A Relax-and-Fix with Fix-and-Optimize Heuristic Applied to Multi-Level Lot-Sizing Problems

Claudio Fabiano Motta Toledo, Márcio da Silva Arantes, Marcelo Yukio Bressan Hossomi
University of São Paulo, Institute of Mathematics and Computer Science
Avenida trabalhador são-carlense, 400, Centro, 13566-590, São Carlos, SP, Brazil
claudio@icmc.usp.br, marcio@icmc.usp.br, marcelo.hossomi@usp.br

Paulo Morelato França
UNESP, Dept. Of Mathematics and Computing
R. Roberto Simonsen, 305, CP 266, 19060-080, Presidente Prudente, SP, Brazil
paulo.morelato@fct.unesp.br

Kerem Akartunalı
Dept. of Management Science, University of Strathclyde
40 George Street, Glasgow G1 1QE, UK
kerem.akartunali@strath.ac.uk

In this paper, we propose a simple but efficient heuristic that combines construction and improvement heuristic ideas to solve multi-level lot-sizing problems. A relax-and-fix heuristic is firstly used to build an initial solution, and this is further improved by applying a fix-and-optimize heuristic. We also introduce a novel way to define the mixed-integer subproblems solved by both heuristics. The efficiency of the approach is evaluated solving two different classes of multi-level lot-sizing problems: the multi-level capacitated lot-sizing problem (MLCLSP) with backlogging and the two-stage glass container production scheduling problem (TGCPSP). We present extensive computational results including four test sets of the MULTILSB (Multi-Item Lot-Sizing with Backlogging) library, and real-world test problems defined for the TGCPSP, where we benchmark against state-of-the-art methods from the recent literature. The computational results show that our combined heuristic approach is very efficient and competitive, outperforming benchmark methods for most of the test problems.

Key words: Lot-Sizing; Heuristics; Relax-and-fix; Fix-and-optimize; Backlogging.

Mathematics Subject Classification (2000): 90C11.

1. Introduction

Manufacturing systems have been analytically studied for more than a century to achieve better efficiencies and outputs, since manufacturing was a key element, if not "the" key element, of the economic advancement of developed countries. 2013 marks the centenary of the renowned "economic order quantity" formula, which was the first attempt to optimize production quantities under very special conditions. Since then, numerous operations researchers in academia and practice have built many more realistic models and proposed various sophisticated solution methods to tackle lot-sizing/production planning problems evident in practice, where decisions such as how much to produce or stock are constrained by various natural limitations such as capacities and setup times.

We investigate two classes of multi-level lot-sizing problems: the multi-level capacitated lot-sizing problem (MLCLSP) with backlogging, and the two-stage glass container production scheduling problem (TGCPSP). The first set of problems, MLCLSP with backlogging, is particularly challenging from a computational point of view, which is also apparent from a number of new lot-sizing problems included in MIPLIB 2010 [20]. Moreover, the theoretical question of the full description of the convex hull of the single-item problem with backlogging has remained open for decades until the recent study of [18], which indicates the sophistication involved in these problems. Finally, in a practical problem setting, backlogging is never prohibited as all manufacturers will sooner or later fall short of satisfying their customer demands and backlog, and therefore the problem with backlogging presents a more realistic case than the one without. The second set of problems, TGCPSP, represents a real-world short-term production planning and scheduling problem with a first mixed-integer programming (MIP) formulation proposed in [4]. The authors in [27] improved the previous MIP formulation for the TGCPSP, proposed a hybrid genetic algorithm to solve it and defined sets of complex test problems. TGCPSP does not allow backlogging like MLCLSP, and takes some problem specific characteristics such as production loss costs and sequence-dependent setup times and costs.

The lot-sizing literature can most appropriately be divided into two main areas due to the nature of solution methods used: i. Exact methods, and ii. Heuristic methods. Although even the capacitated single-item problem is $\mathcal{NP}$-hard (see, e.g., [14]) and expectations for optimal solutions diminish as problems become more realistic, exact methods can be very helpful to understand the underlying difficulties in solving these problems. Such methods include valid inequalities (see, e.g., [8, 19]), extended reformulations (see, e.g., [17, 12, 24]), Lagrangian relaxation (see, e.g., [10]) and Dantzig-Wolfe decomposition (see, e.g., [11]). Most of these studies are mainly focused on simplistic (often single-item) problems, however, many of the developed methods could be extended to realistic problem settings as well. The recent study of [3] provides more insight on the complexities apparent in realistic lot-sizing problems, and an extensive discussion of mathematical programming techniques used in the area can be found in [22].

Although exact methods are powerful since they provide an exposure of complicating structures and a guarantee on solution quality, they exhibit an important drawback on the computational end: even with the modern fast computers and the state-of-the-art optimization packages, solving industrial-size lot-sizing problems is a very complicated (and often an impossible) task. To compensate for the computational shortcomings of exact methods and to provide real time solutions to industrial-size problems, heuristic methods have been extensively used in this area, from very simple frameworks to very sophisticated ones, see, e.g., [29, 30, 16, 1, 28, 6]. We also refer the interested reader to [7] for a recent literature review on general mathematical programming heuristics. Finally, we note that a number of researchers have proposed frameworks using heuristics or meta-heuristics combined with mathematical programming techniques, since the major drawback of heuristic methods is no guarantee of solution quality. Recent results include the ant colony algorithm coupled with reduced MIP solutions for the MLCLSP with overtime proposed by [5], the MIP-based and hybrid simulated annealing heuristics for the stochastic lot-sizing problem proposed by [23], and finally the multi-population genetic algorithm with LP model resolution for the MLCLSP with backlogging proposed by [28].

The method described in this paper combines two heuristics based on mathematical programming. Relax-and-fix (RF), a construction heuristic that solves relaxed MIP subproblems sequentially and fixes binary variables throughout the process for speeding it, has been used by a number of researchers for lot-sizing problems: [9] included a basic RF heuristic in their sophisticated lot-sizing solver, whereas [26] proposed a time-oriented RF for MLCLSP with impressive results. More recent applications of RF in the lot-sizing literature include [13] and [2], where the former iteratively increase the size of the problem for efficient solutions whereas the latter make use of (ℓ, S) inequalities for stronger formulations, outperforming solutions found by Stadtler's heuristic [26]. Fix-and-optimize (FO), an improvement heuristic based on MIP, is firstly described in [15] to solve the MLCLSP with lead times and overtime costs. The authors propose product, resource and process-oriented decompositions for the problem, which define subsets of binary variables to be optimized. [25] extend these decomposition ideas to the multi-level lot sizing and scheduling problem, where the neighborhood decomposition search is combined with FO.

We propose a simple and easy-to-implement solution method that also proves to be computationally effective. Contrary to the recent works of [5, 23, 28] combining complex meta-heuristics with MIP heuristics, our method combines two very simple MIP-based heuristics: relax-and-fix with fix-and-optimize ("RFFO", as we will refer to in the remainder of the paper). The simplicity is one of the key strengths of the proposed method, allowing any interested researcher or practitioner easily implement it if needed. Moreover, we propose novel ways of building subproblems from the classical rolling time horizon approach, which are important components of relax-and-fix and fix-

and-optimize heuristics, and investigate their effectiveness in practice by extensive computational tests, including over some MIPLIB 2010 instances [20]. The method shows impressive computational performance for the majority of difficult test problems of the MCLSP with backlogging, outperforming benchmark methods. Moreover, while many studies such as [15] and [25] explore very specific problem structures for their methodology design, our proposed RFFO framework is designed as generic as possible to avoid taking advantage of a specific problem structure and hence can be extended to other problems if necessary, and in order to support this argument, we have also applied it to TGCPSP, where RFFO was able to find competitive results when compared with the default IBM Ilog Cplex solver and the hybrid genetic algorithm of [27].

To summarize, the proposed RFFO method has two main contributions: i) It is a simple framework combining construction and improvement heuristics, which also returns competitive results in extensive computational tests when compared with state-of-the-art benchmark methods from the recent literature. ii) The rolling horizon window size is oriented not only by column (i.e., period in lot-sizing, which is the common practice) but also by rows (i.e., families of products) as well as by a combination of columns and rows. Therefore, the method allows rolling windows along with different combinations of columns and rows in the two dimensional matrix representation.

The paper is organized as follows. In the next section, we give a brief mathematical description of the problems under investigation. In Section 3, we define in detail our proposed framework, including a discussion of novel ways of building subproblems. Then we present numerical results from extensive computational tests in Section 4 with comparisons to two benchmarks methods from recent literature, showing the effectiveness of the proposed methodology. Finally, we conclude with some future directions in Section 5.

2. Multi-Level Lot Sizing Problems

As we discussed earlier, the RFFO approach developed in this paper is not dependent on the problem structure so that it can be adapted to other MIP problems. In this section, we present the MIP formulations of the two classes of multi-level lot-sizing problems. First, we describe the MIP formulation of [28], based on the formulation of [2], for the MLCLSP with backlogging, and then we describe the MIP formulation for the TGCPSP as presented in [27].

2.1 MLCLSP with Backlogging

In this paper, we consider MLCLSP with families of products, i.e., multiple products are grouped into families based on their similarities. Since backlogging is a natural practice in manufacturing environments due to capacity limitations, it is allowed for products with external demands.

Parameters:

J	Total number of products.
T	Total number of periods.
M	Total number of machines/resources.
F	Total number of families.
a_{mj}	Time necessary to produce one unit of product j on machine m.
B_{jt}	Upper bound for lot-size of product j in period t.
bc_j	Backlogging cost of product j.
C_{mt}	Total capacity of machine m in period t.
D_{jt}	Primary demand (external) of product j in period t.
h_j	Holding cost per unit of product j in one period.
p_{jf}	1 if product j belongs to family f.
r_{jk}	Quantity of product j necessary to produce one unit of product k.
st_{mf}	Setup time for product family f on machine m.
$\delta(j)$	Set of the immediate successors of product j.
Δ	Set of the end products.

