Y}''$ from the current solution $\overline{Y}$ in the VNS. This is an iterative procedure. In each iteration, a setup variable is randomly selected and its value is swapped from 0 to 1 or from 1 to 0. A related MIP subproblem with most binary variables fixed is then solved (re-optimized). If the solution $\overline{Y}'$ of the MIP subproblem satisfies one of the above mentioned conditions, the current solution $\overline{Y}$ will be updated by $\overline{Y}'$ and this swap is tagged as a true (successful) swap. Otherwise, the swap is aborted. Such swap operation will be repeated until k true (successful) swaps are performed, where k is an integer parameter determined by the index of neighbourhood structure N_k in the VNS approach.

In this iterative procedure, in order to keep the diversity realized by all previous swaps, a tabu list TL is introduced to prevent a successful swap made at any previous or the current iteration from being cancelled (undoned) by a swap made at a future iteration. That is, if setup variable Y_{i_r,t_r} is successfully swapped at the current iteration, it will be put in the tabu list, i.e., TL$\leftarrow$TL$\cup\{Y_{i_r,t_r}\}$. This setup variable will be prevented from being selected by any future swap, and its value will be fixed in any fix-and-optimize operation performed later.

In each swap-fix-and-optimize operation, a setup variable Y_{i_r,t_r} is first randomly selected from $\Psi\backslash TL$ and its value is swapped from $\overline{Y}_{i_r,t_r}$ to $1-\overline{Y}_{i_r,t_r}$. A fix-and-optimize operation is then invoked to improve the perturbed solution. This operation is realized by fixing all setup variables in $\mathrm{FB}^l(Y_{i_rt_r},-\Omega)\cup TL\cup\{Y_{i_r,t_r}\}$, while re-optimizing all setup variables in $\mathrm{IR}^l(Y_{i_rt_r},+\Omega)\backslash\{TL\cup\{Y_{i_r,t_r}\}\}$, where $\Omega=\{Y_{it}\mid \hat{Y}_{it}\neq\overline{Y}_{it}\}$ is obtained by solving the linear relaxation of model MLCLSP($\{\overline{Y}, r_k\}$). Note that among the variables to be fixed, Y_{i_r,t_r} is fixed to $1-\overline{Y}_{i_r,t_r}$. If this fix-and-optimize operation leads to a perturbed solution that is acceptable according to the two conditions mentioned above, it is considered successful, and the current solution is updated by the perturbed solution.

Let $\overline{Y}=\{\overline{Y}_{it}\}$ be the current solution of model MLCLSP, and N_k be the neighbourhood structure currently considered by the VNS, the shaking routine is presented in pseudo-code in Algorithm 4.

Algorithm 4. Swap-fix-and-optimize ($\overline{Y}$, l, k)

$TL=\varnothing$;
For $iter=1$ to k **do**
  Solve the linear relaxation of model MLCLSP($\{\overline{Y}, r_k\}$ to get Ω
  Swap=False;
  While Swap=False
    Randomly choose a $Y_{i_r,t_r}\in\Psi\backslash TL$ with the same probability for each element in $\Psi\backslash TL$;
    Swap the value of binary variable Y_{i_r,t_r} from $Y_{i_r,t_r}=\overline{Y}_{i_r,t_r}$ to $Y_{i_r,t_r}=1-\overline{Y}_{i_r,t_r}$;
    Solve the subproblem of model MLCLSP that re-optimizes all setup variables in $\mathrm{IR}^l(Y_{i_rt_r},+\Omega)\backslash\{TL\cup\{Y_{i_r,t_r}\}\}$, while fixing all setup variables in $\mathrm{FB}^l(Y_{i_rt_r},-\Omega)\cup TL\cup\{Y_{i_r,t_r}\}$, where Y_{i_r,t_r} is fixed to $1-\overline{Y}_{i_r,t_r}$ and all other variables Y_{it} in $\mathrm{FB}^l(Y_{i_rt_r},-\Omega)\cup TL$ are fixed to $\overline{Y}_{i,t}$;
    If the solution of the subproblem is acceptable, then
      update $\overline{Y}$ by the setup plan of the solution;
      $TL\leftarrow TL\cup\{Y_{i_r,t_r}\}$;
      Swap=True;
    End if
  End while
End for

