POLITECNICO DI TORINO
Repository ISTITUZIONALE

Matheuristics for the lot sizing problem with back-ordering, setup carry-overs, and non-identical machines

Original
Matheuristics for the lot sizing problem with back-ordering, setup carry-overs, and non-identical machines / Ghirardi, Marco; Amerio, Andrea. - In: COMPUTERS & INDUSTRIAL ENGINEERING. - ISSN 0360-8352. - 127:(2019), pp. 822-831. [10.1016/j.cie.2018.11.023]

Availability:
This version is available at: 11583/2717647 since: 2020-04-24T10:57:32Z

Publisher:
Elsevier

Published
DOI:10.1016/j.cie.2018.11.023

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2019. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.cie.2018.11.023

(Article begins on next page)

20 April 2024

Matheuristics for the Lot Sizing Problem with Back-Ordering, Setup Carry-Overs, and Non-Identical Machines

Marco Ghirardi[1], Andrea Amerio

DIGEP - Politecnico di Torino, Corso Duca degli Abruzzi 24, 10128 Torino, Italy

Abstract

In this study, the *capacitated lot-sizing problem* (CLSP) with back-ordering, setup carry-overs between periods, and non-identical parallel machines (CLSP-BOPM) is considered. The problem is one of the most general extensions of the well-known economic lot scheduling problems (ELSPs). Three matheuristics are designed and implemented starting from the ideas of variable neighborhood local search, local branching, and feasibility pump (FP), adapted and improved by considering the specific characteristics of the problem. Algorithms are tested on a set of medium to large problem instances. The FP algorithm outperforms all other algorithms and two different mixed-integer programming solvers as it requires a shorter computational time. To test the robustness of the algorithm, tests on three particular cases of the general problem, belonging to the family of discrete ELSPs, have been performed. Results from the proposed solver and a known specific state-of-the-art algorithm demonstrate substantial improvements.

Keywords: Scheduling, Matheuristics, Lot sizing, Back-orders, Setup carry-over

1. Introduction and literature review

Many production systems are single-stage processes in a manufacturing facility comprising many production lines or machines. Typically, some of these lines are focused on the production of single high-demand items whereas the remaining lines are dedicated to flexible production where sets of medium- or low-demand items are produced. Each line is characterized by different setup times, limited capacities in each period (finite production rates), and processing or setup costs (some of these vary for each item). Moreover, other item-related costs such as inventory costs or back-ordering costs must be considered. To achieve an efficient solution, good "batching and sequencing" evaluation of the

[1]Corresponding author e-mail: marco.ghirardi@polito.it

production of items on different lines is the core of this type of problem. Typically, in a flexible system, many different types of jobs are considered; each type has different processing parameters and jobs of the same type have identical characteristics. If setup times and setup costs are not important, the problem leads often to an alternated schedule solution that is more efficient than a schedule with long runs of jobs of the same type. However, usually, setup costs and setup times must be considered owing to their high economic relevance. Such a type of production systems are common in several process industries, including the automotive, chemical, cosmetics, electronics, food, packaging, paper, pharmaceutical, printing, textile, wood-processing and semiconductor industries.

The problem related to the production of items on a single line, subject to constant demand rate, is known in the literature as the *economic lot scheduling problem* (ELSP). The traditional ELSP is characterized by a continuous time formulation. It has been widely studied and it is well known to be an NP-hard problem. Later, the *capacitated lot-sizing problem* (CLSP) has been considered as an extension to the ELSP.

- A discrete demand volume for each item, over a predefined number of periods, is given; the machines have a finite capacity (in time units) for each period.
- Switching the production between different items introduces setup cost and setup time is required to set up the machine for production: the latter consumes machine time.
- If an item is produced before its demand period, inventory cost is incurred.

The objective is the minimization of the overall cost (setup and inventory). In the same way as the ELSP, the CLSP is also NP-hard. It has been proven that this is true even when setup times are not considered, whereas if setup times are included, the problem becomes NP-complete (Bitran and Yanasse, 1982). Formulations of the standard CLSP can be found in Billington et al. (1983) or Trigeiro et al. (1989) while an extensive overview of the lot-sizing literature can be found in Pinedo (2009) and Drexl and Kimms (1997).

The problem faced in this study is the CLSP with back-ordering, setup carry-overs, and non-identical parallel machines (CLSP-BOPM). *Back-order* identifies the introduction of a back-ordering policy: the possibility to postpone the production of one or more items in a different period than its demand in order. Postponing the production introduces new costs, the back-ordering costs. These costs are also item-related because each item can have its own priority. *Linked lot-sizes* or *setup carry-over* is the capability to carry-over a setup state through consecutive periods. When, on any machine, the last items produced in a previous period and the first items produced in the following period are of the same type, the setup can be avoided. Hence, the setup carry-over allows to save machine time and cost. Providing *parallel machines* in a single stage means the possibility to sort and route the production of each item between a larger number of machines.

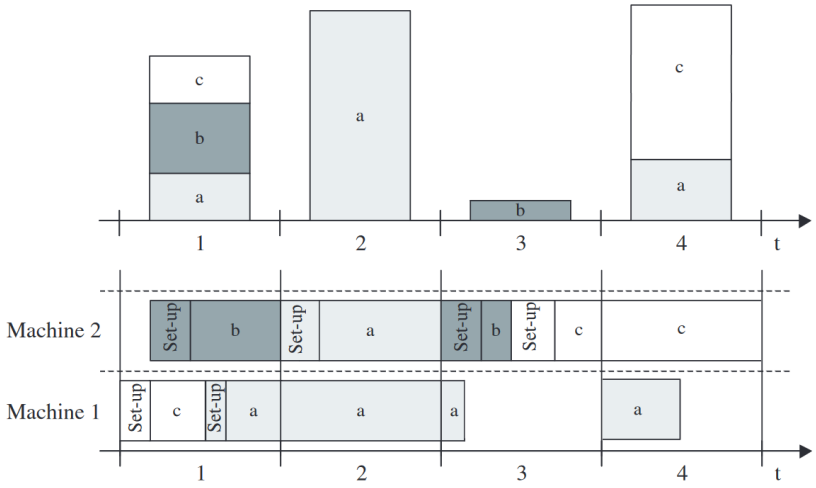

Figure 1: Instance and solution example for a CLSP-BOPM instance

A toy example of an instance and a possible solution of the CLSP-BOPM problem with 3 product types (a,b,c), 2 machines and 4 time periods is given in Figure 1. The figure above represents the production volumes of the three products requested at the end of the corresponding period. The figure below represent a possible solution for the instance. In the first period, all the requested production is manufactured in the period itself (a and c on Machine 1, b on machine 2, with their corresponding set-up times). In the second period, product a is requested in a very large quantity. The solution uses both machines to satisfy the requirement: machine 1 does not need a setup time (setup carry-over from the previous period), while machine 2 requests a setup. The available production time is not enough to complete the requested quantity of product a, hence a small part of the request is back-ordered to the third period (continuing with a setup-carry over on Machine 1). In the third period, a small quantity of product b is allocated to machine 2 which is then setup in advance for the large quantity of product c requested in the forth period. This results in the use of the inventory for product c between the third and the forth period. At last, in the forth period, production of the requested quantity of c is completed, and a smaller quantity of a is allocated to machine 2. Both productions are performed without a setup time (carry over from the previous periods).

