SOME HEURISTICS FOR SOLVING PRODUCTION PLANNING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES

Zrinka Lukač

Faculty of Economics & Business Zagreb
Zagreb, Croatia
zlukac@efzg.hr

Abstract

This paper addresses the problem of production planning of multiple items on two non-identical machines with sequence dependent setup times in order to meet deterministic dynamic demand and minimize overall costs consisting of production, holding and setup costs. The problem is formulated as quadratic 0-1 mixed-integer programming problem. Its linear formulation is also given. Three heuristics based on variable neighborhood search (VNS) are developed. Heuristics are tested and compared to Lagrangean relaxation heuristics, three versions of tabu search heuristics and CPLEX. Computational results show tradeoff between solution quality and run time needed to obtain solution.

Keywords - lot-sizing, sequence dependent setup times, lagrangean relaxation, tabu search, variable neighborhood search, heuristic

1 INTRODUCTION

Ever since the beginning of industrialization, production planning problems have posed an important problem for practitioners and researchers alike. Since 1950s, there has been an ample research in the area resulting in many different theoretical advances and successful applications alike.

Scheduling problems involving setup times and costs are of particular interest in many process industries and have drawn much attention recently. A survey on the subject can be found in Allahverdi et al. [1] and Allahverdi et al. [2]. Scheduling problems often arise on the operational level and deal with daily, weekly or monthly production. An efficient scheduling decision which takes into account setup considerations may result in significant savings for such an industry. In particular, we have encountered this problem in one pharmaceutical company that produces different kinds of drugs and food supplements using two production facilities. Each facility is capable of producing any type of product and has different technological constraints. Within the same time period each facility can produce one type of product only. Furthermore, production changeover from one product type to another requires cleaning of the facility, where cleaning time is sequence dependent. For example, if in a given time period facility produces antibiotics and in the following period vitamins, it needs to be cleaned thoroughly and cleaning process lasts for a long time. On the other hand, if vitamins are being produced first, cleaning the facility should not be so thorough and therefore cleaning process lasts much shorter. Cleaning is performed at the beginning of time period, while the rest of the period can be used for production. If there is no change of production type, no cleaning is needed. The problem is to find production plan that meets demand and minimizes total costs. It can be modeled as lot-sizing problem on two non-identical parallel machines with sequence dependent setup times.

Generally, lot-sizing problems (LSP) deal with how to plan production of multiple products over planning horizon divided into finite number of periods of time so as to meet deterministic dynamic demand and minimize the sum of production costs, inventory holding costs and machine setup costs. In order to do so, decisions regarding exact production sequence and production quantities are needed. Regarding complexity, lot sizing problems are known to be NP-hard [3].

The text is organized as follows. In Section 2 mathematical formulation of the problem is given. The problem is formulated as quadratic 0-1 mixed-integer programming problem. The linear formulation of the model is also given. In Section 3 three different heuristic approaches for the problem are presented: Lagrangean relaxation, Tabu search and VNS. Efficiency of the heuristics proposed is tested by computational study in Section 4. Finally, Section 5 contains conclusions.

2 MODEL FORMULATION

We consider the problem of how to plan production of *n* different items on two non-identical machines over finite planning horizon divided into *T* periods of time in order to fulfill demand, where deterministic dynamic demand is assumed. The model does not allow backlogging. Each machine is capable of producing all types of items. Therefore, an item can be produced on either the first or the second machine, but it is also possible to produce the same item on both machines simultaneously. Moreover, within the same time period each machine may produce one type of item only. Therefore, this model belongs to the class of small-bucket lot sizing models. Whenever there is a switch of production from one item type to another, a sequence dependent setup operation is to be performed on a machine at the beginning of time period, while the remaining time in that period is available for production. Furthermore, setup operations consume machine resources and incur setup costs. We assume that the machine capacity is finite and expressed in time units. The setup costs are sequence independent. The storage of items incurs holding costs. Therefore, the whole production process incurs production costs, holding costs and setup costs. Additional assumption on the model is that the machines are deteriorating with time, thus increasing the setup time as machines are getting older.

2.1 Quadratic formulation

In order to mathematically formulate the problem [8], we use the following notation.

Let $i=1,\ldots,n$ denote item types, let $j=1,2$ denote machine types and let $t=1,\ldots,T$ denote time periods within planning horizon. Furthermore, let d_{it} denote demand for item *i* in period *t*, p_{itj} production cost per unit of item *i* on machine *j* in period *t*, h_{it} holding cost per unit of item *i* in period *t*, k_{itj} fixed setup cost of item *i* on machine *j* in period *t*, u_{iltj} the setup time needed to switch production from item *i* to item *l* in period *t* on machine *j*, c_{tj} capacity of machine *j* in period *t* (in time units), a_{ij} capacity consumption of machine *j* per unit of item *i* (in time units) and M_{itj} upper bound on production of item *i* on machine *j* in time period *t*, where $M_{itj}=\dfrac{c_{tj}}{a_{ij}}$.

The machine deterioration is modeled by assuming that setup times u_{iltj} required for the setup operation from item *i* to item *l* are mutually proportional with respect to some proportionality factor *r>1*, i.e. we assume that $u_{il(t+1)j}=r\cdot u_{iltj}$.

