Column generation for minimizing total completion time on a single machine with parallel batching

Column generation for minimizing total completion time on a single machine with parallel batching

A. Alfieri * **A. Druetto** ** **A. Grosso** ** **F. Salassa** *

* *Department of Management and Production Engineering, Politecnico di Torino, Italy (e-mail: { arianna.alfieri, fabio.salassa } @polito.it).*
** *Department of Computer Science, Universitá di Torino, Italy (e-mail: alessandro.druetto@edu.unito.it, andrea.grosso@unito.it)*

Abstract In manufacturing of integrated circuits, burn in operations are key operations to improve quality by detecting early failures. Such operations are performed in the so-called *batch ovens*, able to process several circuits at the same time. In this paper, the problem of deciding in which sequence to process a given number of circuit boards in a batch oven is addressed. This problem can be considered a scheduling problem in a single machine and parallel batch job processing. A column generation algorithm is developed for the minimization of the total completion time when circuits have an individual size. Results show the efficiency and effectiveness of the proposed algorithm, able to solve instances up to 100 jobs.

Keywords: Dynamic programming, Industry automation, Operations research, Optimization problems, Scheduling algorithms

1. INTRODUCTION

In many manufacturing processes, it is of high importance to be able to test components in stressed conditions before they are assembled in the final products. These tests should force the component to fail in controlled conditions so as to understand the behavior of the component in the real word, improve the manufacturing process (if necessary) and prescribe the final product operating conditions.

One of such process is the production of integrated circuits in which the stress tests are made by operating the semiconductor components at very high temperatures (even higher than 250 Celsius degree), higher to the ones they will go through in the real operating conditions. This testing process is called *burn in process* and is used to detect defects (due to several causes from design to material or manufacturing) that could lead to a component failure, especially in the early stages of its life.

Usually, burn-in operations are performed in ovens (the *batch oven*) that can accommodate several circuits, processed in parallel, and released together as a single batch. There exists an upper limit on the batch capacity and every circuit has an individual size that limits the number of circuits that can be packed together in a single batch.

Hence, when several semiconductor components have to be tested on a single oven, they have to be packed in several batches due to the limited capacity of the oven. Once the batches have been created, how to sequence them has to be decided.

The two decisions (batching and sequencing) are intertwined and depend on the shop floor manager objectives. As an example, if semiconductors have a due dates, the operations manager could be interest in minimizing the number of tardy circuits.

In many manufacturing environment, especially those trying to implement lean manufacturing prescriptions, a common aim is to reduce the work in process (WIP) that, given the company target flow rate (the throughput (TH)), is strictly dependent on the so-called average flow time (FT), due to the Little's law,

$$WIP = TH \cdot FT.$$

Hence, to minimize WIP without decreasing TH, FT must be minimized.

In a deterministic context, when the processing times and the number of jobs to be processed are known, minimizing the average FT is equivalent to minimize the total FT.

The problem discussed above corresponds, in scheduling terms, to the minimization of the total completion time (assuming all the jobs are available at the same time) in a single machine and parallel batch processing.

In this paper, a column generation algorithm is developed to solve such scheduling problem.

The remainder of the paper is structured as follows. Section 2 formally defines the scheduling problem and states the contribution of the paper. The mathematical models and the column generation algorithm are reported in Section 3. Computational results are discussed in Section 4. Conclusions can be found in Section 5.

2. PROBLEM DEFINITION

Using the Kendall's notation, the problem introduced in Section 1 can be considered a $1|\text{p-batch}|\sum C_j$ single-machine scheduling problem. In such problem, a set of

jobs $N = \{1, 2, \ldots, n\}$ is given; each job j has a processing time p_j and a *size* s_j. Jobs are to be scheduled in batches $B \subset N$ such that the total size in a batch (i.e., $\sum_{j \in B} s_j$) does not exceed a given machine capacity C. All the jobs in a batch are processed in parallel and all of them are simultaneously released after a time interval whose length equals the processing time of the longest job. Hence, in a batch B starting at time t, all jobs $j \in B$ will complete at the same time $C_j = C_B = t + p_B$, with $p_B = \max\{p_j : j \in B\}$. The problem asks for partitioning the jobs in a suitable (unknown) number of batches and determining the batch sequence $S = (B_1, B_2, \ldots, B_r)$ for which the objective function $f(S) = \sum_{j=1}^{n} C_j = \sum_{k=1}^{r} |B_k| C_{B_k}$ is the minimum.