In this shaking routine, *iter* represents the number of successful swap-fix-and-optimize operations performed so far, r_k, and $k=1,\ldots,k_m$ are positive integers with $1<r_1<r_2\ldots<r_{k_{\max}}$. For the subproblem solved in the routine, its number of setup variables to be re-optimized does not exceed $|\mathrm{IR}^l(Y_{i_r,t_r})|+|\Omega|$. The difference between this subproblem and the subproblem $SP^l_{i,t}(+\Omega)$ in Algorithm 3 is that except for the setup variables in $\mathrm{FB}^l(Y_{it},-\Omega)$, the setup variables in the tabu list TL and the swapped setup variable Y_{i_r,t_r} are also fixed.

Note that the shaking routine of our VNS is different from that described in Hansen and Mladenovic [20], because the solution generated by this routine may be not in $N_k(\overline{Y})$, where N_k is the current neighbourhood structure of the VNS, and $\overline{Y}$ is the current solution of the problem considered.

6. Numerical results on the MLCLSP

We first evaluated the performances of our fix-and-optimize and VNS approaches for the MLCLSP on the benchmark instances documented by Stadtler and Surie [32]. Five classes (sets) of instances, A+, B+, C, D and E, were considered. Among them, A+, C and E have no setup time, whereas B+ and D have setup times. Each instance in A+ and B+ has 10 products, 24 periods, and 3 resources; each instance in C and D has 40 products, 16 periods, and 6 resources; and each instance in E has 100 products, 16 periods, and 10 resources. The number of instances in class A+, B+, C, D, and E is 120, 312, 180, 80, and 150, respectively. For some instances, whether they have a capacity-feasible solution is not known. According to Stadtler and Surie [32], only 108 instances out of 120 in A+, 132 instances out of 180 in C, and 72 instances out of 80 in D have a known feasible solution without overtime. All instances in class B+ and E have a known capacity-feasible solution.

All algorithms were coded in C++ in the environment of Microsoft Visual Studio 2005, and all instances were tested on a PC of DELL OPTIPLEX 760 with 2.99 GHz CPU and 3 GB RAM. We compared our approaches with the fix-and-optimize approach of Helber and Sahling [3]. All LP and MIP subproblems involved were solved by calling the MIP solver of CPLEX 10.1.

For ease of presentation, in the following, the fix-and-optimize approach of Helber and Sahling [3] is referred to as FO1, and our fix-and-optimize and VNS approaches are referred to as FO2 and VNS, respectively. As mentioned above, for both FO2 and VNS, the initial feasible solution of the MLCLSP is constructed by simply setting the value of each setup variable to one as in FO1. In all tests, the unit overtime cost for each resource was set to 10000 as in [3].

For FO2, the time for CPLEX to solve each subproblem is limited to 2 s, and at most n MIP subproblems are solved in each iteration, where n is the control parameter of FO2 introduced in Section 4. Here, "at most" means that FO2 will terminate if n subproblems are consecutively solved without improvement. Otherwise, the solution of one subproblem is better than the current best solution of the MLCLSP, and FO2 will proceed to a new iteration after the update of the best solution.

The detailed results of all algorithms for the benchmark instances are given in the online version of this paper. Note that the results were obtained by only solving each instance once. Since both FO2 and VNS contain random elements, the solution of each instance obtained by each approach may be different in two different runs. However, since the number of instances in each instance set is quite large (larger than 30), the average cost and computation time for each instance set would not change significantly in two different runs as observed by our numerical experiments and predicated by the law of large numbers in probability theory.

6.1. Comparison of different subproblem structures and control parameters of FO2

In order to choose an appropriate subproblem structure and control parameter for FO2, we tested the algorithm on benchmark

instances A+ (without setup time) and B+ (with setup times). In this test, the control parameter n is taken as $N \times T$. Because considering subproblems of level lager than 3 is too time consuming for large instances, we only tested FO2 with subproblems of level 1, 2, or 3. The results of the three versions of FO2, denoted by FO2-L1, FO2-L2, and FO2-L3, respectively, are given in Table 1. For FO2-L1, the average cost and computation time in seconds for each instance set are given by the columns Cost and Time, respectively. For FO2-Li (i=2, 3), its two columns give the average Relative Cost Reduction and Relative (Computation) Time Increase with respect to FO2-Li-1 (i=2, 3). From Table 1, we can see that with 65.22% to 78.99% increase of computation time, FO2-L2 can reduce cost by 2.84% to 2.86% on average for class A+ and B+ compared with FO2-L1. On the other hand, with 189.67% to 199.32% increase of computation time, FO2-L3 only reduces cost by 1.26% to 1.38% on average for the two classes compared with FO2-L2. It seems that FO2-L2 makes the best trade-off between solution quality and computation time among the three versions of FO2. For this reason, we chose FO2-L2 in all other tests and comparisons.

