Tendências em Matemática Aplicada e Computacional, **18**, N. 3 (2017), 515-529

www.scielo.br/tema
doi: 10.5540/tema.2017.018.03.0515

A New Branching Rule to Solve the Capacitated Lot Sizing and Scheduling Problem with Sequence Dependent Setups

W.A. DE OLIVEIRA[1*] and M.O. SANTOS[2]

Received on December 26, 2016 / Accepted on July 31, 2017

ABSTRACT. In this paper, we deal with the Capacitated Lot Sizing and Scheduling Problem with sequence dependent setup times and costs – CLSD model. More specifically, we propose a simple reformulation for the CLSD model that enables us to define a new branching rule to be used in Branch-and-Bound (or Branch-and-Cut) algorithms to solve this NP-hard problem. Our branching rule can be easily implemented in commercial solvers. Computational tests performed in 240 test instances from the literature show that our approach can significantly reduce the running time to solve this problem using a Branch-and-Cut algorithm of a commercial MIP solver. Therefore, our approach can also improve the performance of other approaches that need to solve partial sub problems of the CLSD model in each iteration, such as Lagrangian approaches and heuristics based on the mathematical formulation of the problem.

Keywords: Lot Sizing, scheduling, Mixed Integer Linear Programming, Branch-and-Bound algorithm.

1 INTRODUCTION

In most production environments, companies need to decide on the size of production lots in order to obtain efficient inventory management and reduce costs. High inventory levels cause high holding costs and low inventory levels may cause undesirable delays in meeting customer demands.

The lot sizing problem (LP) consists of determining the optimal size of production lots with the aim to minimize costs and meet customer demands. The LP has received special attention from researchers due to its importance for the global economy ([10]).

On the other hand, the scheduling problem consists of determining the sequence of production lots in order to minimize the time and cost generated by product changeovers on production lines. When the cost and time generated by product changeovers depends on previously produced items

*Corresponding author: Willy Alves de Oliveira – E-mail: waosoler@gmail.com
[1] Instituto de Matemática – INMA, Universidade Federal de Mato Grosso do Sul – UFMS, 79070-900 Campo Grande, MS, Brasil.
[2] Departamento de Matemática Aplicada e Estatística – ICMC, Universidade de São Paulo – USP, Av. Trabalhador São-Carlense, 400, 13566-590 São Carlos, SP, Brasil . E-mail: mari@icmc.usp.br

and the item to be produced, it can be said that there is a sequence-dependent setup time and/or cost structure.

According to [1], when there is a sequence-dependent setup time/cost structure, the decisions about the size of production lots and the sequence of production need to be taken simultaneously, because the solution obtained by an hierarchical approach may be infeasible or suboptimal. Therefore, the simultaneous lot sizing and scheduling problem (LSP) consists of simultaneously deciding the sizes and production sequences.

In the literature, there are various mathematical models to deal with LSP. We highlight the capacitated lot sizing and scheduling with sequence dependent setups – CLSD model ([13]), the general lot sizing and scheduling problem – GLSP model ([9], [16]) and the reformulation of GLSP model proposed in [5] – CC model. Recent reviews of the models to deal with LSP are presented in [12, 1, 6].

In [12], the authors compare various mathematical models for LSP and by using theoretical and computational results, it could be concluded that the CLSD model is a promising formulation to deal with LSP. The CLSD model has an interesting performance in exact solution approaches such as Brand-and-Bound algorithms from commercial MIP solvers.

In this paper, we introduce a very simple reformulation for the CLSD model (the CLSD^w model) that allows us to introduce a new branching rule able to significantly improve the computational performance of this model in Branch-and-Bound (Branch-and-Cut) algorithms. Therefore, our approach can be used to improve the performance of commercial solvers and approaches to deal with LSP that need to solve partial sub problems, such as Lagrangian approaches and MIP based heuristics.

We use a set of 240 test instances from the literature to compare the performance of the traditional CLSD with our approach in a commercial powerful solver Cplex 12.60. Computational results show that our approach can significantly improve the performance of the solver Cplex, reducing running time and proving optimality for more test instances. This paper is organized as follows: Section 2 presents a literature review for LSP; Section 3 presents the mathematical formulation CLSD^w and the branching rule and Section 4 presents the computational results comparing the performance of the traditional CLSD model with our approach in the Branch-and-Cut algorithm from Cplex 12.60. Finally, the conclusions and future proposals are presented in Section 5. Details of the computational implementation are given in the appendix.

2 LITERATURE REVIEW

[9] introduced the GLSP model to deal with LSP. The GLSP model considered several items to be produced on a single machine (production line) with dynamic deterministic demand and sequence-dependent setup costs and no setup times. This model is based on the idea that consists of splitting to split each period into several micro periods (with varying sizes) where only one item can be produced. Therefore, by determining which items will be produced in each micro period, the production lot schedules can then be automatically determined.