To the authors' knowledge, few results have been published about this problem. Uzsoy (1994) proves NP-completeness, provides a lower bound based on a parallel-machines relaxation, and sketches a branch and bound procedure; Ghazvini and Dupont (1998) give several heuristics; Parsa et al. (2016) develop an ant-colony based metaheuristic procedure.

We propose a very large (exponential in size) model for $1|\text{p-batch}|\sum C_j$ whose continuous relaxation can be handled via column generation, giving the tightest lower bound currently available for the problem. From the final stage of the column generation procedure, a heuristic solution can be easily generated by branch and bound.

3. MODELS AND COLUMN GENERATION

3.1 MILP model

A known (basic) MILP model for problem $1|\text{p-batch}|\sum_j C_j$ is reported in the following where variable $x_{ij} = 1$ iff job i is scheduled in the j-th batch. Variables C_j and c_i represent the completion times of the j-th batch and of job i, respectively. Variable P_j represents the processing time of the j-th batch.

$$\text{minimize} \quad z = \sum_{i=1}^{n} c_i \tag{1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} s_i x_{ij} \le C \qquad j = 1, \ldots, n \tag{3}$$

$$P_j \ge x_{ij} p_i \qquad i = 1, \ldots, n,\ j = 1, \ldots, n \tag{4}$$

$$C_1 \ge P_1 \tag{5}$$

$$C_j \ge C_{j-1} + P_j \qquad j = 2, \ldots, n \tag{6}$$

$$c_i \ge C_j - M(1 - x_{ij}) \qquad i = 1, \ldots, n,\ j = 1, \ldots, n \tag{7}$$

$$P_j, C_j, c_i \ge 0 \qquad i = 1, \ldots, n,\ j = 1, \ldots, n$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n,\ j = 1, \ldots, n$$

The total completion time is expressed by (1). Constraint set (2) ensures that each job is assigned exactly to one batch and, since all the jobs assigned to a batch cannot exceed the batch capacity, we need to define constraint set (3). Since processing time for a batch is the maximum processing time of all the contained jobs, we write constraint set (4). The completion time for the first batch is simply its processing time since it is the first to be processed by the machine, as stated in constraint (5); in fact, constraint set (6) ensures that completion time for all the other batches will be evaluated as the sum of its processing time and the completion time of the precedent batch. Constraint set (7) specifies that the completion time of a job must be the completion time of the corresponding batch.

3.2 A graph-based master problem

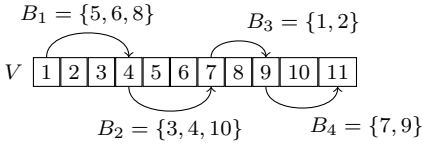
Model (1)–(7) is known to be very weak. A state-of-the art solver like `CPLEX` can waste hours over 15-jobs instances without solving them, nor generating feasible solutions.

We consider an alternative model where a batch sequence is represented as a path on a graph. Let $V = \{1, 2, \ldots, n+1\}$, and let $\mathcal{B} = \{B \subseteq N : \sum_{j \in B} s_j \le C\}$ be the set of all the possible batches. Define a (multi)graph $G(V, A)$ with arc set

$$A = \{(i, k, B) :\ i, k \in V; i < k; B \in \mathcal{B}; |B| = (k - i)\}.$$

The arcs are *labeled* with batches. Each arc (i, k, B) with head k and tail i represents a batch B scheduled in a batch sequence such that exactly $i - 1$ jobs precede batch B, $|B| = k - i$ and exactly $n - i + 1$ jobs are scheduled from batch B up to the end of the sequence.