We then tested FO2-L2 with five different values of n on the two sets of benchmark instances. The results are given in Table 2. In this table, for FO2-L2 with $n=N \times T$, the columns Cost and Time give the average cost and computation time in seconds for each instance set. For FO2-L2 with $n=N \times T/k$ (k=2, 4, 8, 16), its two columns give the average Relative Cost Increase and Relative (Computation) Time Decrease with respect to FO2-L2 with $n=N \times T$. For instances, the average relative cost increase of FO2-L2 with $n=N \times T/8$ with respect to FO2-L2 with $n=N \times T$ is 0.29% for class A+. From Table 2, we can see that the reduction of n from $N \times T$ to $N \times T$ /2 does not significantly increase average cost but reduces average computation time significantly. It seems that a good trade-off is made between solution quality and computation time when we take $n=N \times T/4$ or $n=N \times T/8$. For this reason, we took $n=N \times T/4$ or $N \times T/8$ when FO2-L2 was compared with FO1. Note that in this table, for class A+, the average relative cost increase of FO2-L2 with $n=N \times T/2$ with respect to FO2-L2 with $n=N \times T$ is a very small negative number, this is due to the random nature of the two algorithms as mentioned above.

6.2. Comparison of FO2 and FO1

For this purpose, we also implemented the fix-and-optimize algorithm (FO1) of Helber and Sahling [3] on our PC with the same programming and running environment of FO2. All parameters of FO1 were set according to [3] with $l^{max}=\infty$, i.e., FO1 was run in multiple iterations until no improvement could be found. For FO2-L2, we did not calibrate the parameter of the maximum time for CPLEX to solve each MIP subproblem. The time was set to 2 s for all instances. The comparison results are given in Table 3. From this table, we can see that for class A+ and B+, FO2-L2 is competitive with FO1 in terms of solution quality and computation time. For class C, D and E, FO2-L2 significantly outperforms FO1. If the control parameter n is taken as $N \times T/8$, compared with FO1, FO2-L2 can reduce cost by 0.96% with 5.13% reduction of computation time on average for class C, reduce cost by 7.05% with 23.55% reduction of computation time on average for class D, and reduce cost by 1.39% with 58.92% reduction of computation time on average for class E. It should be noted that FO2-L2 gave impressive results on the hard instances of class D with setup times. Compared with FO1, FO2-L2 with $n=N \times T/8$ can reduce cost by 7.05% with 23.55% reduction of computation time on average, and FO2-L2 with $n=N \times T/4$ can reduce cost by 11.99% with 10.24% increase of computation time on average for class D. This implies that FO2-L2 is much more effective than FO1 for large instances with setup times.

6.3. Comparison of VNS, FO2, and FO1

We next evaluated the performance of our VNS approach for the MLCLSP. For this approach, its k_{max} was set to 10, and its computation time limit was set to 300 s for instance sets A+ and B+, 600 s for instance sets C+ and D, and 1200 s for instance set E, according to their problem size. For the fix-and-optimize local search algorithm and the random swap-fix-and-optimize shaking routine called in the VNS, only subproblems of level 2 were considered by setting l=2 and $r_k=2(k+1)$, k=1, 2,…, k_{max}. Similarly, the time for CPLEX to solve each MIP subproblem in both the local search and the shaking routine was limited to 2 s. We did not calibrate any parameter of the VNS.

The results of the VNS, FO2-L2, and FO1 are given in Table 4. In this table, each cell in a column entitled Relative Cost Reduction (%) contains three numbers. The first number on top gives the average relative cost reduction in percentage obtained by a method (FO2-L2 with $n=N \times T/8$, FO2-L2 with $n=N \times T/4$, or

Table 1
FO2 with different subproblem structures.