CLSPL-BOPM model formulations studied in the literature consider, in general, single machines (Goren et al. (2014), Hindi (1995b), Fiorotto et al. (2017), Ceschia et al. (2017), Ghaniabadi and Mazinani (2017)) or identical parallel machines. CLSP on parallel machines with setup carry-over but without back-ordering is faced in Dillenberger et al. (1993), and extended in Dillenberger et al. (1994). The authors solve problems with up to 40 products, 6 time periods and 15 machines. Gopalakrishnan et al. (1995) introduce a new mixed integer linear programming model for CLSP with parallel machines, setup carry over and

without back-ordering. Setup times and costs are included but are the same for all products. Diaby et al. (1992) introduce a Lagrangean relaxation scheme for a particular CLSP formulation with a single setup time for each time period, no matter how many resources are being used. Their model includes parallel machines but does not include carry-overs or back-orders. Hindi (1995a) considers a problem with multiple machines, but without setups and back-orders, reformulating the model as a transshipment capacitated model. Öznamar and Birbil (1998) propose heuristics for a particular parallel machine model with setup times and overtime costs. No back-ordering and setup carry-overs are allowed. The same model is then extended in Öznamar and Barbarosoglu (1999) to a multiple production stage case. Kang et al. (1999) tackle a particular problem based on real-life cases, in which setups are allowed to be carried over to a subsequent period, but no setup times and back-orders are included. Belvaux and Wolsey (2000) provide a modeling framework for a wide set of lot-sizing problems, including cases with parallel machines and back-orders. Setup carry-overs are not handled. The resulting models are solved by XPress-MP solver. Karimi et al. (2006) propose a tabu search metaheuristic for the CLSP problem with back-orders. Results near to the optimal solution are found for small test instances only, up to 4 products, and 6 periods. Quadt and Kuhn (2005) and Quadt and Kuhn (2009) formulate the CLSP-BOPM as presented in this paper for a multi-stage or single-stage production setting. Moreover, they present an alternative reformulation for the case the machines are identical, using integer variables instead of boolean ones, obtaining substantial improvements. A non-identical parallel machine set leads to more interesting practical situations because each machine has its own unique set of characteristics (for instance, it often happens that in a production system newer and older machines, with different speeds, are both available). Processing time, setup time, and setup cost for all items are machine-related in the problem tackled in this work.

For an extensive literature review of capacitated lot-sizing with extensions, refer to Quadt and Khun (2009), Quadt and Khun (2008), and references therein. More recent research has been dedicated to different extensions of the problem: with setup crossover (Belo-Filho et al., 2014), sequence-dependent setup times (Menenez et al., 2010), multi-stage production (Caserta et al., 2010), machines degradation (Xia et al., 2015), stochastic demand (Zhengyang and Hu, 2018), mainteinance (Xia et al., 2018) or deterioration issues (Pahl et al., 2011). There are no recent results on algorithms for large instances of the general CLSP-BOPM tackled in this study.

This paper is structured as follows. Section 2 introduces the CLSP-BOPM mixed integer linear programming model and section 3 presents the implemented mathematical methods. Section 4 presents three discrete ELSP problems, particular cases for the CLSP-BOPM. Finally, section 5 presents the computational campaign results, and section 6 concludes the paper.

2. Model Formulation

Given a product portfolio P, a set of non-identical parallel machines M, and a planning horizon T, the problem is to find a feasible assignment of the entire production to all the machines, for all the production periods. All demands must be met before the end of the planning horizon. Anticipate or postponing production (increasing inventory or back-order volumes) is allowed, therefore it is not mandatory to meet demands at the end of the connected periods. The objective is to minimize the overall costs, including inventory, back-order, and setup costs. It is possible to carry setup states of a machine over consecutive periods, saving setup costs. In the following, the CLSPL-BOPM model formulation is presented. This model is a slight extension of the one presented in Quadt and Kuhn (2009) for the particular case with identical machines setup costs, setup times and processing times.

Model parameters:

c^i_p	Inventory holding costs of product p per time period
c^s_{pm}	Setup costs of product p on machine m
c^b_p	Back-order costs of product p per time period
t^s_{pm}	Setup time of product p on machine m
t^u_{pm}	Processing time (per unit) of product p on machine m
C_{tm}	Capacity (available time) of machine m in period t
d_{pt}	Demand volume of product p in period t
P	Number of products $\overline{P} = \{1 \dots P\}$
M	Number of machines $\overline{M} = \{1 \dots M\}$
T	Number of periods $\overline{T} = \{1 \dots T\}$
b^0_p	Initial back-order volume of product p
y^0_p	Initial inventory volume of product p
ζ^0_{pm}	Initial setup state of product p on machine m
z	Big number, $z \geq \sum_{p \in \overline{P}, t \in \overline{T}} d_{pt}$

Model variables:

b_{pt}	Back-order volume of product p at the end of period t (continuous)
x_{ptm}	Production volume of product p on machine m in period t (continuous)
y_{pt}	Inventory volume of product p at the end of period t (continuous)
γ_{ptm}	Binary setup variable for product p on machine m in period t ($\gamma_{ptm} = 1$ if, in period t, a setup is performed for product p on machine m)
ζ_{ptm}	Binary linking variable for product p on machine m in period t ($\zeta_{ptm} = 1$ if the setup state for product p on machine m is carried over from period t to $t+1$)

Model CLSPL-BOPM:

$$\min \sum_{p \in \overline{P}, t \in \overline{T}} (c^i_p y_{pt} + c^b_p b_{pt} + \sum_{m \in \overline{M}} c^s_{pm} \gamma_{ptm}) \tag{1}$$

subject to:

$$y_{p,t-1} - b_{p,t-1} + \sum_{m \in \overline{M}} x_{ptm} - d_{pt} = y_{pt} - b_{pt} \qquad \forall\, p \in \overline{P},\, t \in \overline{T} \tag{2}$$

$$\sum_{p \in \overline{P}} (t^u_{pm} x_{ptm} + t^s_{pm} \gamma_{ptm}) \leq C_{tm} \qquad \forall\, t \in \overline{T},\, m \in \overline{M} \tag{3}$$

$$x_{ptm} \leq z(\gamma_{ptm} + \zeta_{p,t-1,m}) \qquad \forall\, p \in \overline{P},\, t \in \overline{T},\, m \in \overline{M} \tag{4}$$

$$\sum_{p \in \overline{P}} \zeta_{ptm} = 1 \qquad \forall\, t \in \overline{T},\, m \in \overline{M} \tag{5}$$

$$\zeta_{ptm} - \gamma_{ptm} - \zeta_{p,t-1,m} \leq 0 \qquad \forall\, p \in \overline{P},\, t \in \overline{T},\, m \in \overline{M} \tag{6}$$

$$\zeta_{ptm} + \zeta_{p,t-1,m} - \gamma_{ptm} + \gamma_{qtm} \leq 2 \qquad \forall\, p,\, q \in \overline{P},\, p \neq q,\; t \in \overline{T},\, m \in \overline{M} \tag{7}$$

$$b_{pT} = 0 \qquad \forall p \in \overline{P} \tag{8}$$

$$b_{p0} = b^0_p, \quad y_{p0} = y^0_p \qquad \forall\, p \in \overline{P} \tag{9}$$

$$\zeta_{p0m} = \zeta^0_{pm} \qquad \forall\, p \in \overline{P},\, m \in \overline{M} \tag{10}$$

$$x_{ptm} \geq 0 \qquad \forall\, p \in \overline{P},\, t \in \overline{T},\, m \in \overline{M} \tag{11}$$

$$y_{pt} \geq 0, \quad b_{pt} \geq 0 \qquad \forall\, p \in \overline{P},\, t \in \overline{T} \tag{12}$$

$$\gamma_{ptm} \in \{0,1\}, \quad \zeta_{ptm} \in \{0,1\} \qquad \forall\, p \in \overline{P},\, t \in \overline{T},\, m \in \overline{M} \tag{13}$$

The objective function (1) to be minimized is the sum of the inventory holding, setup and back-ordering costs. Constraints (2) are ordinary inventory flow conditions, and ensure that demand is met at every period, through inventory, production, or back-orders. Equations (3) model the machine capacities: the time dedicated to processing and setup have to be no longer than the available time of the machines. Equations (4) state that production can only be performed on a machine with a setup performed. This is achieved by either carrying over a setup state from period $t-1$ ($\zeta_{p,t-1,m} = 1$) or performing a setup in period t ($\gamma_{ptm} = 1$). Equations (5) impose that the setup state can be carried over for only one item. Constraints (6) state that if a machine m carries over a setup state for product p from period t to period $t+1$ ($\zeta_{ptm} = 1$), the machine must have a setup for product p in period t ($\gamma_{ptm} = 1$) or the setup state had been carried over from period $t-1$ ($\zeta_{p,t-1,m} = 1$). Constraints (7) state that if machine m carries over a setup state for item p from period $t-1$ to t and from t to $t+1$ ($\zeta_{ptm} = \zeta_{p,t-1,m} = 1$) and, in period t a setup is performed for another product q ($\gamma_{qtm} = 1$), then the machine setup to p must be reset up in period t ($\gamma_{ptm} = 1$).

