Neighbourhood Search Metaheuristics for Capacitated Lotsizing with Sequence-dependent Setups

Bernardo Almada-Lobo, Ross J.W. James

Neighbourhood Search Metaheuristics for Capacitated Lotsizing with Sequence-dependent Setups

Journal:	*International Journal of Production Research*
Manuscript ID:	TPRS-2008-IJPR-0171.R2
Manuscript Type:	Original Manuscript
Date Submitted by the Author:	12-Aug-2008
Complete List of Authors:	Almada-Lobo, Bernardo; Faculty of Engineering of University of Porto, Department of Mechanical Engineering and Industrial Management James, Ross; University of Canterbury, Department of Management
Keywords:	META-HEURISTICS, PRODUCTION PLANNING, LOT SIZING, SCHEDULING, TABU SEARCH
Keywords (user):	CLSP, VNS

Neighbourhood Search Metaheuristics for Capacitated Lotsizing with Sequence-dependent Setups

Bernardo Almada-Lobo[a*] Ross J.W. James[b]

[a] *Faculdade de Engenharia da Universidade do Porto, Porto, Portugal*

[b] *Department of Management, University of Canterbury, Private Bag 4800, Christchurch, New Zealand*

Abstract

We address a problem that often arises in industry, the multi-item capacitated-lotsizing and scheduling problem with sequence dependent setup times and costs. Powerful commercial solvers fail to solve even medium-sized instances of this NP-hard problem, therefore we employ a tabu search and a variable neighbourhood search metaheuristic to solve it and compare the performance of these metaheuristics over time. In contrast to the majority of the literature on this topic, the solution representation explicitly considers production quantities and setup variables, which enables us to develop fast search heuristics. A comprehensive set of computational experiments shows the effectiveness and efficiency of the proposed approaches in solving medium to large size problems.

Keywords: sequence-dependent setup, tabu search, variable neighbourhood search, CLSP.

1 Introduction

In an increasingly competitive global marketplace, production planning plays a key role in the performance of industrial enterprises. Apart from resource acquisition, decisions are typically operational (short-term) or tactical (medium-term) planning problems. One of the most important and challenging production planning problems is lotsizing and scheduling.

The lotsizing problem consists of determining the production orders or lots in order to satisfy demand at minimum cost. The complexity of lotsizing problems depends on the features and assumptions taken into account by the model, much of which is determined by the industry application and the actual problem needing to be solved. The majority of models consider a discrete time scale, dynamic demand and a finite time horizon. The single-level lotsizing problem for multiple items that compete for finite machine capacity is known as the capacitated lotsizing problem (CLSP). The CLSP can be seen as an extension of the Wagner-Whitin model that takes into account capacity constraints. Since several products may be produced per period (i.e. this is a large-bucket model), such a period normally represents a time slot of a week or a month. As a pure lotsizing model, CLSP does not sequence (schedule) products (jobs) in each period, as it requires a setup

*Corresponding author. *E-mail address*: almada.lobo@fe.up.pt; tel: +351 225082133

for an item in any period in which it is produced. By allocating unnecessary setups (that eat into production capacity), such a model does not perform well in practice, this becomes more obvious as the capacity reduces and/or the ratio of the bucket size to average setup times decreases. The inclusion of setup carryovers, meaning that no additional setup is charged if a product is the last one to be produced in a period and the first one in the following period, has lead to the development of the CLSP with linked lot-sizes model (e.g. Haase [1994] and Suerie and Stadtler [2003]). These models incorporate both the lotsizing and partial sequencing decisions but only the first and last products of each period are considered for optimisation as the setups are sequence-independent.

Nevertheless, a wide range of process industries face significant sequence dependent setup times and costs in product changeovers, especially when the same facilities produce items of different family types. Examples can be found in packaging, glass container manufacturing, abrasive manufacturing, chemical manufacturing among many others. Here, the differentiation between lotsizing and scheduling is completely blurred, and it is imperative that both decisions are made simultaneously in order to efficiently use the capacity available. Despite its relevance, little research has been done in this area. The review by Karimi et al. [2003] highlights the scarcity of literature devoted to CLSP with sequence-dependent setups and setup-carryovers, and underlines the need for faster and more efficient heuristics. Jans and Degraeve [2008] give an overview of recent developments in the field of modeling deterministic single-level dynamic lotsizing problems, and point out a further integration of lotsizing and scheduling as an interesting area for future research.

We study single machine CLSP with sequence dependent setup times and costs. Even the standard CLSP is known for its computational intractability (see Maes et al. [1991]), thus motivating the development of heuristics and/or metaheuristics to solve this problem. The success of metaheuristics relies on their flexibility to deal with large and complex problems (Jans and Degraeve [2007]). Those that have studied this specific CLSP, such as Gupta and Magnusson [2005] and Almada-Lobo et al. [2007], report large average deviations from the optimal solution, which demonstrates further how hard this problem is to solve, even for medium-sized instances. Clark et al. [2006] formulate the CLSP with sequence dependent setups as an Asymmetric Traveling Salesman Problem and test it on a real-world instance coming from an animal feed plant, with 21 products and 4 time periods. Clark and Clark [2000] develop a mixed integer programming (MIP) model for this problem on parallel machines that allows for multiple set-ups (up to a pre-determined number) per planning period. Due to its large dimensionality, the authors use approximate models that are tested in a rolling horizon scheme using very small instances.

The CLSP is considered to be a large-bucket model, since several setups can be performed per (lengthy) time period. In small-bucket models, the planning horizon is divided into many short periods, in which at most one setup may be performed, and therefore, depending on the model, at most one or two items may be produced per period. An example of the former is the discrete lotsizing and scheduling problem (DLSP – see Jans and Degraeve [2004]), and of the latter is the proportional setup lotsizing problem (Suerie [2006]). A more flexible lotsizing and scheduling problem is the general lotsizing and scheduling problem (GLSP), e.g. Araujo et al. [2007], that makes use of a two-level time structure, that divides the planning horizon into large buckets which are then divided into a number of small time slots. The reader is referred to Pochet and Wolsey [2006] for an introduction to integer programming formulations of lot-sizing problems.

Neighbourhood-based search algorithms start with an initial solution and iteratively try to find better solutions in the neighbourhood of the incumbent one. When designing such algorithms, one of the key issues is how to represent a solution. In most of the reported approaches to lotsizing and scheduling problems the solution representation only includes the setup (binary) variables.

Given a fixed set of those variables, the remaining linear subproblem is solved (optimally or not) to calculate the previously discarded production quantities. Even if the linear programming (LP) problem is solved efficiently, this step consumes most of the overall computational time. On the other hand, this representation is less complex than the one that considers production quantities and setup variables explicitly, as done in this work.