The original GLSP model was reformulated in [20] using a network flow problem structure and in [5] suppressed the setup state variables in the model. Computational tests performed in [12] showed that both reformulations can provide better dual bounds by solving linear relaxation than the dual bounds obtained by solving the linear relaxation of the GLSP model.

In [13], the CLSD model was introduced using another strategy, consisting of introducing constraints and variables from the travelling salesman problem to map the start time and end time of production of each item in each period, to model the sequencing decisions. The CLSD model considers, originally, a single-stage system where several items have to be produced on a single machine in a finite planning horizon supposing a known dynamic deterministic demand which must be completely satisfied without backlogging.

Mixed integer programming models based on the CLSD model have been proposed to deal with problems from various real world production environments, such as [15], which addressed a yogurt industry and [21], which studied a semiconductor assembly and test manufacturing.

In other papers, extensions of the GLSP model were compared with the extension of the CLSD model to deal with different real problems. For example, [2] addressed the LSP on parallel production lines that need to be equipped with tools for processing. Models inspired by GLSP and CLSD were developed to synchronize these tools on the production lines. Computational tests showed that the CLSD model has a much better performance than the GLSP model considering the runtime and the ability to find feasible solutions.

Computational tests in [12] showed that the CLSD model performs better than all the models that use micro period structures. In particular, the CLSD model presented a lower average deviation from the best known solution (GAP) and running time than other models.

The LSP is a *NP-complete* problem ([9, 16, 17]) where instances based on real world problems can be difficult to solve by exact algorithms at an acceptable computational time. Therefore, various heuristic approaches have been developed to deal with the LSP. For example, [13] proposed a backward oriented heuristic to solve the LSP model without setup times, while [17] proposed a threshold accepting methaheuristic to deal with LSP considering sequence dependent setup times.

Heuristics based on the mathematical formulation of the problem (MIP based heuristics) have been frequently used to solve the LSP, in particular, the *relax-and-fix – RF* and *fix-and-optimise – FO*. FO heuristics have obtained good solutions for various types of lot sizing and scheduling problems and various heuristics approaches combining RF and FO heuristics have been proposed in the literature, such as [3, 7, 8, 22, 19].

In the simplified case, the RF heuristic split the set of all binary variables (B) of the model into a finite number of small subsets ($B^r \subset B, r = 1, \ldots, R$) and, in each iteration $r \in \{1, \ldots, R\}$, the binary variables in the sets B^k with $k < r$ have their values fixed at the incumbent value (obtained from the previous iterations), while the binary variables in the sets B^l with $l > r$ are linearly relaxed and all variables without fixed values are optimized. In this way, in iteration r, a smaller mixed integer linear program is solved with the variables in the set B^r considered as binaries. Usually, RF heuristics are used to obtain an initial solution.

The FO is an improvement heuristic that starts in any feasible solution and in each iteration a subset of binary variables are re-optimized while the other binary variables have their values fixed on the value of the incumbent solution. For lot sizing and scheduling problems, the most used variable partitions are by periods, products and production lines. FO heuristics can also be used with an exact method (matheuristics). For example, [11] combined FO with column generation and [4] integrated FO with the *Variable Neighbourhood Search – VNS* methaheuristic.

Therefore, as MIP based heuristics solve, in each iteration, a small mixed integer linear programming model, these heuristics can be improved if a good model and a good exact solution algorithm are used to solve the sub problems in each iteration. Therefore, the reformulation and the branch rule proposed in this paper can improve the computational performance of some MIP based heuristics to deal with LSP.

3 MODEL FORMULATION AND SOLUTION APPROACH

The traditional CLSD model with setup times can be found in [12] and the following parameters and variables are used to define it:

Parameters:

- T: number of periods (indexed by t);
- J: number of items (indexed by i and j);
- d_{jt}: demand of item j in period t;
- C_t: available capacity time in period t;
- a_j: consumed capacity time for production of a unit of item j;
- h_j: inventory cost of item j;
- sc_{ij}: setup cost for exchange between items i and j;
- st_{ij}: setup time for exchange between items i and j;

Variables:

- I_{jt}: inventory of item j at the end of period t;
- x_{jt}: produced quantity of item j in period t;
- V_{jt}: production order of item j in period t;
- y_{js}: 1, if the item j is the first item produced in period t and 0, otherwise;
- z_{ijs}: 1, if there is an exchange between items i and j in period t and 0, otherwise.

