Polynomial-Size Models to Minimize Total Completion Time in a Parallel Batching Environment

Polynomial-Size Models to Minimize Total Completion Time in a Parallel Batching Environment

Alessandro Druetto * **Andrea Grosso** *

* *Dipartimento di Informatica, Università di Torino, Via Pessinetto 12, 10149 Torino, Italy (e-mail: alessandro.druetto@unito.it, andrea.grosso@unito.it)*

Abstract: We present a new integer linear formulation for the problem of minimizing the total completion time on a single parallel-batching machine. The new formulation is strong, in the sense that it delivers a sharp lower bound, and compact, i.e. polynomial in size, contrasted to recent successful models for the same problem that have exponential size and require to be handled by column generation. The new model is promising: combined with a rounding procedure, it allows to deliver good solutions with small, certified optimality gaps for instances with up to 50 jobs, and we believe it is susceptible of further improvements.

Keywords: Operations Research, Scheduling Algorithms, Parallel Batching, Total Completion Time, Arc-Flow Models

1. INTRODUCTION

We consider the single-machine scheduling problem where a job set $N = \{1, 2, \dots, n\}$ is given in order to be processed on a parallel batching machine with limited capacity. The jobs must be partitioned into batches and arranged into a batch sequence $S = (B_1, B_2, \dots, B_t)$ such that the sum of all jobs' completion times $f(S) = \sum_{j \in N} C_j$ is as small as possible. Such objective is interesting since, being directly related to the *mean* total completion time $\frac{1}{n} \sum_j C_j$, its minimization improves the system's responsiveness.

In the parallel batch environment that is considered, all the jobs in the same batch B_k are processed in parallel and all share the batch completion time C_{B_k}: $C_j = C_{B_k}$ for all $j \in B_k$. Each job $j \in N$ has a given processing time p_j and a size s_j. The machine has a given positive *capacity* b. The longest job in a batch B_k determines the batch processing time, i.e. the whole batch has a duration of $p_{B_k} = \max\{p_j : j \in B_k\}$ during which all its jobs are processed; the total size of the jobs in a batch cannot exceed the machine capacity $b > 0$: $\sum_{j \in B_k} s_j \leq b$. In the classical three-fields notation the problem is denoted $1|p\text{-batch}, s_j| \sum_j C_j$.

Parallel batching problems often arise in the management of furnaces-like facilities, notably in the semiconductor industry (Mathirajan and Sivakumar, 2006; Mönch et al., 2011). For a comprehensive survey, from an algorithmic point of view, a valid reference is still Potts and Kovalyov (2000). In the last decade the literature about batching problems offers plenty of works, but with such a number of very specific variants and technological constraints that making up a coherent survey is a very difficult task. Makespan minimization on one or more machines seems to be best studied variant in parallel batching models; total completion time models occupy a somehow smaller niche. The $1|p\text{-batch}, s_j| \sum_j C_j$ problem has been first tackled in Uzsoy (1994), where it is recognized as NP-hard and a first exact branch-and-bound algorithm is sketched. In (Azizoglu and Webster, 2000) the more general weighted total flow time problem, $1|p\text{-batch}, s_j| \sum_j w_j C_j$, is solved by branch and bound, for instances with up to 25 jobs. A metaheuristic approach, via ant-colony optimization, is proposed in Parsa et al. (2016).

In recently proposed, so-called arc-flow models, the structure of a batch sequence is encoded as a flow configuration on a suitable type network. An original arc-flow ILP model is proposed by (Trindade et al., 2021) for makespan minimization, leading to excellent computational results on very large instances. In Alfieri et al. (2019, 2021) a different arc-flow model is proposed for the $1|p\text{-batch}, s_j| \sum_j C_j$ problem, leading to exact solution of 40-jobs instances and heuristic solution of 100-jobs instances. The size of such model is exponential in the number of jobs, and column generation techniques are necessary to solve it. In Druetto and Grosso (2022) this approach is extended to the aforementioned weighted case.