Karimi et al. [2006] study CLSP with setup carryovers and sequence independent setups, therefore production runs are not sequenced. The authors employ a tabu search procedure with a direct representation of only the binary variables. The subproblem is solved as a minimum-cost network flow problem. Meyr [2000] addresses GLSP with sequence-dependent setups, as follows: after fixing the setup sequence with either threshold accepting or simulated annealing (SA), a minimum cost network-flow problem is solved in order to determine the lotsizes and the holding costs of the candidate. Meyr [2002] considers the same problem on non-identical parallel machines and also fixes the setup sequence by local search and solves the remaining problem as a generalized network flow problem. Araujo et al. [2007] consider a GLSP from a foundry, which is tackled by three local search variants: descent heuristic, diminishing neighbourhood search and simulated annealing. Their solution representation only includes setup variables. We note that in DLSP, the machine either produces at full capacity or is idle (this is called the "all-or-nothing" assumption), which clearly simplifies the problem and therefore the solution procedure. Therefore, all the papers addressing this problem make use of a representation that only includes binary variables (see for example Bruggemann and Jahnke [2000]).

All the aforementioned manuscripts explore local search variants on the combinatorial variables with linear programming on the continuous variables. To the best of our knowledge, Gopalakrishnan et al. [2001] is the only big-bucket lotsizing and scheduling paper that simultaneously determines production quantities and setup variables in their solution representation, which they solve using a tabu search heuristic. However, these authors tackle CLSP with sequence independent setups and propose move types that solely deal with partial sequencing, which simplifies the problem considerably. Ozdamar and Bozyel [2000] and Ozdamar et al. [2002] also consider lot size and setup variables in their model which is solved using simulated annealing and genetic algorithm integrated with tabu search approaches, respectively, to tackle the pure CLSP, but they do not determine the schedule in this version of the problem. By integrating a big-bucket lotsizing and scheduling model and a batching scheduling model (converting lots into jobs) we are able to develop very efficient neighbourhood-based search algorithms, and achieve near optimal solutions, even for the most complicated instances. Tabu Search (TS) and variable neighbourhood search (VNS) metaheuristics are employed to solve this problem. Both metaheuristics use neighbourhoods based on moves that alter either the sequence of jobs processed or both the sequence and the quantities produced by splitting a job in two and moving the split quantity to another part of the sequence. We show that these heuristics clearly outperform CPLEX 10.1 for the larger instances, for which CPLEX fails to produce feasible solutions within a one-hour time limit. Comparing the effectiveness of TS against VNS, the latter constantly beats the former, especially as capacity tightens and setup cost increases. However, as the problem gets very large on the number of products and periods, TS starts to outperform VNS within the one-hour time limit.

The remainder of the paper is organized as follows: Section 2 presents the CLSP formulation. Section 3 describes the solution approach. The metaheuristics used are described in Section 4. Lower bounds for the problem are explained in Section 5. Computational experiments are reported in Section 6 and finally, Section 7 draws some conclusions from this work and suggests directions for future research.

2 The Formulation

We start by introducing the parameters and decision variables of CLSP model. We consider a planning interval with periods $t = 1, \ldots, T$ and products $i, j = 1, \ldots, N$ processed on a single machine. We use the notation $[M]$ to refer to the set $\{1, 2, \ldots, M\}$. Let d_{it} denote the demand of product i in period t, s_{ij} and c_{ij} the time and cost incurred when a setup occurs from product i to product j respectively, h_i the cost of carrying one unit of inventory of product i from one period to the next, p_i the processing time of one unit of product i and C_t the machine capacity available in period t. In addition, M_{it} is an upper bound on the production quantity of product i in period t.

The following decision variables are used: X_{it} refers to the quantity of product i produced in period t, I_{it} the inventory level of product i at the end of period t and V_{it} an auxiliary variable that assigns product i in period t. In addition, the following 0/1 decision variables are defined: T_{ijt} equals one if a setup occurs from product i to product j in period t, and α_{it} equals one if the machine is set up for product i at the beginning of period t.

A conventional model for CLSP with sequence dependent setups is as follows (Almada-Lobo et al. [2007]).

$$\min \sum_i \sum_j \sum_t c_{ij} \cdot T_{ijt} + \sum_i \sum_t h_i \cdot I_{it} \tag{1}$$

$$I_{it} = I_{i(t-1)} + X_{it} - d_{it} \qquad i \in [N], t \in [T] \tag{2}$$

$$\sum_i p_i \cdot X_{it} + \sum_i \sum_j s_{ij} \cdot T_{ijt} \leq C_t \qquad t \in [T] \tag{3}$$

$$X_{it} \leq M_{it} \cdot \left(\sum_j T_{jit} + \alpha_{it} \right) \qquad i \in [N], t \in [T] \tag{4}$$

$$\sum_i \alpha_{it} = 1 \qquad t \in [T] \tag{5}$$

$$\alpha_{it} + \sum_j T_{jit} = \alpha_{i(t+1)} + \sum_j T_{ijt} \qquad i \in [N], t \in [T] \tag{6}$$

$$V_{it} + N \cdot T_{ijt} - (N-1) - N \cdot \alpha_{it} \leq V_{jt} \qquad i \in [N], j \in [N] \setminus \{i\}, t \in [T] \tag{7}$$

$$T_{iit} = 0 \qquad i \in [N], t \in [T] \tag{8}$$

$$I_{i0} = 0 \qquad i \in [N] \tag{9}$$

$$(X_{it}, I_{it}) \geq 0, (T_{ijt}, \alpha_{it}) \in \{0, 1\}, X_{it} \in \mathbb{I}, V_{it} \in \mathbb{R} \tag{10}$$

Objective function (1) minimizes the sum of setup and inventory holding costs. Constraints (2) balance production and inventories with demand. Constraints (3) ensure that production and setups do not exceed the available capacity. The setup forcing constraints are provided by (4). Constraints (5) determine that the machine is set up for one product at the beginning of each time period, whilst constraints (6) keep track of the setup carryover information. Disconnected subtours are eliminated by constraints (7) and (8) (Almada-Lobo et al. [2008]). These constraints apply whenever one subtour occurs in a period, forcing the machine to be set up at the beginning

of that period to one of the products that are part of the subtour. Initial inventory is set to zero in constraints (9). Finally, there are the non-negativity and integrality constraints (10). Note that setup carryovers are guaranteed here, since if a product is the last one to be produced in a period and the first one in the following period no setup is required in the latter period.