The decision variables are: x_{itj} - amount of item *i* produced on machine *j* in period *t*, s_{it} - inventory of item *i* in period I,*t* and $y_{itj}=\begin{cases}1 & \text{if machine } j \text{ is setup for item } i \text{ in period } t\\ 0 & \text{otherwise}\end{cases}$.

The problem can be formulated as the following mixed-integer 0-1 quadratic programming problem [8]:

$$\min\ f(\mathbf{x},\mathbf{y},\mathbf{s})=\sum_{i=1}^{n}\sum_{t=1}^{T}\left(\sum_{j=1}^{2}p_{itj}x_{itj}+h_{it}s_{it}+\sum_{j=1}^{2}k_{itj}y_{itj}\right) \qquad (1)$$

subject to

$$s_{i,t-1}+\sum_{j=1}^{2}x_{itj}=d_{it}+s_{it} \qquad \forall i,t \qquad (2)$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n}u_{iltj}y_{i(t-1)j}y_{ltj}+\sum_{i=1}^{n}a_{ij}x_{itj}\le c_{tj} \qquad \forall t,j \qquad (3)$$

$$x_{itj}\le M_{itj}y_{itj} \qquad \forall i,t,j \qquad (4)$$

$$\sum_{i=1}^{n} y_{itj} \le 1 \qquad \forall t, j \tag{5}$$

$$x_{itj}, s_{it} \ge 0 \qquad \forall i, t, j \tag{6}$$

$$y_{itj} \in \{0,1\} \tag{7}$$

The objective function (1) minimizes the sum of production, holding and setup costs. Constraints (2) are inventory balance constraints. Also, they ensure that demand is to be met. Quadratic constraints (3) are machine capacity constraints stating that the time needed for production and setup operations cannot exceed the available capacity of the machine (capacity is expressed in time units). The quadratic term $\sum_{i=1}^{n}\sum_{l=1}^{n} u_{iltj} y_{i(t-1)j} y_{ltj}$ in constraints (3) which refers to setup operations reduces to one term only: (i) 0, if there is no change of production type from period *t-1* to period *t*; or (ii) setup time u_{iltj} if machine is setup to production of item *i* in period *t*-1 ($y_{i(t-1)j} = 1$) and to production of item *l* in period *l* ($y_{ltj} = 1$) (in other words, setup operation is required at the beginning of period *t*). Due to constraints (4), in period *t* each machine can produce only an item type to which it is setup to. Constraints (5) ensure that in each time period there is at most one setup state for each machine. Finally, constraints (6) and (7) impose non-negativity and binary conditions respectively.

2.2 Linear formulation

Problem (1)-(7) can also be formulated as a linear programming problem. In order to do so and avoid the quadratic term in constraints (3), we introduce additional 0-1 variables w_{iltj}:

$$w_{iltj} = \begin{cases} 1 & \text{if a setup operation from item } i \text{ to item } l \text{ is performed on machine } j \\ & \text{at the beginning of period } t \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The problem can now be formulated as the following linear programming problem:

$$\min f(\mathbf{x},\mathbf{y},\mathbf{s},\mathbf{w}) = \sum_{i=1}^{n}\sum_{t=1}^{T}\left(\sum_{j=1}^{2} p_{itj} x_{itj} + h_{it} s_{it} + \sum_{j=1}^{2} k_{itj} y_{itj}\right) \tag{9}$$

subject to

$$s_{i,t-1} + \sum_{j=1}^{2} x_{itj} = d_{it} + s_{it} \qquad \forall i, t \tag{10}$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n} u_{iltj} w_{iltj} + \sum_{i=1}^{n} a_{ij} x_{itj} \le c_{tj} \qquad \forall t, j \tag{11}$$

$$x_{itj} \le M_{itj} y_{itj} \qquad \forall i, t, j \tag{12}$$

$$w_{iltj} \ge y_{i(t-1)j} + y_{ltj} - 1 \qquad \forall i, l, t, j, i \ne l \tag{13}$$

$$w_{iltj} \le y_{i(t-1)j} \qquad \forall i, l, t, j \tag{14}$$

$$w_{iltj} \le y_{ltj} \qquad \forall i, l, t, j \tag{15}$$

$$\sum_{i=1}^{n} y_{itj} \le 1 \qquad \forall t, j \tag{16}$$

$$x_{itj}, s_{it} \ge 0 \qquad \forall i, t, j \tag{17}$$

$$y_{itj}, w_{iltj} \in \{0,1\} \tag{18}$$

This model differs from model (1)-(7) in constraints (11), (13)-(15). Constraints (11), just like constraints (3), are machine capacity constraints but this time expressed in linear form. Constraints (13)-(15) describe the relationship between variables w_{iltj} and y_{ltj} which can be stated in quadratic form as

$$w_{iltj} = y_{i(t-1)j} \cdot y_{itj}, \qquad \forall i,l,t,j,i \neq l \tag{19}$$

The problem formulation in its linear form can be solved by using commercial software like CPLEX.