The CLSD model is given by (3.1) to (3.10).

$$\text{Min} \quad \sum_{j=1}^{J}\sum_{t=1}^{T} h_j I_{jt} + \sum_{i=1}^{J}\sum_{j=1}^{J}\sum_{t=1}^{T} sc_{ijt} z_{ijt} \tag{3.1}$$

$$\text{subject to} \quad I_{j,t-1} + x_{jt} = d_{jt} + I_{jt}, \qquad \forall t, \tag{3.2}$$

$$\sum_{j=1}^{J} a_j x_{jt} + \sum_{i=1}^{J}\sum_{j=1}^{J} st_{ij} z_{ijt} \leq C_t, \qquad \forall t, \tag{3.3}$$

$$x_{jt} \leq \frac{C_t}{a_j}\left(y_{jt} + \sum_{i=1}^{J} z_{ijt}\right), \qquad \forall j, t, \tag{3.4}$$

$$\sum_{j=1}^{J} y_{jt} = 1, \qquad \forall t, \tag{3.5}$$

$$y_{jt} + \sum_{i=1}^{J} z_{ijt} = \sum_{i=1}^{J} z_{jit} + y_{j,t+1}, \qquad \forall j, \tag{3.6}$$

$$V_{jt} \geq V_{it} + 1 - J(1 - z_{ijt}), \qquad \forall i, j, t, \tag{3.7}$$

$$I_{jt}, V_{jt}, x_{jt} \geq 0, \qquad \forall j, t, \tag{3.8}$$

$$y_{jt} \in \{0, 1\}, \qquad \forall j, t, \tag{3.9}$$

$$z_{ijt} \in \{0, 1\}, \qquad \forall i, j, t. \tag{3.10}$$

The objective function (3.1) reflects the sum of holding costs and product changeover costs, while (3.2) are the inventory balance constraints and (3.3) are the capacity constraints. Constraints (3.4) ensure that item j can only be produced if the production line is set up for it. Constraints (3.5) ensure that only one item is the first produced item in each period, while constraints (3.6) trace the machine configurations. Constraints (3.7) are MTZ (Miller-Tucker-Zemlin) constraints to eliminate sub tours and, finally, constraints (3.8) -(3.10) define the domain of decision variables.

To introduce our branching rule, we firstly define a reformulation of the CLSD model called the CLSD^w model. Consider new binary variables w_{jt}, where $w_{jt} = 1$, if item j is produced in period t and $w_{jt} = 0$, otherwise. Clearly, we have that

$$w_{jt} = y_{jt} + \sum_{i=1}^{J} z_{ijt}, \quad \forall j, t. \tag{3.11}$$

Note that, by (3.11) if $w_{jt} = 1$, then the item j must be scheduled in period t, implying in a setup cost. Therefore, if $x_{jt} = 0$ then in the optimal solution we have that $w_{jt} = 0$.

The CLSD^w model can be obtained introducing constraints (3.11) and (3.13) in the CLSD model and replacing constraints (3.4) by constraints (3.12), where:

$$x_{jt} \leq \frac{C_t}{a_j} w_{jt}, \quad \forall j, t, \tag{3.12}$$

$$w_{jt} \in \{0, 1\}, \quad \forall j, t. \tag{3.13}$$

3.1 Branching rule

Note that by (3.11), if $w_{jt} = 0$, then $y_{jt} = 0$, $z_{ijt} = 0$ and $z_{jit} = 0$, $\forall i$. Therefore, if we can identify that item j is not produced in one period, we can fix directly the value of $2J + 1$ binary variables on zero.

This fact motivated us to introduce the binary variables w_{jt} in the CLSD model, obtaining the CLSD^w model, and performing a branch-and-bound algorithm with a priority of branching for these variables. Note that this reformulation increases the number of binary variables in $J * T$, however it enables us to perform a more efficient branching scheme in a branch-and-bound or branch-and-cut algorithm.

In each search tree node, given an optimal solution of linear relaxation, our branching rule is: if there are variables w_{jt} with no integer values, we firstly perform the branching in these variables before variables y_{jt} and z_{ijt}.

Figure 1 presents an example comparing a traditional branch-and-bound algorithm in the CLSD model and a branch-and-bound algorithm using the CLSD^w model with our branching rule. The instance considered in Figure 1 has 15 products and 5 periods and we notice that with just six nodes explored, the incumbent dual bound found with our branching rule is significantly better than the best dual bound found in a traditional branching scheme. In this test instance, the traditional branch-and-bound algorithm consumed around 56 seconds to solve the CLSD model to optimality, while the branch-and-bound algorithm with the branching rule introduced in this paper consumed only around 6 seconds to solve the CLSD^w model.

