

**008 - 0490**

A new scheduling algorithm for scheduling with limited  
available parallel processors

Oliver Braun

Department of Information and Technology Management,  
Saarland University, D - 66041 Saarbrücken, Germany.

e-mail: [ob@itm.uni-sb.de](mailto:ob@itm.uni-sb.de)

POMS 19th Annual Conference  
La Jolla, California, U.S.A.  
May 9 to May 12, 2008

5th March 2008

**Abstract**

We investigate the problem where a set of  $m$  parallel identical processors has to process a set of  $n$  independent jobs. Processors have periods of offline known non-availability. Jobs may be preempted. The objective is to minimize the makespan. We present an algorithm that constructs a schedule with the minimum schedule length in  $O(n + m \log m)$  time and produces at most  $(m - 1)Q$  preemptions with  $Q$  the number of intervals where processors are available. We show asymptotically and by computational results that our new algorithm outperforms an existing algorithm with respect to running time and induced number of preemptions.

**Key Words:** scheduling, makespan, parallel processors, non-availability

# 1 Introduction

Classical models in scheduling theory assume that a fixed set of processors is continuously available for processing throughout the planning horizon. Often this assumption is not justified. Due to random breakdowns, maintenance requirements, repairing activities or other constraints the processors may not be continuously available for processing. An overview of scheduling problems with respect to limited machine availability is given in [BBS04, SS98]. In this paper we investigate the classical scheduling problem  $P \mid pmtn \mid C_{max}$  with respect to limited available processors. There are  $m$  identical and limited available processors that have to process a set of  $n$  independent jobs with the objective of minimizing the makespan. Non-availability periods of the processors are known *offline*, i.e. prior to schedule generation. A rolling horizon planning system is an example of such a setting.

In case of *constant* processor availability patterns where processors are continuously available, the classical algorithm of McNaughton [McN59] is applicable. McNaughton shows that his algorithm is optimal with respect to the makespan and produces no more than  $m - 1$  preemptions. When precedence constraints are added, Liu and Sanlaville [LS95] show that the problem with precedence constraints among the jobs forming a set of chains can be solved in polynomial time applying processor sharing. Schmidt [Sch84] gives a five-rules based algorithm (we refer to it as *AlgA*) that constructs optimal schedules in  $O(n + m \log m)$  time under the following assumptions: 1. The availability intervals of the processors are rearranged such that they form a *Staircase Pattern*. 2. The set of active processors are known after each availability change. 3. The optimal makespan  $C_{max}^{p*}$  is known. 4. The total processing capacities  $PC_i, i = 1, \dots, m$ , in the interval  $[0, C_{max}]$  of each of the processors are known. Albers and Schmidt [AS01] present algorithms for the problem setting where *nearly-online* and *online* knowledge about processor availability restrictions are given. We present a one-rule-based algorithm (we refer to it as *AlgB*) that solves the offline problem optimal and show

asymptotically and by computational results that this algorithm outperforms *AlgA* with respect to running time and induced number of preemptions.

The paper is organized as follows: In Section 2, we give the detailed problem definition and present an analysis of the running time and number of preemptions of *AlgA*. Section 3 presents and analyzes our new algorithm *AlgB*. In addition, we analyze the asymptotic running time and number of preemptions in comparison to *AlgA*. The two algorithms are compared with respect to running time and induced number of preemptions by an experimental problem setting in Section 4. We show that *AlgB* outperforms *AlgA* with respect to the running time and to the number of preemptions.

## 2 Problem definition

The problem is to schedule a set of  $n$  jobs on  $m$  parallel identical processors with offline known non-availability periods such that the makespan, i.e. the completion time of the last job that finishes, is minimized. At any time, a job can be processed by at most one processor and a processor can execute at any time at most one job. Preemptions of the jobs are allowed in such a way that each job may be preempted at any time and restarted later at no cost, possibly on another processor. All jobs are ready at time zero and we have no setup times.

An interval  $[B_i^r, F_i^r)$  with  $0 \leq r \leq N(i)$  where a processor  $P_i$  is available is called a *processor interval*. A processor is *active* if it is available. In the other case it is *inactive*.  $Q = \sum_{i=1}^m N(i)$  is the total number of all processor intervals, where  $N(i)$  is the total number of all processor intervals of a processor  $P_i$ . We assume that there are at most  $S$  time points  $t_1, t_2, \dots, t_S$  where the processor system (i.e. the set of available processors) changes. An interval  $[t_k, t_{k+1})$  is called a *system interval*. Totally there are  $S$  system intervals.  $\delta = t_{k+1} - t_k$  is the length of a system interval, we set  $t_{S+1} := \infty$ .  $m_k, 1 \leq k \leq S$  is the number of available processors in the  $k^{\text{th}}$  system interval.  $m_k$  may be 0, such