3 HEURISTICS

We propose several heuristics based on VNS method for solving the problem. VNS is a recent metaheuristics [7] which systematically exploits the idea of neighborhood change as a mean of escaping from local optima. The basic form of VNS consists of exploring the set of neighborhoods of the current solution, conducting local search from a neighbor solution to local optima and then moving to it only if it results in improvement of search criterion.

VNS based heuristics proposed here use Lagrangean relaxation heuristic and tabu search heuristic presented in [8]. Therefore first we briefly present these two heuristics.

3.1 Lagrangean relaxation heuristics

This heuristics works on quadratic mixed-integer formulation (1)-(7) of the problem. Hereby we identify constraints (3) as the set of complicating constraints to be relaxed. In order to describe Lagrangean heuristics more easily, we introduce the following notation. Let

$$g_{jt}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n}\sum_{l=1}^{n} u_{iltj}\, y_{i(t-1)j}\, y_{ltj} + \sum_{i=1}^{n} a_{ij} x_{itj} - c_{tj} \qquad \forall t,j \tag{20}$$

let

$$F_{tj}(\mathbf{y}) = \sum_{i=1}^{n}\sum_{l=1}^{n} u_{iltj}\, y_{i(t-1)j}\, y_{ltj} \qquad \forall t,j \tag{21}$$

and let X be the set of all (**x**, **y**, **s)** satisfying constraints (2), (4)-(7). Now the problem (1)-(7) can be written as

$$\min f(\mathbf{x},\mathbf{y},\mathbf{s}) \tag{22}$$

s.t.

$$g_{jt}(\mathbf{x},\mathbf{y}) \leq 0 \qquad \forall t,j \tag{23}$$

$$(\mathbf{x},\mathbf{y},\mathbf{s}) \in X \tag{24}$$

Since the expression $\sum_{i=1}^{n}\sum_{l=1}^{n} u_{iltj}\, y_{i(t-1)j}\, y_{ltj}$ reduces to one term only: (i) 0, if there is no change of production type; and (ii) u_{iltj} if there is a change of production from item *i* to item *l*, we know that $F_{tj}(\mathbf{y}) \in \left[0, F_{tj}^{\max}\right]$, where $F_{tj}^{\max} = \max_{i,l} u_{iltj}$, $\forall t,j$. Let *LB* denote lower bound and let *UB* denote upper bound on optimal value of the problem (22)-(24).

The Lagrangean relaxation heuristics is now as follows.

Lagrangean relaxation heuristics (H-LR):

Step 1 ***Initialization***

Dualize set (19) of complicating constraints.

Let UB = + Infinity; let LB = - Infinity

Step 2 ***Initialize Lagrangean vector***

For *r=1* to *R* **do**

Fix $F_{tj}(\mathbf{y})$ randomly to some value from the interval $\left[0, F_{tj}^{\max}\right]$ and solve problem (P).

Denote the so obtained solution as $\mathbf{x}^r, \mathbf{y}^r, \mathbf{s}^r$.

If $\mathbf{x}^r, \mathbf{y}^r, \mathbf{s}^r$ is feasible to (22)-(24), update upper bound UB.

Use the so obtained solution to initialize Lagrangean vector λ.

Step 2 ***Obtain lower bound***

Solve the relaxed problem

$$\min_{\mathbf{x},\mathbf{y},\mathbf{s}} f(\mathbf{x},\mathbf{y},\mathbf{s}) + \sum_{t=1}^{T}\sum_{j=1}^{2} \lambda_{tj} g_{tj} \tag{25}$$

$$\text{s.t.} \quad (\mathbf{x},\mathbf{y},\mathbf{s}) \in X \tag{26}$$

Denote the so obtained solution as $\mathbf{x}(\lambda), \mathbf{y}(\lambda), \mathbf{s}(\lambda)$.

Update LB; If $\mathbf{x}(\lambda), \mathbf{y}(\lambda), \mathbf{s}(\lambda)$ is feasible to (22)-(24), update UB.

Step 3 ***Stopping criterion***

If $UB - LB < 0.05 \cdot (\frac{UB + LB}{2})$, STOP. Accept $\mathbf{x}(\lambda), \mathbf{y}(\lambda), \mathbf{s}(\lambda)$ as optimal solution.

Otherwise, use $\mathbf{x}(\lambda), \mathbf{y}(\lambda), \mathbf{s}(\lambda)$ to update Lagrangean vector and go to Step 2.

The heuristic stops when the difference between upper and lower bound becomes less then 5% of their arithmetic mean *(UB+LB)/2*.

3.2 Tabu Search heuristics

Next we describe tabu search heuristics presented in [8]. Tabu search [5],[6] has proved to be very efficient in solving many hard combinatorial problems. Good choice of search space and neighborhood structure are crucial for its effective implementation.

In order to formulate the tabu search heuristics for problem (1)-(7), auxiliary 0-1 variables z_{ij} are introduced. They determine which items are going to be produced on which machine:

$$z_{ij} = \begin{cases} 1 & \text{if item } i \text{ is producedon machine } j \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

According to the definition, constraints

$$y_{itj} \le z_{ij} \qquad \forall i,t,j \tag{28}$$

$$z_{ij} \in \{0,1\} \qquad \forall i,j \tag{29}$$

can also be added to the set of problem constraints (2)-(7), without changing the solution space. Therefore, from now on we consider problem (1)-(7),(28)-(29).