Variables:

x_{jt}	Lot-size of product j in period t.
w_{ft}	Setup variable of family f in period t.
i_{jt}	Stock holding quantity of product j in period t.
b_{jt}	Backlogging quantity of product j in period t.

$$Min \quad \sum_{j=1}^{J}\sum_{t=1}^{T}(bc_j \cdot b_{jt} + h_j \cdot i_{jt}) \tag{1}$$

Subject to:

$$i_{jt-1} + b_{jt} + x_{jt} = i_{jt} + b_{jt-1} + D_{jt} \qquad \forall j,t|j \in \Delta \tag{2}$$

$$i_{jt-1} + x_{jt} = i_{jt} + \sum_{k\in\delta(j)} r_{jk} \cdot x_{kt} \qquad \forall j,t|j \notin \Delta \tag{3}$$

$$\sum_{j=1}^{J} a_{mj} \cdot x_{jt} + \sum_{f=1}^{F} st_{mf} \cdot w_{ft} \le C_{mt} \qquad \forall m,t \tag{4}$$

$$x_{jt} \le w_{ft} \cdot B_{jt} \qquad \forall j,f,t|p_{jf} = 1 \tag{5}$$

$$x_{jt}, i_{jt}, b_{jt} \ge 0 \qquad w_{ft} \in \{0,1\} \tag{6}$$

The inventory and backlogging costs are minimized in the objective function (1). We note that we do not include setup costs for the sake of simplicity of the model (and consider setup times only instead), but the proposed model can be easily modified to include them. The flow balance constraints (2) and (3) ensure the satisfaction of external and internal demands, respectively, where the external demand for end products can also be satisfied through backlogging. Here, we note that we use these constraints for the sake of simplicity as well as for consistency with the formulations of [2, 28]; however, external demands as well as backlogging can also be included in higher levels of the echelon. The big bucket machine capacities incorporating both variable processing times and fixed setup times in each period are defined by constraints (4), where we assume that each product belongs to only one product family and there are product family setup times only (rather than for each product). Constraint (5) ensures that a product j cannot be produced (i.e., $x_{jt} = 0$) if there is no setup for its product family (i.e., $w_{ft} = 0$). The upper bound for the lot-size of product j in period t is represented by parameter B_{jt}, which can be defined using the following definitions of (7) and (8) (in a similar fashion to [2]).

$$B_{jt} = \min\left(d_{j(1..T)}, \frac{C_{mt} - st_{mf}}{a_{mj}}\right) \tag{7}$$

$$d_{j(t..T)} = \sum_{u=t}^{T} D_{ju} + \sum_{k \in \delta(j)} r_{jk} \cdot d_{k(t..T)} \tag{8}$$

Note that the equation (7) bounds the lot-size either by the total demand over the horizon (the first term on the right of the equation) or by the maximum capacity available for production (setup time to be subtracted from the total capacity to identify the production time). Finally, the variable domains are established by constraints (6). We note that this formulation can be extended to incorporate other elements of a production system, such as overtime, to make it more realistic. However, we leave it as is for the sake of easier understanding. Finally, we note that due to backlogging allowed to the final period of the horizon, this problem is always feasible. However, as we have observed from our own computational experiences as well as from our discussions with some other researchers, this is a characteristic that makes these problems computationally challenging when attempting to optimize.

2.2 Two-stage Glass Container Production Scheduling Problem

This problem originates from the glass container manufacturing, where a furnace melts the raw material in the first stage of the production process, and molding machines are used in the second stage to finalize the containers. In a typical glass container manufacturer, the daily capacity of the furnace can vary from 100 ton/day to 650 ton/day. We refer the interested reader to [27] for further technical details of the production process. In short, the TGCPSP is a two-level lot sizing and scheduling problem with parallel machines and sequence-dependent setup costs and times. Different than the problem discussed in the previous section, it does not allow backlogging.

Parameters

C : Melting capacity of the furnace in a period (in tonnes).

$\overline{n}_{ik}$: Maximum number of mold cavities of machine k for product i.

$\underline{n}_{ik}$: Minimum number of mold cavities of machine k for product i.

p_{ik} : Amount of product i produced per mold cavity of machine k in a period (in tonnes).

h_i : Holding cost for carrying one tonne of product i into the next period.

c_{ijk} : Cost to set up machine k from product i to product j, $i \neq j$.

s_{ijk} : Capacity necessary to set up machine k from product i to product j, $i \neq j$ (in tonnes).

d_{it} : Demand for product i at the end of period t (in tonnes).

ω : Penalty cost per tonne of furnace under-utilization.

Decision Variables

Y_{itk} : 1 if product i is assigned to machine k in period t; 0 otherwise.

Q_t : 1 if the furnace is active in period t; 0 otherwise.

Z_{ijtk} : 1 if there is a setup changeover from product i in period $t-1$ to product j in period t on machine k; 0 otherwise.

N_{itk} : Number of active mold cavities on machine k dedicated to product i in period t.

I_{it} : Inventory of product i at the end of period t (in tonnes).

$\overline{Id}_t$: Idle capacity of the furnace in period t (in tonnes).

$$Min \quad \sum_{i,j,t,k} c_{ijk} \cdot Z_{ijtk} + \omega \cdot \sum_{t} \overline{Id}_t + \sum_{i,t} h_i \cdot I_{it} \tag{9}$$

Subject to:

$$I_{it} - I_{i,t-1} + d_{it} = \sum_{k} p_{ik} \cdot N_{itk} - \sum_{k,j} s_{jik} \cdot Z_{jitk} \qquad \forall(i,t) \tag{10}$$

$$\sum_{i,k} p_{ik} \cdot N_{itk} + \overline{Id}_t = C \cdot Q_t \qquad \forall(t) \tag{11}$$

$$N_{itk} \leq \overline{n}_{ik} \cdot Y_{itk} \qquad \forall(i,t,k) \tag{12}$$

$$N_{itk} \geq \underline{n}_{ik} \cdot Y_{itk} \qquad \forall(i,t,k) \tag{13}$$

$$\sum_i Y_{itk} \leq 1 \qquad \forall (t,k) \tag{14}$$

$$Q_t = \sum_i Y_{itk} \qquad \forall (t,k) \tag{15}$$

$$\sum_i Y_{itk} \geq \sum_i Y_{i(t+1)k} \qquad \forall (t,k) | t < T \tag{16}$$

$$Y_{jtk} + Y_{i(t-1)k} \leq Z_{ijtk} + 1 \qquad \forall (i,j,t,k) \tag{17}$$

$$\sum_{i,j} Z_{ijtk} \leq Q_t \qquad \forall (t,k) \tag{18}$$

$$p_{ik} \cdot N_{itk} - \sum_j s_{jik} \cdot Z_{jitk} \geq 0 \qquad \forall (i,t,k) \tag{19}$$

$$\big(I_{it}, \overline{Id}_t, Q_t\big) \geq 0, N_{itk} \in \mathbb{Z}_+, (Y_{itk}, Z_{ijtk}) \in \{0,1\} \tag{20}$$

The problem aims to minimize the total cost over the planning horizon that involves inventory and setup costs as well as penalties for the idle capacities of furnaces, as noted in (9). The glass container demands have to be fulfilled without backlogging as ensured by (10), where the "setup time" (i.e., the number of tonnes of products wasted from the capacity) is also taken account of. The constraint (11) enforces the capacity limit of the furnace, and also ensures that the idle time is captured if the furnace is used. The maximum and minimum number of active mold sections in a given machine are enforced by (12) and (13) respectively, when a product is produced. Each machine can produce at most one product in a time period (14), and the two-stage process is synchronized by (15), which would activate the furnace if a product is assigned to a machine. If the furnace is deactivated in period t, then all associated machines will also be idle in the remainder of the horizon as enforced by (16). Constraints (17) and (18) capture product changeovers and ensure that they can happen only when the furnace is active. Constraint (19) enforces that the "setup time" used is not greater than the quantity produced. Finally, (20) defines the variable domains.

3. Proposed Heuristic: Relax-and-Fix with Fix-and-Optimize

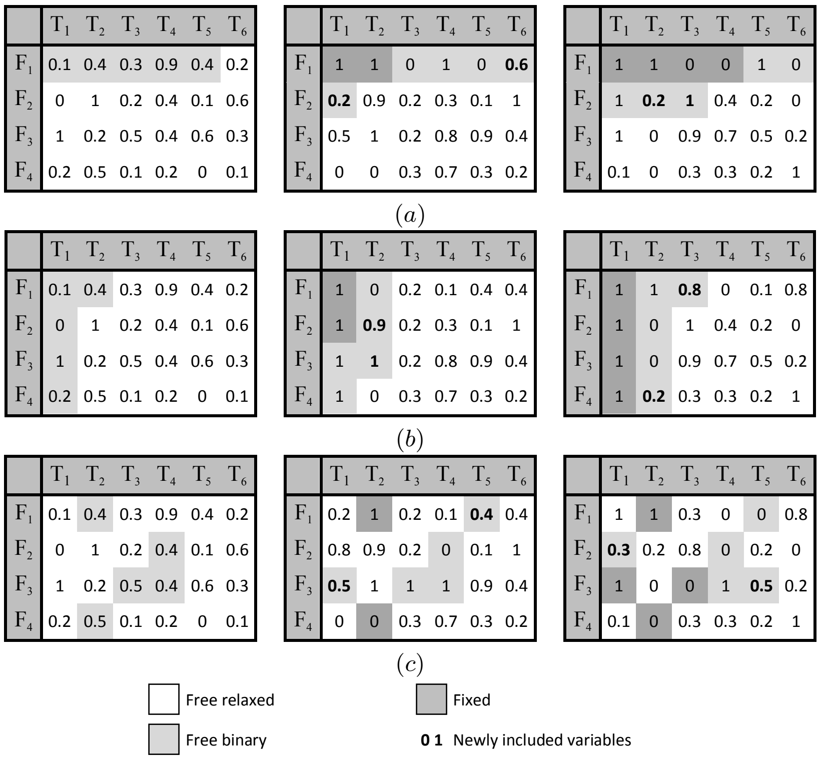
Here we describe the two heuristics and how they are combined to solve the multi-level lot-sizing problems. For both heuristics, let's consider a matrix $F \times T$ where each entry is a binary variable w_{ft}. The relax-and-fix (RF) is a construction heuristic which defines an initial solution by solving several small mixed-integer problems (MIP). This is done by fixing or relaxing most of binary variables, enforcing only few of them to be integer and optimizing them. We call this small set of integer variables as *window* in the remainder of the paper. The pseudo-code of the RF approach proposed in this paper is summarized in Figure 1.

```
1:  function RelaxAndFix($sol.w, windowSize, windowType, overlap, timeLimit$)
2:      $window \leftarrow$ initWindow( $windowSize$, $windowType$, $sol.w$ )
3:      $w_{fix} \leftarrow \emptyset$
4:      $w_{MIP} \leftarrow window$
5:      $w_{LP} \leftarrow sol.w - window$
6:      while fixed solution not reached and $elapsedTime < timeLimit$ do
7:          Solve( $w_{fix}$, $w_{MIP}$, $w_{LP}$ )
8:          $window \leftarrow$ moveWindow( $overlap$, $windowSize$ )
9:          $w_{fix} \leftarrow w_{fix} \cup (w_{MIP} - window)$
10:         $w_{MIP} \leftarrow window$
11:         $w_{LP} \leftarrow w_{LP} - window$
12:     end while
13:     $sol.w \leftarrow w_{fix}$
14:     return $sol.w$
15: end function
```

Figure 1: Pseudocode of Relax-and-Fix.