Problem set	FO2-L1		FO2-L2		FO2-L3	
	Cost	Time	Relative cost reduction (%)	Relative time increase (%)	Relative cost reduction (%)	Relative time increase (%)
A+	152011.13	32.35	2.86	65.22	1.38	189.67
B+	130447.49	33.33	2.84	78.99	1.26	199.32

Table 3
Comparison of FO2 and FO1.

Problem set	FO1		FO2-L2 ($n=N \times T/8$)		FO2-L2 ($n=N \times T/4$)	
	Cost	Time	Relative cost reduction (%)	Relative time reduction (%)	Relative cost reduction (%)	Relative time reduction (%)
A+	148094.96	21.96	−0.14	47.42	0.03	2.59
B+	127093.79	35.53	−0.14	125.10	0.13	53.07
C	1613089.97	98.74	0.96	5.13	1.05	−24.48
D	786095.10	74.90	7.05	23.55	11.99	−10.24
E	2157792.17	656.39	1.39	58.92	1.43	1.78

Table 2
Comparison of FO2-L2 with different control parameter values.

Prob. set	$n=N \times T$		$n=N \times T/2$		$n=N \times T/4$		$n=N \times T/8$		$n=N \times T/16$	
	Cost	Time	Relative cost increase (%)	Relative time decrease (%)	Relative cost increase (%)	Relative time decrease (%)	Relative cost increase (%)	Relative time decrease (%)	Relative cost increase (%)	Relative time decrease (%)
A+	147947.36	50.34	−0.06	20.79	0.13	48.99	0.29	65.33	0.86	75.57
B+	126801.20	58.40	0.01	33.09	0.15	54.04	0.42	68.04	0.96	76.18

VNS) compared with FO1 for a set of instances, and the second and the third number in the parenthesis give the minimum and the maximum value of this relative cost reduction in percentage for all instances of the set. From this table, we can see that the VNS significantly outperforms FO1 and the two versions of FO2 with $n=N\times T/8$ and $n=N\times T/4$, respectively, in terms of cost. For the instances of class D, compared with FO1, the VNS can reduce cost by 16.14% on average, whereas FO2-L2 with $n=N\times T/4$ only reduces cost by 11.99%. Since production planning is a tactical decision for a factory, the MLCLSP considered in this paper is usually solved weekly or monthly. It is thus worth spending more but reasonable time to obtain a significantly better production plan by using our VNS. Note that for class D, the maximum relative cost reduction in percentage is quite large (≥ 236) for the two versions of FO2 and VNS, this is because, compared with FO1, the three methods can significantly reduce overtime costs for certain instances.

6.4. Comparison with the best solutions documented by Stadtler and Suerie [32]

We also compared these approaches in terms of improvement of the best solutions documented by Stadtler and Suerie [32] for the benchmark instances. With this criterion, our VNS outperforms FO2 with $n=N\times T/4$, which in turn outperforms both FO2 with $n=N\times T/8$ and FO1. The results are given in Table 5, where each of the 3rd to 6th columns gives the number of instances for which the corresponding approach can get a better solution compared with that documented in [32] for each instance set. From this table, we can see that the VNS can obtain a better solution for many benchmark instances tested. Note that compared with the documented best solutions, the solutions obtained by the variable neighbourhood decomposition search method of Zhao et al. [5] have a higher average cost for both classes C and D, our VNS thus outperforms theirs in terms of solution quality. We cannot compare the two methods in terms of computation time, because Zhao et al. [5] did not report the computation time of their method on the benchmark instances.

7. Numerical results on the MLCLSP-L

To further evaluate the performance of our fix-and-optimize approach and demonstrate its flexibility, we also conducted numerical experiments on the 1920 benchmark instances of the MLCLSP-L taken from Tempelmeier and Buschkühl [11]. Sahling et al. [4] extended the fix-and-optimize approach of Helber and Sahling [3] to the MLCLSP-L and obtained very good results on the instances. In the following, our fix-and-optimize approach for the MLCLSP-L with subproblems of level 2 is compared with the fix-and-optimize approach of Sahling et al. [4] on the benchmark instances. The two approaches are also referred to as FO2-L2 and FO1, respectively. Two versions of FO2-L2, FO2-L2 with $n=N\times T/8$ and FO2-L2 with $n=N\times T/4$, were tested. In the test, the unit overtime cost for each resource was set to 10000 as in [4]. The detailed results of all algorithms for these instances are given in the online version of this paper.