This heuristic also operates on the quadratic formulation of the problem (1)-(7), (28)-(29). The search space is defined with respect to auxiliary variables z_{ij} defined by (27). Once we have values of variables z_{ij}, values of decision variables **x**, **y**, **s** are obtained by fixing **z,** fixing $y_{itj} = 0$ for all indexes i, t, j such that $z_{ij} = 0$ (because of constraints (28)), and then applying *H-LR* heuristic.

Starting solutions are constructed by decomposing items to the machines, i.e. by determining initial values of $\mathbf{z}$. The underlying rationale is to try to fix $z_{ij} = 0$ for as many indexes as possible, because then according to (28) many of the variables y_{itj} become 0 as well. This significantly reduces the size of the problem. Variables z_{ij} determine the set of items to be produced on each machine. The less time we spend on setup operations, the more time we have available for production. However, even if we know the set of items to be produced on a given machine, we still don't know the exact sequence of production. Therefore, for each machine *j*, *j=1,2*, we consider all pairs of items *(i,l)* to be produced on machine *j* and compute the sum of their setup times $u_{iltj} + u_{litj}$. If this sum is sufficiently small, i.e. if $u_{iltj} + u_{litj} \le s$ for some *s*, we assign both items to machine *j*. The items still left unassigned are then distributed to the machines randomly. For different values of *s* we obtain different starting solutions of the search. In order to create the first starting solution, we use *s* as small as possible.

For each solution from the search space, we construct a neighborhood consisting of four solutions. Given an arbitrary solution $\mathbf{z}$, its neighborhood is constructed as follows. First consider the first machine and find the pair of items *(i,l)* scheduled for production on the first machine for which the setup time u_{iltj} is maximal. The first two neighborhood solutions are obtained by separating these two items: the first solution by sending item *i* to be produced on the second machine ($z_{i1} = 0, z_{i2} = 1$) and keeping item *l* to be produced on the first machine ($z_{l1} = 1$), the second solution by keeping item *i* to be produced on the first machine ($z_{i1} = 1$) and sending item *l* to be produced on the second machine ($z_{l1} = 0, z_{l2} = 1$). The last two neighborhood solutions are obtained in the same manner, but now by fixing the second machine and then separating the items. Again, the pair of products *(i,l)* scheduled for production on the second machine for which setup time u_{iltj} is maximal is found first and then separated, first by sending item *i* to be produced on the first machine ($z_{i2} = 0, z_{i1} = 1$) and keeping item *l* to be produced on the first machine ($z_{l2} = 1$), and then by keeping item *i* to be produced on the second machine ($z_{i2} = 1$) and sending item *l* to be produced on the first machine ($z_{l2} = 0, z_{l1} = 1$).

Tabu list consists of the last *L* moves, where *L* is fixed and prescribed. Aspiration criterion is set to the value of the objective function value in the best solution found so far.

Let *NS* denote the total number of generated starting points, let *S* denote their counter, let *NU* denote the allowed number of uphill moves during the search.

Now the tabu search heuristics is as follows:

<u>T-LR Heuristics:</u>

Step1 ***Initialization***

a) Set *S=1*
b) Set *J=0*. If *S>NS*, go to Step 4; otherwise go to Step 2.

Step 2 ***Choice***

a) Generate *S-th* starting decomposition $\mathbf{z}_{[S]}$. Fix $\mathbf{z}_{[S]}$ and find $\mathbf{x}(\lambda), \mathbf{y}(\lambda), \mathbf{s}(\lambda)$ by using *H-LR*. If *S=1*, set $\mathbf{x}^* = \mathbf{x}_{[S]}, \mathbf{y}^* = \mathbf{y}_{[S]}, \mathbf{s}^* = \mathbf{s}_{[S]},\ f^* = f(\mathbf{x}_{[S]}, \mathbf{y}_{[S]}, \mathbf{s}_{[S]})$.

b) Construct neighborhood solutions $\mathbf{z}'_{[I]}$, *I=1,...,4*. For each *I*, fix $z'_{[I]}$ and find $\mathbf{x}'_{[I]}, \mathbf{y}'_{[I]}, \mathbf{s}'_{[I]}$ by using H-LR. If $\mathbf{z}'_{[I]}$ is not tabu or satisfies aspiration criteria, compute $\bar{f}_{[I]} = f(\mathbf{x}'_{[I]}, \mathbf{y}'_{[I]}, \mathbf{s}'_{[I]})$.

c) As new $\mathbf{x}_{[S]}, \mathbf{y}_{[S]}, \mathbf{s}_{[S]}, \mathbf{z}_{[S]}$, choose the allowable neighborhood point having the minimal objective function value. Let *B* be the index of that point and let $\hat{f}$ be its objective function value.