---

|                                                       |                                                                       |
|-------------------------------------------------------|-----------------------------------------------------------------------|
| $n$                                                   | total number of jobs                                                  |
| $m$                                                   | total number of processors                                            |
| $J_j, j = 1, \dots, n$                                | $j^{\text{th}}$ job                                                   |
| $p_j, j = 1, \dots, n$                                | processing time of job $j$                                            |
| $P = \sum_{j=1}^n p_j$                                | total processing capacity of all jobs                                 |
| $p_{avg} = P/n$                                       | average processing time of all jobs                                   |
| $N(i)$                                                | total number of processor intervals of processor $P_i$                |
| $Q = \sum_{i=1}^m N(i)$                               | total number of processor intervals of all processors                 |
| $B_i^r, i = 1, \dots, m, 0 \leq r \leq N(i)$          | begin time of processor interval $r$ of processor $P_i$               |
| $F_i^r, i = 1, \dots, m, 0 \leq r \leq N(i)$          | finish time of processor interval $r$ of processor $P_i$              |
| $[B_i^r, F_i^r), i = 1, \dots, m, 0 \leq r \leq N(i)$ | $r^{\text{th}}$ processor interval of processor $P_i$                 |
| $S$                                                   | total number of system intervals                                      |
| $[t_k, t_{k+1})$                                      | $k^{\text{th}}$ system interval                                       |
| $t_k, k = 1, \dots, S$                                | time point with a changing of the processor availabilities            |
| $m_k, k = 1, \dots, S$                                | number of available processors in the $k^{\text{th}}$ system interval |

---

Table 1: Notation

that in maximal  $S$  system intervals processors are available. Fig. 1 gives an example.

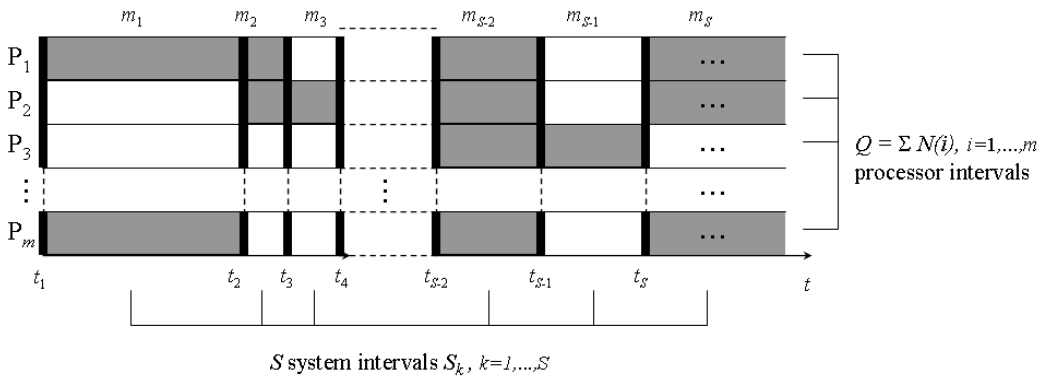


Figure 1: Processor and system intervals

The following lemma gives the relation between system intervals and processor intervals.

**Lemma 1** Let  $S$  the total number of system intervals, and  $Q$  the total number of processor intervals

of  $m$  processors. Then we have

$$S - 1 \leq 2Q \leq (S + 1)m.$$

**Proof:** W.l.o.g. there are  $Q = mh, h \geq 1$ , processor intervals with  $2Q$  begin and end timepoints (an end timepoint may be  $\infty$ ). If all of the  $2Q$  begin and end timepoints of the processor intervals are different, then the set of available processors changes also  $2Q$  times. If at the beginning there are no processors available, we have one additional system interval. Therefore, we have at most  $S \leq 2Q + 1$  system intervals (Fig. 2).

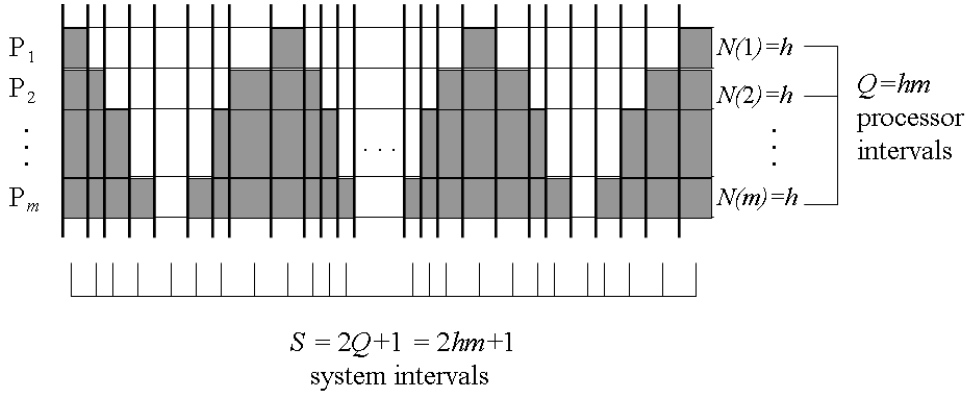
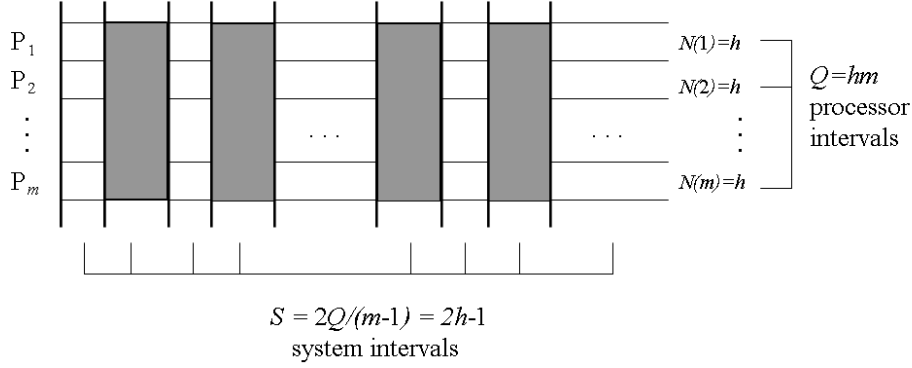


