Croatian Operational Research Review
CRORR **7**(2016), 291–302

Heuristics for batching and sequencing in batch processing machines

Chuda Basnet[1,†]

[1]*Department of Management Systems, Waikato Management School, The University of Waikato, Private Bag 3105, Hamilton, New Zealand*
E-mail: ⟨*chuda@waikato.ac.nz*⟩

Abstract. In this paper, we discuss the "batch processing" problem, where there are multiple jobs to be processed in flow shops. These jobs can however be formed into batches and the number of jobs in a batch is limited by the capacity of the processing machines to accommodate the jobs. The processing time required by a batch in a machine is determined by the greatest processing time of the jobs included in the batch. Thus, the batch processing problem is a mix of batching and sequencing – the jobs need to be grouped into distinct batches, the batches then need to be sequenced through the flow shop. We apply certain newly developed heuristics to the problem and present computational results. The contributions of this paper are deriving a lower bound, and the heuristics developed and tested in this paper.

Keywords: batch processing, permutation flow-shop, heuristic algorithms, environmental stress testing

Received: October 7, 2016; accepted: December 14, 2016; available online: December 30, 2016

DOI: 10.17535/crorr.2016.0020

1. Introduction

This paper addresses a variation of the flow shop sequencing problem, where jobs are formed into batches and each batch is then processed as a unit through the flow shop. The processing time of each batch in each machine is the greatest processing time required in the machine for jobs included in the batch. Once the batches are formed, the problem reduces to the classical flow shop problem.
This problem is of practical importance for manufacturing scenarios where multiple jobs are processed in batches, for example, in ovens or in treatment chambers, such as environmental stress testing (burn-in) of electronic products.

† Corresponding author

This problem, in essence, is a combination of the bin-packing problem, where jobs are batched within a given capacity, and the flow-shop problem, where the processing sequence of the batches is determined. However, these decisions cannot be made independently, given that the processing time depends on the jobs included in the batches.
In this paper, we derive a lower bound for the problem and develop search heuristics, based on the lower bound, to address the problem. In the next section, we present a short overview of the literature; this is followed by a formal problem statement. A lower bound is presented next, and subsequently followed by a presentation of the search heuristics. The final sections present computational results and a conclusion.

2. Literature review

Ikura and Gimple (1986) appear to be the first to have developed algorithms for burn-in operations on a single batch-processing machine, where the processing machine can process a limited number of jobs. Heuristics for minimising make-span were developed.
Lee, Uzsoy, and Martin-Vega (1992) considered an early version of this problem, where batches of jobs were processed in a single machine. The batches were constrained by the number of jobs that could be included, rather than the sizes of the jobs. Processing time for batches was fixed (independent of batch constituents). The objectives were mostly lateness-related, but they also considered make-span. They showed that some of these problems may be solved in polynomial time, but others are NP-hard. Uzsoy (1994) extended this model to allow for varying job sizes and varying process times. The sum of the job sizes in a batch was constrained by machine capacity, and the largest process time in a batch would be the processing time. The objective was makespan. This is BPP, as discussed in this paper, but with a single machine. Algorithms were developed using variations of bin-packing heuristics.
Brucker et al. (1997) considered a single machine where the number of jobs in a batch is constrained (without restrictions on the total job sizes), but the process time is determined by the longest processing time in the batch. Multiple complexity results were developed for various optimisation objectives.
Damodaran, Srihari, and Lam (2007) presented a simulated annealing approach to the single machine problem, as above. In their version, the batching was constrained by both total job size and number of jobs. The simulated annealing approach was shown to be a better approach in comparison to commercial optimisation software. Similarly, Geiger and Uzsoy (2008) used genetic techniques of cross-over and mutation to generate new scheduling rule parameters, which were then evaluated by simulation. The best rules found in this way were tested against currently available scheduling rules, with comparable results.