We assign a cost $c_{ikB} = (n - i + 1)p_B$ to each arc (i, k, B). A feasible batch sequence is represented by a path from node 1 to node $n + 1$ such that the set of jobs is exactly partitioned over the arcs in the path; with the arc costs computed above, the cost of the path is the total flow time of the batch sequence. The optimal sequence is a minimum-cost path from 1 to $n + 1$ satisfying also the partitioning constraint. A 10-jobs example is reported in Figure 1.



$$\begin{aligned} N &= \{1, 2, \ldots, 10\} \\ S &= (B_1, B_2, B_3, B_4) \\ f(S) &= |B_1| p_{B_1} + \\ &\quad |B_2|(p_{B_1} + p_{B_2}) + \\ &\quad |B_3|(p_{B_1} + p_{B_2} + p_{B_3}) + \\ &\quad |B_4|(p_{B_1} + p_{B_2} + p_{B_3} + p_{B_4}) = \\ &\quad 10p_{B_1} + 7p_{B_2} + 4p_{B_3} + 2p_{B_4}. \end{aligned}$$

Figure 1. Batch sequence as a path on a graph. $N = B_1 \cup B_2 \cup B_3 \cup B_4$.

The problem can be modeled as follows:

$$\text{minimize} \quad z = \sum_{(i,k,B) \in A} c_{ikB} x_{ikB} \tag{8}$$

subject to

$$\sum_{\substack{(k,B):\\(i,k,B)\in A}} x_{ikB} - \sum_{\substack{(k,B):\\(k,i,B)\in A}} x_{kiB} = \begin{cases} 1 & i = 1 \\ 0 & i = 2,\dots,n \\ -1 & i = n+1 \end{cases} \quad (9)$$

$$\sum_{(i,k,B)\in A} a_B x_{ikB} = \mathbf{1} \quad (10)$$

$$x_{ikB} \in \{0,1\}\,(i,k,B) \in A$$

where $a_B = (1 \text{ if } j \in B \text{ else } 0) \in \{0,1\}^n$ is the incidence vector of the job set B, and $\mathbf{1} = (1,1,\dots,1)^T$. Variable $x_{ikB} = 1$ if arc (i,k,B) is on the path (i.e., batch B is scheduled after $i-1$ jobs); constraints (9) ensure that a unit flow is sent from node 1 to node $n+1$, therefore capturing the shortest path problem. Constraints (10) enforce the requirement that the job set is exactly partitioned over the arcs selected in the path.

3.3 Pricing procedure

We now outline a column generation procedure that exactly solves the continuous relaxation of the master problem (8)–(10), and can be readily used in order to obtain a heuristic solution for 1|p-batch|$\sum_j C_j$.

We keep a restricted master problem where only a subset $A' \subset A$ is represented, and solve the continuous relaxation of model (8)–(10) on such restricted graph. This provides the optimal simplex multipliers $u_1,\dots,u_{n+1}$ and $v_1,\dots,v_n$ associated to constraints (9) and (10), respectively, such that

$$u_i - u_k + \sum_{j\in B} v_j \le c_{ikB}, \quad \forall\,(i,k,B) \in A'.$$

Then, by the general theory of linear programming, we know that either

$$u_i - u_k + \sum_{j\in B} v_j \le c_{ikB}, \quad \forall\,(i,k,B) \in A,$$

and the continuous solution is optimal for the relaxed master problem, or there exist an arc with negative reduced cost

$$r_{ikB} = c_{ikB} - \big(u_i - u_k + \sum_{j\in B} v_j\big), \quad (11)$$

which can enter the simplex basis in order to decrease the objective function. We seek for the arcs with minimum reduced cost for each given i, k and each given batch processing time p_B.

Assume to index the jobs such that $p_1 \ge p_2 \ge \cdots \ge p_n$. Finding the batch B that minimizes (11) for each given i,k and p_B can be done by exploiting the dynamic programming state space of a family of cardinality-constrained knapsack problems, i.e.,

$$\max\left\{\sum_{j=r}^{n} v_j y_j :\ \sum_{j=r}^{n} s_j y_j \le \tau\ \sum_{j=r}^{n} y_j = m,\ y_j \in \{0,1\}\right\}. \quad (12)$$

Let $g_r(\tau,m)$ be the optimal value of a knapsack of type (12), limited to items $r, r+1,\dots,n$ and total size $\le \tau$. Such $g_r(\tau,m)$ can be recursively computed as

$$g_r(\tau,m) = \max\begin{cases} g_{r+1}(\tau - s_r, m-1) + v_r & (y_r = 1) \\ g_{r+1}(\tau,m) & (y_r = 0) \end{cases}$$

with boundary conditions

$$g_r(\tau,1) = \begin{cases} v_r & \text{if } s_r \le \tau\ (y_r = 1) \\ 0 & \text{otherwise } (y_r = 0) \end{cases}$$
$$r = 1,\dots,n,\ t = 0,\dots,C$$
$$g_r(\tau,0) = 0 \qquad r = 1,\dots,n,\ \tau = 0,\dots,C$$
$$g_r(\tau,m) = -\infty \qquad \text{if } m > n-r+1 \text{ or } \tau < 0.$$