3 The Solution Approach

3.1 Problem Representation

We represent the CLSP problem as a sequence of jobs, each with a predefined production quantity. A job is defined as the production of a quantity of one type of product and is subject to a deadline. The deadlines of the jobs are determined by the demand for the product in a given period, therefore the demand for a given period should be met by one or more jobs which have a deadline of that period's end date.

In order to schedule a given sequence of jobs, we start at the final job in the sequence and work backwards to the first job. The final job is scheduled to finish on the job's deadline. The last unscheduled job in the sequence is then scheduled to complete at minimum of that job's deadline or the starting time of the last job scheduled. This process is repeated until all jobs are scheduled. When a job is scheduled on its deadline, this essentially inserts idle time into the schedule and therefore we refer to this procedure as the idle time insertion algorithm. The pseudo code for the algorithm is presented in Figure 1.

Let S = the sequence of all jobs over all periods
Let S_i = the job being produced in the i^{th} position in the sequence S
Let P_{S_i} = the processing time of the job in the i^{th} position
Let $SU_{i,j}$ = the setup time for changing from job i to job j
Let D_{S_i} = the deadline for the job in the i^{th} position
Let C_{S_i} = the completion time of the job in the i^{th} position
$i = |S|$
$time = D_{S_i}$
do
    if $(time > D_{S_i})$ then
        $time = D_{S_i}$
    $C_{S_i} = time$
    $time = time - P_{S_i} - SU_{S_{i-1},S_i}$
    $i = i - 1$
until $i = 0$

Figure 1: Idle Time Insertion Pseudo Code

Any violation of a deadline constraint for a job is penalised in the objective function in proportion to the number of units of time the deadline is violated, however note that using the idle time insertion algorithm ensures no deadline penalties are incurred. Any production scheduled to be performed before time zero is also penalised by the number of units of time the schedule required

production before time zero. This penalty is similar to the idea, proposed in the literature, of having an initial inventory that is costed at a different rate to other inventory holdings, for example Vanderbeck [1998] and Degraeve and Jans [2007].

Recall that holding costs are only incurred if the product is held in inventory between time periods. If a job is started in one time period and completed in another, then the the number of units of the job completed before the end of the period are deemed to incur a holding cost.

For example, assume we have the demand, setup and and capacity data for a problem as given in Table 1.

Table 1: Example Data

	Demand in Period		Setup Time To Product		
Products	1	2	1	2	3
1	2	1	0	1	2
2	3	2	2	0	1
3	1	3	2	1	0
Capacity in Period	12	10			

Also assume each product takes one unit of time to produce. Assume also that we have the sequence as shown in Figure 2.

Position	1	2	3	4	5	6
Product	3	1	1	2	2	3
Quantity	1	2	1	3	2	3
Deadline period	1	1	2	1	2	2

Figure 2: Initial Sequence

In order to determine the schedule, the idle time insertion algorithm will start with the last job, producing 3 units of product 3, and schedule this on its deadline date, in this case at the end of period 2 or at time 22. The algorithm repeats this process scheduling jobs from the back of the sequence to the front, setting the completion time of the job either on the job's deadline date or at the start time of the last job that was scheduled, essentially scheduling the job **before** the last scheduled job. Hence the production of 2 units of product 2 is scheduled **before** the production of product 3, taking into account the necessary setup times. In the end we end up with the schedule shown in Figure 3, where an X indicates a setup is being done at this time.

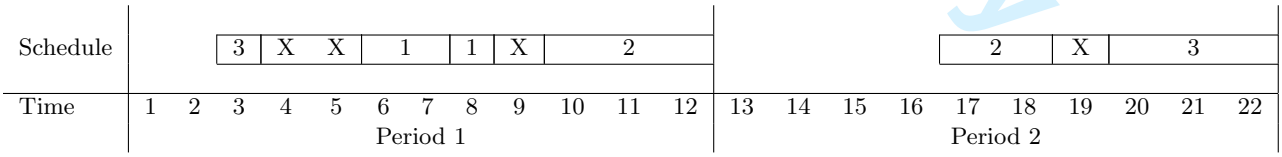

Figure 3: Sequence with Idle Time Inserted

3.2 Moves

We have implemented three different types of moves which form the basis for the neighbourhoods used in the metaheuristics we developed. These moves are:

- *Insert*: This takes a job and inserts it before another job in the sequence. The completion times of the jobs affected by this move are then re-evaluated and the objective value of the schedule is recalculated. Note that inserting a job before itself or before the job after itself are explicitly excluded from the neighbourhood as they do not change the schedule of jobs and therefore are non-productive moves. This neighbourhood also explicitly excludes moves that simply resequence a contiguous series of jobs producing the same product, as these moves do not change the objective function value.

- *Swap*: This takes two jobs in the sequence and swaps their location in the sequence. The sequence then has its idle time and objective value recalculated.

- *Fractional Insert*: This move is similar to an insert move however it allows the option of splitting a job into two jobs, where the total quantity produced is the same as the original job. One of these new jobs is left in the same position as the original job while the second job is inserted into a new location in the sequence. This move tries to preserve the periods in which jobs are currently produced, hence the quantity moved is dictated by the capacity available in the period where part of the job is being moved to. If there is enough capacity, then all of the job will be moved, if not then only the amount that can fit in the period will be moved. All locations within the period are tested as the capacity available in the period will vary depending on position the new job is inserted into and the jobs surrounding it because of the sequence dependency of the setup times. For example if we take the schedule in Figure 3 and try to insert the production of product 2, due in period 2 into the first position in the sequence then we find that this would not fit into the two remaining units of capacity in this period due to the required setup between products 2 and 3, so the fractional insert would split this job into two and insert one unit of production into the schedule at time 1, and produce the second unit at time 18. In the end we get a schedule that looks like Figure 4. Note that if we were to insert this production into the fourth position in the sequence then all of this job would be moved, as there is no extra setup required in this position and all two units could be produced in the capacity available.

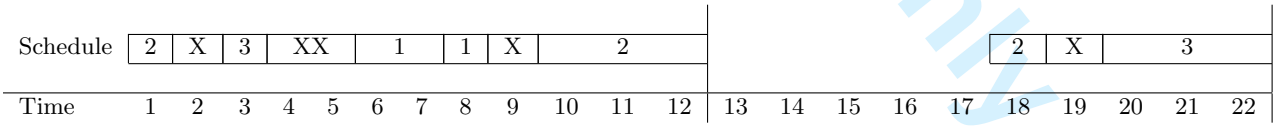

Figure 4: Fractional Insert Move - Splitting Jobs

Note that when there are two jobs producing the same product next to each other in the sequence it has the same overall processing time and cost as if this was one job, hence there is no need for an aggregation move that amalgamates two consecutive jobs that produce the same product and are due at the same time.