Damodaran and Srihari (2004) have proposed a mixed integer programming (MIP) model for the two-machine case of BPP. This model has both batching variables and sequencing variables. An example problem with 10 jobs was solved using commercial optimisation software.
Kashan and Karimi (2009) have improved on the MIP model proposed by Damodaran and Srihari (2004) by merging the batching and sequencing variables, thus considerably reducing the number of variables. Kashan and Karimi (2009) also extended the formulation to multiple machines. They developed multiple lower bounds of the problem and provided computational results on the quality and run-times of these bounds. However, no algorithms were offered for the solution of the problem.
A newer version of this problem divides the set of jobs into incompatible families. Thus, only compatible jobs can be batched together. Lei and Wang (2011) have proposed a neighbourhood search for this problem, where jobs are randomly swapped between batches and batches are randomly swapped in their processing sequence for a set number of iterations.
It is clear that although some work has been done for the single or two machine case of the BPP, particularly in terms of complexity results, there is not much work on development of suitable algorithms and heuristics for the BPP. This paper is an attempt to contribute to this area. In this paper, we address a version of the batch processing problem, where batching is constrained by the machine capacity and batch processing time is determined by the largest processing time in the batch. In the next section, we present a formal statement of the problem.

3. Problem statement

Batch Processing Problem (BPP)

In this formulation, batches are formed in the order of their processing sequence in the machines. The following formulation is adopted from Kasham and Karimi (2009).

Notation

$i \in 1 \ldots n$, index of jobs

$j \in 1 \ldots m$, index of machines, in order of processing in the flow-shop.

$b \in 1 \ldots n$, index of batches, in the sequence of processing in each machine.

Sj = Capacity of machine j, $j = 1, \ldots, m$

s_i = Size of job i, $\quad i = 1 \ldots n$, $s_i \leq \min(S_j)$

p_{ij} = Processing time of job i in machine j

P_{bj} = Processing time of batch b in machine j

C_{bj} = Completion time of batch b in machine j
C_{max} = Makespan
x_{bi} = 1 if batch b includes job i; 0 otherwise

Model
The optimisation problem is:

Minimise makespan, Min $C_{\max}$
s.t.

$$\sum_{b=1}^{n} x_{bi} = 1,\ i = 1,....,n$$

A job can be assigned to a particular batch only.

$$\sum_{i=1}^{n} s_i x_{bi} \le Min(S_1 ..., S_m),\quad b = 1,...,n$$

The total of job sizes in a batch cannot exceed the minimum capacity of machines

$$P_{bj} \ge x_{bi} p_{ij}, i = 1,..., n; b = 1,..., n;\ j = 1,...., m$$

Processing time for a batch in a machine is determined by the job with the longest processing time included in the batch.

$$C_{b1} = \sum_{k=1}^{b} P_{k1}, b = 1,...., n$$

All batches may be processed in the first machine, after the earlier-sequenced batch is finished.

$$C_{bj} \ge C_{b-1,\, j} + P_{bj}\ ,\ b = 2,...., n;\ \ j = 2,..., m$$

A batch cannot be processed in a machine, until the earlier-sequenced batch is finished processing.

$$C_{bj} \ge C_{b,\, j-1} + P_{bj}\ ,\ b = 1,...., n;\ \ j = 2,..., m$$

A batch cannot be processed in a machine, until it has finished processing in the earlier machine.

$$C_{\max} \geq C_{bn}, b = 1,...,n$$

This constraint defines C_{max}.

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

The decision variable, x_{ij} is binary.
The usual assumptions for this problem are:

No pre-emption is allowed.
All the jobs are available at the beginning of the schedule.
Batches remain unchanged once they are formed.
The underlying flow-shop problem is the permutation flow-shop, i.e., the batches are processed in the same order in each machine.

The BPP is NP-hard (Damodaran and Srihari, 2004), which justifies the search for efficient heuristics to solve the problem. This paper contributes to this endeavour. In the next section, a lower bound for this problem is presented. This lower bound is applicable to a partial solution of the problem – some jobs have been formed into batches but other jobs are yet to be batched.

