ISSN :2584-2706 DOI: https://doi.org/10.67317/IJMSRT26NOV049

Eligibility-Aware Scheduling of Parallel Production Machines with Planned Downtime

Adenuga Michael Ayotunde
Mechanical Engineering, Ajayi Crowther University, Oyo Town, Nigeria

Corresponding author

Abstract

Background. Production jobs can have different processing times across machines, and equipment qualification or tooling restrictions may limit where each job can run. Planned maintenance further constrains machine assignments and job sequences.

Methods. This computational study compares three dispatching rules for minimising makespan on three unrelated parallel machines with a planned unavailability window on each machine. The rules use the same earliest-completion assignment step and differ in how they prioritise jobs. An exact subset dynamic program provides a benchmark. Sixty synthetic eight-job batches were evaluated under three downtime durations, yielding 180 batch–calendar combinations.

Results. When each job had one or two eligible machines, the eligibility-first rule reduced mean relative optimality gaps from 4.80%, 13.28%, and 11.21% under the longest-job-first rule to 1.41%, 9.21%, and 7.80% for downtime durations of zero, six, and twelve time units, respectively. When jobs had two or three eligible machines, longest-job-first achieved a lower mean gap without downtime, while the two rules gave nearly equal mean gaps with downtime.

Conclusion. In these generated batches, eligibility-first priority was most beneficial when machine options were scarce. The results identify a condition under which planners can compare dispatch rules against exact schedules for small production batches.

Keywords: *Production Scheduling, Unrelated Parallel Machines, Machine Eligibility, Planned Maintenance, Dispatching Rules, Makespan.*

I. INTRODUCTION

A planner assigning a batch of jobs to parallel production machines often works with a qualification matrix and a maintenance calendar. A job may require a fixture fitted to only one machine, while another may run on several machines at different speeds. If the planner commits a flexible job to a machine early, that assignment can delay a job with fewer alternatives. Even when the machine can process both jobs, a planned outage can make the order of the jobs consequential. For example, if a job would overlap the beginning of maintenance, it must wait until the machine reopens under a nonpreemptive production policy.

Lenstra *et al.* [1] formalised the makespan minimisation on unrelated parallel machines (UPM), which is when the processing time of a job is dependent on the machine it is processed on. Lee [2] and Schmidt [3] present scheduling problems under limited machine availability and the significance of the time when the machines are unavailable. This study combines machine-dependent processing times, explicit eligibility, and fixed maintenance windows in a small-batch computational experiment. It systematically changes the eligibility and downtime of the machines and makes comparisons with an exact benchmark schedule for each instance, following the approach of Hall and Posner [4], which focused on controlled instance generation. The research question is whether it is beneficial to sort jobs by ascending eligibility count, compared to dispatching them in input order, or longest job first, when all three rules have a common machine assignment component that is calendar-aware. The experiment also tests whether the relative performance of the rules changes between more restrictive and less restrictive eligibility regimes. The contribution is a reproducible comparison of these specific rules and a compact exact benchmark for assessing their performance.

ISSN :2584-2706 DOI: https://doi.org/10.67317/IJMSRT26NOV049

II. MATERIALS AND METHODS

2.1 Production environment

A batch comprises n independent jobs available at time zero and m unrelated machines. E_j is the nonempty set of machines eligible for job j. The processing time p_{jk} is a positive integer on each eligible machine k. Each job is assigned to one eligible machine and processed in a single uninterrupted interval. Machines process at most one job at a time, and each machine k has a fixed half-open maintenance interval $[a_k, b_k)$. A job may end at a_k or begin at b_k. When $a_k = b_k$, the interval is empty. Setups, transfer times, shared tooling capacities, and release-date differences are outside the model. The objective is to minimise C_{max}, the latest job completion time.

2.2 Dispatching rules

Each dispatch rule maintains the completion time t_k of the last job appended to each machine, initially zero. For a job of duration p, define $F_k(t,p)$ as $b_k + p$ when $a_k < b_k$, $t < b_k$ and $t + p > a_k$. In all other cases, $F_k(t,p) = t + p$. The rule assigns jobs to eligible machines in the specified priority order, with the $F_k(t_k,p_{jk})$ value of the eligible machine that is the smallest, and updates the completion time for the assigned machine. The machine index is used to resolve assignment ties. All three rules attach jobs to the machine sequences, and idle time in front of a maintenance interval occurs only if the last job appended does not end before the next maintenance interval.