4 Metaheuristic Approaches

We compare two different metaheuristics for solving this problem. The first is a Variable Neighbourhood Search (VNS) and the second is a Tabu Search (TS).

4.1 Variable Neighbourhood Search (VNS)

VNS is a metaheuristic approach that has been used in for solving many different types of combinatorial optimization problems, see [Hansen and Mladenovic, 2003] for a review. The Basic VNS requires a list of neighbourhoods schemes that are going to be used to generate neighbours during the search. It incorporates a "Shaking Phase", that randomly chooses a neighbouring solution from the current neighbourhood scheme, and a "Local Search" phase that improves this solution. This new solution is then compared to the incumbent and if it is better it becomes the incumbent and we reset the current neighbourhood to the first neighbourhood. If it does not provide an improvement we move to the next neighbourhood scheme. [Hansen and Mladenovic, 2003] state that the most successful applications of VNS have replaced the local search phase with a variable neighbourhood descent (VND). VND starts with an incumbent solution and a list of neighbourhood schemes. Starting with the first neighbourhood scheme VND finds the best move using this neighbourhood scheme. If it is an improvement, move to it and reset the current neighbourhood scheme to the first neighbourhood scheme. If it does not improve on the incumbent, move to the next neighbourhood in the list and try again. Stop when all neighbourhoods have been tried and no improvement is found.

For this problem we use a VNS with a VND instead of the local search phase. The Fractional Insert and Swap neighbourhood schemes are used in the Shaking Phase, while the Insert and the Swap neighbourhood schemes are used in the VND Phase. Fractional Insert was chosen for the shaking phase as it can provide changes in the structure of the problem by changing both the sequence and the lot sizes in the problem, and hence provide diversification to the search. Insert was used in the VND phase as Insert provides the ability to fine tune the sequence, and hence provide a more intense set of neighbours. Swap was used in both the Shaking and VND phases as it provided a route to solutions which are difficult to get to via insert moves only due to the capacity constraints and penalties. Preliminary testing confirmed that this combination of neighbourhoods produced a very effective form of search.

4.2 Tabu Search (TS)

Tabu Search has been used successfully to solve many different types of combinatorial optimisation problems (see Glover and Laguna [1997]) for a review). Our TS implementation only includes short term memory, however it does alternate between two neighbourhood schemes, the Insert and Fractional Insert neighbourhoods.

The search initially starts off with the insert neighbourhood and after visiting a given number of local minima changes to the fractional insert neighbourhood. After visiting a given number of local minima using this scheme (which may be different from the number for the insert neighbourhood), the search then reverts back to the insert neighbourhood, and this process repeats until the stopping criteria is met.

A fixed tabu list was used that recorded the product type and the old completion time for the job being moved. Various tabu restrictions were tested. Tabu restrictions based on the position in the sequence did not perform well as the position in the sequence does not directly correlate to

the objective function, hence sequence based tabu restrictions allowed cycling in subsequences of jobs. The scheme that was found to work the best was to tabu the moved job's product and its original completion time in combination and to use a tabu restriction that did not allow a move if it produced a schedule where a job of that product type completed at the tabued completion time. Finding a job completing at a given time in the schedule could slow the search down, so for computational efficiency the tabu list also recorded the position the job has been moved from. This is the starting point in the sequence where the search will look for jobs completing around the time that is tabu. The rationale being that the completion times will probably not change dramatically around this position and, even if the completion times do change, it is fairly quick to scan the sequence from this position for the correct job.

The aspiration criteria used was the standard aspiration criteria used in most TS application, that is a tabu move was accepted if the solution it produced was better than the best solution found to date.

4.3 Starting Point

The initial solution to the problem for all of these metaheuristics was generated using the heuristic of Almada-Lobo et al. [2007]. It contains five (forward and backward) steps that are able to find feasible solutions efficiently, even for very tight and large problem instances.

This solution was then converted into jobs that correspond to the demands in the problem. If a job produced products for more than one period then this job was split into two consecutive jobs producing the same product, but with different due dates. This ensured that deadlines for each job represented demand exactly.

5 Lower Bound

In order to assess the overall performance of the metaheuristics, a good lower bound is needed to assess the quality of the metaheuristic results.

To generate good lower bounds we rely on the alternative stronger formulation presented in Almada-Lobo et al. [2007] that uses an exponential number of constraints (that can be separated in polynomial time). This formulation is obtained by replacing constraints (7) by the following set of inequalities:

$$\sum_{i\in S}\sum_{j\notin S} T_{ijt} + \sum_{i\in S} \alpha_{i(t+1)} \geq \sum_j T_{jkt} \qquad t \in [T], k \in S, S \subseteq [N]. \tag{11}$$

To find the most violated (t, S, k) inequalities, we implement the separation algorithm introduced by Almada-Lobo et al. [2007], which is based on the use of minimum cuts in directed graphs.

In order to tighten this second formulation, we use the (W_t) inequalities developed in Almada-Lobo et al. [2007], and the well known (l, S) inequalities for uncapacitated single-item lotsizing (Barany et al. [1984]). Hence, the lower bounds are obtained through the LP relaxation of this model, strengthened with these inequalities.

6 Computational Experiments

Random data sets were generated using the approach of [Almada-Lobo et al., 2007]. Elements of the problem were generated from a uniform distribution and then rounded to the nearest integer, or calculated from elements that were generated this way. The ranges used for the elements are as follows:

- Setup Times between 5 and 10 time units.
- Setup Costs are proportional to the setup time by a specified parameter (Cost of Setup per unit of time).
- Holding costs between 2 and 9 penalty units per period.
- Demand between 40 and 59 per period.
- Period Capacity is proportional to the total demand in that period as defined by a parameter (Capacity Utilisation per period, Cut): $C_t = \sum_i d_{it}/Cut$.
- Processing time for one unit = one unit of time.

Forty eight different problems types were created from the combinations of the following problem parameters:

- Number of Products (15, 25)
- Number of Periods (5, 10, 15)
- Capacity Utilisation per period: Cut (0.6, 0.8)
- Cost of Setup per unit of time: θ (50, 100)

In each case 10 different instances were generated, meaning a total of 240 problem instances were solved. Each type of instance can be characterized by the quadruple N, T, Cut and θ.