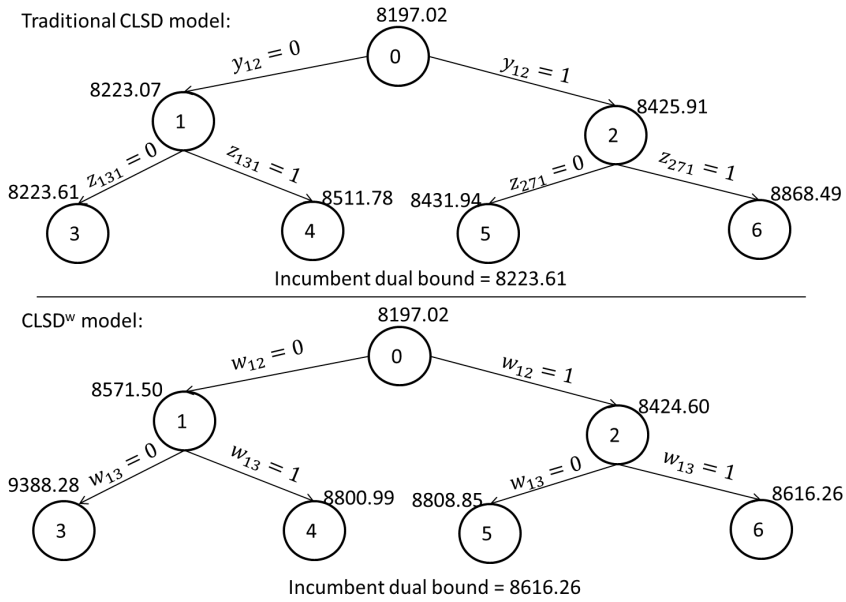

Figure 1: Representation of our branching rule.

We empirically observe that with some values of variables w_{jt} fixed in binary numbers, the value of linear relaxation for variables y_{jt} and z_{ijt} also tends to be binary. Clearly, if $w_{jt} = 0$, then $y_{jt} = 0$, $z_{ijt} = 0$ and $z_{jit} = 0, \forall i$ (see (3.11)). Consider now a simplified case where just two items can be produced in each period and suppose that in a given node, the values of variables w_{it} and w_{jt} were fixed in one for some i, j and t. Suppose that $st_{ij} < st_{ji}$ and $sc_{ij} < sc_{ji}$. Therefore, in the optimal solution of the linear relaxation in this node, we will have $y_{it} = 1$, $y_{jt} = 0$, $z_{ijt} = 1$ and $z_{jit} = 0$.

Our branching scheme has another advantage. It can be easily implemented using a commercial solver, and therefore, we can benefit from a general powerful branch-and-cut algorithm and various general heuristics to improve the convergence.

In Section 4, we present computational results to compare the performance of the traditional CLSD model with the performance of the CLSD^w model using our branching scheme implemented in the Cplex 12.60 solver on 240 test instances from the literature.

4 COMPUTATIONAL RESULTS

4.1 Test environment

We implemented the traditional CLSD model, the CLSD^w model and our branching rule in C++ language using the library Concert Technology of the Cplex 12.60 solver. We denote by CLSD$^w_{BR}$ the results of the model CLSD^w using our branching rule in the general Branch-and-Cut algorithm of the Cplex solver.

We ran the tests on a computer with two Intel Xeon processors, 2.8 GHz and 128 GB DDR3 RAM memory. The maximum running time was fixed to one hour (3600 seconds). For each instance, we captured the best feasible solution and the best dual bound found. The deviation of the best feasible solution from the lower bound (GAP) was computed as $GAP = 100 * \left(\frac{z^f - z^d}{z^f}\right)$, where z^f is the best feasible solution and z^d is the incumbent dual bound.

4.2 Test instance features and computational results

To test the performance of model CLSD^w with our branching rule, we used a set of 240 test instances from the literature, and compared it with the performance of traditional CLSD model. The test instances were presented in [14] and can be obtained in `http://www.mang.canterbury.ac.nz/people/rjames`. The test instances have the following features:

1. $J \in \{15, 25\}$, $T \in \{5, 10, 15\}$;
2. $h_j \in \{2, \ldots, 9\}$;
3. $d_{jt} \in \{40, \ldots, 59\}$;
4. $st_{ij} \in \{5, \ldots, 10\}$ and $sc_{ij} = \theta st_{ij}$, where θ is a positive parameter;
5. $a_j = 1$;
6. $C_t = \frac{\sum_j d_{jt}}{Cut}$, where $0 < Cut < 1$ is a parameter that defines the capacity utilization.

Another parameter $CutVar$ was introduced in order to control the amount of capacity variation. The parameter $CutVar$ represents and controls the maximum total allowed variation from Cut, and therefore the actual capacity can vary ([14]). The value for $CutVar$ was fixed to 0.5 for all the test instances as in [14].