Equations (8) ensure that the whole demand volume is produced before the end of the planning periods. Equations (9) and (10) define the initial states. Equations (11) and (12) define the production/inventory/backorder decision variables (as common in the lot sizing literature, these variables are supposed to be continuous), and (13) the boolean decision variables.

3. Matheuristic Approaches

Recently, meta-heuristics and exact methods have been hybridized, leading to a branch of algorithms called *matheuristics* (see, for instance, Ball (2011), Della Croce et al. (2013), and Boschetti et al. (2009)), exploiting the combination of mathematical programming and heuristics approaches to solve combinatorial optimization problems, usually embedding a general *mixed-integer linear programming* (MILP) solver as a subroutine of a heuristic algorithm, with the task of solving subsets of the general problem. Two improvement procedures based on variable partitioning local search (VPLS) and local branching are presented in section 3.1 and a constructive feasibility-pump-based algorithm FP is introduced in section 3.2.

3.1. Improvement Procedures

3.1.1. Variable Partitioning Local Search

VPLS was developed from the classical *local search* algorithms. According to Della Croce et al. (2013), VPLS is an iterative procedure that runs, at each iteration, the mixed-integer programming (MIP) solver to perform a local search inside a variable neighborhood of a current solution. The subproblem is a permutation problem where a certain number of binary variables can be commuted to obtain an improving solution with respect to the current one. In particular, referring to a general linear programming with binary variables of a solution X, define S as a subset of a defined size of variable indices $\{1, 2, \ldots, n\}$. All the solutions belonging to a neighborhood $N(\overline{X})$ are those that maintain all the jth binary variables equal to the jth in X, for all $j \notin S$. Furthermore, to add differentiation to the search, the algorithm is randomized: at each iteration, variables involved are chosen randomly. The algorithm described here is actually inspired both by the general VPLS formulation and by the matheuristic method in Bollapragada et al. (2011) applied to a simpler discrete time ELSP.

The pseudocode of the algorithm is presented in algorithm blocks 1. The first part of the algorithm is dedicated to the choices of the neighborhood. The *FREEITEMS* and *FREEMACHINES* parameters represent the number of items and machines that can be changed in the subproblem. After preliminary tests, to speed up the algorithm, we chose to dynamically reduce the neighborhood size with respect to the running iteration. *FREEITEMS* is always set to value $\frac{P}{4}$, whereas *FREEMACHINES* is set to $\frac{P*M*T}{8}$ first, and then reduced to $\frac{P*M*T}{12}$ and finally to $(\frac{P*T}{2})$. The indexes to fix are selected randomly by the algorithm and stored in the arrays *i_fix* and *j_fix*, marking the items/machines pairs (i, j). The second part of the algorithm fixes the real variables. The variables γ and ζ identified by the pairs (i, j) are set free for all the periods $t \in \overline{T}$. All other variables are blocked to their values assumed in the current solution. When a setup binary variable $\gamma(i, j, t)$ is set free (or blocked), the corresponding carry-over binary variable $\zeta(i, j, t)$ is also freed (or blocked) to avoid unfeasible neighbors. Finally, in the last part of the algorithm, a call to the MIP solver is performed. Then, if the current best solution is updated, the iteration counter

Algorithm 1 VPLS pseudocode

Require: Starting_Solution
Begin(VPLS)
Current_Solution = Starting_Solution
while iteration $\leq$ MAX_iterations **do**
    if iteration $\leq$ MAX_iterations/3 **then**
        FREEITEMS = $P/4$; FREEMACHINES = $PMT/8$
    else if MAX_iterations/3 $<$ iteration $\leq$ 2MAX_iterations/3 **then**
        FREEITEMS = $P/4$; FREEMACHINES = $PMT/12$
    else
        FREEITEMS = $P/4$; FREEMACHINES = $PT/2$
    end if
    Set i_fix(i) = 1 for FREEITEMS randomly selected items i , 0 otherwise.
    Set j_fix(j) = 1 for FREEMACHINES randomly selected machines j, 0 otherwise.
    for all the items $i \in \overline{P}$, the machines $j \in \overline{M}$, the periods $t \in \overline{T}$ **do**
        if i_fix(i) = 1 and j_fix(j) = 1 **then**
            Free the corresponding $\gamma(i,j,t)$ and $\zeta(i,j,t)$ variable
        else
            Set $\gamma(i,j,t)$ and $\zeta(i,j,t)$ value as in Current_Solution
        end if
    end for
    Current_Solution = **Solve**(MILP)
    if totalcost(Current_Solution) $\leq$ totalcost(Best_Solution) **then**
        Best_Solution = Current_Solution
        iteration = 0 ▷ Restart counting iterations
    end if
    if Current_Time $\geq$ Time_Limit **then**
        iteration = MAX_iterations ▷ Jump to the last iteration to abort the procedure
    else
        iteration ++
    end if
end while
End(VPLS)

is set newly to 0 to restart the iteration count. If the time limit is reached, the procedure ends.

3.1.2. Improved Local Branching Technique

Local branching is another well-known improvement matheuristic procedure (Fischetti et al., 2004). It consists of an iterative local search, where each iteration is composed of three phases (*DRT* structure): *diversification*, *refining*, and *tight refining*. The binary set B is organized to identify the subsets B_1 and B_2, corresponding to the first-level and second-level variables, respectively. Beginning from an *initial solution* $\overline{X}$, the DRT structure is iterated until a time limit is reached.

1. *Diversification*: Dynamically adding a constraint that forces a move away from the current solution for a (Hamming) distance between k_D^{min} and a maximum k_D^{max}:
$$k_D^{min} \leq \Delta_1(X, \overline{X}) \leq k_D^{max} \tag{14}$$
The subscript 1 means that the constraint is set on the first-level variables set only. The exclusion of the current solution from the new search space is done by adding a static tabu constraint that hides this solution for the entire iteration cycle:
$$\Delta_1(X, \overline{X}) \geq 1 \tag{15}$$
The (k_D^{min}, k_D^{max}) range can be modified on the run.

2. *Refining*: The solution $\overline{X}$ is refined by removing the dynamic *diversification constraint* (14) and by adding the new dynamic *refining constraint*:
$$\Delta_1(X, \overline{X}) \leq k_R \tag{16}$$
If the local optimum solution of this new problem is not reached before the time limit, a tight refining step will be performed.

3. *Tight refining*: The dynamic refining constraint (16) on the first-level variable is maintained. An additional dynamic constraint (17) is added, affecting only second-level variables
$$\Delta_2(X, \overline{X}) = k_{TR} \tag{17}$$
Re-optimization is then performed for different values of k_{TR}.