Step 3 ***Update***

a) Update tabu list and aspiration criterion. If $\hat{f} \geq f^*$, set *J=J+1*; otherwise set $\mathbf{x}^* = \mathbf{x}'_{[B]}, \mathbf{y}^* = \mathbf{y}'_{[B]}, \mathbf{s}^* = \mathbf{s}'_{[B]}, \mathbf{z}^* = \mathbf{z}'_{[B]}, f^* = \hat{f}$ and set *J=0*.

b) If *J>NU*, set *S=S+1* and go to Step 1b. Otherwise go to Step 2b.

Step 4 ***Termination***

STOP. The optimal solution has been achieved.

3.3 Variable Neighborhood Search heuristics

Here three versions of the VNS based heuristics for the problem in question are being proposed.

The search space of these three heuristics is defined with respect to auxiliary variables z_{ij} defined by (27). They determine which items are going to be produced on which machines. Therefore heuristics consider quadratic formulation of the problem (1)-(7), (28)-(29). As in T-LR heuristics, once we have values of variables z_{ij}, the values of decision variables **x**, **y**, **s** are obtained by using *H-LR* heuristics.

The first VNS heuristic is based on basic VNS scheme. The search space of this heuristic is defined with respect to variables $\mathbf{z}$. The heuristic uses two types of neighborhood structures: (i) $N_k(\mathbf{z})$, $k = 1,..., k_{\max}$, where $k_{\max}$ is prescribed; and (i) $LN(\mathbf{z})$. Here $N_k(\mathbf{z})$ is defined as the set of all $\mathbf{z}'$ for which the Hamming distance from $\mathbf{z}$ is equal to k. In other words, $N_k(\mathbf{z})$ is the set of all $\mathbf{z}'$ that have exactly k coordinates different from $\mathbf{z}$. Neighborhood structure $LN(\mathbf{z})$ consists of four constructed solutions and two randomly generated solutions. For a given $\mathbf{z}$, the first four neighborhood solutions of $LN(\mathbf{z})$ neighborhood structure are constructed in a same manner as neighborhood structure for T-LR heuristics. In other words, for each machine we find the pair of items for which the setup time is maximal and then separate them. The last two neighborhood solutions of $LN(\mathbf{z})$ neighborhood structure are obtained by generating two solutions which differ from $\mathbf{z}$ in a random number of coordinates.

The local search step of the first VNS heuristics uses T-LR heuristic, as described in section 3.2.

The VNS1 heuristics is now as follows:

<u>VNS1 Heuristics:</u>

Step 1 ***Initialization***

a) Select the set of neighborhood structures $N_k(\mathbf{z})$ and $LN(\mathbf{z})$
b) Generate initial value of $\mathbf{z}$; fix $\mathbf{z}$; find $\mathbf{x}, \mathbf{y}, \mathbf{s}$ by using *H-LR* heuristics
c) Set $\mathbf{x}^* = \mathbf{x}, \mathbf{y}^* = \mathbf{y}, \mathbf{s}^* = \mathbf{s}, \mathbf{z}^* = \mathbf{z}, f^* = f(\mathbf{x}, \mathbf{y}, \mathbf{s})$.

Step 2 ***Repeat*** *until the objective function value is being improved:*

a) Set *k=1*
b) Repeat until *k=k_{max}:*

i. *Shaking:* Generate a point $\mathbf{z}' \in N_k(\mathbf{z})$ at random. Find $\mathbf{x}', \mathbf{y}', \mathbf{s}'$ by using *H-LR* heuristics
ii. *Local Search:* Apply *T-LR* heuristic with $\mathbf{z}'$ as initial solution. For *T*-LR use $LN(\mathbf{z}')$ neighborhood structure. Denote with $(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'', \mathbf{z}'')$ the so obtained local optima.
iii. *Move or not:* If $f(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'') < f^*$, set $\mathbf{x}^* = \mathbf{x}'', \mathbf{y}^* = \mathbf{y}'', \mathbf{s}^* = \mathbf{s}'', \mathbf{z}^* = \mathbf{z}''$, $f^* = f(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'')$ and set *k=1.* Otherwise set *k=k+1.*

The VNS2 heuristic is also based on basic VNS scheme. It uses the same search space and the same two types of neighborhood structures $N_k(\mathbf{z}), k = 1,..., k_{\max}$ and $LN(\mathbf{z})$ as VNS1. The main difference to VNS1 is in its local search step. Instead of using *T-LR*, here we look for the best neighbor within the $LN(\mathbf{z})$ neighborhood structure of the current solution.

VNS2 Heuristics:

Step 1 ***Initialization***

a) Select the set of neighborhood structures $N_k(\mathbf{z})$ and $LN(\mathbf{z})$
b) Generate $\mathbf{z}$; fix $\mathbf{z}$; find $\mathbf{x}, \mathbf{y}, \mathbf{s}$ by using *H-LR* heuristics
c) Set $\mathbf{x}^* = \mathbf{x}, \mathbf{y}^* = \mathbf{y}, \mathbf{s}^* = \mathbf{s}, \mathbf{z}^* = \mathbf{z}, f^* = f(\mathbf{x}, \mathbf{y}, \mathbf{s})$.