4. A lower bound

Some authors have provided lower bounds for the permutation flow-shop problem. Based on these lower bounds, a lower bound for a partial solution of the batch processing problem may be created. The lower bounds are created by determining the release dates or "heads" and delivery times or "tails" of the batches.
Our lower bound is designed to find a lower bound of partial solutions, where some jobs have been assigned into batches and the remaining jobs are yet to be assigned.
Assume some jobs (i= 1,...k) are assigned to batches (b = 1, ..., p). The remaining jobs (i = k+1, ..., n) are "free", yet to be assigned. In this scenario, the minimum number of additional batches required for the free jobs is:

$$q = \left\lceil \frac{\sum_{i=k+1}^{n} s_i}{Min.(S_1 ... S_j)} \right\rceil$$

Thus, additional q new batches ($b = p + 1, \ldots, p+q$) may be created successively in the following manner.

$$P_{bj} = (b-p)th \text{ minimum of } (p_{ij}), i = k+1,...,n;\ j = 1,...m, b = p+1,...p+q$$

Thus, a total of $p+q$ batches of jobs are formed. The lower bound of this batching assignment may be calculated using any available lower bound of the permutation flow-shop problem. In this paper, we use the LB1 provided by Ladhari and Haouaria (2005), as follows.

Release dates (or heads), h_{bj}:

$$h_{b1} = 0, b = 1,..., p+q$$

$$h_{bj} = \sum_{l=1}^{j-1} P_{bl}, b = 1,..., p+q;\ j = 2,..., m$$

Delivery times (or tails), t_{bj}:

$$t_{bj} = \sum_{l=j+1}^{m} P_{bl}, b = 1,..., p+q; j = 1,..., m-1$$

$$t_{bm} = 0, b = 1,..., p+q$$

A lower bound is obtained for each machine j:

$$LB_j = \min_b (h_{bj}) + \sum_{b=1}^{p+q} P_{bj} + \min_b (t_{bj}),\ j = 1,..., m$$

The overall lower bound is:

$$LB = Min(LB_j)$$

As mentioned above, this lower bound has been developed for partial solutions of the BPP, where some jobs are batched and others are yet to be batched. The

lower bound applies to the optimum makespan, as presented in the earlier formulation of the BPP. This bound is useful for searches where batches are sequentially formed, as is the case in this paper.

5. Solution approaches

We describe below a few algorithms designed to solve the problem at hand. Then we will present a computational comparison of these algorithms.

5.1 Greedy heuristic (GH)

The batch processing problem (BPP) has two aspects: 1) batching, where one prefers to create fewer batches, and 2) sequencing, where one prefers to complete the processing of batches quickly. In this heuristic, we separate these tasks by forming the batches first. Straight forward pseudo-code to form the batches is given below:

Prepare a job-list of the jobs in non-increasing order of the total process times in all machines ($\sum_{j=1}^{m} p_{ij}$)

Until all the jobs are assigned into batches

Create a new batch of jobs

While the total of size of the first job in the job-list and sizes of all jobs already included in the new batch is less than minimum capacity of machines

Include the first job in the new batch

Remove the first job from the job-list

End While

End Until

The above heuristic assigns all jobs into feasible batches. These batches may then be sequenced by using any of the available flow-shop sequencing heuristics. In this paper, we use the well-known Cambell, Dudek, and Smith (1970) (CDS) algorithm for this purpose.

5.2 Bounded enumeration heuristic (BEH)

This heuristic enumerates maximal subsets of jobs to form batches. A maximal subset is defined here as a batch of jobs, such that including any other job into this batch would violate the capacity constraints of the machines.

The bounded enumeration heuristic exhaustively enumerates all maximal subsets in its search for the best solution to the BPP. Pseudo-code for the exhaustive search is given below.

```
    Enumerate all the maximal subsets of jobs.
For each of the maximal subsets,
        Enumerate all maximal subsets of the remaining jobs.
        For each of these maximal subsets,
            Enumerate all maximal subsets of the remaining jobs.
                For each of these maximal subsets,
                 (etc., recursively, until there are no more remaining jobs)
```

At each stage of the iteration of the above algorithm, we have a partial solution, i.e., some jobs have been assigned into batches (or maximal subsets) and some jobs are not assigned into batches yet. In an attempt to reduce the search space, the lower bound developed earlier is used at each iteration of the recursive algorithm to evaluate the partial solution. If the lower bound is higher than a previously found solution, the partial solution is discarded, and the search backtracks to the earlier node.
Once all jobs have been assigned into batches, the CDS algorithm is used to sequence the batches and determine the makespan. The least makespan found in the search is saved together with the associated batching.
It is important to clarify that neither maximal subsets nor CDS sequencing are requirements for an optimal solution of the BPP. Thus, even though the BEH heuristic described here enumerates maximal subsets, the heuristic cannot guarantee optimal solutions and remains a heuristic.