Parameters k_D^{min}, k_D^{max}, k_R, k_{TR}, and k_D^{step} must be chosen reasonably small to allow a fast solution search by the MIP solver.

In our case, the setup carry-over variables ζ have a greater influence over the setup variables γ rather than the contrary. We have chosen to consider ζ as the first-level variables set and γ as the second-level variables set. To summarize the DRT procedure, diversification and refine steps add constraints on the ζ variables, whereas the tight-refine step works on γ variables. The algorithm pseudocode 2 shows the main scheme of the DRT procedure structure. The exit

Algorithm 2 Local branching pseudocode

```
Require: Starting_Solution
Begin(LocalBranching)
Current_Solution = Starting_Solution
no_improv_counter = 0
while Current_Time ≤ Time_Limit do
    Add the TABU constraint (15) with respect to Current_Solution
    if no_improv_counter ≥ no_improv_threshold then
        no_improv_counter = 0                    ▷ Erase the no-improvements counter
        DIV(Current_Solution,BIG)                ▷ Do a BIG Diversification
    else
        DIV(Current_Solution,Normal)             ▷ Do a Normal Diversification
    end if
    REF(Current_Solution)                        ▷ Refine the new Current_Solution
    if the refinement step reached its time limit then
        TREF(Current_Solution)                   ▷ Do a Tight Refining
    end if
    if totalcost(Current_Solution) ≤ totalcost(Best_Solution) then
        Best_Solution = Current_Solution
        no_improv_counter = 0
    else
        no_improv_counter ++                     ▷ Count iterations without improvements
    end if
    Remove the Refining, Tight Refining and Tabu constraints (16), (17), and (15)
end while
End(LocalBranching)
```

condition for the loop is a time limit *total-time limit* on the execution time. A tabu constraint (15) to hide the entering current solution is added, valid for the entire current DRT cycle. The diversification step is performed by calling the *DIV()* function. When *no_improv_counter* consecutive iterations performing a normal diversification do not produce improvements, a BIG diversification is performed. The current solution is then refined. If the solver has not been able to find the optimal solution for the refining (REF) subproblem, the procedure proceeds into the tight refining (TREF) step. Then, an update for the best solution found is performed. If the current solution improved the, until now, best solution found, the no-improvement counter is reset; otherwise, it is increased. Before the next iteration, all the REF (16), TREF (17), and tabu (15) constraints are removed.

Diversification. The *DIV()* algorithm (3) receives as input the current solution and the normal/BIG parameter that regulates the size of the diversification subproblem. Here k_D^{min}/k_D^{max} represent the minimum/maximum number of variables that must be changed during the re-optimization of the problem (the minimum/maximum distance between the current solution and the new one). If a new solution is found, it is returned to the main procedure. In each iteration, the DIV search space is augmented by the k_D^{step} parameter and bounds k_D^{max} and k_D^min are updated. The diversification procedure ends if a time limit has been reached, a limit on the maximum iterations (parameter $k_D^{MAX}\_limit$) has been reached, or an improved solution has been found.

Already during the preliminary computational testing, the classical diversification step has been proven to be slow, even with a small k_D^{step}, mainly due to the fact that the diversification subproblems remain too large and the optimal solutions are hardly reached by the solver. Hence, the classical *DIV()* method has been hybridized with the VPLS-based procedure given in the previous section, obtaining substantial speed improvements. Two fixing parameters (*FREEITEMS*, *FREEMACHINES*) and two index arrays (i_fix, j_fix) are introduced. For all the pairs, if (i, j) indexes are marked as free over the entire periods set, all the corresponding first-level (ζ) and second-level (γ) variables are let free. All the other variables are blocked to their values with reference to the current solution.

Algorithm 3 Local branching: diversification

Begin(DIV(Current_Solution, Type))
Entering_Time = Current_Time
if Type = BIG **then**
    k_D^{step} = BIG_step
else
    k_D^{step} = Normal_step
end if
while (Current_Time $\leq$ Entering_Time + DIVTime_Limit) and ($k_D^{MAX} \leq k_D^{MAX}\_limit$) and (a solution has not yet been found) **do**
    $k_D^{MAX} = k_D^{min} + k_D^{step}$
    Add the Diversification constraint (14) with respect to the Current_Solution
    Set i_fix(i) = 1 for FREEITEMS randomly selected items i, 0 otherwise.
    Set j_fix(i) = 1 for FREEMACHINES randomly selected machines j, 0 otherwise.
    for all the items $i \in \overline{P}$, the machines $j \in \overline{M}$, the periods $t \in \overline{T}$ **do**
        if i_fix(i) = 1 and j_fix(j) = 1 **then**
            Free the corresponding $\gamma(i,j,t)$ and $\gamma(i,j,t)$ variables
        else
            Set $\gamma(i,j,t)$ and $\zeta(i,j,t)$ as in Current_Solution
        end if
    end for
    Current_Solution = **Solve**(MILP) ▷ Run the solver
    Remove the Diversification constraint (14)
    $k_D^{min} = k_D^{min} + k_D^{step}$
end while
if totalcost(Current_Solution) $\leq$ totalcost(Best_Solution) **then**
    Best_Solution = Current_Solution
end if
Return(Current_Solution)
End(DIV)

Refining and Tight Refining. The REF algorithm is described by the algorithm pseudocode 4. First-level variables are now interested by the refining constraint (16) with respect to the current solution but always remembering the tabu constraint (15) referring to the original current solution of the main iteration cycle. This constraint is dynamically added, but it is also maintained for the eventually performed tight refining step. The TREF step (algorithm 5) is composed by a main loop. During each iteration, the size of the corresponding

Algorithm 4 Local branching: refining procedure, pseudocode

```
1: Begin(REF)
2: Add the Refining constraint (16) with respect to the Current_Solution
3: Current_Solution = Solve(MILP)
4: Return(Current_Solution)
5: End(REF)
```

Algorithm 5 Local branching: tight refining procedure, pseudocode

```
1: Begin(TREF)
2: Entering_Time = Current_Time
3: k_TR = 0                                          ▷ Set the initial Hamming Distance
4: while (k_TR ≤ k_TR_limit) and (Current_Time ≤ Entering_Time + REFtime_Limit) do
5:     Add the Tight Refining constraint (17) with respect to the Current_Solution
6:     New_Solution = Solve(MILP)
7:     if totalcost(New_Solution) ≤ totalcost(Current_Solution) then
8:         Current_Solution = New_Solution
9:         k_TR = 0
10:    else
11:        k_TR = k_TR + k_TR^step                   ▷ Increase the non-improvements counter
12:    end if
13: end while
14: Return(Current_Solution)
15: End(TREF)
```

subproblem is expanded by a step parameter k_{TR}^{step}. The TREF constraint (17) is added to the model with respect to the second-level variables and the current solution. The current solution is updated every time the new solution provided by the MIP solver improves it. In that case, the TREF constraint must be adapted to that solution for the next iteration. At each improvement, k_{TR} is also reset; otherwise, it is augmented by adding k_{TR}^{step}. The main loop is time limited by a k_{TR}^{max} threshold.

3.2. The Proposed FP Algorithm

The *feasibility pump* (Fischetti et al., 2005) aims to find a feasible integer solution starting from the optimal (or almost-optimal) solution for the LP-relaxation of the problem. The procedure is structured in an iterative way.

(I) Given a continuous solution obtained by relaxing the binary variables γ and ζ, a nearest-integer rounding is performed. The obtained integer solution in general is not feasible for the MIP problem.

(II) Search for a new relaxed optimum close to the rounded solution.

The two steps are repeated until a feasible solution for the MIP problem is found. This happens when the rounded solution in (I) and the LP-relaxation optimum generated by (II) match.