The test instances were grouped into twenty four classes, with 10 test instances each class, according to the value of parameters J, T, Cut, $CutVar$ and θ. The values of parameters for each class and the results are given in Table 1 while the results grouped by number of products (J), number of periods (T) and capacity utilization (Cut) are presented in Table 2.

Table 1: Test instances features and computational results.

Class	J	T	Cut	θ	CLSD		Our approach	
					GAP	Time	GAP	Time
1	15	5	0.6	50	0.00	1.77	0.00	1.58
2	15	5	0.6	100	0.00	3.23	0.00	2.24
3	15	5	0.8	50	0.00	2.25	0.00	1.72
4	15	5	0.8	100	0.00	17.32	0.00	4.78
5	15	10	0.6	50	0.00	16.78	0.00	10.72
6	15	10	0.6	100	0.02	1199.49	0.00	167.24
7	15	10	0.8	50	0.00	53.96	0.00	17.74
8	15	10	0.8	100	0.65	3073.14	0.00	881.05
9	15	15	0.6	50	0.00	1061.70	0.00	38.27
10	15	15	0.6	100	1.12	3600.00	0.60	3600.00
11	15	15	0.8	50	0.03	910.96	0.00	115.94
12	15	15	0.8	100	2.25	3534.61	1.18	3600.00
13	25	5	0.6	50	0.00	16.39	0.00	16.84
14	25	5	0.6	100	0.00	22.88	0.00	17.47
15	25	5	0.8	50	0.00	34.19	0.00	18.43
16	25	5	0.8	100	0.00	569.32	0.00	77.59
17	25	10	0.6	50	0.05	2348.67	0.03	1379.36
18	25	10	0.6	100	2.12	3600.00	0.81	3600.00
19	25	10	0.8	50	0.19	2960.61	0.02	2091.43
20	25	10	0.8	100	2.95	3600.00	1.41	3600.00
21	25	15	0.6	50	0.01	731.67	0.00	495.04
22	25	15	0.6	100	0.72	3481.34	0.13	2811.65
23	25	15	0.8	50	0.01	1143.15	0.01	518.13
24	25	15	0.8	100	1.40	3600.00	0.50	3600.00
Average					0.39	1068.63	0.17	801.77

In Table 1, it can be observed that our approach was able to reduce the average GAP in 12 classes. Moreover, the obtained GAP by our approach does not increase in any class compared to the traditional approach. The general average GAP was reduced by around 56% (GAP_{CLSD}

Table 2: Results by parameters.

	GAP(%)		Time (sec)		Optimality	
	CLSD	$CLSD^w_{BR}$	CLSD	$CLSD^w_{BR}$	CLSD	$CLSD^w_{BR}$
J						
15	0.34	0.15	1041.25	703.44	91	100
25	0.62	0.24	1842.35	1518.83	66	76
T						
5	0.00	0.00	83.42	17.58	80	80
10	0.75	0.28	2106.58	1468.44	40	54
15	0.69	0.30	2135.40	1847.38	37	42
Cut						
0.6	0.34	0.13	1258.64	1011.70	84	91
0.8	0.62	0.26	1624.96	1210.57	73	85

= 0.39 and $GAP_{CLSD^w_{BR}}$ = 0.17), while the general average time was reduced by around 24% ($Time_{CLSD}$ = 1068.63 and $Time_{CLSD^w_{BR}}$ = 801.77).

The average time was reduced by 19 classes, remaining constant in 4 classes and increased in only one class (class 12). The increase is due to an instance from class 12 the random access memory limit (128 GB) was exceeded before reaching the time limit by the traditional CLSD model, and therefore, the running time was slightly reduced. The largest reduction in the average time occurred in classes 6, 11 and 16 where our approach reduces the mean running time by around 86%. In Table 2, it can be observed that our approach could prove optimality in 176 test instances while the traditional approach could prove optimality in 157 test instances.

Table 2 shows that when the number of products or periods increased, then the problem becomes more difficult to solve resulting in longer average deviations and running times, as well as a reduction in the number of instances considering optimality. This fact is not impressive, because the branch-and-cut algorithm grows exponentially when the number of decisions is increased.

Table 3 presents the number of nodes explored in the search and the number of iterations (including the cuts applied). We can observe that using our branching rule, the average number of nodes explored was reduced by around 39%, while the average number of iterations was reduced by around 6.9%.

The number of iterations performed in the $CLSD^w_{BR}$ approach was greater than in CLSD approach for classes 10, 12, 14, 18, 19, 20, 22 and 24. In theses classes (except for Class 14), there are instances in which the optimal solution was not found because there are residual GAPs. For this reason, the maximum running time was reached for some instances. Therefore, in the same running time, the $CLSD^w_{BR}$ approach can perform a greater number of iterations than CLSD approach.