The inputs of the RF are the set of binary variables (*sol.w*), the number of binary variables (*windowSize*) to be chosen, the selection criteria to choose variables (*windowType*), the overlap rate of binary variables to be re-optimized (*overlap*) and the execution time limit (*timeLimit*). Initially, all binary variables in the RF solution (*sol.w*) are relaxed which means they can take any value between 0.0 and 1.0. A *window* is defined as a set that includes a fixed amount (*windowSize*) of variables (line 2, Figure 1). The variables inside the window are enforced to be integer in the set w_{MIP} (lines 4 and 10, Figure 1), while the others are kept relaxed in w_{LP} (lines 5 and 11, Figure 1). The resulting MIP is then solved (line 7, Figure 1). Next, a new set of variables (*window*) is defined, a subset of integer variables is fixed (w_{fix}), and another sets of integer and relaxed variables are optimized (lines 8 to 11, Figure 1).

The window type (*windowType*) defines how variables are selected to compound the window, as well as how it is moved after each iteration. We propose three different window types: row-wise, in which the window moves along rows; column-wise, in which the window moves along columns; and value-wise, in which the window selects the variables with relaxed values closest to 0.5. The window moves *step* variables at each iteration, with $step = \lfloor overlap * windowSize \rfloor$ (line 8, Figure 1), where $overlap \in [0, 1]$ is the overlap rate defined by the user. Fixing happens to all variables that leave the window in the next iteration (line 9, Figure 1), and the same number of relaxed variables are enforced to be integer. The algorithm continues processing in this fashion until all variables are fixed. We note that the RF process would benefit if the problem considered, such as the MLCLSP with backlogging, has always a feasible solution; this is a property commonly exploited by other researchers using RF as well (see, e.g., [26, 2]).



(a)

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	0.1	0.4	0.3	0.9	0.4	0.2
F_2	0	1	0.2	0.4	0.1	0.6
F_3	1	0.2	0.5	0.4	0.6	0.3
F_4	0.2	0.5	0.1	0.2	0	0.1

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	1	1	0	1	0	**0.6**
F_2	**0.2**	0.9	0.2	0.3	0.1	1
F_3	0.5	1	0.2	0.8	0.9	0.4
F_4	0	0	0.3	0.7	0.3	0.2

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	1	1	0	0	1	0
F_2	1	**0.2**	**1**	0.4	0.2	0
F_3	1	0	0.9	0.7	0.5	0.2
F_4	0.1	0	0.3	0.3	0.2	1

(b)

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	0.1	0.4	0.3	0.9	0.4	0.2
F_2	0	1	0.2	0.4	0.1	0.6
F_3	1	0.2	0.5	0.4	0.6	0.3
F_4	0.2	0.5	0.1	0.2	0	0.1

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	1	0	0.2	0.1	0.4	0.4
F_2	1	**0.9**	0.2	0.3	0.1	1
F_3	1	**1**	0.2	0.8	0.9	0.4
F_4	1	0	0.3	0.7	0.3	0.2

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	1	1	**0.8**	0	0.1	0.8
F_2	1	0	1	0.4	0.2	0
F_3	1	0	0.9	0.7	0.5	0.2
F_4	1	**0.2**	0.3	0.3	0.2	1

(c)

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	0.1	0.4	0.3	0.9	0.4	0.2
F_2	0	1	0.2	0.4	0.1	0.6
F_3	1	0.2	0.5	0.4	0.6	0.3
F_4	0.2	0.5	0.1	0.2	0	0.1

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	0.2	1	0.2	0.1	**0.4**	0.4
F_2	0.8	0.9	0.2	0	0.1	1
F_3	**0.5**	1	1	1	0.9	0.4
F_4	0	0	0.3	0.7	0.3	0.2

	T_1	T_2	T_3	T_4	T_5	T_6
F_1	1	1	0.3	0	0	0.8
F_2	**0.3**	0.2	0.8	0	0.2	0
F_3	1	0	0	1	**0.5**	0.2
F_4	0.1	0	0.3	0.3	0.2	1

□ Free relaxed	▩ Fixed
▨ Free binary	**0 1** Newly included variables

Figure 2: Relax-and-fix different types of windows: (a) row-wise; (b) column-wise; (c) value-wise. For each window, three iterations are shown in sequence. Window size = 5, overlap rate = 60%.

Figure 2 shows examples of the three window types as well as how they proceed for $windowSize = 5$ and $step = 2$. Figure 2 (a) illustrates the row-wise window, where variables from w_{F_1,T_1} to w_{F_1,T_5} are first include in *window* to be optimized as binary variables. After finding the solution of this MIP, variables w_{F_1,T_1} and w_{F_2,T_2} are fixed and leave the *windows* set, while variables w_{F_1,T_6} and w_{F_2,T_1} are enforced to be binary variables. This procedure allows re-optimizing variables w_{F_3,T_1}, w_{F_4,T_1} and w_{F_5,T_1} in this step. A similar idea is applied in the column-wise window as illustrated by Figure 2 (b).

When using the value-wise window, the model is first solved with all variables relaxed so that the relaxed solution is obtained for evaluation. This is shown by the first matrix on the left side of Figure 2 (c), where w_{F_1,T_2}, w_{F_4,T_2}, w_{F_3,T_3}, w_{F_2,T_4} and w_{F_3,T_4} variables are the closest to 0.5. Thus, they are selected to be optimized as binary variables in *window* set. When two or more variables have the same value, those within first columns are preferred. If the variable with same value are in the same column, then first rows will be picked first. In the two-dimensional matrix $F \times T$ defined for lot-sizing problems, this means to choose products in the earlier periods (first columns) and end products (first rows). These criteria are also used to decide how to fix variables, after the MIP problem is solved. For example, variables w_{F_1,T_2} and w_{F_4,T_2} are chosen to be fixed (middle matrix, Figure 2 c), once they are in the first column among variables in the *window* set. In this step, variables w_{F_3,T_1} and w_{F_1,T_5} are now included in *window*, and w_{F_3,T_3}, w_{F_2,T_4} and w_{F_3,T_4} remain to be re-optimized.

In our framework, the solution built by the RF will be used as the solution to initiate the fix-and-optimize (FO). The steps executed by FO are very similar to RF, where several MIP problems need to be solved. Figure 3 shows the FO pseudo-code. A rolling window, covering *windowSize* number of variables in $F \times T$ matrix (*sol.w*), is also defined for FO and these variables are adjusted as binary to be optimized by a solver. However all variables outside the window are kept fixed in the FO heuristic. At each iteration, after solving the MIP subproblem, the window is moved *step* variables forward with $step = |overlap * windowSize|$ (line 9, Figure 3).

```
1: function FixAndOptimize($sol.w, windowSize, overlap, timeLimit$)
2:     $windowType \leftarrow 0$
3:     repeat
4:         $window \leftarrow$ initWindow( $windowSize$, $windowType$ )
5:         $w_{MIP} \leftarrow window$
6:         $w_{fix} \leftarrow sol.w - window$
7:         while window not reached end do
8:             Solve( $w_{fix}$, $w_{MIP}$ )
9:             $window \leftarrow$ moveWindow( $overlap$, $windowSize$)
10:            $w_{MIP} \leftarrow window$
11:            $w_{fix} \leftarrow sol.w - window$
12:            $sol.w \leftarrow w_{fix} \cup w_{MIP}$
13:        end while
14:        $windowType \leftarrow 1 - windowType$
15:    until $elapsedTime < timeLimit$ and $windowType \neq 0$
16: end function
```

Figure 3: Pseudocode of Fix-and-Optimize.

FO improves the binary values following row and column directions and hence two window types are defined. The first window type combines the row-wise, which covers the matrix along the rows (Figure 4 a) and column-wise, which does the same along the columns (Figure 4 b) following the same idea defined for RF. However, FO applies both windows types during its execution adjusting *windowType* in lines 2 and 14 (Figure 3). In this case, $windowType = 0$ means to apply row-wise and $windowType = 1$ column-wise.

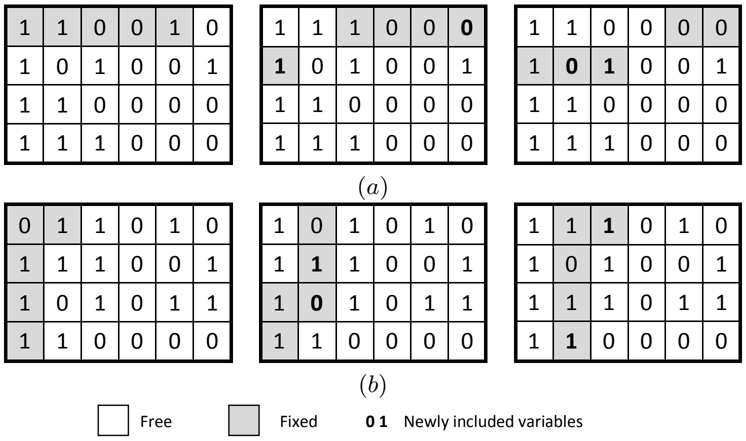


Figure 4: Fix-and-Optimize: (a) row-wise and (b) column-wise. Three iterations of each direction are shown with window size = 5 and overlap rate = 60%.

The second type is a square-wise window that covers the matrix along rows (Figure 5 a) and columns (Figure 5 b) simultaneously, compounding a square through the matrix. Note that this square overlaps in both sides with the same overlap rate, and *step* is rounded down to the closest integer multiple of the square side. The square moves along rows and columns during the FO execution according to the *windowType* value. In this case, $windowType = 0$ means moving the square in the row direction and $windowType = 1$ moves it in the column direction.

The Relax-and-Fix with Fix-and-Optimize (RFFO) heuristic proposed is summarized in Figure 6. After the RF execution is complete, FO tries to improve this initial solution until the time limit has been reached. If the improvement achieved by a FO solution did not satisfy a given tolerance *tol*, the window size is increased by *inc* variables, a user-defined parameter. Thereafter, the MIP subproblems become larger as an attempt to find better solutions. When using the square window, the increment will affect the window area, making its growth faster when small, and slower when larger. The window area will be rounded to the closest perfect square integer so that the square window can be formed, but the rounded value will not be used in future increment calculations.