From a practical point of view, it can be appealing to get rid of the technicalities that arise in column generation. We present the first strong Integer Linear Programming (ILP) model for problem $1|p\text{-batch}, s_j| \sum_j C_j$ which is also "compact" in size, i.e. with a number of variables and constraints bounded by a polynomial in the number of jobs. The continuous relaxation of such model delivers a sharp lower bound, competitive with the bounds provided

by the column generation techniques of (Alfieri et al., 2021). Combined with a variable rounding heuristic it can generate very good solutions with certified optimality gaps for instances with up to 50 jobs. Although this performance is still behind that of column generation, the compact model is extremely promising, and sifting procedures (currently under development) are expected to further enlarge the size of solvable instances.

The paper is organized as follows. In Section 2 we develop arc-flow models for $1|p\text{-batch}, s_j|\sum_j C_j$; in Section 3 we illustrate heuristic procedures based on variable rounding, and in Section 4 we discuss testing and computational results for the given models and procedures. Conclusions and directions for future research are drawn in Section 5.

2. ARC-FLOW MODELS

An ILP model for this problem, as given by Parsa et al. (2016) is 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.

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

subject to

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

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

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

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

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

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

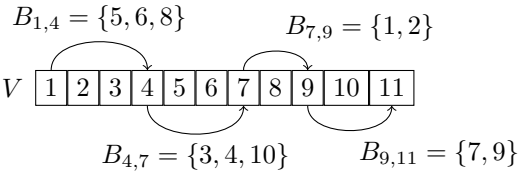
$$P_j, C_j, c_i \ge 0 \qquad i = 1, \ldots, n,\ j = 1, \ldots, n \tag{8}$$

$$x_{ij} \in \{0, 1\} \qquad i = 1, \ldots, n,\ j = 1, \ldots, n \tag{9}$$

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, constraint set (3) has to be defined. Constraint set (4) represents the fact that the processing time of a batch is the maximum processing time of all the contained jobs. 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). Constraint set (6), instead, ensures that completion time for all the other batches is 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 (the constant M must be very large). Model (1)–(9) is known to be very weak. A state-of-the-art solver like `CPLEX` can waste hours over 15-jobs instances, with optimality gaps at the root branching node of 100%.

2.1 Arc-flow models

Arc-flow model for parallel batching problems are a recent development where batch schedules are mapped on flow configurations on a suitable network. In Trindade et al. (2021) an arc-flow model for minimizing makespan in a single parallel-batching machine is proposed, with very good results. In Alfieri et al. (2019, 2021) an arc-flow model is proposed for the $1|p\text{-batch}, s_j|\sum_j C_j$ problem, leading to handle 100-jobs instances. This arc-flow model is exponential in size and must be handled via column generation techniques; here we develop, from the same modeling ideas, arc-flow models whose size is polynomial in the number on jobs n.



$$N = \{1, 2, \ldots, 10\} \implies V = \{1, 2, \ldots, 11\}$$
$$P_S = [(1,4), (4,7), (7,9), (9,11)]$$
$$S = (B_{1,4}, B_{4,7}, B_{7,9}, B_{9,11})$$

$$c_{1,4} = p_{B_{1,4}}(10 - 1 + 1) = 10 p_{B_{1,4}}$$
$$c_{4,7} = p_{B_{4,7}}(10 - 4 + 1) = 7 p_{B_{4,7}}$$
$$c_{7,9} = p_{B_{7,9}}(10 - 7 + 1) p_{B_{79}} = 4 p_{B_{7,9}}$$
$$c_{9,11} = p_{B_{9,11}}(10 - 9 + 1) = 2 p_{B_{9,11}}$$

$$\begin{aligned} f(S) = &|B_{1,4}| p_{B_{1,4}} + \\ &|B_{4,7}|(p_{B_{1,4}} + p_{B_{4,7}}) + \\ &|B_{7,9}|(p_{B_{1,4}} + p_{B_{4,7}} + p_{B_{7,9}}) + \\ &|B_{9,11}|(p_{B_{1,4}} + p_{B_{4,7}} + p_{B_{7,9}} + p_{B_{9,11}}) = \\ = &10 p_{B_{1,4}} + 7 p_{B_{4,7}} + 4 p_{B_{7,9}} + 2 p_{B_{9,11}} = c(P_S). \end{aligned}$$

Fig. 1. Example of mapping between batch sequences and paths on a graph. The path P_S shown corresponds to a batch sequence where the job set N is partitioned into $B_{14}, B_{47}, B_{79}, B_{9,11}$. The cost of P_S equals the total completion time for the batch sequence S. The arc-flow model should determine the arcs in the path *and* how the jobs must be partitioned over the arcs.