The optimal item sets $S_r(\tau,m)$ can be retrieved by backtracking. All the relevant arcs with minimum reduced cost (for given i,k) can be generated by the following procedure:

1: Set $H = \emptyset$;
2: **for** $m = 1,\dots,n$ **do**
3: **for** $l = 1,\dots,n$ **do**
4: Compute $g_{l+1}(C - s_l, m-1)$ and $S_{l+1}(C - s_l, m-1)$;
5: Set $B = S_{l+1}(C - s_l, m-1) \cup \{l\}$;
6: **for** $i = 1,\dots,n-m+1$ **do**
7: Set $k = i + m$;
8: Compute $r_{ikB} = p_l(n-i+1) - (u_i - u_k) - \sum_{j\in B} v_j$;
9: **if** $r_{ikB} < 0$ **then**
10: Set $H := H \cup \{(i,k,B)\}$;
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: **return** H;

We store the states $g_r(\tau,m)$ in a dynamic programming table, so that no state needs to be computed more than once (line 4). There are no more than $O(n^2C)$ states to be computed, with constant-time operations needed for each of them. Hence, filling the dynamic programming table contributes for a computation time limited by $O(n^2C)$ for the whole procedure. The three **for** loops (lines 2,3,6) contribute for an $O(n^3)$ computation time. Hence, the overall time complexity for the pricing procedure is $O(\max(n^3, n^2C))$.

We note that a similar pricing procedure is outlined in Parsa et al. (2010) for a similar problem.

3.4 Main procedure

Once the relaxed master problem is optimally solved (no more arc with negative cost is found), the optimal value provides a valid lower bound for 1|p-batch|$\sum_j C_j$. Also, a heuristic solution can be easily generated by keeping all the generated columns and solving via branch and bound the integer version of the restricted master problem.

The whole procedure can be summarized as follows:

1: $A' \leftarrow$ starting column subset
2: $G(V,A') \leftarrow$ restricted master problem
3: **while** true **do**
4: $z \leftarrow$ continuous optimum of $G(V,A')$
5: $H \leftarrow$ columns from A with negative reduced cost
6: **if** |H| = 0 **then**
7: `CG-LB` $\leftarrow$ z — continuous optimum, lower bound
8: `CG-UB` $\leftarrow$ integer computed over $G(V,A')$
9: **break**
10: **end if**
11: $A' \leftarrow A' \cup H$

12: **end while**

We refer to the lower bound as CG-LB and to the upper bound as CG-UB in the next section.

4. COMPUTATIONAL RESULTS

Following the experiments made in Uzsoy (1994), for the job processing time p_i and size s_i we used random uniform distribution. For the processing time we considered the range $p_i \in [1, 100]$ and for the job size s_i we used four different ranges, $s_1 \in [1, 10], s_2 \in [2, 8], s_3 \in [3, 10], s_4 \in [1, 5]$.

The number n of jobs ranged from 10 to 100 and the batch capacity C was set to two different values, 10 and 30.

We generated 10 test instances for each combination of C, n and s_i, for a total number of 400 instances. The gap between CG-LB and CG-UB is evaluated as

$$gap = \frac{\texttt{CG-UB} - \texttt{CG-LB}}{\texttt{CG-LB}} \cdot 100\%.$$

All the tests have been done in a UNIX environment with Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz processor, the entire program was developed in C++ 5.4.0 language, and the optimizer used to solve relaxed and integer problems is IBM(R) ILOG(R) CPLEX(R) 12.8.0.0 directly called from C++ environment using CPLEX callable libraries.

Tables 1 and 3 show the results over an increasing number of jobs with fixed batch capacity; the term limit is used whenever CPLEX reaches its 60-secs time limit during upper bound evaluation. Values are shown as average over each 10-instance group for the time and as average, maximum and minimum over each 10-instance group for the gap. Column *opt* reports the number of instances (over the 10) in which the solution can be certified to be the optimum, as explained in the following.