In our case, good solutions have a 0-structure that is largely in common with the starting continuous solution. Moreover, all the ζ variables should always

be set free to avoid unwanted infeasibility situations. Re-optimization is then performed on the entire ζ set and on a certain number of selected γ variables.

The matheuristic procedure is structured as follows.

1. Obtain the continuous solution from the LP-relaxation of the problem. Assume it is the current solution (MIP-infeasible).
2. Select the first pair of periods $(1, 2)$.
3. - With respect to the current solution: fix the γ variables assuming 0 value; set free all the non-zero γ variables and all the ζ variables.
   - Set free all the γ variables belonging to the selected pair of periods.
   - Re-optimize the MIP subproblem and update the current solution.
   - Select the next pair of periods and repeat until the iterations limit is reached.

Hence, re-optimizations are performed inside a "freedom region" that reflects the non-zero region of the continuous solution, plus the iteration-related additional γ variables.

Table 1: FP free variables example

Machines	1				2				M			
Items	1	2	3	P	1	2	3	P	1	2	3	P
Period 0	0	0	0	0	0	0	0	0	0	0	0	0
Period 1	x	x	x	x	x	x	x	x	x	x	x	x
Period 2	x	x	x	x	x	x	x	x	x	x	x	x
Period 3	0	0	x	x	0	x	0	0	x	0	x	0
Period 4	x	0	x	0	0	0	x	0	0	0	0	x
Period T	0	x	0	0	0	x	0	x	0	x	0	0

An example of the re-optimization problem mask at the first iteration is represented in Table 1. Period 0 is related to the initial states. All the iteration freed variables are marked by an "x" character. It can be seen as variables related to the first pair of periods ($t = 1, 2$) are all freed.

At each iteration, the pair of "freeing-periods" moves. During the first cycle (a complete scan of the periods set), the advancing step is set to be 1 period. When the first cycle is finished, a new cycle is run, with the step augmented to *FREEPERIODS*. Extensive preliminary tests showed that it was unnecessary to continue after the second cycle.

Algorithm block 6 presents the detailed pseudocode of the algorithm. The first part of the algorithm solves the LP-relaxed problem. The next steps are inside a loop, in which exiting conditions are iterations and time limits together with a maximum cycle limit. To begin iterations, some parameters have been defined. The $FREEPERIODS = 2$ parameter gives the number of periods in which the rest of the procedure will work. Maximum iterations are set through preliminary tests to one-third of the planning horizon ($\frac{T}{3}$). In the main loop,

Algorithm 6 FP matheuristic

Begin(FP)
Current_Solution = **Solve**(relaxed LP)
if Current_Solution is an integer feasible solution **then**
    Return(Current_Solution)
end if
cycle = 0; FREEPERIODS = 2; MAX_Iterations = $T/3$; t_t = 1
while (iteration $\leq$ MAX_Iterations) and (Current_Time $\leq$ Time_Limit) and (cycle < 2) **do**
    for all Items $i \in \overline{P}$, Machines $j \in \overline{M}$, Periods $t \in \overline{T}$ **do**
        if $\gamma(i,j,t) = 0$ **then**
            Set constraint $\gamma(i,j,t) = 0$
        else
            Free the $\gamma(i,j,t)$ variable
        end if
        Add constraints $\gamma(i,j,t) \in \{0,1\}$ and $\zeta(i,j,t) \in \{0,1\}$
    end for
    Define FP_Range as [t_t, $t\_t + FREEPERIODS$)
    for all Periods $t \in$ FP_Range **do**
        for all Items $i \in \overline{P}$, Machines $j \in \overline{M}$ **do**
            Free the $\gamma(i,j,t)$ variable
        end for
    end for
    Free all the ζ variables
    Current_Solution = **Solve**(MILP)
    if totalcost(Current_Solution) $\leq$ totalcost(Best_Solution) **then**
        Best_Solution = Current_Solution
        if (cycle = 1) **then** MAX_Iterations ++
        end if
    end if
    if in the next iteration FP_Range exceeds T **then**
        t_t = 1; Cycle++ ▷ New Cycle
    else
        if cycle = 0 **then** t_t++ ▷ Small step
        else if cycle = 1 **then** t_t = t_t + FREEPERIODS ▷ Large step
        end if
    end if
    iteration ++
end while
End(FP)

every setup relaxed variable γ, which in the current solution assumes a value equal to 0, is constrained to this value. All the other γ variables, together with the entire ζ variables set, are maintained free. Moreover, all the γ and ζ variables are now constrained to binary values. The new MIP subproblem constraints define a sort of "LP-relaxation mask." Then, re-optimization is defined by all the already-free variables ζ and γ (hence, those γ variables outside the zeros grid); in addition, all the γ variables belonging to periods from t_t and $t\_t + FREEPERIODS - 1$. In the first iterations, the selected t_t period is the first one in the periods set. This means that at the first while-loop iteration, γ variables corresponding to periods from 1 and 2 are set free and added to all the other already-free variables. Iteration advancing moves these period pairs inside the set of planning periods.

Tests have demonstrated that the best solution is almost always found in one-third of the first cycle. Hence, the iteration limit has been set to $\frac{T}{3}$. However, it has been proven that this can be improved by applying the following rule: every time an improvement is found, the iterations limit is increased by one. On average, the algorithm will run a little bit more than the original $\frac{T}{3}$ iterations but, in general, will stop before the end of the first cycle. This reflects the fact that FP finds a feasible solution in a very limited computational time.

As stated previously, several tests have been performed before defining the presented algorithm structure. In particular, this approach has also been tested when defining some *FREEITEMS* and *FREEMACHINES* parameters as triplet-based ($i \in \overline{P}$, $j \in \overline{M}$, $t \in \overline{T}$) variables freeing as in the VPLS approach. However, the best results have been obtained with the presented algorithm scheme. However, the hybridization idea has been successfully used for the particular case problems described in section 4.

4. Discrete ELSP

In Bollapragada et al. (2011) a particular case of the discrete ELSP was dealt with that has many points in common with the CLSPL-BOPM. Three models (A,B,C) were presented, which differ from CLSPL-BOPM basically by the addition of production costs contributions, the avoidance of setup carry-over, and the possibility of late delivery or lost sales penalties. Refer to the original paper for the complete models and the recovering beam search (RBS) algorithm presented to solve them. Here, only the main differences with the CLSPL-BOPM are highlighted. FP has been tested with them to evaluate the effect of matheuristic structure behavior on some problem characteristics and to compare our algorithm results with the state-of-the-art algorithm for those specific subcases. In the following and in Table 2, the differences between CLSP-BOPM and the three models are presented and computational results are presented in section 5.5.

Model A. With respect to the CLSPL-BOPM model (2), model A differs in the following ways.

Table 2: Differences between CLSP-BOPM and ELSP in the three proposed versions

	CLSP-BOPM	ELSP-A	ESLP-B	ELSP-C
Non-identical parallel machines	✓	✓	✓	✓
Unitary production costs		✓	✓	✓
Setup carry-overs	✓			
Lost sales			✓	
Back-orders	✓			✓

- Setup carry-over has been forbidden. Constraint (5) becomes

$$\sum_{p \in \overline{P}} \zeta_{ptm} = \mathbf{0} \qquad \forall\, t \in \overline{T},\, m \in \overline{M} \tag{18}$$

- A production costs parameter has been added to the model and the production contribution has been added to the objective function 19

  c^s_{pm} Production costs of product p on machine m

- Back-orders have been removed.

Hence, the objective function (1) has been modified as

$$\min \sum_{p \in \overline{P}, t \in \overline{T}} (c^i_p y_{pt} + \sum_{m \in \overline{M}} (c^s_{pm} \gamma_{ptm} + c^u_{pm} x_{ptm})) \tag{19}$$

Model B. Model A is extended with the hazard of lost sales at the end of each period. This introduces lost-sales penalty costs into the objective function.