Table 1. Dispatch priorities

Rule	Primary job order	Tie break
Input order	Ascending job index	Job index
Longest job first	Descending minimum eligible processing time	Job index
Eligibility first	Ascending number of eligible machines	Descending minimum eligible processing time, then job index

The minimum processing time is the minimum time that a job can be processed in. Sorting according to this quantity is a machine-dependent variant on the longest-job-first idea. Eligibility 1st is applied based on the same duration priority in groups of equal eligibility. Input order is provided as a reference, as all the generated jobs are released simultaneously and the index in the input file is the first order in which they will appear.

2.3 Exact makespan benchmark

For each machine k and subset S, $D_k(S)$ is the earliest time at which the machine can finish all jobs in S. Set $D_k(\emptyset) = 0$. For each nonempty S, try each eligible job $j \in S$ as the final job. The candidate finish is $F_k(D_k(S \setminus \{j\}),p_{jk})$. $D_k(S)$ is the minimum candidate finish, with infeasible states assigned infinity. A later predecessor finish cannot produce an earlier completion for a fixed final job because F_k is nondecreasing in its first argument. Induction on the subset size therefore establishes that the recurrence evaluates the best sequence for each subset.

Enumerate each assignment of jobs to machines and evaluate the maximum of their corresponding subset completion times. The minimum of these maxima is the global optimum because assignments partition the jobs and the machine sequences are independent under the model assumptions. The benchmark uses $O(m\ n\ 2^n + (n + m)m^n)$ time and $O(m\ 2^n)$ working storage, including the assignment masks and enumeration state. It is used here for small batches. Each dispatch rule requires at most $O(n \log n + nm)$ time.

2.4 Instance generation and performance measures

The experiment used Python 3.12 and seed 73129. Each batch had eight jobs and three machines. For each job, three potential processing times were drawn independently and uniformly from the integers 3 through 15. For the larger eligibility regime, the set of eligible machines was chosen uniformly from the set {2,3}. It was uniformly sampled from {1, 2} in the scarce eligibility regime. Under this count, the sampling of machines was done uniformly without replacement. Processing times were set to zero for ineligible. One pseudorandom stream was used to generate 30 batches, in the different order mentioned above. The following maintenance times of 0, 6 and 12 units were used for each batch. The maintenance started at times 8, 12 and 16 on machines 1, 2 and 3, respectively. Pairing batches with durations provides calendar comparisons. There are six cells with 30 observations each; those cells represent duration and regime. The primary statistic was the arithmetic mean of instance-level relative gaps, computed as 100 × (rule makespan − exact makespan) / exact makespan. The analysis also recorded mean makespan, exact matches and the largest observed relative gap. Results are descriptive summaries of the generated instances.

The implementation asserted that the exact makespan never exceeded a dispatching rule's makespan. A separate enumeration of all eligible assignments and within-machine permutations verified the benchmark on a five-job test batch with rows (7,3,0), (5,0,4), (9,8,0), (0,7,6), and (4,0,5). This check found makespans of 10, 11 and 11 for durations of 0, 6 and 12, respectively, in agreement with the subset recurrence.

ISSN :2584-2706 DOI: https://doi.org/10.67317/IJMSRT26NOV049

III. RESULTS AND DISCUSSION

Table 2. Mean makespan in time units across 30 batches per cell

Eligibility	Downtime	Exact	Input order	Longest job first	Eligibility first
Broader	0	18.97	22.20	20.73	21.17
Broader	6	26.17	31.77	29.93	29.83
Broader	12	32.17	37.77	35.93	35.83
Scarce	0	27.40	29.73	28.57	27.73
Scarce	6	35.90	41.33	40.33	39.00
Scarce	12	41.90	47.33	46.33	45.00

Table 3. Mean relative gap from exact makespan in per cent

Eligibility	Downtime	Input order	Longest job first	Eligibility first
Broader	0	17.00	9.21	11.31
Broader	6	21.98	14.14	14.00
Broader	12	17.75	11.51	11.36
Scarce	0	9.01	4.80	1.41
Scarce	6	15.97	13.28	9.21
Scarce	12	13.53	11.21	7.80