The number of nodes explored in the CLSD^{w}_{BR} approach was greater than CLSD approach for classes 13, 17, 19 and 21 and it was smaller for the other classes. Considering only the classes in which all instances were solved to optimality, the number of nodes explored in the CLSD^{w}_{BR} approach was around 76% smaller than the number of nodes explored in the CLSD approach. Considering the classes with residual GAPs, the number of nodes explored in the CLSD^{w}_{BR} approach was around 11% smaller than CLSD approach.

Table 3: Number of nodes and iterations performed.

Class	Number of nodes explored		Number of iterations performed	
	CLSD	CLSD^{w}_{BR}	CLSD	CLSD^{w}_{BR}
1	1541	671	50134	24784
2	5324	4151	172363	169440
3	3842	1400	130092	49398
4	36452	8769	1224504	420539
5	21534	14202	913286	552771
6	1426276	202249	100289584	15004754
7	76071	25676	3970066	998653
8	3719591	978368	187900007	92400732
9	123995	47022	8097885	2112733
10	2339063	1915179	162079035	204888190
11	1031393	151758	67064913	9687699
12	2138835	1733453	125684903	202960893
13	26452	37367	859010	846972
14	16540	11259	695352	709076
15	47227	27330	1496080	678274
16	1426276	92446	37124290	5811404
17	765350	802356	52081996	34160326
18	504313	397020	42277202	58011746
19	857055	1209576	52552985	58049001
20	482558	441854	45284620	61073639
21	368987	382150	17561192	15858221
22	1193224	727726	77532729	110469561
23	547339	523243	31833302	22167379
24	1060625	963640	80127545	117803858
Average	729807	442590	45381429.2	42221972.49

4.3 Impact of the proposed branching rule

In this section we present some results to try to identify the impact of the branching rule proposed considering the results obtained in Section 4.2. Table 4 presents results with the aim of comparing the performance of the model with the branching rule (CLSD^{w}_{BR}) and without the rule (CLSD^{w}).

Table 4: Studing the impact of the proposed branching rule.

Class	Experiment I: traditional Branch-and-Cut algorithm				Experiment II: default settings of the Cplex solver			
	GAP		running time		GAP		running time	
	$CLSD^w$	$CLSD^w_{BR}$	$CLSD^w$	$CLSD^w_{BR}$	$CLSD^w$	$CLSD^w_{BR}$	$CLSD^w$	$CLSD^w_{BR}$
1	0	0	1.40	**1.16**	0	0	1.73	**1.58**
2	0	0	2.33	**2**	0	0	**2.20**	2.24
3	0	0	1.88	**1.32**	0	0	1.82	**1.72**
4	0	0	6.09	**3.43**	0	0	7.10	**4.78**
5	0	0	37.40	**35.40**	0	0	16.58	**10.72**
6	0.42	**0.16**	1889.95	**1611.01**	0	0	286.41	**167.24**
7	0	0	102.07	**68.65**	0	0	26.36	**17.74**
8	1.37	**0.99**	3335.19	**3220.64**	0	0	**770.33**	881.05
9	**0.02**	0.07	**881.03**	1201.12	0	0	63.45	**38.27**
10	3.18	**2.66**	3600	3600	0.73	**0.60**	3600	3600
11	0.19	**0.15**	1800.97	**1288.20**	0	0	220.34	**115.94**
12	5.96	**3.61**	3600	3600	1.23	**1.18**	3600	3600
13	0	0	67.37	**37.19**	0	0	17.53	**16.84**
14	0	0	27.37	**16.41**	0	0	18.54	**17.47**
15	0	0	64.85	**26.12**	0	0	22.03	**18.43**
16	0	0	304.24	**136.2**	0	0	108.07	**77.59**
17	0.80 (1)	**0.22**	**3503.21**	3600	0.03	0.03	1752.35	**1379.36**
18	8.53 (1)	**2.39**	3600	3600	0.99	**0.81**	3600	3600
19	2.00 (2)	**0.54**	3600	**3450.55**	0.04	**0.02**	2233.86	**2091.43**
20	7.42 (1)	**3.24**	3600	3600	1.42	**1.41**	3600	3600
21	0.09	**0.07**	2152.58	**1668.59**	0	0	628.62	**495.04**
22	2.66	**1.07**	3408.49	**3343.67**	0.23	**0.13**	2894.97	**2811.65**
23	0.18 (1)	**0.07**	2468.21	**1612.57**	**0**	0.01	700.08	**518.13**
24	2.43	**1.56**	**3574.67**	3600	**0.47**	0.50	3600	3600
Average	1.43	**0.70**	1779.76	**1640.08**	0.22	**0.19**	1157.18	**1111.13**

The Cplex solver (using default settings) triggers heuristics to identify the nodes that must be explored first, i.e., the Cplex solver can infer branching rules by exploring the structure of the $CLSD^w$ model. In this way, with the aim of better understanding the impact of the branching rule, we performed Experiment I turning off all existing heuristics and presolve techniques in the Cplex solver obtaining a traditional Branch-and-Cut algorithm.