- Lost sales volume variable L_{pt}: lost sales volume of product p at the end of period t.
- Lost sales costs parameter c^l_p: lost sales penalty costs of product p.

The objective function is

$$\min \sum_{p \in \overline{P}, t \in \overline{T}} (c^i_p y_{pt} + c^l_p L_{pt} + \sum_{m \in \overline{M}} (c^s_{pm} \gamma_{ptm} + c^u_{pm} x_{ptm})) \tag{20}$$

Model C. The Model C removes the lost sales penalties and re-introduces the back-orders policy into the objective function:

$$\min \sum_{p \in \overline{P}, t \in \overline{T}} (c^i_p y_{pt} + c^b_p b_{pt} + \sum_{m \in \overline{M}} (c^s_{pm} \gamma_{ptm} + c^u_{pm} x_{ptm})) \tag{21}$$

5. Computational Testing

5.1. Generating Instances

Tests have been performed on groups of CLSPL-BOPM instances of different sizes (P-M-T) (considered number of items, machines, and periods): $(20, 20, 20)$, $(25, 25, 20)$, $(25, 25, 25)$, $(30, 30, 25)$, and $(30, 30, 30)$. A total of 20 instances for each dimension have been generated. All the results tables list the average result over all the instances of a dimension, and detailed results for the larger group only. The choice of a mostly balanced value in the three problem sizes P, M and T has been made according to the following consideration. Being the number of binary variables of the model $2 \cdot P \cdot M \cdot T$ we expect that increasing one of the other dimension will have a very similar impact on the solver/algorithms performances. The number of continuous variables is instead usually less influent.

In order to randomly generate the problem instances, we relied on the data value generation described in Bollapragada et al. (2011), where the generation scheme was based on a randomized perturbation of values inspired from real productive systems. That scheme has been adapted to our case adding the few missing values. The time unit is 1 day; hence, 1 period corresponds to 1 working day (*daylength).* Data have been generated as follows:

- inventory holding costs $c^i_p = \frac{\sum_{m \in \overline{M}} c^s_{pm}}{M} \cdot rand(0.2,\, 0.4)$;
- setup costs $c^s_{pm} = rand(200,\, 800)$;
- back-order costs $c^b_p = \frac{\sum_{m \in \overline{M}} c^s_{pm} + c^i_p}{M}$;
- setup times $t^s_{pm} = rand(0.05,\, 0.25)$;
- processing times $t^u_{pm} = rand(0.001,\, 0.01)$;
- capacity $C_{tm} = rand(0.75,\, 1.00) \cdot daylength$;
- $AverageCapacity_m = \frac{\sum_{t \in \overline{T}} C_{tm}}{T}$;
- $AverageSetupTime_p = \frac{\sum_{m \in \overline{M}} t^s_{pm}}{M}$;
- $ProductionRate_p = M \cdot \frac{AverageCapacity_m}{t^u_{pm} + [AverageSetupTime_p \cdot (\frac{P}{M} - intrand[0,1])]}$;
- demand volumes $d_{pt} = ProductionRate_p \cdot rand(0.8,\, 1.3)$;
- initial states have been considered null, $b^0_p = y^0_p = \zeta^0_{pm} = 0$.

5.2. Algorithms settings

Initial solutions have been obtained by selecting, for each instance, the best solution provided by two pre-solving methods. The first method comprised a truncated application of the MIP solvers (both Xpress and Cplex), providing two partial solutions. The second method was a heuristic method comprising a five-iteration cycle where, starting from the LP-relaxation of the problem, the two solvers perform a partial rounding on one-fifth of the variables at each iteration, while all variables have been rounded to integer values.

Time limitations for each stage have been set according to the problem sizes, as described in table 3.

Note that all the algorithms (standalone or initial solution + improvements) have the same total time limits on a given dimension so that the results are comparable. Time limits have been chosen in order to give the solver enough

Table 3: Procedures time limits with respect to problem size [s]

	20-20-20	25-25-20	25-25-25	30-30-25	30-30-30
XPRESS/CPLEX/FP	150	500	500	1500	3000
Initial sol. for VPLS/LB	50	200	200	600	1000
VPLS/LB	100	300	300	900	2000

time to provide a good solution. Setting a larger time limit may clearly result in an improving solution. Anyway, on the largest tested instances, this would happen very slowly. In fact, the computer RAM would begin to be filled by the open nodes of the search tree and hence the operative system would begin using hard disk as virtual memory, resulting in a very slow behavior.

Computational tests have been performed on a 64-bit system *Intel ®Core i5 @2.5 GHz* with 8 GB of RAM. Both solvers (*FICO Xpress 7.6* and *ILOG Cplex 12.5*) have been tested. The total estimated computational time spent was about 284 h of non-stop computations.

Parameters have been tuned by preliminary testing and are summarized as follows.

- VPLS fixing parameters have been set as explained in section 3.1.1.
- LB k-parameters have been set as follows:
  $k_D^{min} = 1$
  $k_D^{step} = 2$ (normal) or 10 (big)
  $no\_improv\_threshold = 5$ iterations.
  $FREEITEMS = P/4$; $FREEMACHINES = M/4$
  $k_R = 1$
  $k_{TR}^{step} = 2$, $k_{TR}^{max} = 10$.
- The solver time limit has always been set to the maximum between 10 s and the moment when Xpress finds the first feasible solution to the problem.

5.3. MIP solvers results

Tables 4 and 5 report the results achieved by the two solvers on the test instances. The first table contains the detailed report of testing on the largest problems, whereas the second presents the average results on all tested sizes. The bold cells, in this table and all the subsequent ones, refer to the best solution value obtained on each line/instance of the table. The "Best LB" column contains the higher of the lower bounds found by the MIP solvers. The average (AVG) row contains the average values for each column. Note that on this specific problem *Xpress* performs better than *ILOG Cplex*. Xpress has reached the optimal solution for five of the largest problem instances under test, whereas Cplex never did. Even if Xpress seems to perform better than its opponent, it also highlights a floating behavior. In contrast, whereas the average result is worse than its opponent's, the Cplex solver seems to have more

Table 4: 30-30-30: solver results

INSTANCE	Best LB	Solution	
		Xpress	Cplex
1	7976.066	**9870,428**	12638235,714
2	7475.758	3204125,421	**3091508,596**
3	7480.759	32357976,880	**4426832,901**
4	7614.791	**3143287,988**	6509580,352
5	7533.408	**133617,369**	2543112,400
6	7372.601	**7372,601**	4275878,973
7	7699.353	41994111,900	**4622219,545**
8	8481.065	**8481,065**	12389799,237
9	7836.178	**266436,978**	11904963,864
10	7846.872	**1203094,028**	11989134,554
11	6996.676	27678157,610	**4847169,981**
12	7473.605	**7473,605**	5383834,206
13	7491.706	5819587,822	**3880130,075**
14	7341.890	4402038,568	**4353532,645**
15	7092.496	**1614167,891**	5397827,863
16	7380.767	**7380,767**	12080254,361
17	7922.878	**2081899,725**	9253161,207
18	7994.171	**7994,171**	4785393,293
19	7534.016	**595345,446**	4391619,242
20	7466.557	**174507,954**	3406922,308
AVG	7600.581	**6235846,410**	6608555,566

stable behavior. As a result of this computational campaign, the results in the following sections refer to the algorithms implemented with Xpress.