Table 4. Number of batches attaining the exact makespan out of 30

Eligibility	Downtime	Input order	Longest job first	Eligibility first
Broader	0	6	10	7
Broader	6	2	6	6
Broader	12	2	6	6
Scarce	0	15	20	27
Scarce	6	5	9	11
Scarce	12	5	9	11

3.1 Effect of eligibility on dispatch performance

Eligibility first had the lowest mean optimality gap in all three scarce-eligibility cells. Its advantage over longest job first ranged from 3.39 to 4.07 percentage points. With no downtime, eligibility first attained the exact makespan on 27 of 30 batches, compared with 20 for longest job first and 15 for input order. With six or twelve units of downtime, eligibility first attained the optimum on 11 batches in each cell. The larger gap under downtime suggests that maintenance interactions leave scope for changing sequences or assignments even after jobs are prioritised by eligibility.

Under broader eligibility and no downtime, longest job first achieved the smallest mean gap, 9.21%, compared with 11.31% for eligibility first. With downtime, the two rules' mean gaps were within 0.15 percentage points of one another. Eligibility first strictly improved on longest job first on eight of 30 scarce-eligibility batches at each downtime level. In these cells, the largest observed gap was 21.74%, 27.03% and 23.26% for durations of zero, six and twelve, respectively. Those values show the size of the losses that can remain in individual instances even when mean performance improves.

3.2 Effect of scheduled downtime

Mean exact makespan increased from 18.97 to 26.17 and 32.17 units in the broader regime and from 27.40 to 35.90 and 41.90 units in the scarce regime. For a fixed batch, enlarging each downtime interval with its start held fixed removes feasible processing times. The exact makespan therefore cannot improve as duration increases. Relative gaps can move in either direction because they compare the change in the rule's makespan with the change in the exact makespan. In both eligibility regimes, each rule had a smaller mean percentage gap at twelve units of downtime than at six, even though every rule had a higher mean makespan.

3.3 Scheduling implications and limitations

A planner can build an eligibility matrix from machine–job combinations approved, enter processing-time estimates, and insert maintenance windows from the current calendar. The three rules make use of the same inputs; therefore, a planner can use the resulting makespans for each of the three rules before releasing a small batch. When there is only one or a few machines available for a job, scheduling that job early can free up capacity for flexible jobs. Its working mechanics are based on the exact time and how long the work is done. Processing long jobs early can be beneficial for a better workload balance if jobs are broad and can be processed in a timely fashion. The study employs a synthetic batch release time of 1, three machines, eight jobs and one fixed window per machine. Processing time and eligible subsets are used randomly, whereas industrial time qualification patterns can bunch jobs around machine families or machine-few fixtures. The results are for the append-only dispatch implementation described here. Rules that insert jobs into earlier gaps, or that jointly optimise maintenance timing or that take into account due dates, may

ISSN :2584-2706 DOI: https://doi.org/10.67317/IJMSRT26NOV049

have different behaviour. A plant-level evaluation would necessitate the observed processing times, eligibility records, calendar histories and an agreed operational measure of schedule quality.

IV. CONCLUSION

Ordering jobs by the number of eligible machines reduced the mean optimality gaps with ordering by input job size, in all six cells, and with ordering by LJF, in all three cells with scarce eligibility. The longest-job-first rule worked best in the broad eligibility cell with no downtime. This experiment demonstrates the use of exact small batch schedules in assessing priority rules for various qualification patterns and maintenance calendars for planning. Additional experiments using plant information would confirm if the differences occur using real routing and processing-time scenarios. Data and Code Availability

All instances are generated from the seed and procedure in Appendix A. The appendix reproduces the reported aggregate results with Python 3.12 and the standard library.

ACKNOWLEDGEMENT

NONE

Conflict of Interest
The authors declare no conflicts of interest.

Funding
This study was self-funded and received no external funding.