The results of this test are given in Table 6. From this table, we can see that for most instance sets, FO2-L2 with $n=N\times T/4$ can obtain better solutions in terms of average cost compared with FO1. The average computation time of this FO2-L2 is shorter than that of FO1 for small instance sets but longer for large instance sets. If we consider both cost and computation time, FO2-L2 is competitive with FO1. FO2-L2 is also competitive with the corridor approach of Caserta and Voß [6], since the results obtained by the approach are similar to those obtained by FO1.

8. Conclusion

In this paper, we have presented a new fix-and-optimize (FO) approach for both the MLCLSP and the MLCLSP-L. This is an iterative local search approach. In each iteration, multiple MIP subproblems are solved. Each subproblem is derived from an MIP model of the MLCLSP or the MLCLSP-L by fixing most binary variables, while re-optimizing the other binary variables. Our FO approach determines the binary variables to be re-optimized in each subproblem based on the interrelatedness of binary variables in the constraints of the MIP model. Based on this approach, we also developed a variable neighbourhood search (VNS) approach for the MLCLSP. Numerical experiments show that our FO approach outperforms the FO approach of Helber and Sahling for the MLCLSP and is competitive with the FO approach of Sahling, Buschkühl, Tempelmeier, and Helber for the MLCLSP-L. Moreover, for the MLCLSP, our VNS can further improve the solution obtained by our FO approach through the diversification of the search space. As a result, new best solutions have been obtained for many benchmark instances tested.

Compared with the FO approach of Helber and Sahling, our FO approach is more general. It may be extended and applied to other 0–1 mixed integer programming problems, since its subproblem definition is based on the interrelatedness of binary variables in a 0–1 MIP model rather than based on problem-specific decompositions. Since many combinatorial optimization problems can be formulated as a 0–1 MIP in which binary variables are interrelated

Table 5
Number of better solutions found by each approach.

Problem set	Number of instances tested	FO1	FO2 with $n=N\times T/8$	FO2 with $n=N\times T/4$	VNS
A+	120	36	33	37	86
B+	312	119	120	133	232
C	180	72	113	117	133
D	80	12	25	26	37
E	150	11	32	31	32

Table 4
Comparison of VNS, FO2 and FO1.

Problem set	FO1		FO2-L2 ($n=N\times T/8$)		FO2-L2 ($n=N\times T/4$)		VNS	
	Cost	Time	Relative cost reduction (%)	Time	Relative cost reduction (%)	Time	Relative cost reduction (%)	Time
A+	148094.96	21.96	−0.14(−4.57,3.46)	15.79	0.03(−8.87, 3.44)	23.33	1.28(−2.83, 4.35)	300
B+	127093.79	35.53	−0.14(−4.68, 6.65)	16.47	0.13(−6.28, 8.70)	23.96	1.16(−8.69, 8.56)	300
C	1613089.97	98.74	0.96(−3.78, 6.03)	104.46	1.05(−2.49, 5.52)	149.17	1.41(−1.96, 6.73)	600
D	786095.10	74.90	7.05(−21.5, 236)	67.96	11.99(−3.56, 236)	99.71	16.14(−1.13,310)	600
E	2157792.17	656.39	1.39(−5.15, 8.54)	546.57	1.43(−5.07, 8.13)	794.74	1.53(−3.48, 7.81)	1200

Table 6
Comparison of FO2 and FO1 for the MLCLSP-L.