Table 5: Solver results: average performance

Dimension	Xpress	Cplex
(20-20-20)	**1321390,830**	5776237,768
(25-25-20)	**269906,546**	2070374,969
(25-25-25)	**1617744,753**	5794782,803
(30-30-25)	14296892,571	**6270828,099**
(30-30-30)	**6235846,410**	6608555,566

5.4. Matheuristic approaches

Table 6 collects tests results of the matheuristics application on the large problem instances (30-30-30). Clearly, FP widely outperforms the other matheuristics. Among the other algorithms, the VPLS approach wins. Table 7 lists the average results for all instance sizes. All the previous considerations hold. Regarding computational time, all the matheuristics have a fixed time limit, except for the FP approach and Xpress (in the case it produces an optimal solution). Table 8 lists the average computational time on all instances dimensions. All time values are expressed in seconds. It can be seen how the end times of matheuristic FP were considerably lower than the Xpress end times (and of all other matheuristic approaches). Hence, FP also outperforms the other procedures with regards to CPU time.

Table 6: 30-30-30: matheuristic approaches

Instance	Xpress	VPLS	LB+VPLS	FP
1	9870,428	13640,687	61555,743	**8003,546**
2	3204125,421	13556,447	28743,641	**7544,201**
3	32357976,880	25203,821	78095,925	**8172,668**
4	3143287,988	22630,797	52307,588	**7869,087**
5	133617,369	16342,582	43634,644	**7558,202**
6	**7372,601**	18109,324	49351,231	**7372,601**
7	41994111,900	31137,724	168272,137	**8023,548**
8	**8481,065**	28717,844	76531,295	**8481,065**
9	266436,978	21551,097	61947,777	**7926,621**
10	1203094,028	20592,150	61163,984	**7935,282**
11	27678157,610	21114,695	71964,399	**7585,174**
12	**7473,605**	14643,093	72006,738	**7473,605**
13	5819587,822	24472,170	51417,744	**7913,804**
14	4402038,568	26911,995	159238,531	**7579,515**
15	1614167,891	19277,196	82295,904	**8058,504**
16	**7380,767**	12849,165	69852,748	**7380,767**
17	2081899,725	26395,745	170260,410	**7933,104**
18	**7994,171**	12392,444	42269,187	**7994,171**
19	595345,446	15122,808	47907,301	**7617,988**
20	174507,954	14953,313	65588,910	**7549,962**
AVG	6235846,410	19980,754	75720,291	**7798,670**

Table 7: Matheuristic methods: average performance

	20-20-20	25-25-20	25-25-25	30-30-25	30-30-30
XPRESS	1321390,830	269906,546	1617744,753	14296892,570	6235846,410
VPLS	9610,257	17076,715	20558,658	20091,626	19980,754
LB	27942,582	37181,497	60390,967	59752,297	75720,291
FP	**4610,885**	**5655,585**	**5721,848**	**6858,179**	**7798,670**

Table 8: XPRESS versus FP matheuristic: average time comparison [s]

Algorithm	20-20-20	25-25-20	25-25-25	30-30-25	30-30-30
FP	33	64	135	391	1607
Xpress	101	372	464	1405	2819
Others (time limit)	150	500	500	1500	3000

To further improve the FP results, a test has been performed providing the solution returned by FP to the VPLS improvement procedure. For all tested instances, no improvements have been observed by VPLS after 100 procedure iterations. Hence, no results are presented for the FP + VPLS algorithm.

Table 9: ELSP - Model A

Instance	Xpress	RBS	FP	FPb	FPb+VPLS
1	804995	805656	808685	799656	**799616**
2	750217	748343	746087	733696	**733528**
3	752902	755343	757689	739907	**739561**
4	946569	947555	949460	922287	**916632**
5	814168	813602	813558	798440	**798186**
6	671428	671436	671374	**665392**	**665392**
7	896759	895439	891762	871615	**865864**
8	884018	894323	893211	878233	**878072**
9	835057	837541	826246	797197	**795565**
10	728850	732344	734830	725024	**723770**
11	891080	905340	902116	883368	**883014**
12	736038	746555	746953	731350	**731047**
13	770684	771225	769857	766610	**766525**
14	925677	925723	925027	899649	**898605**
15	981553	919344	915185	885475	**875181**
16	744118	743133	744331	735478	**735467**
17	844764	843520	843975	819737	**819571**
18	842519	845439	846519	830957	**827981**
19	914245	919233	921793	898177	**898112**
20	1084913	1064522	1012814	983664	**982272**
AVG	841028	839281	836074	818296	**816698**

5.5. Discrete ELSP Results

A set of 20 instances of size (20-20-25) have been created with the same data characteristics seen in section 5.1 but adding the new parameters required, according to Bollapragada et al. (2011):

- production costs $c^p_{pm} = rand(3, 9) + rand(2, 6) + 1/(100 + t^u_{pm})$;
- lost sales penalty costs $c^l_p = 10 * c^b_p$.

Tables 9, 10, and 11 list the testing results with a time limit of 500 s. It can be seen that FP outperforms both Xpress and RBS for models A and B, whereas RBS is performs best on model C, where FP is also outperformed by Xpress.

FP was then adapted slightly to perform better for the specific problems, generating algorithm FPb. In particular, it has been shown that the best configuration for the algorithm was to remove the iteration limit and consider a time limit only. Then, to allow a direct comparison, the time limit has been set to 500 s. Then, it has been noted that subproblems sizes are often too large for the solver to obtain an optimum solution for each re-optimization. Hence, another hybridization with VLSP has been implemented, selecting a

Table 10: ELSP - Model B

Instance	Xpress	RBS	FP	FPb	FPb+VPLS
1	811227	810239	815707	799300	**799081**
2	748732	752361	751733	734791	**733494**
3	750730	756323	755713	740081	**739807**
4	937384	939400	939406	922878	**914796**
5	812769	812232	811972	802557	**801941**
6	671371	670343	673716	**665475**	**665475**
7	910072	899436	898430	873544	**867983**
8	899345	889454	898430	873544	**867983**
9	827485	912334	895379	878966	**878688**
10	**733671**	805456	833386	808726	799474
11	911555	833123	734911	**723328**	**723328**
12	**744653**	894544	901295	882475	881834
13	771188	761003	748431	731155	**730917**
14	927213	843455	773137	**766340**	**766340**
15	978098	954001	924038	902302	**901138**
16	**750684**	875456	910115	882875	875755
17	837029	810226	747982	735242	**734907**
18	845079	844344	846843	820150	**819387**
19	911884	888419	846316	826484	**826228**
20	1038871	1042293	914761	899437	**899189**
AVG	840952	849722	831085	813483	**811387**

Table 11: ELSP: model C

Instance	Xpress	RBS	FP	FPb	FPb+VPLS
1	805064	802543	815600	799579	**799342**
2	745991	739232	751437	734582	**734312**
3	749672	773491	757573	740661	**740621**
4	931497	933452	953437	1006057	**919378**
5	819909	815433	819270	800142	**800040**
6	668301	671211	676677	**665581**	**665581**
7	899984	887451	901834	870748	**865490**
8	888110	**878342**	892172	879471	879397
9	823535	835451	833304	803978	**797739**
10	733850	741232	744930	723622	**723504**
11	899442	900012	904332	886479	**882799**
12	737443	739121	748825	730440	**730117**
13	772325	769433	778333	769168	**767153**
14	910042	915436	926214	902264	**899607**
15	951895	915659	922359	889800	**878159**
16	742584	741234	751563	735471	**735286**
17	842620	839454	847585	820451	**819658**
18	840484	841322	846981	916808	**829436**
19	913185	914192	917338	901534	**900965**
20	1034147	1034345	1024537	992074	**983354**
AVG	835504	834402	840715	828445	**817597**

smaller number of variables for the re-optimization problem. Again, the parameters $FREEITEMS$, $FREEMACHINES$, and $FREEPERIODS$ as in the VLSP description, indicating the number of items/machines/periods that can be set as free, have been introduced and set as follows:

- $FREEPERIODS = 2$;
- $FREEITEMS = a \cdot P$;
- $FREEMACHINES = b \cdot M$;

where a and b are randomly drawn from the uniform distribution $U(0.8, 0.9)$.