REFERENCES

[1]. J. K. Lenstra, D. B. Shmoys, and É. Tardos, “Approximation algorithms for scheduling unrelated parallel machines,” Mathematical Programming, vol. 46, pp. 259–271, 1990. https://doi.org/10.1007/BF01585745
[2]. C.-Y. Lee, “Machine scheduling with an availability constraint,” Journal of Global Optimisation, vol. 9, pp. 395–416, 1996. https://doi.org/10.1007/BF00121681
[3]. G. Schmidt, “Scheduling with limited machine availability,” European Journal of Operational Research, vol. 121, no. 1, pp. 1–15, 2000. https://doi.org/10.1016/S0377-2217(98)00367-1
[4]. N. G. Hall and C. N. Posner, “Generating experimental data for computational testing with machine scheduling applications,” Operations Research, vol. 49, no. 6, pp. 854–865, 2001. https://doi.org/10.1287/opre.49.6.854.10014

APPENDIX A REPORDUCTION CODE

Save the code as a Python file and execute it with Python 3.12. Each output row gives the eligibility regime flag (True for scarce), downtime duration, four mean makespans (exact, input order, longest job first, eligibility first), three mean gaps, three exact-match counts, two eligibility-first win counts (against input order and longest job first), and three worst observed gaps. The three-rule statistics follow the order shown in Table 1.

```
from itertools import product
from random import Random
from statistics import mean
from math import inf

def finish(t, p, window):

a, b = window  if a < b and t < b and t + p > a:

 t = b

 return t + p

def optimum(P, W):

 n = len(P)

 size = 1 << n
```

ISSN :2584-2706 DOI: https://doi.org/10.67317/IJMSRT26NOV049

```
 dp = [[0] + [inf] * (size - 1) for _ in W]
 for m, window in enumerate(W):
 for mask in range(1, size):
 for j in range(n):
 bit = 1 << j
 if mask & bit and P[j][m]:
 t = finish(dp[m][mask ^ bit], P[j][m], window)
dp[m][mask] = min(dp[m][mask], t)
 best = inf
for assignment in product(range(3), repeat=n):
 masks = [0, 0, 0]
for j, m in enumerate(assignment):
 masks[m] |= 1 << j
best = min(best, max(dp[m][mask] for m, mask in enumerate(masks)))
 return best
def dispatch(P, W, order):
 T = [0, 0, 0]
 for j in order:
t, m = min((finish(T[m], p, W[m]),
 for m, p in enumerate(P[j]) if p)
 T[m] = t
 return max(T)
```

APPENDIX A CONTINUED

```
def main():
 rng = Random(73129)
 for scarce in (False, True):
 cases = []
for _ in range(30):
  P = []
 for j in range(8):
 durations = [rng.randint(3, 15) for _ in range(3)]
  k = rng.choice((1, 2) if scarce else (2, 3))
```

ISSN :2584-2706 DOI: https://doi.org/10.67317/IJMSRT26NOV049

```
  E = rng.sample(range(3), k)
  P.append([p if m in E else 0
 for m, p in enumerate(durations)])
  cases.append(P)
  for d in (0, 6, 12):
  W = [(a, a + d) for a in (8, 12, 16)]
   rows = []
   for P in cases:
   fifo = range(8)
  lpt = sorted(fifo, key=lambda j: (-min(filter(None, P[j])), j))
  constrained = sorted(fifo, key=lambda j:
  (sum(p > 0 for p in P[j]), -min(filter(None, P[j])), j))
  opt = optimum(P, W)
 values = [dispatch(P, W, order) for order in (fifo, lpt, constrained)]
assert opt <= min(values)
 rows.append([opt, *values])
 means = [mean(col) for col in zip(*rows)]
  gaps = [mean(100 * (row[k] - row[0]) / row[0] for row in rows)
  for k in (1, 2, 3)]
 hits = [sum(row[k] == row[0] for row in rows) for k in (1, 2, 3)]
  wins = [sum(row[3] < row[k] for row in rows) for k in (1, 2)]
 worst = [max(100 * (row[k] - row[0]) / row[0] for row in rows)
 for k in (1, 2, 3)]
print(scarce, d, *(f"{x:.2f}" for x in means+gaps),
*hits, *wins, *(f"{x:.2f}" for x in worst))
if __name__ == "__main__":
 main()
```