Tables 2 and 4, instead, show the comparison between our lower bound CG-LB and the other lower bound known in literature PR defined by Uzsoy (1994). To compare the two bounds, we simply divided the raw values and showed the average, maximum and minimum over each 10-instance group.

In Table 1 we can see that with 10 jobs our algorithm has no problem to produce an excellent lower bound, since running CPLEX upon the batch subset sometimes produces values for the CG-UB strictly close to the CG-LB (consider that the average gap is under 1% in all cases).

For many instances our algorithm is so good that, when running CPLEX on the batches produced by the column generation, it actually testifies that the lower bound found is in fact the upper bound and, hence, the optimum of the problem.

Unfortunately, it happens for a lot of 10-jobs instances but when increasing the number of jobs, it decreases rapidly (we are only able to prove optimality of 2 of the 100-jobs instances, and all inside s_3 distribution).

Even when we have to stop CPLEX after 60 seconds during the upper bound evaluation (if the $C = 10$, $n = 100$, and $s_i = s_4$), we are able to find excellent solutions with an average gap over 5% only in the s_4 case and under 1% in all other cases.

Param		Times (s)		Gap (%)			
n	s_i	LB time	UB time	avg	worst	best	opt
10	s_1	0.007569	0.019876	0.13	1.16	0.00	8
	s_2	0.007221	0.019370	0.72	6.15	0.00	7
	s_3	0.006075	0.016860	0.13	1.34	0.00	9
	s_4	0.008757	0.023490	1.02	3.57	0.00	3
20	s_1	0.015767	0.047490	1.21	4.07	0.00	2
	s_2	0.013482	0.032899	1.32	3.12	0.00	2
	s_3	0.010421	0.023087	0.65	3.46	0.00	7
	s_4	0.025562	0.074103	2.37	4.86	0.83	0
30	s_1	0.029203	0.065790	0.27	0.82	0.00	3
	s_2	0.028965	0.075451	0.76	1.91	0.00	2
	s_3	0.021027	0.037646	0.30	1.15	0.00	7
	s_4	0.039657	0.285791	2.39	4.98	0.00	1
50	s_1	0.071697	0.512053	0.73	1.60	0.09	0
	s_2	0.051811	0.194932	0.33	0.85	0.09	0
	s_3	0.040634	0.138985	0.58	1.71	0.00	3
	s_4	0.195869	3.859571	2.11	3.04	0.96	0
100	s_1	1.027188	4.849005	0.46	0.85	0.08	0
	s_2	0.634684	7.887090	0.43	0.80	0.15	0
	s_3	0.420390	0.635104	0.18	0.69	0.00	2
	s_4	2.325456	limit	6.48	12.15	1.25	0

Table 1. Results for CG-UB and CG-LB with $C = 10$

Param		CG-UB / PR		
n	s_i	avg	worst	best
10	s_1	1.33	1.22	1.54
	s_2	1.29	1.19	1.40
	s_3	1.25	1.16	1.36
	s_4	1.38	1.29	1.50
20	s_1	1.25	1.19	1.31
	s_2	1.22	1.20	1.27
	s_3	1.19	1.15	1.21
	s_4	1.29	1.23	1.37
30	s_1	1.18	1.15	1.21
	s_2	1.19	1.14	1.27
	s_3	1.19	1.11	1.22
	s_4	1.22	1.17	1.36
50	s_1	1.15	1.12	1.18
	s_2	1.16	1.12	1.20
	s_3	1.17	1.14	1.21
	s_4	1.15	1.11	1.17
100	s_1	1.13	1.11	1.14
	s_2	1.14	1.11	1.16
	s_3	1.15	1.12	1.20
	s_4	1.09	1.08	1.10

Table 2. Comparison of CG-LB over PR with $C = 10$

From Table 2, it can be easily seen that CG-LB performances are much better than PR in every combination, ranging from an average 9% when $n = 100$ and $s_i = s_4$ to an average 38% when $n = 10$ and $s_i = s_4$.

Percentages are, in fact, the relative increase in the lower bound value and it is better to have, at every time, the highest possible lower bound value (e.g., to be used in a branch-and-price algorithm).

Our algorithm performs better in this comparison. In fact, it gives a higher lower bound value when we have a low number of jobs, and it starts to behave more and more similarly to the other one when the number of jobs increases (since the number of needed batches also increases).