We recall the key modeling idea for arc-flow representations of $1|p\text{-batch}, s_j|\sum_j C_j$ from Alfieri et al. (2019, 2021). Consider a graph $G(V, A)$ with node and arc sets

$$\begin{aligned} V &= \{1, 2, \ldots, n+1\}, \\ A &= \{(i,k) \colon i, k \in V;\ i < k\}. \end{aligned} \tag{10}$$

The structure of a batch sequence S can be mapped onto a path P_S from node 1 to node $n+1$ in the graph G:

- each arc $(i,k) \in P_S$ corresponds to a batch B_{ik} — we label the batch with the arc tail and head — in the sequence where exactly $k - i$ jobs are processed; by suitably partitioning the job set N on the arcs of P_S, such that $|B_{ik}| = k - i$ and $\sum_{j \in B_{ik}} s_j \le b$, the path P_S correctly represents a feasible batch sequence;
- by assigning to each arc (i,k) in P_S the cost $c_{ik} = p_{B_{ik}}(n - i + 1) = \max\{p_j \colon j \in B_{ik}\}(n - i + 1)$,

the total completion time of the batch sequence S equals the computed cost of path P_S:

$$f(S) = \sum_{j \in N} C_j = \sum_{(i,k) \in P_S} c_{ik} = c(P_S).$$

We omit a detailed proof for the sake of conciseness, but refer to Alfieri et al. (2021) for the proof of a very similar result. Figure 1 conveys the idea.

2.2 A polynomial-size flow-based model

In this section, we formulate an ILP model that calls for finding a minimum-cost path P_S from node 1 to $n+1$ on graph G, partitioning jobs on the arcs of P_S.

$$\text{minimize} \quad \sum_{(i,k) \in A} (n-i+1)\pi_{ik} \tag{11}$$

subject to

$$\sum_{(i,k) \in A} x_{ik} - \sum_{(k,i) \in A} x_{ki} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n+1 \end{cases} \tag{12}$$

$$\sum_{(i,k) \in A} y_{ikj} = 1 \qquad \forall j \in N \tag{13}$$

$$\sum_{j=1}^{n} s_j y_{ikj} \leq b x_{ik} \qquad \forall (i,k) \in A \tag{14}$$

$$\sum_{j=1}^{n} y_{ikj} = (k-i) x_{ik} \qquad \forall (i,k) \in A \tag{15}$$

$$\pi_{ik} \geq p_j y_{ikj} \qquad \forall j \in N,\ (i,k) \in A \tag{16}$$

$$\pi_{ik} \geq 0 \qquad \forall (i,k) \in A \tag{17}$$

$$x_{ikt},\ y_{jikt} \in \{0,1\} \qquad \forall j \in N,\ (i,k) \in A \tag{18}$$

Each decision variables x_{ik} will be set to 1 if arc (i,k) belongs to P_S; constraints (12) are flow conservation constraints, with a unit flow routed from node 1 to node $n+1$; we note that this is the classical ILP formulation of a path construction problem. Each decision variable $y_{ikj} = 1$ if job j is scheduled in the batch B_{ik} corresponding to arc (i,k). Constraints (13) require that each job is assigned to an arc/batch, constraints (14) limits the total size packed into a batch, and constraints (15) enforce the correct cardinality of a batch. Variable π_{ik} is set by constraints (16) to the processing time of the longest job in the batch. The total completion time is expressed by (11), multiplying processing times π_{ik} for the number of still-to-be-scheduled jobs *in advance*, since completion time of each job j is the processing time of the batch it is currently scheduled in, plus the processing time of all previous batches.

2.3 A stronger model

Model (11)–(18) requires $\mathcal{O}(n^3)$ variables and constraints. It is a reasonably compact ILP, but computational experience shows that it still has severe limitations; although the lower bound delivered by the continuous relaxation of (11)–(18) is much higher than the one computed on (1)–(9), the optimality gap is still large, and `CPLEX` cannot solve (11)–(18) on large instances. A stronger arc-flow model can be developed at the cost of using a larger (but still polynomial) number of variables. Let $\mathcal{T} = \{p_j : j \in N\}$ the set of all processing times listed in the considered problem instance. The ILP model still calls for finding a minimum cost path P_S from node 1 to node $n+1$; we use a larger set of decision variables x_{ikt}, where $x_{ikt} = 1$ iff an arc (i,k) is in the paths and the corresponding batch B_{ik} has processing time $p_{B_{ik}} = t \in \mathcal{T}$. Another set of decision variables y_{jikt} is defined, with $y_{jikt} = 1$ iff job j is in the batch B_{ik} and the latter has batch processing time $p_{B_{ik}} = t$.