We can observe in the results from Experiment I shown in Table 4 that the use of the branching rule causes a reduction in the general average GAP of around 51% ($GAP_{CLSD^w} = 1.43$ and $GAP_{CLSD^w_{BR}} = 0.70$) and a reduction in the average running time of around 7.80% ($TIME_{CLSD^w} = 1779.76$ and $TIME_{CLSD^w_{BR}} = 1640.08$).

Using our branching rule, the average GAP was reduced for 13 classes and increased only for one class (class 9). Considering only the instances of classes where all instances were solved to optimality (classes 1 to 5, 7 and 13 to 16), the average running time was reduced for all classes and the average running time using our branching rule for these instances was around 317

seconds, while without the rule, the average running time was around 347 seconds. Therefore, our approach reduces the time needed to solve these instances to optimality by around 8.8%.

We also observe that, without the branching rule, no feasible solutions were found for 6 instances, while using the rule, feasible solutions were found for all test instances. The number of instances that the CLSD^w approach could not find a feasible solution is given in parenthesis in the GAP column of Experiment I in Table 4.

In Classes 17 and 24, the running time for CLSD^w was less than that for CLSD$^w_{BR}$, because the RAM memory limit was exceeded before the timeout expired for some instances in the CLSD^w model without our branching rule.

Experiment II was performed using default settings of the Cplex solver with the aim of to identify the impact of our branching rule when heuristics are used (in the background) to determine the nodes to be explored. We can observe in Table 4 that the use of our branching rule causes a reduction of approximately 10% in the average GAP ($GAP_{CLSD^w} = 0.22$ and $GAP_{CLSD^w_{BR}} = 0.19$) and a reduction in the average running time ($TIME_{CLSD^w} = 1157.18$ seconds and $TIME_{CLSD^w_{BR}} = 1111.13$ seconds) of around 4%.

It can be observed that, considering only the instances in which the optimal solution was found (from classes 1 to 9, 11, 13 to 16, 21 and 23) in Experiment II, the average running time without the branching rule is 192 seconds and, using the rule, this time is 158 seconds. Therefore, the time needed to solve these instances to optimality was reduced by around 17% when our branching rule was used.

Experiments I and II showed that our branching rule causes a positive impact in the computational performance of the CLSD^w model. This impact is more evident when a traditional Branch-and-Cut algorithm is used. However, even using heuristics to determine the nodes that are explored (default settings of the Cplex solver), making our branching rule explicit can significantly accelerate the computational convergence of the Branch-and-Cut algorithm.

5 CONCLUSIONS AND FUTURE STUDIES

In this paper, we deal with the lot sizing and scheduling problem with sequence dependent setup costs and times. We proposed a simple reformulation for the CLSD model originating the CLSD^w model. The CLSD^w model consists of explicitly specifying the binary variables (w) to indicate if an item is produced in one period or not. This formulation allowed us to define a new branching rule to improve the performance of branch-and-bound algorithms. Our branching rule consists of firstly performing the branching in variables w before the other binary variables. We implemented the CLSD^w model and our branching rule in the Cplex 12.60 solver, tested the performance of our approach in 240 test instances from the literature and compared them with the performance of the traditional CLSD model. The computational results show that our approach can significantly reduce the running time and the average GAP.

The branching rule proposed in this paper can improve the performance of algorithms that need to partially solve the CLSD model in each iteration, such as the mixed integer programming based heuristics and Lagrangian based heuristics. As future studies, we highlight the investigation of these approaches using the CLSD^w model with the branching rule proposed in this paper.

ACKNOWLEDGEMENTS

The authors would like to thank the following funding agencies for the financial support: Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), and Fundação de Amparo Pesquisa do Estado de São Paulo (FAPESP) via CEPID No. 2013/07375-0.

APPENDIX: COMPUTATIONAL IMPLEMENTATION

According to the Cplex User's Manual, in the search, Cplex makes decisions about which variable to branch on at a node and the user can control the order in which Cplex branches on variables by issuing a priority order. Cplex performs branches on variables with a higher assigned priority number before variables with a lower priority number.

From Concert Technology, the user can set priority for variables using the method *setPriority*. By default, the variables not assigned an explicit priority value by the user are treated as having a priority value of zero. In this way, our branching rule can be implemented setting a higher priority for variables $w_{jt}, \forall j, t$, than the other binary variables. In this paper, we set a priority of 2 for all variables w_{jt} and use the default value (0) for all variables y_{jt} and z_{ijt}.