Param		Times (s)		Gap (%)			
n	s_i	LB time	UB time	avg	worst	best	opt
10	s_1	0.005430	0.016408	0.01	0.06	0.00	9
	s_2	0.009738	0.026623	0.00	0.00	0.00	10
	s_3	0.008556	0.021751	0.35	2.29	0.00	8
	s_4	0.005473	0.018949	0.00	0.00	0.00	10
20	s_1	0.027192	0.083138	1.12	4.48	0.00	3
	s_2	0.023559	0.078249	1.13	5.95	0.00	5
	s_3	0.031489	0.078068	1.19	5.78	0.00	4
	s_4	0.027081	0.089415	0.00	0.00	0.00	10
30	s_1	0.068000	0.398944	2.23	4.09	0.44	0
	s_2	0.077995	0.372680	2.63	5.16	0.27	0
	s_3	0.056954	0.271775	2.76	5.01	1.00	0
	s_4	0.198553	0.496824	0.33	1.59	0.00	4
50	s_1	0.599840	16.744630	4.41	7.01	0.62	0
	s_2	0.609479	4.575961	3.97	5.90	1.48	0
	s_3	0.319882	8.072704	4.79	6.68	2.48	0
	s_4	2.749605	4.631010	1.97	4.42	0.00	2
100	s_1	6.098725	`limit`	16.64	23.38	8.16	0
	s_2	4.859389	`limit`	11.02	13.20	6.04	0
	s_3	2.795998	`limit`	12.78	24.44	2.33	0
	s_4	27.279960	`limit`	9.32	19.84	2.71	0

Table 3. Results for `CG-UB` and `CG-LB` with $C = 30$

Param		`CG-LB` / `PR`		
n	s_i	avg	worst	best
10	s_1	1.72	1.50	2.15
	s_2	1.59	1.33	1.85
	s_3	1.52	1.40	1.63
	s_4	2.05	1.77	2.28
20	s_1	1.46	1.36	1.66
	s_2	1.39	1.29	1.49
	s_3	1.35	1.30	1.40
	s_4	1.81	1.62	2.11
30	s_1	1.37	1.31	1.54
	s_2	1.33	1.25	1.42
	s_3	1.25	1.22	1.31
	s_4	1.57	1.40	1.70
50	s_1	1.25	1.23	1.28
	s_2	1.22	1.17	1.25
	s_3	1.17	1.15	1.20
	s_4	1.43	1.35	1.50
100	s_1	1.16	1.13	1.19
	s_2	1.13	1.12	1.14
	s_3	1.10	1.09	1.11
	s_4	1.26	1.23	1.29

Table 4. Comparison of `CG-LB` over `PR` with $C = 30$

Looking at Table 3, we notice that performances with 10 jobs are better than with the previous capacity, since the algorithm spends time generating a lot of batches; we are able to prove optimality of the majority of instances with better average gaps.

The same consideration applies when we consider 20 and 30 jobs; with 50 jobs we notice some curious behavior. With job distribution s_1, the column generation finds with no problem the column subset that contains the continuous optimum, but `CPLEX` has a hard time to extract from that batch subset the integer optimum. Except from that, the performance is rather good, with an average gap of less than 5% in all the combinations.

If we have 100 jobs, considering a capacity of 30, the possible batches can be quite a lot. It can be easily seen if we consider the computation times for `CG-LB` especially with job distribution s_4: it takes an average of about 30 seconds for our algorithm to generate batches containing the continuous optimum, but the search space for the lower bound is very large and `CPLEX` reaches the 60 seconds time limit in all the combinations.

From Table 4, it can be noticed that our algorithm performs way better when $n \leq C$, and this fact is amplified when jobs are only 10.

Especially with jobs distribution s_4 (and partially with s_1), if we have to schedule 10 jobs, `CG-LB` performs more than 100% better than `PR` in its best and average values.

Overall, increasing capacity also increments the distance between the two bounds; we perform again better in every combination, ranging from an average 10% more when $C = 30$, $n = 100$, and $s_i = s_3$ to an average 105% more when $C = 30$, $n = 10$, and $s_i = s_4$.