The MIP construction heuristic requires one parameter, which is the number of periods to solve at a time. For this we tested 2 and 3 period solutions. There is little point solvng single period solutions as this has little chance of creating good solutions. More than 3 periods at a time potentiallty took a lot of CPU time, hence resticting these experiments to only 2 and 3 periods at a time. For the MIP Improvement heuristic there are three parameters, the number of periods to solve, . The parameters that appeared to work well for most instances were a tabu list size of 10, 40 insert local minima and five fractional insert local minima before swapping neighbourhood schemes. As TS is deterministic, only one run of TS was required for each problem instance.

Computational experiments were performed on machines with Core 2 Duo 6600 CPUs running at 2.4 GHz, each with 2GB of random access memory. CPLEX 10.1 from ILOG was used as the mixed integer programming solver, while the separation algorithm, used to add the most violated (t, S, k) and (l, S) inequalities to the problem, was coded in C++.

An instance of type $N = 25, T = 15, Cut = 0.8, \theta = 100$ produces an MIP, as formulated in Section 2, with $11,636$ rows, $10,515$ columns and $130,180$ nonzeros. This is a fairly large MIP that is very hard to solve to optimality in a reasonable amount of time. Tables 2 and 3 present the average number of branch-and-bound nodes, the gap (%) between the upper bound obtained

by CPLEX 10.1 and the lowerbound, the number of instances (out of ten) that CPLEX found the optimal solution (No. Optimal), and the number of instances that CPLEX failed to produce any feasible solution (No. Undefined), and the average time to solve the problem instances. In our computational study, lower bounds are obtained through the LP relaxation of the model introduced in Section 5, strengthened with the (l, S) and (W_t) inequalities. The CPU time to compute the lower bound were always less than one minute. Each instance had a CPU time limit of one hour. An empty gap field means that CPLEX 10.1 was not able to generate at least one feasible solution within the time limit. As the size of the instance gets bigger (both in N and T), as capacity gets tighter and as setup costs increase, the CPLEX results worsen since the value of #opt. (#undef.) decreases (increases). When CPLEX solves these instances we find that, with all the other parameters being constant, the gap decreases as N increases. This seems counterintuitive, however it is due to the fact that the lower bound quality improves considerably as N increases, and does not come from a better upper bound. When we compare the gap between the best solution and the best bound obtained in the same run, the gap exponentially worsens as a function of N. It is also clear from the substantially lower number of branch-and-bound nodes that as N and T increase, the LPs become much more difficult to solve. CPLEX was not able to solve instances with $N = 25$ and $T > 5$ and also for some particular cases with $N = 15$ and even $N = 10$. As expected, and acknowledged by other researchers, the hardest instances are for $Cut = 0.8$ and $\theta = 100$.

Table 2: CPLEX 10.1 optimality gap (%) and nodes within the one hour time limit for $\theta = 50$

		$Cut = 0.6$				$Cut = 0.8$			
T	N	5	10	15	25	5	10	15	25
5	Nodes	212	10,034	157,228	61,455	1,564	273,742	104,974	50,001
	Gap	1.1%	0.2%	0.2%	0.5%	5.4%	0.7%	0.5%	
	No. Optimal	10	10	7	0	10	8	4	0
	No. Undefined	0	0	0	0	0	0	0	5
	Time	0.7	45.0	1521.5	3621.9	2.0	1191.2	3058.7	
10	Nodes	61,484	106,713	94,551	26,771	223,813	158,217	81,655	20,243
	Gap	1.6%	0.6%			4.8%	1.3%		
	No. Optimal	10	6	0	0	9	1	0	0
	No. Undefined	0	0	2	10	0	0	6	10
	Time	127.7	1799.2			713.7	3445.8		
15	Nodes	809,348	194,837	60,915	17,811	805,277	123,125	47,843	13,314
	Gap	2.3%	0.6%			5.2%			
	No. Optimal	4	0	0	0	1	0	0	0
	No. Undefined	0	0	9	10	0	1	8	10
	Time	2984.9	3654.9			3737.1			

To generate an upper bound, we implemented both the VNS and TS approaches from Section 4 in C++ by using Visual Studio 6.0. Both the VNS and TS metaheuristics stopped after a given amount of computational time. The time given for each search was determined from the size of the problem, as determined by $N \times T$. If $N \times T$ was less than 100 then each instance was run for 20 minutes, if less than 150 it was run for 40 minutes, and if greater or equal to 150 it was run for one hour. The running times of the starting point solution were always less than one second for all of the instances, and these starting points were generated before the search time allowed commenced.

The results are measured as a deviation from the lower bound, which is the same lower bound obtained through the LP relaxation of the model introduced in Section 5, and used in Tables 2 and 3. Tables 4 and 5 show the minimum, average, and maximum gap of the VNS solution from the lower bound, for $\theta = 50$ and $\theta = 100$, respectively.

Table 3: CPLEX 10.1 optimality gap (%) and nodes within the one hour time limit for $\theta = 100$

		$Cut = 0.6$				$Cut = 0.8$			
T	N	5	10	15	25	5	10	15	25
5	Nodes	577	165,580	280,034	41,037	12,395	498,100	222,649	54,427
	Gap	6.8%	2.2%	1.5%		11.2%	3.4%	2.1%	
	No. Optimal	10	10	0	0	10	5	1	0
	No. Undefined	0	0	0	1	0	0	0	3
	Time	1.1	735.4	3693.1		13.3	2433.0	3339.5	
10	Nodes	518,013	341,359	117,713	14,840	617,127	146,614	57,974	10,660
	Gap	9.0%	4.0%			12.8%	5.6%		
	No. Optimal	9	0	0	0	0	0	0	0
	No. Undefined	0	0	2	10	0	0	10	10
	Time	1301.0	3661.4			3725.1	3811.9		
15	Nodes	843,969	119,878	37,999	6,661	345,011	98,189	33,711	5,651
	Gap	11.8%	5.5%			16.2%			
	No. Optimal	0	0	0	0	0	0	0	0
	No. Undefined	0	0	5	10	0	10	10	10
	Time	3920.2	3966.4			3726.8			

Table 4: Gap (%) between the lower bound and VNS solution for $\theta = 50$

	$Cut = 0.6$			$Cut = 0.8$		
N	$T = 5$	$T = 10$	$T = 15$	$T = 5$	$T = 10$	$T = 15$
5	0.2/2.7/5.4	0.8/2.8/5.1	1.0/3.1/6.2	1.6/7.0/10.8	2.4/6.9/9.6	1.8/6.8/10.3
10	0.2/1.4/2.5	1.0/2.1/3.5	1.6/2.7/4.0	0.5/2.0/4.4	0.9/2.2/3.8	1.8/3.0/4.2
15	1.0/1.6/2.2	1.7/2.7/3.7	2.2/3.5/5.0	1.1/1.9/2.6	1.8/2.7/3.9	2.7/3.6/5.0
25	1.7/2.6/3.8	3.1/4.6/6.3	3.1/5.2/6.7	1.8/2.8/4.1	2.9/4.6/5.8	3.4/5.4/6.5

minimum / average / maximum gap (%)