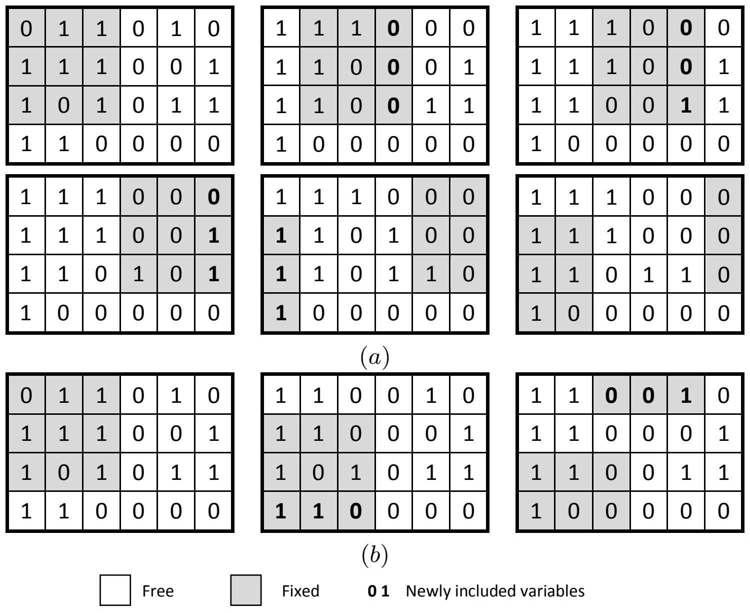


Figure 5: Fix-and-Optimize with square window following: (a) row and (b) column directions. Six iterations of column direction and three of row direction are shown, both with window size = 9 and overlap rate = 70%.

4. Computational Results

The computational tests reported here were run on a PC with Intel i7 processor, 2.6 GHz, and 8GB RAM, and all mathematical models were implemented and solved using IBM ILOG Cplex 12.2 callable library. We have implemented RFFO first to solve the MLCLSP with backlogging, and therefore, we discuss in detail parameter tuning of the method as well as extensive results achieved for the MLCLSP next. Then, the application of RFFO to TGCPSP will follow with computational results obtained for instances based on parameters obtained from a glass container manufacturer.

```
function RFFO($rfSize, rfOverlap, foSize, foOverlap, tol, inc, timeLimit$)
    $sol \leftarrow$ RelaxAndFix( $rfSize, rfOverlap, timeLimit$ )
    $prevCost \leftarrow sol.cost$
    while $timeElapsed < timeLimit$ do
        FixAndOptimize( $sol, foSize, foOverlap, timeRemaining$ )
        if $(sol.cost - prevCost)/prevCost < tol$ then
            $foSize \leftarrow foSize + inc$
        end if
    end while
end function
```

Figure 6: Pseudocode of RFFO.

4.1 Results for MLCLSP with Backlogging

In order to evaluate the effectiveness of our method solving the MLCLSP, we compare the RFFO framework to two state-of-the-art methods from the recent literature: Aheur [2] and LugNP [30]. The executable codes of these methods were kindly provided by the respective authors so that we could run all methods on the same computer for a fair comparison.

We used all the four test sets (SET1 to SET4) of [21] for our computational experiments, where for all problems multi-item and backlogging are allowed. Each of these test sets has 30 instances with 6 machines, 78 products (divided into 11 product families) and 16 periods, except that SET2 instances have 24 periods. A product can be component for only one product in the bill of materials (assembly structure) defined for these instances. The resource utilization factor is 1.05 for SET1 and SET2, 2.0 for SET3 and 1.25 for SET04. This factor determines how much of the total maximum resource capacity is required to supply all the demand, which means a factor greater than 1.0 implies that backlogging in the last period is necessary. The backlogging costs are set to twice the inventory holding cost for SET1 and SET2, and 10 times the inventory holding costs for SET3 and SET4. These characteristics make SET3 and SET4 harder to solve, as also noted by [3]. Some of the hardest instances from this test are recently included in the MIPLIB 2010 library [20] as “open problems”. We make a practical remark that very high utilization factors of SET3 and SET4 make these test instances unrealistic in practice. However, the computational challenges they offer as well as the fact that other researchers have used them make these instances appealing, giving us a significant opportunity to benchmark.

All three methods were executed for 100 seconds in SET1, 150 seconds in SET2 and 300 seconds in SET3 and SET4, to remain consistent with the computational times used by [30] for LugNP, where the Aheur and LugNP results achieved smaller duality gaps against those returned by the branch-and-cut (B&C) algorithm embedded in Cplex.

Initially we set RFFO parameter values empirically, based on preliminary tests executed over randomly chosen instances, where RF and FO apply value-wise and row/column-wise, respectively. RFFO was set with an initial window size of 140 variables for RF windows and 5 for FO windows. Overlap rate for RF was set to 80%, and for FO to 40%. FO improvement tolerance used was set to 5%, and failing to obtain such improvement would increase the window size by 40 variables. Next, the effects of changing these initial parameters were evaluated with computational tests conducted over SET1 to SET4 of MLCLSP as shown by Figure 7. The new parameter values chosen from the initial ones are indicated with circles, and they allow us to customize RFFO to achieve better results for the benchmark set of instance of MLCLSP. The average deviation for all instances in each set is outlined and such results are compared to LugNP and Aheur. We calculate the deviation (denoted by Dev(%)) for all instances, using the equation (21), where Sol^{Ref} refers to the "Reference" solution, i.e., LugNP and Aheur.

$$Dev(\%) = \left(\frac{Sol^{RFFO} - Sol^{Ref}}{Sol^{Ref}} \cdot 100 \right) \tag{21}$$

The effect of changing parameter values is negligible for the instances of SET1 and SET2, and it also remains limited for the instances of SET3, whereas SET4 instances seem to be in general quite sensitive to parameter changes. A large window for RF seems to be not as efficient as one with 40 variables, where the results for SET4 seem to fluctuate as shown in Figures 7 (a) and (b). The RF overlap rate of 80% seems to be slightly better than low values (Figures 7 (c) and (d)), and increasing the preferable value of 1% for FO improvement tolerance worsens the results over SET4 (Figures 7 (e) and (f)). A large initial FO windows size with 40 variables gives some improvement for all sets (Figures 7 (g) and (h)), with significant fluctuation for SET4 instances. Once FO fails to improve solutions by 1%, an increment of 10 variables seems to be working best for increasing submodels (Figures 7 (i) and (j)). Finally, the overlap rate of 50% produces slightly better average deviations than the other values (Figures 7 (k) and (l)). We also present the improvement of deviations after each parameter change from the initial settings in Figure 8, which indicates that these adjustments have the biggest impact on SET4 instances.

As shown on all these cases, the RFFO performs better when RF and FO start solving MIP sub-problems with the RF and FO window size of 40 variables. RF is able to obtain better solutions for FO when 80% of its variables can be re-optimized (Overlap rate), while FO works better re-optimizing 50% of its variables. Such behavior seems to be related to the fact that FO is an improvement heuristic and RF is a construction heuristic, so RF needs to review past decision more often to converge to a feasible solution. We also note that we have experimented with the sensitivity of other parameters but seen insignificant differences in many cases. For example, RFFO achieves average values less than 0.1% different when FO improvement tolerance is set as 5% and 10% in SET4. An exhaustive finer evaluation of these parameters and experimenting with other test sets might potentially lead RFFO to achieve "optimal" performance. However, as noted earlier, our main focus in this paper is to evaluate strategies regarding choices of decompositions of MIP sub-models apparent in RF and FO, which we will discuss next.

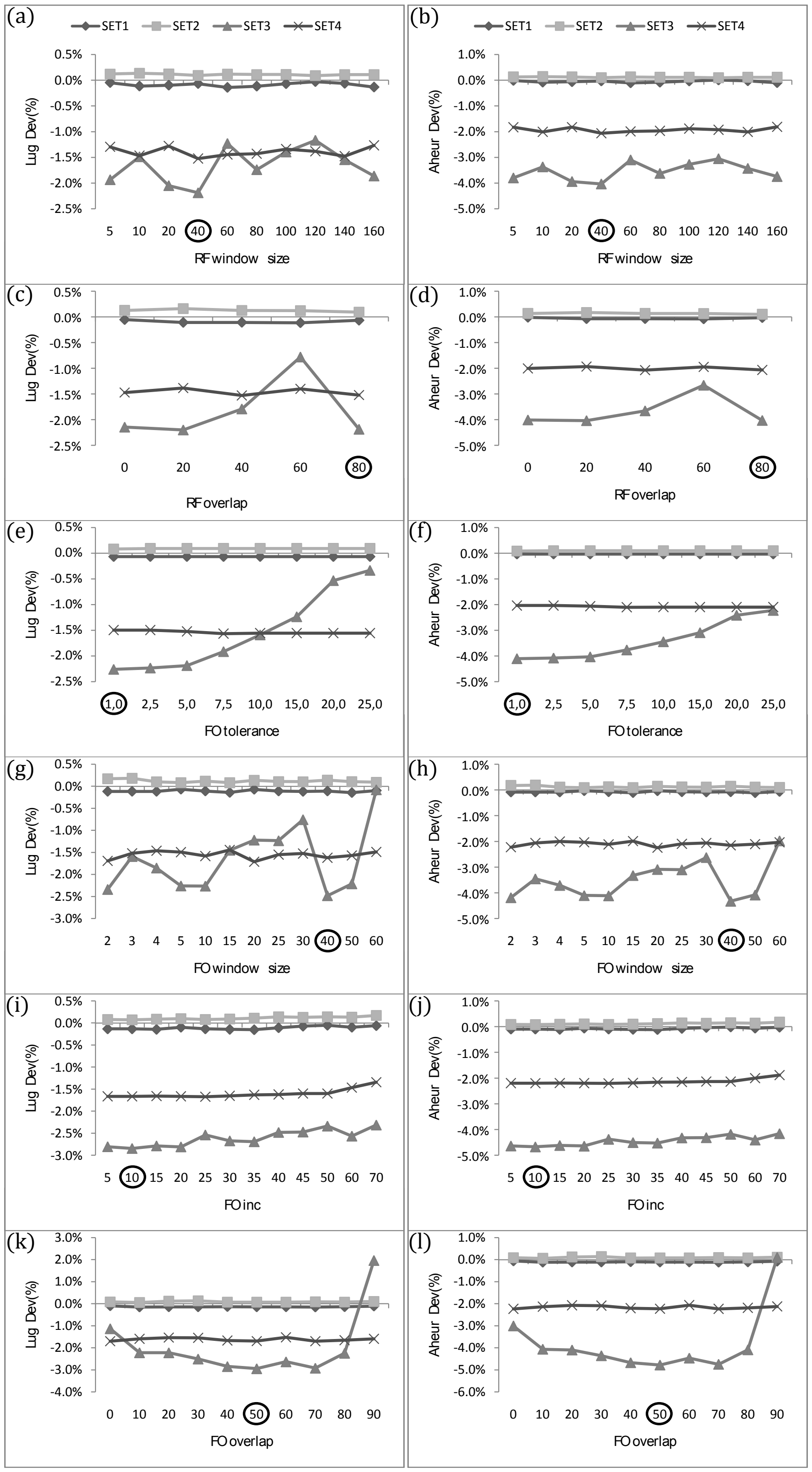
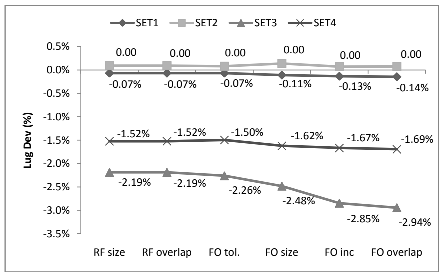