The cost c_{ikt} for an arc whose batch B_{ik} has processing time t is defined as

$$c_{ikt} = t(n-i+1).$$

The complete model can then be written as follows.

$$\text{minimize} \quad \sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} c_{ikt} x_{ikt} \tag{19}$$

subject to

$$\sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} x_{ikt} - \sum_{\substack{(k,i) \in A \\ t \in \mathcal{T}}} x_{kit} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \dots, n \\ -1 & i = n+1 \end{cases} \tag{20}$$

$$\sum_{\substack{(i,k) \in A \\ t \in \mathcal{T}}} y_{jikt} = 1 \qquad \forall j \in N \tag{21}$$

$$\sum_{j \in N} s_j y_{jikt} \leq b x_{ikt} \qquad \forall (i,k) \in A,\ t \in \mathcal{T} \tag{22}$$

$$\sum_{j \in N} y_{jikt} \leq (k-i) x_{ikt} \qquad \forall (i,k) \in A,\ t \in \mathcal{T} \tag{23}$$

$$y_{jikt} \leq x_{ikt} \qquad \begin{array}{l} \forall j \in N,\ (i,k) \in A, \\ t \in \mathcal{T} \end{array} \tag{24}$$

$$x_{ikt},\ y_{jikt} \in \{0,1\} \qquad \begin{array}{l} \forall j \in N,\ (i,k) \in A, \\ t \in \mathcal{T} \end{array} \tag{25}$$

Objective function (19) requires to minimize the sum of total completion time for all selected arcs. Constraints in (20) represents a classical flow model with unitary flow, from source 1 to destination $n+1$, counting as positive the flow to outgoing arcs and as negative the flow from incoming arcs. Exact partitioning for all jobs is enforced in (21), and capacity constraints (22) are defined for all selected arcs.

The number of variables for this model grows theoretically to $\mathcal{O}(n^4)$, but the number of variables can be sensibly trimmed since not all pairs i,k $(i < k)$ can correspond to a batch because of the machine capacity limit. Also, constraints (24) can be used as *lazy constraints*, to be dynamically separated when needed.

3. VARIABLE ROUNDING HEURISTIC

Once programs (11)–(18) or (19)–(25) has been solved, a lower bound is available; a simple strategy for getting an upper bound could be setting all variables back to the binary type and solve the integer version of the problem by branch and bound, truncating the process when a limit on computation time is reached.

We also investigated a simple strategy based on rounding fractional variables in order to generate feasible solutions within shorter computation times. The value of such solution is called `VR-UB` in the following. This approach is inspired by Mourgaya and Vanderbeck (2007) where rounding heuristics are applied to vehicle routing problems.

3.1 Variable Rounding for (19)–(25) *model*

Using a basic rounding approach, we build a partition path by rounding flow values, starting from the source and moving towards the sink node. The full procedure is as follows (a detailed pseudocode is not included here for the sake of conciseness and will be presented at the conference).

- Given the optimal fractional flow on the problem, we search for tuple (i, k, t) corresponding to arc (i, k) with processing time t and maximal nonzero flow that lies as near as possible to the source, and round its flow value to 1. This corresponds to the fixing of a variable x_{ikt} to 1.
- We search for all tuples (j, i, k, t) corresponding to jobs j associated to the fixed arc (i, k) with processing time t that present nonzero flow; potentially those jobs can be *more* than the required $k - i$ jobs to fill the arc.
- Amongst all subsets of the jobs found in the previous step, we select the *best* combination that can fit in a batch, that is of length $k-i$, and with *maximum flow value* amongst the sum of flow values for all included jobs; those jobs will have their flow value rounded to 1. This corresponds to the fixing of some variables y_{jikt} to 1. This step is admittedly combinatorial, but never involves more than a handful of jobs.
- The problem is optimized again to obtain new flow values for the remaining arcs and jobs.

We then iterate this rounding step until an arc reaching the sink node $n + 1$ is fixed, thus completing a feasible solution.