5.3 Truncated search heuristic (TSH)

It is obvious that the BEH heuristic, given above, conducts an exhaustive search of maximal subsets. The search-space can be quite large. One approach to reduce the search space is to truncate the search tree aggressively. We use a “beam” search, which is a breadth-first approach, where all the child-nodes are evaluated

for possible truncation. Only a fixed number (so-called "beam width") of child-nodes survive, which are the most promising nodes. These nodes then propagate further child-nodes.

Figure 1 shows the search tree in a beam search with beam width 2; meaning only two nodes are allowed to sprout at any one stage. The first node 1 sprouts three nodes 2, 3, and 4. These three nodes are evaluated, and node 2 is not considered very promising and dropped from further consideration. Nodes 3 and 4 together sprout nodes 5, 6, 7, 8, and 9. After evaluation, only nodes 7 and 8 survive at this stage. Similarly, nodes 12 and 13 are deleted after evaluation at the third stage.

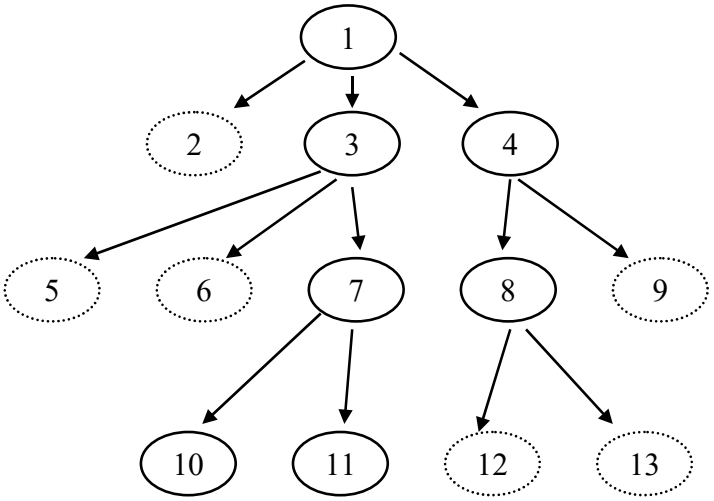

Figure 1: *Scheme of a beam search*

Depending on the beam width, the search tree can be truncated quite severely. Each node of the search tree represents a partial solution. The success of this approach depends obviously on the evaluation of the nodes for possible propagation. We use the lower bound, developed earlier, to evaluate nodes for possible truncation.

6. Computational evaluation

To evaluate the heuristics described above, we generated some problems and applied the above algorithms to them. A beam width of 4 was employed for the

truncated search. For the sake of comparison, CPLEX, a commercial optimisation package, was also used to find the optimal solutions.

The problem generator generated a number of jobs in the range of [6, 100] and a number of machines in the range of [3, 100]. Job sizes were randomly generated in the range of [1, 5]. Process times of jobs were randomly generated in the range of [1, 25]. Machine capacity of all machines was fixed at 25.

The problem generator and the heuristics were all programmed in Microsoft Visual C and run in a PC (i5-4570 CPU, 3.20 GHz, 8 GB RAM, 64-bit OS).

The computational results are presented in the following table. Problem sizes are shown in the table as [number of jobs, number of machines]. Fifteen problems of each size were generated.