Figure 7: Analysis of parameter values: (*a*,*b*) RF window size, (*c*,*d*) RF overlap rate, (*e*,*f*) FO tolerance, (*g*,*h*) FO window size, (*i*,*j*) FO increment, (*k*,*l*) FO overlap rate



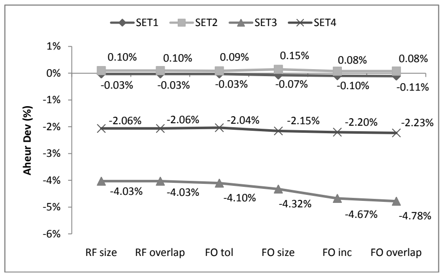


Figure 8: Successive improvements in the parameters: (a) Lug Dev(%), (b) Aheur Dev(%)

A total of six parameter setups were defined in order to determine which window type combination has better performance for MIP sub-models in RF and FO. Parameter setups #1 to #3 use RF row-wise, column-wise and value-wise windows, respectively, combined with FO row/column window (first type). Setups #4 to #6 use all RF windows combined now with FO square window (second type). We executed all the six setups for all test instances, and the results were compared to LugNP. Table 1 summarizes the results obtained by each setup, showing the number of better solutions found by RFFO, LugNp and the draws. It is considered draw when the deviation of solution values for some instance is less than 0.01%. The last column in Table 1 has the average deviation of RFFO solutions against LugNP for all instances in all sets.

Table 1: Average deviation and number of better solutions in all sets for each window type combination

	Windows		Better Solutions			
Setup	RF	FO	RFFO	draw	LugNP	Dev. (%)
#1	Row	Row/Column	61	38	21	-0.49%
#2	Column	Row/Column	68	41	11	-0.98%
#3	Value	Row/Column	68	37	15	-1.18%
#4	Row	Square	41	36	43	1.48%
#5	Column	Square	57	43	20	-0.55%
#6	Value	Square	49	32	39	0.20%

The setup #3, which combined RF value-wise window and FO row/column-wise window, showed the best performance with an average deviation of -1.18% as well as 68 wins over LugNP. The combination of RF row-wise and column-wise with FO row/column-wise also returned improvement from LugNP. However, the FO with square approach seems to be better only when combined with RF column-wise. Thus, for the remainder of computational tests, we used setup #3.

Next, we present Figure 9, summarizing how the FO heuristic can improve the initial solution built by RF. It shows the average deviation of the solutions found by RF and RFFO from LugNP and Aheur for SET1 to SET4. RFFO was executed with the parameters values and window types discussed earlier. RF on its own returns on average solutions with less quality compared to the benchmark methods. However, the FO improves these initial solutions significantly for all of these four test sets, in particular for SET3 and SET4.

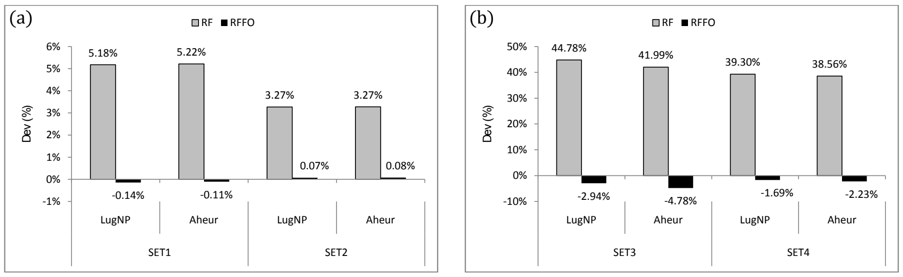


Figure 9: Comparing RFFO with RF: (*a*) SET1 and SET2, (*b*) SET3 and SET4

Finally, we discuss the results comparing the proposed RFFO approach against Aheur and LugNP, as summarized in Table 2. Regarding average percentage improvement, the results for SET1 and SET2 are not necessarily improved by RFFO, where it achieved almost the same performance as the benchmark methods with average deviations around 0.0%. This can be also seen by the high number of draws, but RFFO was able to return better solutions than LugNp and Aheur for both SET1 and SET2. On the other hand, RFFO outperforms the two benchmark methods in SET3 and SET4, achieving more than 4% and 2% of average improvement, respectively. This is quite significant, since these sets include the most challenging instances. Considering the number of better final solution values, our proposed framework outperformed benchmark approaches for more than 20 out of 30 instances in each of the sets SET3 and SET4.

Table 2: Number of better solutions and deviation values by set

	RFFO vs. Aheur				RFFO vs. LugNP			
Set	RFFO	draw	Aheur	Dev. (%)	RFFO	draw	LugNP	Dev. (%)
SET1	8	21	1	-0.11%	10	19	1	-0.14%
SET2	9	15	6	0.08%	10	14	6	0.07%
SET3	27	0	3	-4.78%	22	0	8	-2.94%
SET4	28	2	0	-2.23%	26	4	0	-1.69%

Next we discuss detailed results for each data set, where tables with detailed results for all instances are provided in the Appendix. We start with Tables 4 and 5 showing results for SET1 and SET2 instances, respectively. We also provide the root node lower bounds with (ℓ, S) inequalities of [3], shown as XLP, to indicate the computational complexity of the instances. It can be noticed that deviations are low and most of them are draws, with several deviations between 1.00% and 0.00%, explaining the low average improvement. Compared to Aheur and LugNP, SET1 has the most positive deviation of 0.24% and the most negative deviation (best improvement) of -0.80% from Aheur and of -1.83% from LugNP, respectively. In SET2, it is worth to note that there are more negative than positive values, but the high positive deviations for SET2_23, SET2_1 and SET2_7 are the main reasons for the positive average deviation reported in Table 2. Our computational experience is that the instances in SET1 and SET2 are quite easy to solve in general, and therefore harder to improve, most likely because the results from literature are already very good and close to optimality. This can be verified by the results and comparisons carried on [2] and [30] to support the performance of Aheur and LugNP against B&C.

Table 6 shows the results for SET3, where the results dominantly indicate negative deviations, reaching -11.77% from Aheur and -9.18% from LugNP for the instances SET3_21 and SET3_16, respectively. Another important remark to make is that RFFO improves Aheur and LugNP solutions significantly (more than 5%) for 15 and 9 instances, respectively, whereas the worst performance for RFFO is below 3.1% compared to these two benchmarks (in case of SET3_14, with 1.14% against Aheur and 3.06% against LugNP in SET3_29). This is important since SET3 includes hardest to solve instances in these problems.

Table 7 summarizes the results for SET4. The results are in line with the results of SET3, indicating noticeable negative deviations (though slightly less significant compared to SET3). There is no considerable positive deviation with 2 and 4 results considered draws, respectively, against Aheur and LugNP. Negative deviations reach -8.53% against Aheur and -5.66% against LugNP, whereas RFFO improves Aheur and LugNP solutions more than 3% for 8 and 7 instances, respectively.

Finally, we present in Figure 10 computational performance of different methods (including default Cplex) with extended computational times, where the average value of the best solutions found by each method is given. All methods were executed for 10-fold time limits compared to our original time limits, i.e., $1,000$ seconds for SET1 instances (Figure 10(a)), $1,500$ seconds for SET02 instances (10(b)), and $3,000$ seconds for SET03 and SET04 instances (Figures 10(c) and 10(d)), respectively.

We have omitted some initial values (e.g., Cplex and AMH first values for SET02), as they were out of the scale of the graphs used and would have deteriorated the visualization otherwise. As the graphs indicate, RFFO finds high quality solutions quickly and only improve these solutions slightly during the extended times. Moreover, RFFO solutions over the extended times are only outperformed in SET02, albeit slightly, by LugNP and Aheur, where Aheur is able to do so only after

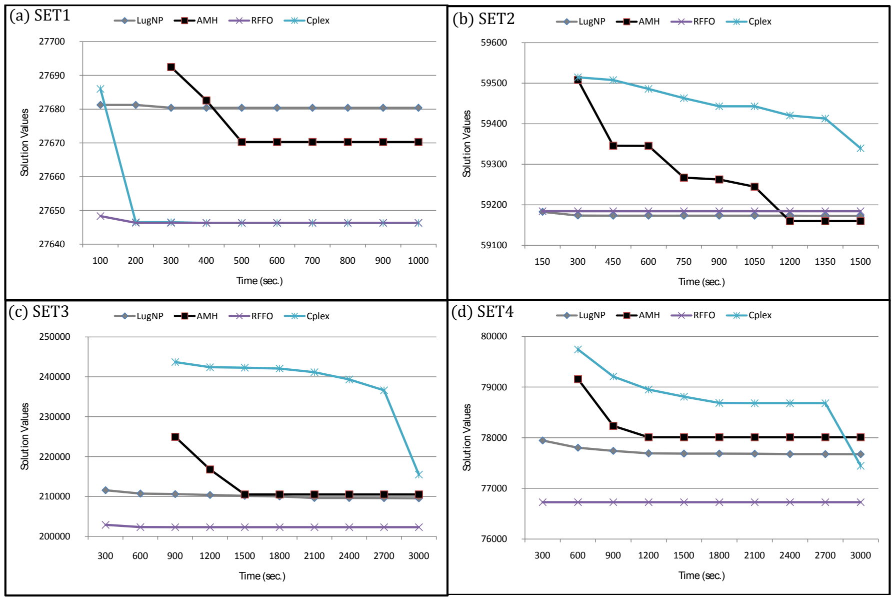


Figure 10: Computational performance of different methods with extended execution times.