Problem set	FO1		FO2-L2 ($n=N\times T/8$)		FO2-L2 ($n=N\times T/4$)	
	Cost	Time	Relative cost reduction (%)	Relative time reduction (%)	Relative cost reduction (%)	Relative time reduction (%)
1_AA	1802.8637	0.876483	0.1539	360.3744	0.2092	157.2061
1_AC	1774.8492	0.881842	−0.0287	304.2784	0.1258	139.9074
1_GA	2830.6842	0.782792	−0.1687	505.9947	0.1664	208.3002
1_GC	2621.5046	0.779800	−0.6165	454.2723	−0.0122	199.6320
2_AA	4311.1130	1.512083	0.0353	95.5936	0.1816	9.7741
2_AC	4170.3882	1.391575	0.0032	91.6306	0.0343	2.0873
2_GA	5006.7420	1.395333	−0.1614	127.8753	0.3421	26.8224
2_GC	4707.1809	1.362075	−0.2557	111.3096	0.0433	13.4167
3_AA	10359.8150	4.243933	−0.0059	30.9729	0.1201	−3.7620
3_AC	10192.1858	4.013517	0.1574	13.4503	0.2499	−25.5228
3_GA	14518.4433	3.723400	−0.1318	64.4532	0.0084	−14.8951
3_GC	13465.1258	3.807783	−0.2349	50.2086	−0.0782	−15.9086
4_AA	22669.5967	9.725517	0.2135	−57.9352	0.1889	−66.4152
4_AC	21826.0700	9.393483	0.0151	−58.1843	−00.0576	−57.8693
4_GA	32626.0083	8.774983	0.0928	−19.1573	0.1076	−57.9391
4_GC	29518.0183	8.092717	−0.0790	−26.4683	−0.0398	−54.5742
5_AA	13862.8883	10.663583	−0.0006	−6.3321	0.3133	−41.1605
5_AC	13692.0517	10.381517	0.0484	12.4528	0.2052	−31.2883
5_GA	16010.1983	9.732817	0.1535	18.0614	0.0162	−19.5931
5_GC	15435.2450	10.131450	−0.4385	45.2745	−0.2317	−7.8735
6_AA	30722.2317	27.852883	0.0276	−54.8496	0.1321	−71.0312
6_AC	30261.9717	28.397133	−0.0834	−39.4173	0.0682	−61.2961
6_GA	36467.4233	22.226867	0.1089	−51.3003	0.0387	−62.8559
6_GC	35119.7900	29.747567	−0.1403	−36.3880	−0.01674	−62.7234

through constraints, our FO approach may be generalized to become a generic MIP-based local search approach for these problems. For instances, in a warehouse location model, a real variable that indicates the quantity delivered to a customer by a warehouse depends on the binary variable that indicates whether the warehouse is created. If only one warehouse is created, all binary variables in the model are interdependent through the constraint formulating the condition. For this model, a fix-and-optimize approach like that in this paper may be developed based on the interrelatedness of binary variables.

The work of this paper has opened several directions for future research. One direction is to further improve the performance of our FO approach for the MLCLSP by finding a more effective way to select MIP subproblems in each iteration. Another is to extend the FO approach to solve other 0-1 MIPs. Finally, it is possible to develop more effective metaheuristics that use the FO as a local search engine for the MLCLSP.

Acknowledgements

The author is grateful to Prof. Florian Sahling at Leibniz University of Hannover, Prof. Hartmut Stadtler at University of Hamburg, Prof. Horst Tempelmeier at University of Koln, and Prof. Marco Caserta at IE Business School for their providing the data of the benchmark instances of the MLCLSP and the MLCLSP-L. He would like to also thank three anonymous reviewers for their constructive comments, which have helped him to improve previous versions of this paper.

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.omega.2015.03.002.