Param		Times (s)		Gap (%) / `PR`		
n	s_i	`HMMAS`	`CG-UB`	`HMMAS`	`CG-UB`	`CG-UB` / `CG-LB`
20	s_1	1.29	0.05	1.25	1.27	1.01
	s_2	1.41	0.03	1.25	1.24	1.01
	s_3	1.56	0.02	1.21	1.20	1.01
	s_4	0.98	0.07	1.28	1.32	1.02
40	s_1	5.55	0.48	1.19	1.21	1.01
	s_2	5.93	0.49	1.19	1.18	1.01
	s_3	6.32	0.12	1.18	1.19	1.01
	s_4	3.68	1.19	1.20	1.22	1.03
60	s_1	20.61	2.71	1.17	1.18	1.01
	s_2	19.26	1.61	1.16	1.14	1.01
	s_3	23.41	0.43	1.18	1.17	1.01
	s_4	10.90	27.53	1.18	1.18	1.02
80	s_1	57.54	8.27	1.16	1.15	1.01
	s_2	55.72	22.91	1.16	1.12	1.01
	s_3	62.91	1.04	1.16	1.15	1.01
	s_4	28.87	60.77	1.16	1.18	1.06
100	s_1	109.72	4.84	1.16	1.13	1.01
	s_2	106.39	7.89	1.15	1.14	1.01
	s_3	135.90	0.64	1.15	1.16	1.01
	s_4	53.58	61.27	1.15	1.16	1.06

Table 5. Comparison between `HMMAS` and `CG-UB` algorithms

We also made an attemp to compare our upper bound `CG-UB` to the one implemented by Parsa et al. (2016). Unfortunately, we do not have access neither to their algorithm nor to their instances, so we managed to do our comparison generating the random instances in the same way they did, and evaluating the execution times with extreme care since our operational environment is

not the same as the one they used. Moreover, they do not provide a new lower bound; in fact, they used the previously mentioned PR described in Uzsoy (1994). Also, for the comparison, we only used capacity $C = 10$ since authors only tested such value.

Our tests, as can be seen in Table 5, had shown that the performance of our upper bound CG-UB is very similar for all the combinations in comparison of the hybrid ant system HMMAS over the known lower bound PR.

We cannot explicitly compare to results of Parsa et al. (2016) but, since the results appear to be very close, we can speculate that the two algorithms could give similar results for the upper bound, when they are run on the same instance set.

On the other hand, considering the CG-LB lower bound, differences appear, as it can be noticed from the last column of the table. In fact, the use of CG-LB certifies that our found upper bounds are very close to the integer optima for all the combinations (maximum gap is around 6%). This last consideration even more certifies that CG-LB is a better approximation of the integer optimum than the previously known PR.

5. CONCLUSIONS

In this paper, the scheduling of a single machine and parallel batch has been addressed. This problem is a very relevant problem in many manufacturing environments, especially in the semiconductor industry. In this industry, in fact, stress tests in batch ovens are usually performed to detect defects that would lead to a component failure in the early phases of the system life.

In order to reduce the WIP without affecting the throughput rate, the minimization of the total completion time has been considered as objective function.

The scheduling problem has been represented through a novel graph-based mathematical model, the relaxation of which can be, however, solved using a time efficient column generation algorithm.

The computational results showed that the column generation algorithm is able to find lower bounds that outperform all the ones currently available in the literature. Moreover, once the optimal solution of the relaxed problem (i.e., the lower bound) has been found, a simple procedure relying on the integer version of the restricted master problem with the generated columns can give a good quality heuristic solution, as shown by the computational results.

Future research will deal with the extension of the proposed solution approach to parallel batch scheduling problems in other production layouts such as flow lines and parallel machine stations.

REFERENCES

Ghazvini, F.J. and Dupont, L. (1998). Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes. *International Journal of Production Economics*, 55(3), 273 – 280. doi: https://doi.org/10.1016/S0925-5273(98)00067-X.

Parsa, N.R., Karimi, B., and Husseini, S.M. (2016). Minimizing total flow time on a batch processing machine using a hybrid maxâĂŞmin ant system. *Computers & Industrial Engineering*, 99, 372 – 381. doi: https://doi.org/10.1016/j.cie.2016.06.008.

Parsa, N.R., Karimi, B., and Kashan, A.H. (2010). A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers & Operations Research*, 37(10), 1720 – 1730. doi:https://doi.org/10.1016/j.cor.2009.12.007.

Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32(7), 1615–1635. doi: 10.1080/00207549408957026.