1200 sec. RFFO is always better on average for all other sets showing a more stable performance when compared with the other methods, achieving these solutions very quickly.

4.2 Results for TGCPSP

The effectiveness of our method is now evaluated solving the TGCPSP, where one of the key differences compared to previous computational tests is that feasibility of the problems are not guaranteed, which was ensured with the backlogging to the last period in case of MLCLSP. We benchmark our results against those returned by the default Cplex solver and the hybrid genetic algorithm (HGA) of [27], which is a custom-designed method specifically for TGCPSP. HGA runs a genetic algorithm (GA) with several populations, where their individuals are hierarchically structured in trees, and integrated with simulated annealing (SA) and the so-called cavity heuristic (CV). SA is applied over the best individual found by the GA at each generation to intensify the search over its neighborhood. CV determines the number of mold cavities and, consequently, the efficiency of the machine during the production process of containers. HGA ran 10 times over each test problem within 1 hour, and the same time limit was spent by Cplex to solve each test problem using the model described in section 2.2. More details about the algorithm and parameters used can be found in [27].

The test problems, based on data provided by real-world glass container plants, are compounded by 150 *artificial* and 150 *real* problems. The artificial set is generated randomly in an academic fashion not necessarily representing a real-world scenario and it involves small to moderate size instances with $T \in \{7, 14\}$ days, $K \in \{1, 2, 3, 4, 5\}$ machines and $N \in \{5, 10, 20\}$ products per week. The real set corresponds to actual scenarios that happen in the glass container plants, where the production process involves $T \in \{14, 28, 56\}$ days with a number of products around 10 to 90 per week, and $K \in \{2, 3, 4, 5\}$ machines per furnace. For each set of problems, the type of solution returned by the Cplex solver within a 1 hour time limit is used to classify test problems as Optimal (solver returns an optimal solution), Feasible (solver returns a feasible solution without guaranteed optimality), and Unknown (solver does not return a feasible solution). Table 3 indicates some characteristics of these test sets as well as their subsets.

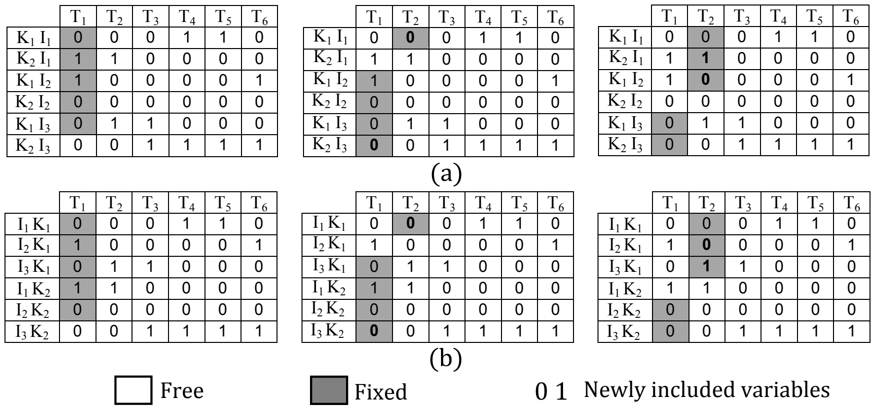
Table 3: TGCPSP problem instances

Artificial Test Problems				
			Average	
Type	Status	# instances	CPU(sec.)	Gap(%)
O0	Optimal	27	1.8	0.0
O1	Optimal	27	419	0.0
F0	Feasible	24	3600	3.2
F1	Feasible	24	3600	9.8
F2	Feasible	25	3600	17.1
U	Unknown	23	3600	-

Real Test Problems				
			Average	
Type	Status	# instances	CPU(sec.)	Gap(%)
O0	Optimal	3	1294	0.0
F0	Feasible	20	3600	9.1
F1	Feasible	21	3600	20.6
U	Unknown	106	3600	-

We recall that for the MLCLSP problem discussed in the previous section, RFFO optimized the setup variables w_{ft} combining RF with value-wise window and FO with row/column-wise window, meaning that FO first searches through rows (families f) and then through columns (periods t) in the two-dimensional data structure of w_{ft}. Since the setup variables Y_{itk} of the TGCPSP are three-dimensional, a further elaboration is necessary for the RFFO framework. This does not pose a problem in executing RF with value-wise window, but a strategy to execute FO needs to be adapted from the previous row/column-wise window. Based on our preliminary testing with various options, we concluded to execute FO following first the sequence product-machine-period and then machine-product-period as illustrated by Figure 11.

In Figure 11(a), the window includes variables selecting indexes by machines K_i first followed



	T_1	T_2	T_3	T_4	T_5	T_6
$K_1 I_1$	0	0	0	1	1	0
$K_2 I_1$	1	1	0	0	0	0
$K_1 I_2$	1	0	0	0	0	1
$K_2 I_2$	0	0	0	0	0	0
$K_1 I_3$	0	1	1	0	0	0
$K_2 I_3$	0	0	1	1	1	1

	T_1	T_2	T_3	T_4	T_5	T_6
$K_1 I_1$	0	0	0	1	1	0
$K_2 I_1$	1	1	0	0	0	0
$K_1 I_2$	1	0	0	0	0	1
$K_2 I_2$	0	0	0	0	0	0
$K_1 I_3$	0	1	1	0	0	0
$K_2 I_3$	0	0	1	1	1	1

	T_1	T_2	T_3	T_4	T_5	T_6
$K_1 I_1$	0	0	0	1	1	0
$K_2 I_1$	1	1	0	0	0	0
$K_1 I_2$	1	0	0	0	0	1
$K_2 I_2$	0	0	0	0	0	0
$K_1 I_3$	0	1	1	0	0	0
$K_2 I_3$	0	0	1	1	1	1

(a)

	T_1	T_2	T_3	T_4	T_5	T_6
$I_1 K_1$	0	0	0	1	1	0
$I_2 K_1$	1	0	0	0	0	1
$I_3 K_1$	0	1	1	0	0	0
$I_1 K_2$	1	1	0	0	0	0
$I_2 K_2$	0	0	0	0	0	0
$I_3 K_2$	0	0	1	1	1	1

	T_1	T_2	T_3	T_4	T_5	T_6
$I_1 K_1$	0	0	0	1	1	0
$I_2 K_1$	1	0	0	0	0	1
$I_3 K_1$	0	1	1	0	0	0
$I_1 K_2$	1	1	0	0	0	0
$I_2 K_2$	0	0	0	0	0	0
$I_3 K_2$	0	0	1	1	1	1

	T_1	T_2	T_3	T_4	T_5	T_6
$I_1 K_1$	0	0	0	1	1	0
$I_2 K_1$	1	0	0	0	0	1
$I_3 K_1$	0	1	1	0	0	0
$I_1 K_2$	1	1	0	0	0	0
$I_2 K_2$	0	0	0	0	0	0
$I_3 K_2$	0	0	1	1	1	1

(b)

Figure 11: Fix-and-Optimize: (a) machine-product-period and (b) product-machine-period. Three iterations of each direction are shown with window size = 5 and overlap rate = 60%.

for items I_i (products) and periods T_i. After to optimize on this way, the window in this three-dimensional data-structure selects variables indexes in Figure 11(b) by items followed for machines and periods.

We have executed RFFO for each test instance within the same time limit of 1 hour, where we use the initial parameter settings presented in the previous section. First of all, to test the flexibility of our approach, we executed RFFO on all the "Unknown" instances that were identified by the default Cplex, which correspond to 15.3% and 70.7% of the artificial and real test problems, respectively. In the same 1 hour limit, RFFO was able to find solutions for 56.5% and 20.8% of these unknown instances, respectively, achieving failure rates of 6.67% and 56% in the overall sets of artificial and real problems, respectively. Although the improvement over Cplex for the unknown artificial instances is significant, the unknown real instances still present a challenge, in particular due to their immense sizes and high number of binary variables (on average 3,329 for real problems). In addition, the involvement of general integer variables complicate these problems significantly and they were not specifically dealt within our RFFO framework in order to preserve the simple structure presented earlier for mixed binary problems. Moreover, the performance might also be affected by the fact that the accessibility of a feasible solution is less straightforward compared to MLCLSP with backlogging, where the simple solution of zero production and backlogging total demand to the last period is always feasible (but costly). We are currently investigating these areas more thoroughly as needed and plan to address these challenges in our future research outcomes.

In order to evaluate the solution quality RFFO can achieve for TGCPSP, we have next executed RFFO for all test instances that are not "Unknown". Using the same time limit of 1 hour as Cplex and HGA, we present our computational results executing RFFO with the initial parameter values

($RFFO^d$) and with the better parameter setting for the benchmark instances of the MLCLSP ($RFFO$). Thus, the idea here is to evaluate the performance of RFFO running with the initial values empirically obtained as well as with those parameter values customized to solve MLCLSP instances.

In Figure 12, we compare all methods for the five subsets of artificial test problems involving 127 instances, where the gap (%) is calculated by $Gap(\%) = (UpperBound - LowerBound)/(UpperBound)$ using the best solution obtained by each method for $UpperBound$ and the lower bound returned by the branch & cut algorithm of Cplex for $LowerBound$.

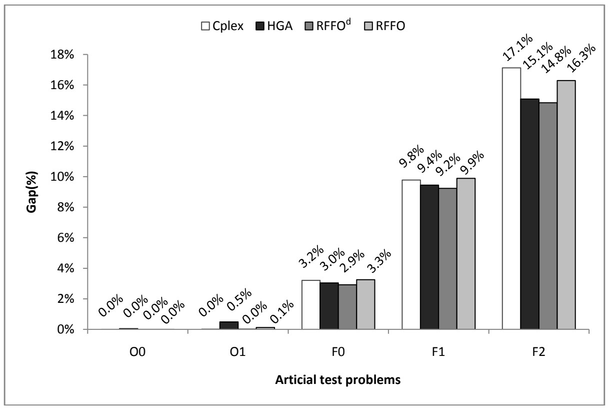


Figure 12: GAP(%) comparisons for artificial test problems.

As Figure 12 indicates, $RFFO^d$ was able to find optimal solutions in the set of instances, whose optimal values were returned by Cplex ($O0$ and $O1$), whereas HGA and $RFFO$ could not return optimal solutions for all test problems belonging to set $O1$ with average gaps 0.5% and 0.1%, respectively. For the other three subsets of problems, where only feasible solutions were returned by Cplex, all methods had similar performance with regards to solution quality, but $RFFO^d$ managed to achieve consistently the lowest average gap value. This is promising, in particular considering that HGA is a custom-designed method for these problems.