Table 5: Gap (%) between the lower bound and VNS solution for $\theta = 100$

	$Cut = 0.6$			$Cut = 0.8$		
N	$T = 5$	$T = 10$	$T = 15$	$T = 5$	$T = 10$	$T = 15$
5	5.6/9.6/13.2	8.4/12.3/18.4	9.3/13.4/16.0	9.7/14.0/18.6	11.4/15.7/18.5	9.7/16.7/20.3
10	3.2/4.8/6.7	4.9/6.7/8.3	6.3/8.5/10.3	3.7/5.9/8.2	5.7/7.7/10.3	7.2/9.3/12.5
15	2.8/4.8/6.3	4.2/6.7/9.3	5.8/8.5/10.8	3.5/4.7/5.7	5.8/7.2/8.1	6.7/9.0/10.1
25	3.6/5.1/7.4	5.7/7.6/8.9	8.1/10.0/12.6	3.7/4.5/5.3	5.5/7.4/8.6	8.6/9.6/11.9

minimum / average / maximum gap (%)

When we compare Tables 2 and 4 and Tables 3 and 5, it is clear that CPLEX outperforms the heuristics in terms of deviation from the lower bound for the smaller sized problems. In fact, for most of the instances with $N = 5$ and a significant portion of instances with $N = 10$, CPLEX found the optimal solution. However, the heuristics outperformed CPLEX for the larger problem instances, as CPLEX does not find feasible solutions to these problems within the one hour time limit. Looking at the number of nodes CPLEX is able to investigate for these larger problems, we also see that as the problem gets bigger the number of branch-and-bound nodes CPLEX can consider in one hour reduces due to the extra complexity of the problem. From these trends we can see that even if we provided significantly more computational time or significantly faster processors it is still unlikely CPLEX would be able to solve these problems.

If we exclude the smallest instances, with $N = 5$, the results from the VNS indicate an average gap below 5% for $\theta = 50$ and below 10% for $\theta = 100$. We stress that most of the gap of the smallest instances comes from a weak lower bound. As an example, the instance $N = 5, T = 5, Cut = 0.8, \theta = 100$ for which was reported one of the worst results (14% of mean gap), presents an average gap between the heuristic solution and the CPLEX's upper bound (corresponding to the optimal solution – see Table 3) of 2.6%. The quality of the solution seems to be independent of the number of products (N) and deteriorates slightly as the number of periods increase (T). The results also show that, on average, the gap of the instances with $\theta = 50$ ($Cut = 0.6$) is lower than the gap of the instances with $\theta = 100$ ($Cut = 0.8$). Starting with the solution generated by the heuristic of Almada-Lobo et al. [2007], we were able to improve its quality considerably. For instance, *Almada-Lobo et al* present a gap of 24% for $N = 25, T = 5, Cut = 0.6$ and $\theta = 100$ against 5.1% displayed in Table 5. Table 6 gives the average gap between the lower bound and the starting point, TS and VNS solutions. Clearly, the deviation reduction obtained by the aforementioned techniques over the starting solution is significant.

Table 6: Gap (%) between the lower bound and the starting solution (StartPt), TS and VNS

	$T = 5$			$T = 10$			$T = 15$		
N	StartPt	TS	VNS	StartPt	TS	VNS	StartPt	TS	VNS
5	20.12	9.88	8.35	25.92	12.20	9.40	28.27	13.29	10.02
10	12.06	4.97	3.54	14.42	6.25	4.68	15.01	7.09	5.86
15	14.96	4.62	3.25	16.84	5.88	4.79	18.30	6.76	6.15
25	18.42	4.88	3.74	21.27	6.29	6.05	22.20	7.35	7.56

We now compare the effectiveness of TS against VNS. Let UB_{TS} and UB_{VNS} denote the values of the upper bounds obtained by TS and VNS approaches, respectively. Table 7 presents the gap $\frac{UB_{TS}-UB_{VNS}}{UB_{VNS}}$. Generally, VNS clearly outperforms TS for $\theta = 100$ and $Cut = 0.8$. However, as the problems get very large TS starts to outperform VNS in the one hour time limit provided for these problems. This can be seen more clearly in the deviation versus time graphs in Figures 5–7. As an example, the series TS $10 - 15$ in Figure 5 refers to the instance type $N = 10, T = 15, Cut = 0.8, \theta = 100$ tackled by TS. Note that the graphs stop at the point where the search found its best solution, which will normally be before the total time the search was allowed to run for. In Figure 7 we can see how VNS initially provides better solution faster, before being overtaken by TS at some point, and then slowly VNS converges to the TS solution. This is in contrast to the deviation versus time graphs of the smaller problems in Figure 5 where TS finds better solutions faster than VNS, but VNS quickly overtakes the TS result.

The reason for this pattern in performance can perhaps be explained by considering the balance between intensification and diversification between the two search techniques. VNS, as it is configured here, is a more diverse search than the TS that has been implemented here. Therefore with the smaller problems, TS looks carefully around a small area of the solution space which is based around the initial solution hence makes some quick progress. VNS, on the other hand, moves in a more diverse manner and finds other areas of the solution space that are perhaps more promising. This takes a little longer but provides better long run results.

When the size of the solution space is expanded considerably, the 'good' area of the solution space around the starting point expands considerably and therefore the VNS can quickly find good