3.2 Variable Rounding for (11)–(18) *model*

The underlying approach is similar to the one implemented for the other model: we build a partition path by rounding flow values, starting from the source and moving towards the sink node. Difference between this procedure and the one described for (19)–(25) model are as follows.

Since variables relative to arcs and jobs does not depend on a specific processing time, we have to fix the variable x_{ik} that corresponds to the arc (i, k) with maximal nonzero flow to 1, search for its associated jobs j with nonzero flow, select some of them with the same criteria as before, and fix those variables y_{ikj} to 1.

4. COMPUTATIONAL RESULTS

Very few heuristics and/or relaxations are available in literature for $1|p\text{-batch}, s_j|\sum_j C_j$, among them are both a relaxation and a greedy heuristic (Uzsoy, 1994; Azizoglu and Webster, 2000) in the context of a branch and bound algorithm for $1|p\text{-batch}, s_j|\sum_j C_j$. Such procedures are labeled `AW-LB` and `AW-UB` respectively.

The following naming convention is valid for all tables, for the results of both models. All gaps in the following tables are evaluated by the relative difference between upper bound and lower bound, using the following formula.

$$\text{Gap} = \frac{UB - LB}{UB}$$

Gap values for the `AW-UB` procedure by Azizoglu and Webster (2000) are evaluated using `AW-LB` as the lower bound.

The values for the lower bound obtained by solving the continuous relaxation of the problem are denoted by `CR-LB`, and with `BB-UB` we refer to the upper bound found by running the integer version of the problem. The gap values in this case are evaluated using the best continuous lower bound returned by the solver after its termination (optimum found, or computation time limit reached). Time limit for the solver is set to 600 seconds.

The values for the upper bound obtained by Variable Rounding, as described in Section 3, are denoted by `VR-UB`. The gap values in this case are evaluated using `CR-LB` as the lower bound.

4.1 Testing environment

All the tests ran in a Linux environment with `Intel Core i7-6500U CPU @ 2.50GHz` processor; all algorithms (`AW-LB`, `AW-UB` and `VR-UB`) have been implemented in `Python 3.6`; the LP-ILP solver used was `CPLEX 12.8`.

A number of test instances were generated following the de-facto standard for this type of parallel batching problems.

- The processing times p_j were randomly drawn from the uniform discrete distribution $[1, 100]$.
- The machine capacity was fixed to $b = 10$.
- The job sizes s_j were randomly drawn from four possible uniform discrete distributions, labeled by σ:

$$\sigma_1 : s_j \in [1, 10] \qquad \sigma_3 : s_j \in [3, 10]$$
$$\sigma_2 : s_j \in [2, 8] \qquad \sigma_4 : s_j \in [1, 5]\,.$$

A batch with 10 instances for each σ class was generated, considering an increasing number of jobs $n \in [10, 20, 30, 40, 50]$. The following tables report average values evaluated over all (n, σ) combinations.

4.2 Results for model (11)–(18)

Table 1 contains the average number of nodes opened during evaluation of `BB-UB` by the solver, and the number of optima found, all within the 600 seconds threshold.

Unfortunately, performance of this model is poor: the branch and bound requires to explore a lot of nodes, and even with only 20 jobs the solver cannot find even an optimal result for all instances.

Comparison between upper bounds, in Table 2, confirms the poor performance of this model. Although the lower

Table 1. Number of opened branch and bound nodes and optima found for model (11)–(18).

Param		Avg Nodes	# Opt
n	σ	BB-UB	BB-UB
10	σ_1	7472	10
	σ_2	22256	10
	σ_3	3892	10
	σ_4	5440	10
20	σ_1	148518	0
	σ_2	613856	0
	σ_3	930158	0
	σ_4	102653	0

bound CR-LB is very fast to execute, even with 20 jobs where in less than one second the result is found, the integer optimum BB-UB is very hard to find. After the entire allotted time limit of 600 seconds, the gap obtained by the solver is, in the majority of the cases, even worse than the gap obtained using the Azizoglu and Webster (2000) polynomial bounds.

Since the quality of the lower bound CR-LB is bad, even the Variable Rounding upper bound VR-UB is of poor quality. Here, even for 10 jobs the obtained gap is worse than the gap obtained with the aforementioned polynomial bounds.