In Figure 13, we present the same evaluation for three subsets of real test problems involving 44 instances. Based on the results obtained by $RFFO^d$ for the artificial test problems, we have also implemented a slightly modified version of $RFFO^d$, named $\overline{RFFO}$, where the order to optimize variables in the FO is changed. In this case, the setup variables Y_{itk} are optimized by FO following first the sequence machine-product-period and then the sequence product-machine-period. Similar to artificial instances, the three RFFO versions were able to find optimal solutions for the real instances, for which Cplex could return their optimal values, while HGA did not manage to find the optimal solution for all instances of this set $O0$. For the two feasible sets, $RFFO^d$ and $RFFO$ return

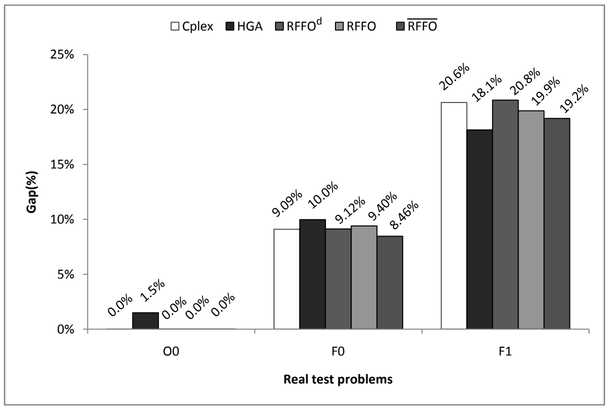


Figure 13: GAP(%) comparisons for real test problems.

almost the same average gap value as Cplex, whereas the modified version $\overline{RFFO}$ outperforms Cplex and HGA for the set $F0$ and outperforms Cplex for set $F1$. On the other hand, HGA generates better solutions on average than RFFO for the set $F1$, but our results are competitive, considering that HGA is specifically designed for these problems. As the experimentation with $\overline{RFFO}$ indicated, we remark that other changes in RFFO parameters can potentially improve its performance as it was done for the benchmark set of MLCLSP. However, as a general framework, it works effectively. Finally, we note that RFFO is currently designed to optimize only binary variables, but a more sophisticated RFFO framework could handle general integer variables of the TGCPSP more efficiently, which we plan to address in near future.

5. Conclusion

A hybrid method, RFFO, was proposed by combining two well-known heuristics, relax-and-fix (RF) and fix-and-optimize (FO). A simple combination is proposed, where RF is used to build an initial solution which is further improved by the FO in available computational time. The RFFO is applied to the Multi Level Capacitated Lot-Sizing Problem (MLCLSP) with backlogging and Two-stage Glass Container Production Scheduling Problem (TGCPSP). Using various test problems available from the literature, the proposed method was benchmarked to state-of-the-art methods from literature: Aheur of [2] and LugNP of [30] for MLCLSP, which are also heuristics based on mathematical programming, and HGA of [27] for TGCPSP, which is a genetic algorithm.

In the proposed approach, both heuristics use mathematical programming to solve mixed-integer subproblems defined by a certain amount of binary variables. These variables define a window that moves in the solution matrix using different orientations. Also, the number of binary variables under the window is increased if the solution is not sufficiently improved in a single execution of the FO.

Different strategies to traverse the matrix optimizing the binary variables were proposed and tested, where the best one reported combines a value-wise RF with row/column-wise FO. Thus, the results reported indicate better initial solutions returned by RF when the optimization is focused on relaxed variables closer to 0.5, followed by FO working better trying to optimize separately rows and columns oriented variables. The best setup found allowed the proposed method significantly outperform the benchmark approaches in two out of four test sets of MLCLSP. However, it was also able to return competitive results in the other two sets. More importantly, the results indicate a better performance of RFFO in the more complex test instances of SET3 and SET4. Similarly, three configuration of RFFO were also able to return competitive results for the more sophisticated problem of TGCPSP, outperforming default Cplex regularly and obtaining better or comparable results with HGA, which is a fast and efficient custom-built method for these problems. We believe that RFFO is overall an effective method for lot-sizing problems with varying characteristics.

As future work, we plan to conduct extensive computational testing on different combinations of parameter values. This would give better insight into sensitivity of different sizes for the MIP's solved by RF and FO, as well as different overlap rate values. We are also currently investigating combining RF and FO with other meta-heuristics. For instance, RF could be used to provide different initial solutions if a random criteria is incorporated in the value-wise strategy. Also the proposed FO heuristic could be applied as local search to improve better solutions found by other meta-heuristics. Another area to investigate is the potential improvement of the method if it exploited the specific problem structure. We plan to study this for different settings, e.g., for overtime.

Finally, we note that the design proposed in this paper is generic and problem-independent. To verify its robustness, we plan to extend this approach to more general MIP problems that naturally have a sequential decision making structure, including problems with general integer variables. In this case, it is in particular our special interest to investigate MIP problems where the RF heuristic could fail to determine an initial solution. Thus, another construction heuristics could be applied taking advantage from the partial solution provided by the proposed RF. We are currently investigating some crew scheduling problems with this framework.

Acknowledgements

We would like to thank Dr. Tao Wu for providing us the code of LugNP heuristics used in the comparisons. The work conducted by the first three authors was supported by Fundação de Amparo e

Pesquisa do Estado de São Paulo (FAPESP) projects 2010/10133-0, 2011/15534-5 and 2011/15581-3 and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) projects 483474/2013-4 and 312967/2014-4.

References

[1] N. Absi, B. Detienne, and S. Dauzère-Pérès. Heuristics for the multi-item capacitated lot-sizing problem with lost sales. *Computers & Operations Research*, 40(1):264 – 272, 2013.

[2] K. Akartunalı and A.J. Miller. A heuristic approach for big bucket multi-level production planning problems. *European Journal of Operational Research*, 193(2):396–411, 2009.

[3] K. Akartunalı and A.J. Miller. A computational analysis of lower bounds for big bucket production planning problems. *Computational Optimization and Applications*, 53(3):729–753, 2012.

[4] B. Almada-Lobo, D. Klabjan, M.A. Carravilla, and J.F. Oliveira. Multiple machine continuous setup lotsizing with sequence-dependent setups. *Computational Optimization and Applications*, 47(3):529–552, 2010.

[5] C. Almeder. A hybrid optimization approach for multi-level capacitated lot-sizing problems. *European Journal of Operational Research*, 200:599–606, 2010.

[6] M.F. Baki, B.A. Chaouch, and W. Abdul-Kader. A heuristic solution procedure for the dynamic lot sizing problem with remanufacturing and product recovery. *Computers & Operations Research*, 43(0):225 – 236, 2014.

[7] M.O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16(1):21 – 38, 2011.

[8] I. Barany, T. J. van Roy, and L. A. Wolsey. Uncapacitated lot sizing: The convex hull of solutions. *Mathematical Programming Study*, 22:32–43, 1984.

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

[10] P.J. Billington, J.O. McClain, and L.J. Thomas. Heuristics for multilevel lot-sizing with a bottleneck. *Management Science*, 32:989–1006, 1986.

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

[12] G. D. Eppen and R. K. Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.

[13] A. Federgruen, J. Meissner, and M. Tzur. Progressive interval heuristics for multi-item capacitated lot sizing problem. *Operations Research*, 55(3):490–502, 2007.

[14] M. Florian, J. K. Lenstra, and H. G. Rinnooy Kan. Deterministic production planning: Algorithms and complexity. *Management Science*, 26(7):669–679, 1980.

[15] S. Helber and F. Sahling. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, 123:247–256, 2010.

[16] S. Kébé, N. Sbihi, and B. Penz. A lagrangean heuristic for a two-echelon storage capacitated lot-sizing problem. *Journal of Intelligent Manufacturing*, 23(6):2477–2483, 2012.

[17] J. Krarup and O. Bilde. *Plant location, set covering and economic lotsizes: An O(mn) algorithm for structured problems. Optimierung bel Graphentheoretischen und Ganzzahligen Probleme*, pages 155–180. BirkhauserVerlag, 1997.

[18] S. Küçükyavuz and Y. Pochet. Uncapacitated lot-sizing with backlogging: The convex hull. *Mathematical Programming*, 118(1):151–175, 2009.

[19] A.J. Miller, G.L. Nemhauser, and M.W.P. Savelsbergh. On the polyhedral structure of a multi-item production planning model with setup times. *Mathematical Programming*, 94(2-3):375–405, 2003.

[20] MIPLIB. A library of pure and mixed integer problems. `http://miplib.zib.de/`. Last accessed on 29/12/2014, 2010.

[21] Multi-LSB. Multi-item lot-sizing problems with backlogging: A library of test instances. DOI: 10.15129/252b7827-b62b-4af4-8869-64b12b1c69a1, `http://dx.doi.org/10.15129/252b7827-b62b-4af4-8869-64b12b1c69a1`. Last accessed on 29/12/2014, 2014.

[22] Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2006.

[23] R. Ramezanian and M. Saidi-Mehrabad. Hybrid simulated annealing and mip-based heuristics for stochastic lot-sizing and scheduling problem in capacitated multi-stage production system. *Applied Mathematical Modelling*, 37(7):5134 – 5147, 2013.

[24] R.L. Rardin and L.A. Wolsey. Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems. *European Journal of Operational Research*, 71(1):95–109, 1993.

[25] F. Seeanner, B. Almada-Lobo, and H. Meyr. Combining the principles of variable neighborhood decomposition search and the fix&optimize heuristic to solve multi-level lot-sizing and scheduling problems. *Computers & Operations Research*, 40(1):303–317, January 2013.

[26] H. Stadtler. Multilevel lot sizing with setup times and multiple constrained resources: Internally rolling schedules with lot-sizing windows. *Operations Research*, 51:487–502, 2003.

[27] C.F.M. Toledo, M. da Silva Arantes, R.R.R. de Oliveira, and B. Almada-Lobo. Glass container production scheduling through hybrid multi-population based evolutionary algorithm. *Applied Soft Computing*, 13(3):1352 – 1364, 2013.

[28] C.F.M. Toledo, R.R.R. de Oliveira, and P.M. França. A hybrid multi-population genetic algorithm applied to solve the multi-level capacitated lot sizing problem with backlogging. *Computers & Operations Research*, 40(4):910 – 919, 2013.