Step 2 ***Repeat*** *until the objective function value is being improved:*

a) Set *k=1*
b) Repeat until *k=k$_{max}$:*
    i. *Shaking:* Generate a point $\mathbf{z}' \in N_k(\mathbf{z})$ at random. Find $\mathbf{x}', \mathbf{y}', \mathbf{s}'$ by using *H-LR* heuristics
    ii. *Local Search:* Find the best neighbor $\mathbf{z}'' \in N_k(\mathbf{z}')$. Obtain $\mathbf{x}'', \mathbf{y}'', \mathbf{s}''$ by using *H-LR* heuristics. Denote with $(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'', \mathbf{z}'')$ the so obtained local optima.
    iii. *Move or not:* If $f(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'') < f^*$, set $\mathbf{x}^* = \mathbf{x}'', \mathbf{y}^* = \mathbf{y}'', \mathbf{s}^* = \mathbf{s}'', \mathbf{z}^* = \mathbf{z}''$, $f^* = f(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'')$ and set *k=1.* Otherwise set *k=k+1.*

Unlike VNS1 and VNS2, VNS3 is based on reduced VNS scheme. It still uses the same solution space as VNS1 and VNS2, but it uses neighborhood structures $N_k(\mathbf{z}), k = 1,..., k_{\max}$ only. The stopping criterion is the maximum number of iterations controlled by *max_iter* parameter.

VNS3 Heuristics:

Step 1 ***Initialization***

a) Select the set of neighborhood structures $N_k(\mathbf{z})$ and $LN(\mathbf{z})$
b) Set *iter=1.*
c) Generate $\mathbf{z}$; fix $\mathbf{z}$; find $\mathbf{x}, \mathbf{y}, \mathbf{s}$ by using *H-LR* heuristics
d) Set $\mathbf{x}^* = \mathbf{x}, \mathbf{y}^* = \mathbf{y}, \mathbf{s}^* = \mathbf{s}, \mathbf{z}^* = \mathbf{z}, f^* = f(\mathbf{x}, \mathbf{y}, \mathbf{s})$.

Step 2 ***Repeat*** *until iter = max_iter:*

b) Set *k=1*
c) Repeat until *k=k$_{max}$:*
    i. Set *k=1.*
    ii. *Shaking:* Generate a point $\mathbf{z}' \in N_k(\mathbf{z})$ at random. Find $\mathbf{x}', \mathbf{y}', \mathbf{s}'$ by using *H-LR* heuristics
    iii. *Move or not:* If $f(\mathbf{x}', \mathbf{y}', \mathbf{s}') < f^*$, set $\mathbf{x}^* = \mathbf{x}', \mathbf{y}^* = \mathbf{y}', \mathbf{s}^* = \mathbf{s}', \mathbf{z}^* = \mathbf{z}'$, $f^* = f(\mathbf{x}'', \mathbf{y}'', \mathbf{s}'')$ and set *k=1.* Otherwise set *k=k+1.*

4 COMPUTATIONAL RESULTS

We have generated 9 problem classes. Each class is defined by number of different items to be produced and length of planning horizon. Table 1 shows the size of generated problem classes and corresponding number of variables, constraints and 0-1 variables for the quadratic formulation of the problem, while Table 2 shows the size of the problem classes and the corresponding number of variables and constraints for the linear formulation of the problem

Table 1. Number of variables, constraints and 0-1 variables for quadratic formulation of the problem

Problem Class (T,n)	**No. of Variables**	**No. of Constraints**	**No. of 0-1 Variables**
(4,3)	66	76	30
(4,4)	88	96	40
(6,4)	128	144	56
(10,5)	260	290	110
(15,5)	385	435	160
(15,8)	616	660	256
(20,5)	510	580	210
(20,10)	1020	1080	420
(20,15)	1530	1580	630

Table 2. Number of variables, constraints and 0-1 variables for linear formulation of the problem

Problem Class (T,n)	**No. of Variables**	**No. of Constraints**	**No. of 0-1 Variables**
(4,3)	138	220	102
(4,4)	216	384	168
(6,4)	320	574	248
(10,5)	760	1490	610
(15,5)	1135	2235	910
(15,8)	2536	5700	2176
(20,5)	1510	2980	1210
(20,10)	5020	11880	4420
(20,15)	10530	26780	9630

For each problem class 30 problem instances were generated. Demand matrix was generated by randomly choosing d_{i1} from the interval $[30,110]$ and then randomly generating demands d_{it} for time periods t>1 from the interval $[d_{it}-10, d_{it}+50]$ by using uniform distribution. Production costs p_{itj} were randomly generated in a similar manner. First production costs p_{i1j} for the first time period were chosen from the interval $[30,110]$ and then production costs p_{itj} for time periods t>1 were generated from the interval $[p_{itj}-10, p_{itj}+30]$ by using uniform distribution. Holding costs h_{it} were chosen from the interval $[10,20]$, fixed setup costs k_{itj} from the interval $[20,40]$ and machine capacity consumption a_{ij} from the interval $[1,6]$, all by using uniform distribution. Setup times u_{iltj} were generated as follows. First, setup times u_{il1j} for the first time period were randomly chosen from the interval $[5,15]$ by using uniform distribution, and then proportionality factor *r* was chosen from the interval $[1.001,1.15]$. Setup times u_{iltj} for *t>1* were calculated by using the rule $u_{il(t+1)j} = r \cdot u_{iltj}$. Furthermore, $u_{iitj} = 0$. In order to generate machine capacities, average overall demand per single time period D_{avg} was computed by using the formula:

$$D_{avg} = \frac{\sum_{i=1}^{n}\sum_{t=1}^{T} d_{it}}{T} \tag{30}$$

Then machine capacities c_{tj} were determined according to the rule

$$c_{tj} = U_{tj} \cdot a_{\max} \cdot D_{avg} \tag{31}$$

where $a_{\max} = \max_{i,j} a_{ij}$ is maximal time needed for production of any item on any of the machines and U_{tj} are random numbers generated from the interval $[0.4,2]$ by using uniform distribution. Here $a_{\max} \cdot D_{avg}$ is an upper bound on production of average overall demand per time period. Setup times are not included in the process of generating capacities, so machine capacities are determined as a random percentage of that upper bound.

Each problem instance was solved by 7 heuristics (using quadratic formulation of the problem) and by CPLEX 9.0 (using linear MIP formulation of the problem). The heuristics are: three different versions of T-LR heuristics, VNS1, VNS2, VNS3 and H-LR heuristics. Three versions of T-LR heuristics are as follows: T-LR as presented in section 3.2., T-LR1 obtained from T-LR by expanding the neighborhood structure consisting of 4 constructed solutions with 2 randomly generated solutions and T-LR2 obtained from T-LR by expanding the neighborhood structure consisting of 4 constructed solutions with 4 randomly generated solutions. All heuristics were implemented in AMPL programming language [4]. Tests were run on PC Pentium IV with 2.4 GHz processor and 1Gb RAM. The MIP sub-programs appearing in various steps of heuristics were solved by using CPLEX 9.0.

Table 3 displays the size of the problem classes together with average CPU time (in seconds) required by each heuristics and CPLEX to reach termination.

Table 3. Average CPU time (in seconds) to reach termination

Problem Class (T,n)	**T-LR**	**T-LR1**	**T-LR2**	**VNS1**	**VNS2**	**VNS3**	**H-LR**	**CPLEX**
(4,3)	2,84	4,39	5,93	10,52	8,67	3,85	0,19	0,04
(4,4)	3,58	5,49	7,18	13,51	15,92	6,01	0,30	0,05

(6,4)	4,59	8,52	12,14	40,03	28,26	4,44	0,49	0,09
(10,5)	8,14	31,68	39,97	114,39	88,33	13,12	2,59	1,18
(15,5)	9,73	34,92	44,78	123,65	111,16	19,61	1,60	3,02
(20,5)	17,39	56,56	94,39	331,46	230,70	19,80	2,86	7,45
(15,8)	174,05	1041,90	1414,11	2966,98	2288,51	334,70	1633,61	11046,45
(20,10)	661,00	1211,93	1951,12	4049,99	2569,64	478,58	-	86400,00
(20,15)	1513,93	2569,64	3161,59	4316,54	2385,05	654,64	-	86400,00

Computational time for each problem instance was limited to 86400 seconds (24 hours). For problem classes $(20{,}10)$ and $(20{,}15)$ H-LR heuristics was unable to produce any solution within the time limit, as indicated by missing entry. For the same problem classes CPLEX did not reach termination within the time limit, so the process was stopped and the solution obtained within the time limit was used for comparison. As expected, parameters *T* and *n* have significant impact on CPU run-time. By comparing run-times for problem classes (10,5), (15,5) and (15,8), we see that CPU time grows much faster as *n* increases than as *T* increases. For small problem instances (problem classes (4,3), (4,4), (6,4) and (10,5)) CPLEX is superior to any heuristics. For these problem classes, H-LR is the only heuristics with run time close to CPLEX. All other heuristics are much slower. For problem classes (15,5) and (20,5) the best run time is obtained by H-LR. These are the first problem classes for which some heuristics outperforms CPLEX, though CPLEX outperforms heuristics other than H-LR. For problem class (15,8), the best run-time is obtained by T-LR, followed by H-LR. Run times of other heuristics are greater by factor 10, still outperforming CPLEX with run time of approximately 3 hours. For problem classes (20,10) and (20,15) heuristic H-LR was unable to produce any solution within the limit. The best run time was obtained by VNS3 heuristic, followed by T-LR.

Table 4 displays comparison of results in terms of solution quality. The comparison was made with respect to objective function value obtained by CPLEX. Therefore, Table 4 displays average optimality gap, where optimality gap for a single problem instance and a single heuristic is defined as:

$$\text{Gap}(\%) = \frac{(\text{CPLEX objective value}) - (\text{heuristic objective value})}{\text{CPLEX objective value}} \times 100\,\% \qquad (32)$$

The average optimality gap for a certain problem class is calculated by computing the average optimality gap of 30 problem instances generated for this problem class.