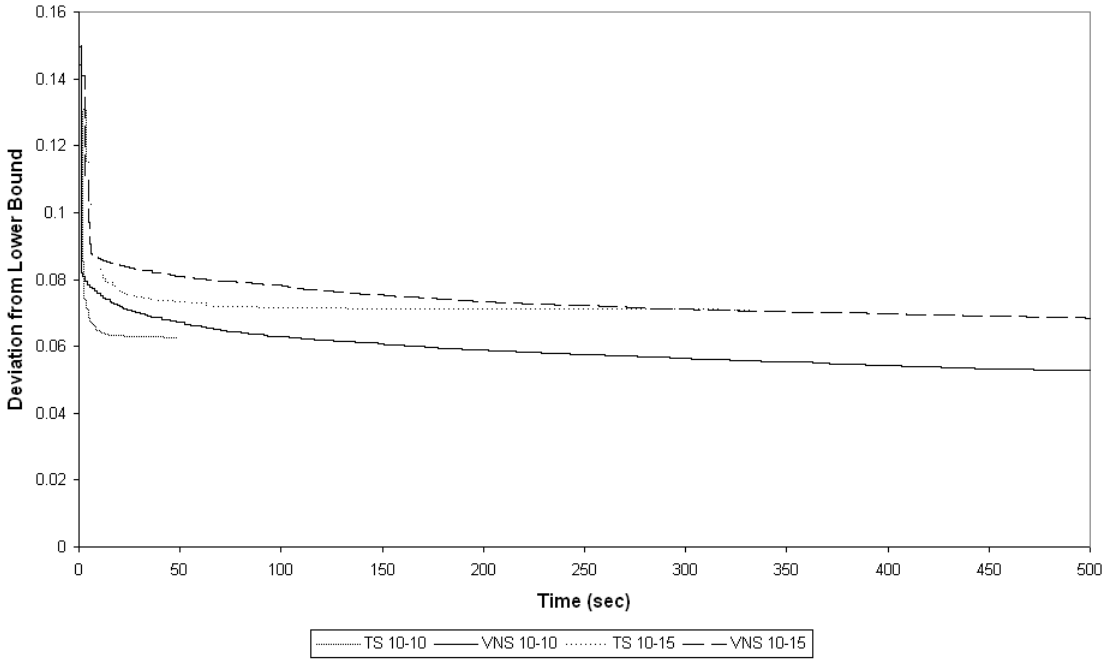

Figure 5: Average deviation from the lower bound over time for $N = 10, Cut = 0.8$ and $\theta = 100$

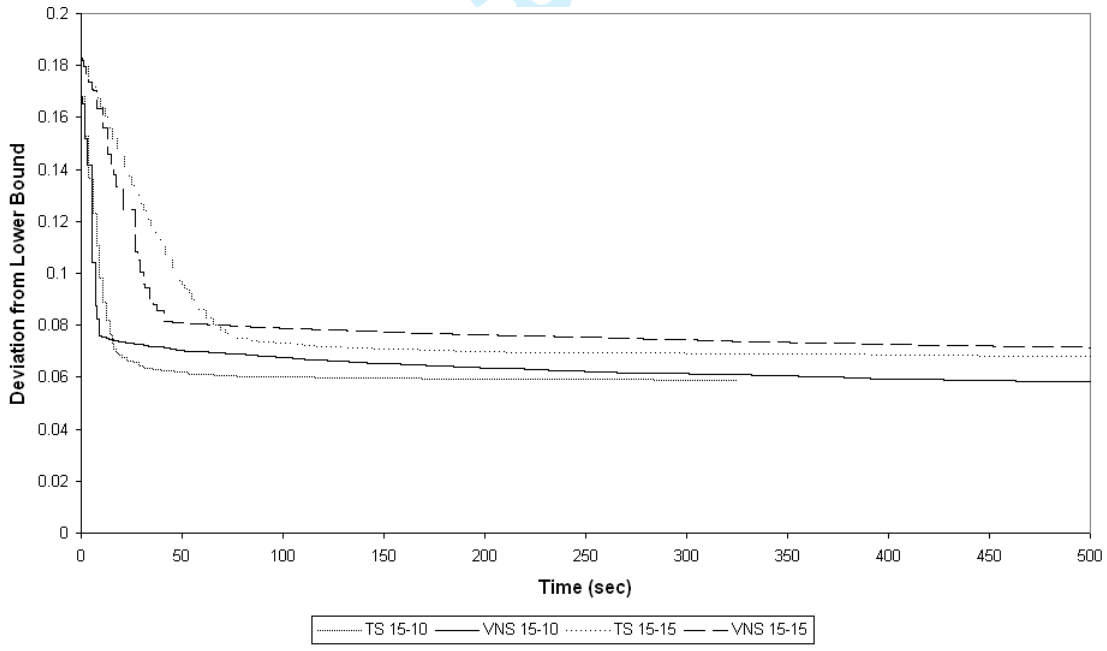

Figure 6: Average deviation from the lower bound over time for $N = 15, Cut = 0.8$ and $\theta = 100$

solutions in this area as it moves to other areas of the solution space. TS, on the other hand, considers more points in this area and therefore eventually finds a better points than what VNS

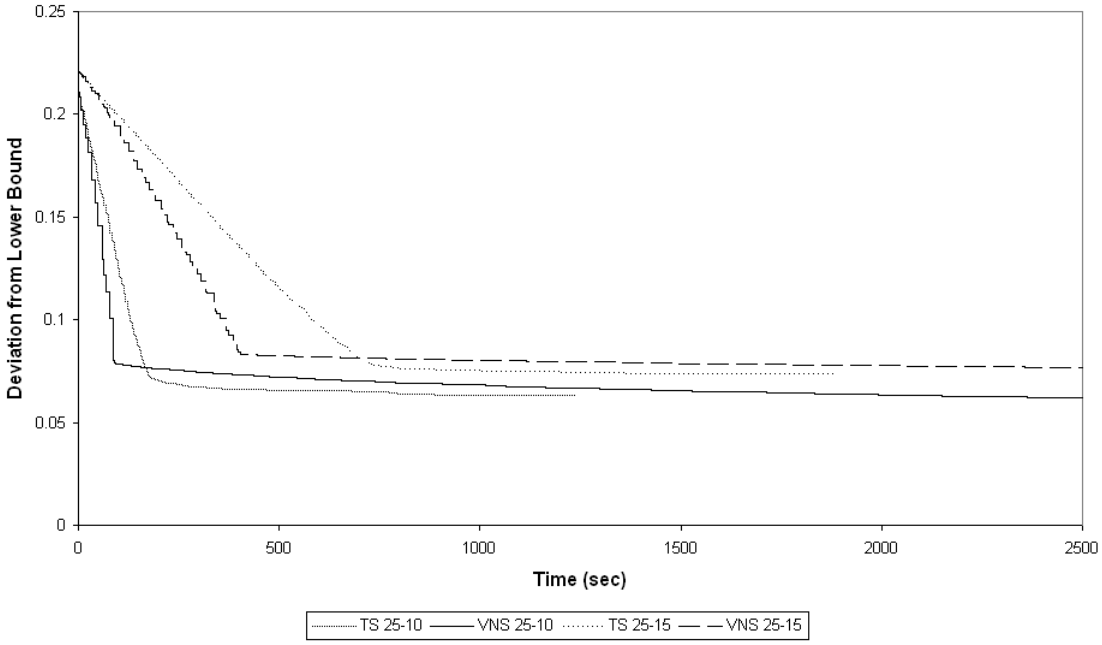

Figure 7: Average deviation from the lower bound over time for $N = 25, Cut = 0.8$ and $\theta = 100$

has found. As time progresses VNS slowly catches up to TS. If we gave these searches more computational time, then we would predict that VNS would eventually out perform the TS due to its more diverse nature.