References

[1] Tempelmeier H, Derstroff. M. A Lagrangean-based heuristic for dynamic multilevel multi-item constrained lotsizing with setup times. Management Science 1996;42:738–57.
[2] Trigeiro WW, Thomas LJ, McClain JO. Capacitated lot sizing with setup times. Management Science 1989;35:353–66.
[3] Helber S, Sahling. F. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. International Journal of Production Economics 2010;123:247–56.
[4] Sahling F, Buschkühl L, Tempelmeier H, Helber S. Solving a multi-level capacitated lot sizing problem with multi-period setup carry-over via a fix-and-optimize heuristic. Computers and Operations Research 2009;36:2546–53.
[5] Zhao Q, Xie C, Xiao. Y. A variable neighborhood decomposition search algorithm for multilevel capacitated lot-sizing problems. Electronic Notes in Discrete Mathematics 2012;39:129–35.
[6] Caserta M, Voß. S. A MIP-based framework and its application on a lot sizing problem with setup carryover. Journal of Heuristics 2013;19:295–316.
[7] Billington PJ, McClain JO, Thomas LJ. Mathematical programming approaches to capacity constrained MRP systems: review, formulation and problem reduction. Management Science 1983;29:1126–41.
[8] Akartunali K, Miller. AJ. A computational analysis of lower bounds for big bucket production planning problems. Computational Optimization and Applications 2012;53:729–53.
[9] Buschkühl L, Sahling F, Helber S, Tempelmeier H. Dynamic capacitated lot-sizing problems: a classification and review of solution approaches. OR Spectrum 2010;32:231–61.
[10] Jans R, Degraeve Z. Meta-heuristics for dynamic lot sizing: a review and comparison of solution approaches. European Journal of Operational Research 2007;177:1855–75.
[11] Tempelmeier H, Buschkühl. L. A heuristic for the dynamic multi-level capacitated lot sizing problem with linked lot sizes for general product structures. OR Spectrum 2009;31:385–404.
[12] Karimi B, Fatemi Ghomi SMT, Wilson. JM. The capacitated lot sizing problem: a review of models and algorithms. Omega 2003;31:365–78.
[13] Robinson P, Narayanan A, Sahin F. Coordinated deterministic dynamic demand lot-sizing problem: a review of models and algorithms. Omega 2009;37:3–15.
[14] Belvaux G, Wolsey. LA. BC-PROD: a specialized branch-and-cut system for lot-sizing problems. Management Science 2000;46:724–38.
[15] Stadtler H. Multilevel lot sizing with setup times and multiple constrained resources: internally rolling schedules with lot-sizing windows. Operations Research 2003;51:487–502.
[16] Suerie C, Stadtler. H. The capacitated lot-sizing problem with linked lot sizes. Management Science 2003;49:1039–54.
[17] Pochet Y, Wolsey LA. Production planning by mixed integer programming. New York: Springer; 2006.
[18] Stadtler H, Sahling. F. A lot-sizing and scheduling model for multi-stage flow lines with zero lead times. European Journal of Operational Research 2013;225:404–19.

[19] Chen H, Chu. C. A Lagrangian relaxation approach for supply chain planning with order/setup costs and capacity constraints. Journal of Systems Science and Systems Engineering 2003;12:98–110.
[20] Hansen P, Mladenovic N. Variable neighborhood search: principles and applications. European Journal of Operational Research 2001;130:449–67.
[21] Hindi KS, Fleszar K, Charalambous. C. An effective heuristic for the CLSP with set-up times. Journal of Operational Research Society 2003;54:490–8.
[22] Xiao Y, Kaku I, Zhao Q, Zhang. R. A variable neighborhood search based approach for uncapacitated multilevel lot-sizing problems. Computers and Industrial Engineering 2011;60:218–27.
[23] Seeanner F, Almada-Lobo B, Meyr H. Combining the principles of variable neighbourhood decomposition search and the fix&optimize heuristic to solve multi-level lot-sizing and scheduling problems. Computers and Operations Research 2013;40:303–17.
[24] Almeder. C. A hybrid optimization approach for multi-level capacitated lot-sizing problems. European Journal of Operational Research 2010;200:599–606.
[25] Muller LF, Spoorendonk S, Pisinger. D. A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. European Journal of Operational Research 2012;218:614–23.
[26] Li XY, Baki F, Tian P, Chaouch. BA. A robust block-chain based tabu search algorithm for the dynamic lot sizing problem with product returns and remanufacturing. Omega 2014;42:75–87.
[27] Tempelmeier. H. A column generation heuristic for dynamic capacitated lot sizing with random demand under a fill rate constraint. Omega 2011;39:627–33.
[28] Barany I, Van Roy TJ, Wolsey LA. Uncapacitated lot sizing: the convex hull of solutions. Mathematical Programming Studies 1984;22:32–43.
[29] Belvaux G, Wolsey LA. Modelling practical lot-sizing problems as mixed-integer programs. Management Science 2001;47:993–1007.
[30] Van Vyve M, Wolsey LA. Approximate extended formulations. Mathematical Programming, Series B 2006;105:501–22.
[31] Danna E, Rothberg E, Le Pape C. Exploring relaxation induced neighbourhoods to improve MIP solutions. Mathematical Programming 2005;102:71–90.
[32] H. Stadtler, C. Suerie. Description of MLCLSP test instances. at ⟨www.vwl.tu-darmstadt.de/bwl1/forschung/ti_mlclsp/ti_mlclsp.php⟩; last accessed on April 29, 2014.