Table 4. Average optimality gap of objective function value

Problem Class (T,n)	**T-LR**	**T-LR1**	**T-LR2**	**VNS1**	**VNS2**	**VNS3**	**H-LR**
(4,3)	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
(4,4)	0,34%	0,31%	0,31%	0,31%	0,00%	0,00%	0,00%
(6,4)	3,49%	2,53%	2,38%	-0,25%	-0,03%	0,21%	-0,28%
(10,5)	3,38%	-0,11%	-0,14%	-0,26%	-0,19%	-0,01%	-0,44%
(15,5)	3,67%	0,24%	0,34%	-0,13%	-0,10%	-0,26%	-0,26%
(15,8)	3,55%	0,56%	0,36%	0,07%	0,09%	0,17%	-0,04%

(20,5)	4,32%	1,07%	0,38%	-0,13%	0,01%	0,36%	-0,20%
(20,10)	1,71%	1,13%	1,01%	0,42%	0,60%	0,60%	-
(20,15)	0,41%	0,12%	0,10%	0,16%	0,19%	0,63%	-

For problem class (4,3) all heuristics obtained the same objective function value as CPLEX. For problem class (6,4) and (15,5), in average heuristics VNS1, VNS2, VNS3 and H-LT obtained better results than CPLEX, with H-LR achieving the best result. For problem class (10,5) all heuristics but T-LR outperform CPLEX, the best one being H-LR. For problem class (15,8) heuristic H-LR slightly outperforms CPLEX. However, heuristics other than T-LR also obtain good results, with optimality gap less than 1%. For problem class (20,5) the best results were again obtained by H-LR, which outperforms CPLEX just as VNS1. For problem classes (20,10) and (20,15) H-LR was unable to produce any solution. However, other heuristics have produced results within 2% gap from objective value obtained by CPLEX in 24 hours computational time in significantly less time. For problem class (20,10) the best results were obtained by VNS1 with average gap of 0.42% and only 478,58 CPU seconds of computational time. For problem class (20,15), the best solution quality is obtained by T-LR2 heuristics, but with 3161,59 CPU run time. However, VNS3 finds solution in 654,64 CPU seconds with 0.63% gap.

The results show that the choice of neighborhood structure effects both solution quality and computational time. Regarding solution quality, neighborhood structures containing 4 constructed solutions only might be a bit too restrictive, thus preventing thorough diversification of the search. Better results are obtained for neighborhood structures expanded by 2 or 4 solution obtained by changing random number of coordinates of variable **z**. However, such structures increase run times of heuristics in question, thus creating a tradeoff between solution quality and run time needed to obtain solution.

5 CONCLUSIONS AND FUTURE WORK

This paper proposes three different VNS heuristic procedures for the quadratic formulation of capacitated lot-sizing and scheduling production planning problem with sequence dependent setup times. The first two are based on basic VNS scheme. Hereby the first one uses tabu search heuristics in its local search step, while the second performs the search for the best neighbor only. The third VNS heuristic is based on reduced VNS scheme. Since the search space of these heuristics is different from the problem solution space, all three heuristics use Langrangean relaxation of machine capacity constraints to construct complete solutions. Heuristics were tested and compared to Lagrangean relaxation heuristics and three versions of tabu search heuristic, with each version having a different neighborhood structure in terms of number of solutions it contains. Heuristics were also compared to CPLEX, were CPLEX was used to solve the linear formulation of the problem. The effectiveness of the heuristic procedures proposed was tested on nine generated problem classes, each class having 30 problem instances. Computational results show that for large problem instances computational time required by heuristics to reach solution is significantly less than CPLEX run time. However, there is a tradeoff between the solution quality and computational time needed to obtain solution.

Since the real world industrial processes require fast and efficient solutions for large problem instances, future work should deal with creating VNS solution procedures using some of constructive methods for creating solutions. This might result in further improvement of computational time as well as in improvement of solution quality for large problem instances. Moreover, such a procedure would not require use of sophisticated solvers, thus making it more accessible to implementation in actual production industries.

References

[1] Allahverdi A, Gupta J N D, Aldowaisan T (1999) A review of scheduling research involving setup considerations. OMEGA The International Journal of Management Science 27: 217-239

[2] Allahverdi A, Ng C T, Cheng T C E, Kovalyov M Y (2008) A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187: 985-1032

[3] Florian M, Lenstra J K, Rinnooy Kan G (1980) Deterministic production planning: algorithms and complexity. Management Science 26: 669-679

[4] Fourer R., Gay D M, Kernighan B W (1990) A Modeling Language for Mathematical Programming. Management Science 36: 519-554

[5] Glover F (1989) Tabu Search – Part I. ORSA Journal of Computing 1: 190-206

[6] Glover F (1990) Tabu Search – Part II. ORSA Journal of Computing 2: 4-32

[7] Hansen P, Mladenović N (2001) Variable neighborhood search: Principles and applications. European Journal of Operations Research 130: 449-467

[8] Lukač Z, Šorić K, Vojvodić-Rosenzweig V (2005) One Heuristics for Production Planning Problem with Sequence Dependent Setups, The International Conference on Industrial Engineering and Systems Management IESM 2005.