We provide the complete code in C++ language including the implementation of the models and the branching rule on the website `http://conteudo.icmc.usp.br/pessoas/mari/Pesquisas.htm`.

RESUMO. Neste artigo tratamos do desafiador problema integrado de dimensionamento de lotes e sequenciamento da produção na existência de tempos e custos de preparação para produção dependentes da sequência. Mais especificamente, nossa atenção é fixada no modelo CLSD, proposto em [13]. Propõe-se, neste trabalho, uma reformulação para o modelo CLSD (intitulada CLSD^w), bem como, uma nova regra de *branching* para ser utilizada em algoritmos do tipo *Branch-and-Bound* para solução do modelo CLSD^w. Por meio de testes computacionais realizados com base em instâncias da literatura, foi possível observar que a abordagem de solução proposta neste artigo é bastante promissora, uma vez que proporcionou significante redução no tempo computacional para solução problema, elevada redução no desvio médio (GAP) entre a melhor solução e o melhor limitante dual conhecidos e uma significante elevação no número de instâncias resolvidas até a otimalidade quando comparado com a abordagem tradicional.

Palavras-chave: Dimensionamento e sequenciamento de lotes, Programação matemática inteira mista, Algoritmo *Branch-and-Bound*.

REFERENCES

[1] B. Almada-Lobo et al. Industrial insights into lot sizing and scheduling modeling. *Pesquisa Operacional*, **35**(3) (2015), 439–464.

[2] C. Almeder & B. Almada-Lobo. Synchronisation of scarce resources for a parallel machine lotsizing problem. *International Journal of Production Research*, **49**(24) (2011), 7315–7335.

[3] S.A. de Araujo, M.N. Arenales & A.R. Clark. Lot sizing and furnace scheduling in small foundries. *Computers & Operations Research*, **35**(3) (2008), 916–932.

[4] H. Chen. Fix-and-optimize and variable neighborhood search approaches for multi-level capacitated lot sizing problems. *Omega*, **56** (2015), 25–36.

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

[6] K. Copil et al. Simultaneous lotsizing and scheduling problems: a classification and review of models. *OR Spectrum*, **39**(1) (2017), 1–64.

[7] D. Ferreira, R. Morabito & S. Rangel. Solution approaches for the soft drink integrated production lot sizing and scheduling problem. *European Journal of Operational Research*, **196**(2) (2009), 697–706.

[8] D. Ferreira, R. Morabito & S. Rangel. Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants. *Computers & Operations Research*, **37**(4) (2010), 684–691.

[9] B. Fleischmann & H. Meyr. The general lotsizing and scheduling problem. *Operations-Research-Spektrum*, **19**(1) (1997), 11–21.

[10] C.H. Glock, E.H. Grosse & J.M. Ries. The lot sizing problem: A tertiary study. *International Journal of Production Economics*, **155** (2014), 39–51.

[11] L. Guimaraes, D. Klabjan & B. Almada-Lobo. Pricing, relaxing and fixing under lot sizing and scheduling. *European Journal of Operational Research*, **230**(2) (2013), 399–411.

[12] L. Guimarães, D. Klabjan & B. Almada-Lobo. Modeling lotsizing and scheduling problems with sequence dependent setups. *European Journal of Operational Research*, **239**(3) (2014), 644–662.

[13] K. Haase. Capacitated lot-sizing with sequence dependent setup costs. *Operations-Research-Spektrum*, **18**(1) (1996), 51–59.

[14] R.J.W. James & B. Almada-Lobo. Single and parallel machine capacitated lotsizing and scheduling: New iterative MIP-based neighborhood search heuristics. *Computers & Operations Research*, **38** (2011), 1816–1825.

[15] G.M. Kopanos, L. Puigjaner & C.T. Maravelias. Production planning and scheduling of parallel continuous processes with product families. *Industrial & engineering chemistry research*, **50**(3) (2010), 1369–1378.

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

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

[18] H. Meyr & M. Mann. A decomposition approach for the General Lotsizing and Scheduling Problem for Parallel production Lines. *European Journal of Operational Research*, **229**(3) (2013), 718–731.

[19] W. Wei et al. Tactical production and distribution planning with dependency issues on the production process. *Omega*, **67** (2017), 99–114.

[20] L.A. Wolsey. MIP modelling of changeovers in production planning and scheduling problems. *European Journal of Operational Research*, **99**(1) (1997), 154–165.

[21] J. Xiao et al. A hybrid Lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, **63** (2015), 72–82.

[22] S. Çağrı & B. Bilgen. Hybrid simulation and MIP based heuristic algorithm for the production and distribution planning in the soft drink industry. *Journal of Manufacturing Systems*, **33**(3) (2014), 385–399.