Table 2. Comparison of bounds quality and execution times for model (11)–(18).

Param		Avg Times (s)			Avg Gaps		
n	σ	CR-LB	BB-UB	VR-UB	AW-UB	BB-UB	VR-UB
10	σ_1	0.02	3.27	0.02	0.37	0.00	0.67
	σ_2	0.01	3.14	0.01	0.26	0.00	0.48
	σ_3	0.01	0.72	0.01	0.22	0.00	0.36
	σ_4	0.03	3.60	0.02	0.37	0.00	0.70
20	σ_1	0.30	600.00	0.61	0.33	0.46	0.70
	σ_2	0.20	600.00	0.19	0.26	0.32	0.59
	σ_3	0.18	600.00	0.11	0.21	0.16	0.44
	σ_4	0.31	600.00	1.07	0.37	0.57	0.81

We decided to test this model only for instances with 10 and 20 jobs, given these performances.

4.3 Results for model (19)–(25).

Table 3 contains the average number of nodes opened during evaluation of BB-UB by the solver, and the number of optima found, all within the 600 seconds threshold.

The performance of this model is way better than the previous: the branch and bound requires to explore a very small number of nodes. For 10 jobs, in fact, the entire problem is solved in the root node, in the majority of the cases.

However, since the model is heavier than the previous one, even the computation of the root node is intensive. As we can see for 50 jobs, for the hardest distribution σ_4 the solver uses the entirety of its 600 seconds allotted time only for the root node evaluation.

Comparison between upper bounds, in Table 4, shows the interesting performance of this model. The lower bound CR-LB is slightly slower to execute than in the previous

Table 3. Number of opened branch and bound nodes and optima found for model (19)–(25).

Param		Avg Nodes	# Opt
n	σ	BB-UB	BB-UB
10	σ_1	0	10
	σ_2	0	10
	σ_3	1	10
	σ_4	0	10
20	σ_1	753	10
	σ_2	1036	10
	σ_3	783	10
	σ_4	442	10
30	σ_1	648	7
	σ_2	6071	7
	σ_3	1846	10
	σ_4	143	2
40	σ_1	188	1
	σ_2	138	1
	σ_3	5189	3
	σ_4	2	0
50	σ_1	136	1
	σ_2	642	1
	σ_3	4752	4
	σ_4	0	0

case, but in return we have a way better Variable Rounding upper bound VR-UB in all cases, having a gap lower than 0.10 and performing way better than the gap evaluated with the Azizoglu and Webster (2000) polynomial bounds. When the upper bound BB-UB starts breaking the 600 seconds threshold in its evaluation, for 40 and 50 jobs, the gap evaluated by VR-UB is very competitive in comparison, even surpassing the branch and bound for distribution σ_4 that is known to be hard. It is worth noting that the time required for the Variable Rounding upper bound is way lower than the branch and bound one, with an order of magnitude very similar to the continuous lower bound.

Increasing the number of jobs for the instances, it is clear that VR-UB will be even better in time performance than BB-UB, and will always retain its good gap quality.

5. CONCLUSIONS

Our analysis over two arc-flow models for the parallel batching problem $1|p\text{-batch}, s_j|\sum_j C_j$ shows that model (19)–(25) is capable to deliver a good continuous relaxation lower bound CR-LB and a good Variable Rounding upper bound VR-UB, leading to excellent performance in gap quality within an acceptable amount of time.

In Table 5 we compared the lower bound CR-LB obtained with the best performing model, (19)–(25), with the current state-of-the-art lower bound known in literature, CG-LB obtained by Alfieri et al. (2021). The comparison ratio is evaluated by dividing the two lower bounds.

$$Ratio = \frac{\texttt{CR-LB}}{\texttt{CG-LB}}$$

The higher this fraction is, the better model (19)–(25) performs.

Table 4. Comparison of bounds quality and execution times for model (19)–(25).