As can be seen in the final columns of tables 9, 10, and 11, FPb outperforms all the previous methods. Moreover, differently than with the CLSP-BOPM problem, results can be further improved by applying a VPLS improvement phase after the FPb algorithm (column FPb+VPLS), at the expense of some additional computational time.

6. Conclusions

In this study, an extension of the well-known CLSP considering back-order, setup carry-overs, and parallel machines has been investigated. Three matheuristics have been developed, starting from the basic schemes of the well-known variable neighborhood local search, local branching, and feasibility pump algorithms. Computational testing has been performed comparing the results of the three algorithms and two different MIP solvers. The best-performing algorithm was the FP algorithm. The strengths of this algorithm have been confirmed by an additional computational test on three different problems, particularly, cases of the CLSP-BOPM. A slightly modified version of FP outperformed both the solver and a specialized algorithm from the literature. Future studies will be devoted to adapting the algorithmic approaches to more general production problems (i.e. multi-stage production, setup crossover, and sequence-dependent setups).

M. O. Ball. Heuristic based on mathematical programming. *Surveys in Operation Research and Management Science*, 16:21–38, 2011.

M. Belo-Filho, F. Toledo, and B. Almada-Lobo. Models for capacitated lot-sizing problem with backlogging, setup carryover and crossover. *Journal of the Operational Research Society*, 65, 2014.

G Belvaux and LA Wolsey. bc-prod: a specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46(5):724–738, 2000.

PJ Billington, JO McClain, and LJ Thomas. Mathematical programming approaches to capacity-constrained mrp systems: review, formulation and problem reduction. *Management Science*, 29(10):1126–1141, 1983.

GR Bitran and HH Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28(10):1174–1186, 1982.

R. Bollapragada, F. Della Croce, and M. Ghirardi. Discrete-time, economic lot scheduling problem on multiple, non-identical production lines. *European Journal of Operational Research*, 215(1):89–96, 2011.

A. Boschetti, M., V. Maniezzo, M. Roffilli, and A. Boluf Rohler. Matheuristics: Optimization, simulation and control. *Lecture Notes in Computer Science*, 5818:171–177, 2009.

M. Caserta, A. Ramirez, and S. Voß. A math-heuristic for the multi-level capacitated lot sizing problem with carryover. In *European Conference on the Applications of Evolutionary Computation, EvoApplications 2010: Applications of Evolutionary Computation*, pages 462–471, 2010.

S Ceschia, L Di Gaspero, and A Schaerf. Solving discrete lot-sizing and scheduling by simulated annealing and mixed integer programming. *Computers & Industrial Engineering*, 114:335–343, 2017.

F. Della Croce, A. Grosso, and F. Salassa. *Matheuristics: embedding MILP solvers into heuristic algorithms for combinatorial optimization problems. In: Heuristics, Theory and Applications*. NOVA Publisher, 2013.

M. Diaby, HC Bahl, MH Karwan, and S Zionts. A lagrangean relaxation approach for very-large-scale capacitated lot-sizing. *Management Science*, 38 (9):1320–1340, 1992.

C Dillenberger, LF Escudero, A Wollensak, and W Zhang. On solving a large-scale resource allocation problem in production planning. In G Fandel, T Gulledge, and A Jones, editors, *Operations research in production planning and control*, pages 105–119. Springer, Berlin, 1993.

C Dillenberger, LF Escudero, A Wollensak, and W Zhang. On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research*, 75(2):275–286, 1994.

A. Drexl and A. Kimms. Lot sizing and scheduling survey and extensions. *European Journal of Operational Research*, 99:221–235, 1997.

DJ Fiorotto, R Jans, and SA de Araujo. An analysis of formulations for the capacitated lot sizing problem with setup crossover. *Computers & Industrial Engineering*, 106:338–350, 2017.

M. Fischetti, C. Polo, and M. Scantamburlo. A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44(2):61–72, 2004.

M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104, 2005.

M Ghaniabadi and A Mazinani. Dynamic lot sizing with multiple suppliers, backlogging and quantity discounts. *Computers & Industrial Engineering*, 110:67–74, 2017.

M Gopalakrishnan, D Miller, and C Schmidt. A framework for modelling setup carryover in the capacitated lot sizing problem. *International Journal of Production Research*, 33:1973–1988, 1995.

G. Goren, H., S. Tunali, and R. Jans. A hybrid approach for the capacitated lot sizing problem with setup carryover. *International Journal of Production Research*, 50(6):1582–1597, 2014.

KS Hindi. Algorithms for capacitated multi-item lot-sizing without set-ups. *Journal of the Operational Research Society*, 46:465–472, 1995a.

KS Hindi. Solving the single-item, capacitated dynamic lot-sizing problem with startup and reservation costs by tabu search. *Computers & Industrial Engineering*, 28(4):701–707, 1995b.

S Kang, K Malik, and LJ Thomas. Lotsizing and scheduling on parallel machines with sequence-dependent setup costs. *Management Science*, 45(2):273–289, 1999.

B Karimi, SMT Fathemi Ghomi, and JM 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.

A. Menenez, A., A. Clark, and B. Almada-Lobo. Capacitated lotsizing and scheduling with sequence-dependent, period overlapping and not triangular setups. *Journal of Scheduling*, 14(2):209–219, 2010.

L Öznamar and G Barbarosoglu. Hybrid heuristics for the multi-stage capacitated lot sizing and loading problem. *Journal of the Operational Research Society*, 50:810–825, 1999.

L Öznamar and SI Birbil. Hybrid heuristics for the capacitated lot sizing and loading problem with setup times and overtime decisions. *European Journal of Operational Research*, 110(3):525–547, 1998.

G.. Pahl, S. Voß, and L. WoodRuff, D. Discrete lot-sizing and scheduling with sequence-dependent setup times and costs including deterioration and perishability constraints. In *Proceedings of the 44th Hawaii International Conference on System Sciences*, 2011.

L. Pinedo, M. *Planning and Scheduling in Manufacturing and Services*. Springer, xviii ed. edition, 2009.

D. Quadt and H. Khun. Capacitated lot-sizing with extensions: a review. *4OR*, 6:61–83, 2008.

D. Quadt and H. Khun. Capacitated lot-sizing and scheduling with parallel machines, back-orders and setup carry-over. *Naval Research Logistics*, 56(4): 366–384, 2009.

D Quadt and H Kuhn. A conceptual framework for lot-sizing and scheduling of flexible flow lines. *International Journal of Production Research*, 43(11): 2291–2308, 2005.

D Quadt and H Kuhn. Capacitated lot-sizing and scheduling with parallel machines, back-orders, and setup carry-over. *Naval Research Logistics*, 56(4): 366–384, 2009.

WW Trigeiro, LJ Thomas, and JO McClain. Capacitated lot sizing with setup time. *Management Science*, 35(3):353–366, 1989.

T Xia, X Jin, L Xi, and J Ni. Production-driven opportunistic maintenance for batch production based on mamapb scheduling. *European Journal of Operational Research*, 240(3):781–790, 2015.

T Xia, L Xi, S Du, L Xiao, and E Pan. Energy-oriented maintenance decision-making for sustainable manufacturing based on energy saving window. *Journal of Manufacturing Science and Engineering*, 140(5):1–12, 2018.

H Zhengyang and G Hu. A multi-stage stochastic programming for lot-sizing and scheduling under demand uncertainty. *Computers & Industrial Engineering*, 119:157–166, 2018.