Table 7: Comparison (%) of TS and VNS: $\frac{UB_{TS}-UB_{VNS}}{UB_{VNS}}$

	$Cut = 0.6$			$Cut = 0.8$		
N	$T = 5$	$T = 10$	$T = 15$	$T = 5$	$T = 10$	$T = 15$
5	0.1/1.4	1.0/2.6	1.6/5.4	1.4/2.6	2.1/4.4	2.0/2.8
10	1.0/2.1	0.4/2.2	0.7/1.3	0.5/1.9	1.0/2.4	1.0/1.7
15	0.9/1.4	0.6/0.8	0.3/0.4	1.2/1.7	1.0/1.7	0.4/1.1
25	0.8/0.7	−0.4/ − 0.2	−0.5/ − 1.0	0.9/2.0	0.1/1.3	−0.1/0.8

$\theta = 50$ / $\theta = 100$

7 Conclusions

Industrial lotsizing and scheduling poses some very hard analytical problems, especially challenging is the CLSP with sequence dependent setup times and costs that appears in different manufacturing contexts. We give new insights into current literature by developing very efficient Tabu Search and Variable Neighbourhood Search metaheuristics based on a solution representation that enables us to determine simultaneously the production quantities (lotsizing decision) and the setup variables

(sequencing decision). By integrating a big-bucket lotsizing and scheduling model and a batching scheduling model (converting lots into jobs), very good results are reported, even for highly capacitated instances with high setup to holding costs ratio. The research also showed that the preferred metaheuristic for solving this problem varied with the size of the problem. However if sufficient time was available, in general we found that the VNS as formulated here would eventually outperform the TS as formulated here.

In order to improve the computational results reported for this and similar problems and to address additional complexities stemming from the inclusion of other industrial features, such as parallel machines, inventory constraints and uncertainty to name but a few, the combination of different solution approaches is mandatory. Given their flexibility in dealing with complex manufacturing settings, metaheuristics will lie at the very heart of these hybrid approaches, if well combined with exact methods. The extension of the techniques proposed in this paper to the multiple machine CLSP with sequence dependent setups and setup carryovers is an interesting area for future research.

Acknowledgments

The authors thank the two anonymous referees for their valuable suggestions.

References

B. Almada-Lobo, D. Klabjan, M. A. Carravilla, and J. F. Oliveira. Single machine multi-product capacitated lotsizing with sequence-dependent setups. *International Journal of Production Research*, 45(20):4873–4894, 2007.

B. Almada-Lobo, J. F. Oliveira, and M. A. Carravilla. A note on "The capacitated lot-sizing and scheduling problem with sequence-dependent setup costs and setup times". *Computers and Operations Research*, 35(4):1374–1376, 2008.

S. Araujo, N. Arenales, and A. R. Clark. Joint rolling-horizon scheduling of materials processing and lot-sizing with sequence-dependent setups. *Journal of Heuristics*, 13(4):337–358, 2007.

I. Barany, T. J. Vanroy, and L. A. Wolsey. Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10):1255–1261, 1984.

W. Bruggemann and H. Jahnke. The discrete lot-sizing and scheduling problem: Complexity and modification for batch availability. *European Journal of Operational Research*, 124(3):511–528, 2000.

A. R. Clark and S. J. Clark. Rolling-horizon lot-sizing when set-up times are sequence-dependent. *International Journal of Production Research*, 38(10):2287–2307, 2000.

A. R. Clark, R. M. Neto, and E. A. V. Toso. Multi-period production setup-sequencing and lot-sizing through ATSP subtour elimination and patching. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 80–87, University of Nottingham, 2006.

Z. Degraeve and R. Jans. A new dantzig-wolfe reformulation and branch-and-price algorithm for the capacitated lot-sizing problem with setup times. *Operations Research*, 55(5):909–920, 2007.

F. Glover and M. Laguna. *Tabu Search*. Boston, 1997.

M. Gopalakrishnan, K. Ding, J. M. Bourjolly, and S. Mohan. A tabu-search heuristic for the capacitated lot-sizing problem with set-up carryover. *Management Science*, 47(6):851–863, 2001.

D. Gupta and T. Magnusson. The capacitated lot-sizing and scheduling problem with sequence-dependent setup costs and setup times. *Computers and Operations Research*, 32(4):727–747, 2005.

K. Haase. *Lot sizing and scheduling for Production Planning*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin, 1994.

P. Hansen and N. Mladenovic. Variable neighborhood search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer, 2003.

R. Jans and Z. Degraeve. An industrial extension of the discrete lot-sizing and scheduling problem. *IIE Transactions*, 36(1):47–58, 2004.

R. Jans and Z. Degraeve. Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches. *European Journal of Operational Research*, 177(3):1855–1875, 2007.

R. Jans and Z. Degraeve. Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, 46(6):1619–1643, 2008.

B. Karimi, S. M. T. F. Ghomi, and J. Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378, 2003. TY - JOUR.

B. Karimi, S. M. T. F. Ghomi, and J. M. Wilson. A tabu search heuristic for solving the clsp with backlogging and set-up carry-over. *Journal of the Operational Research Society*, 57(2):140–147, 2006.

J. Maes, J. O. McClain, and L. N. Vanwassenhove. Multilevel capacitated lotsizing complexity and lp-based heuristics. *European Journal of Operational Research*, 53(2):131–148, 1991.

H. Meyr. Simultaneous lotsizing and scheduling by combining local search with dual reoptimization. *European Journal of Operational Research*, 120(2):311–326, 2000.

H. Meyr. Simultaneous lotsizing and scheduling on parallel machines. *European Journal of Operational Research*, 139(2):277–292, 2002.

L. Ozdamar and M. A. Bozyel. The capacitated lot sizing problem with overtime decisions and setup times. *IIE Transactions*, 32(11):1043–1057, 2000.

L. Ozdamar, S. I. Birbil, and M. C. Portmann. Technical note: New results for the capacitated lot sizing problem with overtime decisions and setup times. *Production Planning and Control*, 13 (1):2–10, 2002.

Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer Series in Operations Research and Financial Engineering. Springer, New York, 2006.

C. Suerie. Modeling of period overlapping setup times. *European Journal of Operational Research*, 174(2):874–886, 2006.

C. Suerie and H. Stadtler. The capacitated lot-sizing problem with linked lot sizes. *Management Science*, 49(8):1039–1054, 2003.

F. Vanderbeck. Lot-sizing with start-up times. *Management Science*, 44(10):1409–1425, 1998.