[29] M. Van Vyve and Y. Pochet. A general heuristic for production planning problems. *INFORMS Journal on Computing*, 16(3):316–327, 2004.

[30] T. Wu, L. Shi, J. Geunes, and K. Akartunalı. An optimization framework for solving capacitated multi-level lot-sizing problems with backlogging. *European Journal of Operational Research*, 214(2):428–441, 2011.

A. Appendix

Table 4: Comparison for SET1 instances (time limit = 100s)

		Solution values			Deviation (%)	
SET1	XLP	Aheur	LugNP	RFFO	Aheur	LugNP
1	17,888	22,382.5	22,460.7	22,382.1	0	-0.35
2	23,534	27,584.8	27,584.8	27,584.6	0	0
3	21,227	25,187.3	25,187.3	25,246.6	0.24	0.24
4	22,232	26,334.7	26,334.7	26,334.7	0	0
5	21,446	25,145.5	25,145.5	25,145.8	0	0
6	22,974	26,667.4	26,770.8	26,667.5	0	-0.39
7	20,360	24,123.8	24,123.8	24,124.2	0	0
8	25,582	29,640.4	29,640.4	29,639.8	0	0
9	16,321	20,971.2	21,362.7	20,971.0	0	-1.83
10	17,998	22,645.8	22,647.5	22,562.8	-0.37	-0.37
11	11,080	12,955.6	12,955.6	12,955.3	0	0
12	24,721	26,831.3	26,831.3	26,831.1	0	0
13	20,782	23,127.8	23,127.8	23,128.5	0	0
14	22,264	25,035.8	25,035.8	25,036.0	0	0
15	12,401	14,118.1	14,118.1	14,117.9	0	0
16	15,122	17,540.2	17,400.1	17,400.1	-0.80	0
17	20,468	23,007.5	23,007.5	22,996.2	-0.05	-0.05
18	11,075	12,973.8	12,973.8	12,973.8	0	0
19	13,276	16,502.9	16,502.9	16,349.2	-0.93	-0.93
20	14,101	17,158.6	17,158.6	17,158.7	0	0
21	10,159	12,421.2	12,421.2	12,421.1	0	0
22	38,040	40,158.3	40,188.7	40,158.4	0	-0.08
23	29,331	30,605.7	30,605.7	30,605.5	0	0
24	28,858	32,190.4	32,145.5	32,007.2	-0.57	-0.43
25	51,371	52,989.2	52,959.9	52,960.3	-0.05	0
26	39,379	41,221.5	41,221.5	41,221.0	0	0
27	40,838	43,319.7	43,319.7	43,289.6	-0.07	-0.07
28	39,846	40,993.5	41,019.8	40,993.5	0	-0.06
29	23,155	25,492.6	25,322.3	25,322.0	-0.67	0
30	68,989	70,863.7	70,863.7	70,863.7	0	0

Table 5: Comparison for SET2 instances (time limit = 150s)

		Solution values			Deviation (%)	
SET2	XLP	Aheur	LugNP	RFFO	Aheur	LugNP
1	46,116	52,050.7	52,050.7	52,339.4	0.55	0.55
2	47,780	53,863.4	53,713.4	53,713.0	-0.28	0
3	40,551	46,894.5	47,053.2	46,893.3	0	-0.34
4	36,347	43,009.8	42,977.1	43,063.1	0.12	0.20
5	45,395	51,757.6	51,757.6	51,768.8	0.02	0.02
6	45,902	51,858.1	51,858.1	51,858.4	0	0
7	52,825	58,153.8	58,153.8	58,425.2	0.47	0.47
8	48,033	54,396.2	54,449.6	54,182.9	-0.39	-0.49
9	37,553	43,737.8	43,737.8	43,690.0	-0.11	-0.11
10	38,751	45,278.8	45,278.8	45,305.8	0.06	0.06
11	65,210	68,488.8	68,646.4	68,487.8	0	-0.23
12	62,792	66,561.9	66,474.5	66,475.4	-0.13	0
13	34,778	39,120.3	39,082.7	38,852.7	-0.68	-0.59
14	62,907	66,373.7	66,383.2	66,325.1	-0.07	-0.09
15	59,079	61,574.1	61,574.1	61,574.0	0	0
16	75,682	79,364.8	79,385.0	79,363.9	0	-0.03
17	36,809	41,298.6	41,282.4	41,192.6	-0.26	-0.22
18	77,873	81,561.8	81,562.9	81,562.5	0	0
19	54,981	58,426.1	58,426.1	58,425.4	0	0
20	119,568	122,827.6	122,827.6	122,829.0	0	0
21	22,281	24,013.2	24,014.2	24,013.3	0	0
22	51,279	52,887.1	52,887.1	52,886.8	0	0
23	29,793	32,618.2	32,708.8	33,713.9	3.36	3.07
24	65,891	68,640.6	68,575.1	68,574.8	-0.10	0
25	75,627	78,064.3	78,088.2	78,064.2	0	-0.03
26	60,952	63,275.2	63,285.6	63,273.2	0	-0.02
27	53,016	54,794.1	54,794.1	54,793.9	0	0
28	44,545	46,607.9	46,607.9	46,607.6	0	0
29	93,631	96,278.0	96,157.4	96,152.0	-0.13	0
30	68,324	71,408.0	71,408.0	71,408.7	0	0

Table 6: Comparison for SET3 instances (time limit = 300s)

		Solution values			Deviation (%)	
SET3	XLP	Aheur	LugNP	RFFO	Aheur	LugNP
1	65,668	188,294.0	189,400.6	179,554.0	-4.64	-5.20
2	82,342	216,700.4	217,283.4	216,401.0	-0.14	-0.41
3	74,209	216,517.4	207,362.6	198,215.0	-8.45	-4.41
4	78,282	214,175.7	220,062.4	203,208.0	-5.12	-7.66
5	76,607	220,928.0	220,686.4	201,723.0	-8.69	-8.59
6	79,093	213,987.2	210,339.0	203,253.0	-5.02	-3.37
7	72,979	206,793.3	208,245.8	193,804.0	-6.28	-6.93
8	88,610	231,333.9	224,404.5	226,042.0	-2.29	0.73
9	64,180	198,594.1	183,327.9	178,576.0	-10.08	-2.59
10	66,878	201,771.0	192,069.0	188,790.0	-6.43	-1.71
11	42,946	132,466.6	130,055.9	132,231.0	-0.18	1.67
12	86,047	213,445.5	211,726.2	195,981.0	-8.18	-7.44
13	74,643	199,471.6	197,240.0	195,772.0	-1.85	-0.74
14	85,209	198,005.1	200,193.9	200,257.0	1.14	0.03
15	40,715	135,491.1	125,875.5	127,045.0	-6.23	0.93
16	46,548	144,580.2	149,411.0	135,689.0	-6.15	-9.18
17	71,555	200,971.1	199,875.3	184,830.0	-8.03	-7.53
18	39,533	98,901.8	97,031.1	98,106.0	-0.80	1.11
19	47,495	149,973.9	151,618.8	138,420.0	-7.70	-8.71
20	58,189	170,524.4	163,785.9	163,740.0	-3.98	-0.03
21	44,182	141,578.2	134,625.9	124,919.0	-11.77	-7.21
22	130,235	256,283.6	245,549.4	246,270.0	-3.91	0.29
23	96,810	229,468.8	215,893.6	209,798.0	-8.57	-2.82
24	105,300	272,965.6	245,491.9	241,071.0	-11.68	-1.80
25	203,044	329,382.0	333,236.6	324,800.0	-1.39	-2.53
26	145,184	286,229.0	289,459.6	280,060.0	-2.16	-3.25
27	145,420	294,614.0	297,025.5	286,754.0	-2.67	-3.46
28	145,227	225,567.2	224,734.0	227,483.0	0.85	1.22
29	79,813	189,879.7	185,569.7	191,242.0	0.72	3.06
30	274,018	415,185.0	407,150.8	399,907.0	-3.68	-1.78

Table 7: Comparison for SET4 instances (time limit = 300s)

		Solution values			Deviation (%)	
SET4	XLP	Aheur	LugNP	RFFO	Aheur	LugNP
1	16,353	57,483.0	53,168.4	53,062.3	-7.69	-0.20
2	31,541	80,772.8	77,346.7	73,884.2	-8.53	-4.48
3	24,864	68,176.9	67,097.7	66,030.7	-3.15	-1.59
4	27,786	72,989.4	68,995.1	68,662.6	-5.93	-0.48
5	25,450	67,329.1	66,993.7	66,328.8	-1.49	-0.99
6	30,632	75,042.4	74,601.8	70,698.3	-5.79	-5.23
7	22,650	62,993.3	64,132.5	62,974.3	-0.03	-1.81
8	40,532	81,200.6	84,586.1	80,914.5	-0.35	-4.34
9	13,490	55,901.5	52,041.0	51,457.5	-7.95	-1.12
10	15,542	55,602.2	57,297.3	55,341.7	-0.47	-3.41
11	12,802	28,415.7	28,323.5	28,207.1	-0.73	-0.41
12	43,341	73,653.4	72,084.8	71,886.9	-2.40	-0.27
13	28,152	52,525.0	55,251.9	52,518.4	-0.01	-4.95
14	56,174	79,086.4	80,501.7	78,903.9	-0.23	-1.98
15	14,628	25,927.5	25,286.3	24,568.9	-5.24	-2.84
16	17,171	35,048.5	35,138.7	34,569.2	-1.37	-1.62
17	29,001	51,396.2	51,671.9	51,266.5	-0.25	-0.78
18	19,184	26,101.5	26,282.3	26,037.4	-0.25	-0.93
19	10,724	31,585.8	33,006.4	31,139.0	-1.41	-5.66
20	18,718	38,796.1	38,781.4	37,179.1	-4.17	-4.13
21	15,812	25,727.0	25,840.8	25,713.0	-0.05	-0.49
22	91,715	120,008.2	119,481.0	118,749.0	-1.05	-0.61
23	55,058	74,180.4	73,297.4	73,296.6	-1.19	0
24	58,919	82,349.4	82,260.2	80,733.2	-1.96	-1.86
25	171,987	196,626.7	196,025.1	196,023.0	-0.31	0
26	110,570	137,224.6	134,856.0	134,854.0	-1.73	0
27	101,114	135,936.6	132,463.3	132,451.0	-2.56	0
28	112,892	126,553.7	126,157.1	125,872.0	-0.54	-0.23
29	51,149	66,131.1	66,217.4	66,131.4	0	-0.13
30	241,678	262,380.7	263,042.1	262,378.0	0	-0.25