Problem Size	Greedy Heuristic		Truncated Search		Bounded Enumeration		CPLEX Package		
	Cmax	Time	Cmax	Time	Cmax	Time	Cmax	Time	MIPGAP
[6, 3]	77	<1	71.4	<1	70.2	<1	70.2	85	0
[6, 5]	117.4	<1	109.8	<1	107.7	1.1	107.7	117	0
[6, 25]	533	<1	501.2	<1	444.8	13.6	443.5	1594	0
[10, 5]	151	<1	136.5	2.1	132.3	139.6	132.2	5773	0
[10, 25]	568.7	<1	529.9	5.2	505.3	5801	503.6	300000	.106
[10, 50]	1081.7	<1	1008.7	11.5	908.3	13978	909.5	300000	.161
[20, 25]	644.4	<1	596.27	2881.3	♥		601.5	300000	.903
[20, 50]	1168.1	<1	1083.4	8028.1	♥		1130	300000	.920
[50, 25]	863.3	<1	♥		♣		880.9	300000	.958
[50, 50]	1411.9	<1	♥		♣		1439.4	300000	.966
[100, 50]	1221.4	1	♥		♣		1791.5	300000	1.0
[100, 100]	1780.3	2	♥		♣		2941.9	300000	1.0

Time = Avg. computer time in milliseconds; ♣ = Ran out of computer memory;
♥ = Ran out of allotted time of 5 min.
MIPGAP = (Current feasible solution of CPLEX – Current lower bound of CPLEX) / Current feasible solution of CPLEX

Table 1: *Computational results*

The solutions obtained by the three heuristics for the smaller problems are quite comparable. The bounded enumeration performs close to the optimisation package, but requires much less time. However, it runs out of space and time

relatively quickly. The truncated search can solve larger problems but not significantly larger ones. A greedy search finds solutions relatively quickly, however the solution quality is poor for the higher problem sizes. As can be expected, the quality of the solution depends on the computational time spent.

7. Conclusion

We have presented the batch processing problem and some heuristics to solve the problem at hand. The main contribution of the paper has been a lower bound for partial solutions to the problem, which was employed in an enumeration search and truncated search.
Computational experiments suggest that the bounded enumeration heuristic performs close to that of a commercial package but at a fraction of the time. The greedy heuristic performs well for small problems, but the solutions are inadequate for larger problems. The truncated search performed close to the bounded enumeration but was unable to reduce the search space significantly.
The success of both the bounding and the truncation depends on the evaluation function used. The lower bound, developed and tested in this paper, did perform its intended function, but further work needs to be carried out to develop a better lower bound.

Acknowledgement

IBM Corporation for providing the IBM ILOG CPLEX software for educational / research use.

References

[1] Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C., Tautenhahn, T., & Van De Velde, S. (1997). Scheduling a Batching Machine.

[2] Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. Management Science, 16(10), B-630.

[3] Damodaran, P., & Srihari, K. (2004). Mixed integer formulation to minimize makespan in a flow shop with batch processing machines. Mathematical and Computer Modelling, 40(13), 1465-1472.

[4] Damodaran, P., Srihari, K., & Lam, S. S. (2007). Scheduling a capacitated batch-processing machine to minimize make-span. Robotics and Computer-Integrated Manufacturing, 23(2), 208-216.

[5] Geiger, C. D., & Uzsoy, R. (2008). Learning effective dispatching rules for batch processor scheduling. International Journal of Production Research, 46(6), 1431-1454.

[6] Ikura, Y., & Gimple, M. (1986). Efficient scheduling algorithms for a single batch processing machine. Operations Research Letters, 5(2), 61-65.

[7] Kashan, A. H., & Karimi, B. (2009). An improved mixed integer linear formulation and lower bounds for minimizing makespan on a flow shop with batch processing machines. The International Journal of Advanced Manufacturing Technology, 40(5-6), 582-594.

[8] Lee, C. Y., Uzsoy, R., & Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. Operations Research, 40(4), 764-775.

[9] Ladhari, T., & Haouari, M. (2005). A computational study of the permutation flow shop problem based on a tight lower bound. Computers & Operations Research, 32(7), 1831-1847.

[10] Lei, D., & Wang, T. (2011). An effective neighborhood search algorithm for scheduling a flow shop of batch processing machines. Computers & Industrial Engineering, 61(3), 739-743.

[11] Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. International Journal of Production Research, 32(7), 1615-1635.