Param		Avg Times (s)			Avg Gaps		
n	σ	CR-LB	BB-UB	VR-UB	AW-UB	BB-UB	VR-UB
10	σ_1	0.04	0.12	0.02	0.37	0.00	0.04
	σ_2	0.02	0.12	0.01	0.26	0.00	0.05
	σ_3	0.01	0.07	0.01	0.22	0.00	0.04
	σ_4	0.06	0.14	0.01	0.37	0.00	0.02
20	σ_1	0.93	41.26	1.04	0.33	0.00	0.09
	σ_2	0.57	14.03	0.57	0.26	0.00	0.07
	σ_3	0.33	4.38	0.29	0.21	0.00	0.04
	σ_4	1.60	100.97	1.51	0.37	0.00	0.04
30	σ_1	4.94	238.89	8.44	0.32	0.01	0.08
	σ_2	2.71	231.89	3.92	0.28	0.00	0.06
	σ_3	1.58	19.91	1.71	0.22	0.00	0.03
	σ_4	9.03	516.16	13.43	0.34	0.05	0.06
40	σ_1	24.64	576.59	55.11	0.37	0.06	0.08
	σ_2	16.58	567.71	48.09	0.27	0.05	0.09
	σ_3	8.12	531.04	14.13	0.24	0.01	0.07
	σ_4	43.00	600.00	81.08	0.29	0.11	0.05
50	σ_1	63.40	583.14	123.19	0.35	0.09	0.07
	σ_2	26.33	591.03	37.28	0.24	0.01	0.07
	σ_3	15.49	367.34	11.06	0.19	0.00	0.03
	σ_4	74.65	600.00	213.89	0.27	0.16	0.05

Table 5. Comparison between lower bound CR-LB obtained by the (19)–(25) model, and lower bound CG-LB obtained by the column generation approach described in Alfieri et al. (2021).

Param		Avg Ratio
n	σ	CR-LB/CG-LB
10	σ_1	0.98
	σ_2	0.96
	σ_3	0.97
	σ_4	0.99
20	σ_1	0.95
	σ_2	0.96
	σ_3	0.98
	σ_4	0.99
30	σ_1	0.96
	σ_2	0.97
	σ_3	0.98
	σ_4	0.99
40	σ_1	0.97
	σ_2	0.96
	σ_3	0.97
	σ_4	0.99
50	σ_1	0.97
	σ_2	0.97
	σ_3	0.99
	σ_4	0.99

As it can be seen, the quality of our lower bound CR-LB is excellent, and stays tightly close to the state-of-the-art lower bound delivered by column generation; especially for the hardest distribution σ_4. Thus we claim that model (19)–(25) is promising and deserves further study, in order to improve its performances; in particular we are studying sifting techniques for speeding up the solution of the continuous relaxation, reducing as well the computational time required by the Variable Rounding heuristic. Also, the extension to multiple parallel machines (identical or unrelated) variant of the problem would be easily handled. Extension of this approach also to the *weighted* total completion time case is under investigation.

REFERENCES

Alfieri, A., Druetto, A., Grosso, A., and Salassa, F. (2019). Column generation for minimizing total completion time on a single machine with parallel batching. *IFAC-PapersOnLine*, 52(13), 969–974. doi: 10.1016/j.ifacol.2019.11.320. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

Alfieri, A., Druetto, A., Grosso, A., and Salassa, F. (2021). Column generation for minimizing total completion time in a parallel-batching environment. *Journal of Scheduling*, 24(6), 569–588. doi:10.1007/s10951-021-00703-9.

Azizoglu, M. and Webster, S. (2000). Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research*, 38(10), 2173–2184. doi:10.1080/00207540050028034.

Druetto, A. and Grosso, A. (2022). Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine. *Computers & Operations Research*, 139(105639). doi: 10.1016/j.cor.2021.105639.

Mathirajan, M. and Sivakumar, A.I. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9), 990–1001. doi:10.1007/s00170-005-2585-1.

Mourgaya, M. and Vanderbeck, F. (2007). Column generation based heuristic for tactical planning in multi-period vehicle routing. *European Journal of Operational Research*, 183(3), 1028–1041. doi: 10.1016/j.ejor.2006.02.030.

Mönch, L., Fowler, J.W., Dauzère-Pérès, S., Mason, S.J., and Rose, O. (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6), 583–599. doi:10.1007/s10951-010-0222-9.

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

Potts, C.N. and Kovalyov, M.Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2), 228–249. doi:10.1016/S0377-2217(99)00153-8.

Trindade, R.S., de Araújo, O.C.B., and Fampa, M. (2021). Arc-flow approach for single batch-processing machine scheduling. *Computers & Operations Research*, 134, 105394. doi:10.1016/j.cor.2021.105394.

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.