Figure 2:  $S - 1 \leq 2Q$

On the other hand, at most  $2Q/m$  begin and end timepoints of the processor intervals can be identical. In that case, the set of available processors changes  $(2Q/m)$  times. If each of the end timepoints of the last processor intervals are  $\infty$ , we have one system interval less. Therefore, we have at least  $S \geq 2Q/m - 1$  system intervals (Fig. 3).  $\square$

A pattern is called *staircase* if for all *system intervals*, i.e. the intervals where the processor system remains unchanged, the availability of processor  $P_i$  implies the availability of processor  $P_{i+1}$ . Fig. 2 is an example of a *staircase* availability pattern. It is easy to see that after reordering an arbitrary processor availability pattern to a *staircase pattern*, there are at most  $\sum_{j=1}^m 2j - 1 h = m^2 h = mq$  availability pieces in the newly formed *combined processor*.



*AlgA* [Sch84] schedules each job on at most two processors according to five rules that are described in detail on pp. 156-160 in [Sch84]. The following assumptions are used:

1. The availability intervals of the processors are rearranged such that they form a *Staircase Pattern*.
2. The set of active processors are known after each availability change.
3. The optimal makespan  $C_{max}^{p*}$  is known.
4. The total processing capacities  $PC_i, i = 1, \dots, m$  in the interval  $[0, C_{max}]$  of each of the processors are known.

It is shown that *AlgA* needs under the above assumptions  $O(n + m \log m)$  time and uses at most  $Q-1$  preemptions. In the next lemma, we analyze the number of preemptions for the case of arbitrary availability patterns.

**Lemma 2** *AlgA produces at most  $(m-1)(Q+1)$  preemptions on arbitrary availability patterns.*

**Proof:** After reordering the processors into a *staircase pattern*, a combined processor  $P_i, 1 \leq i \leq m$  consists of at most  $mQ$  pieces. Therefore we have in the worst case  $mQ - Q = (m-1)Q$  preemptions induced by processor changings forced by the reordering process. *AlgA* schedules each job on at

most two processors. Therefore we have in the worst case  $(m - 1)Q + (m - 1) = (m - 1)(Q + 1)$  preemptions.  $\square$

### 3 New algorithm

Given the same assumptions as for *AlgA*, our algorithm *AlgB* schedules the jobs in LPT order starting at processor  $P_m$  with the smallest processing capacity, proceeding with  $P_{m-1}$  and so on by obeying the following condition:

$$\text{At any time a job can be scheduled on at most one processor.} \quad (1)$$

The complete algorithm is displayed in Fig. 4. We assume that the jobs are given in LPT order, i.e.

$$p_1 \geq p_2 \geq \dots \geq p_n.$$

---

```

begin
1.  $z := m$ ;
2. for ( $j = 1..n$ ):
3. {
4.    $i := z; r_j := p_j; RC_i := PC_i$ ;
5.   while ( $r_j \neq 0$ )
6.   {
7.     Schedule as much as possible of job  $J_j$  for  $\delta$  time units
       in free intervals of  $P_i$  by obeying condition (1);
8.      $r_j := r_j - \delta$ ; // reduce remaining processing time of job  $J_j$ 
9.      $RC_i := RC_i - \delta$ ; // reduce processing capacity of processor  $P_i$ 
10.    if ( $RC_i == 0$ ) then  $z := z - 1$ ; // processor  $P_i$  is completely filled
11.     $i := i - 1$ ; // switch to the processor above
12.  }
13. }
end;

```

---

Figure 4: New algorithm

$z$  (line 1) serves as a counter for the processor indices of the completely filled processors. The outer for-loop (line 2) schedules the jobs one by one (line 5). The remaining processing times  $r_j$  and the remaining processor capacities  $RC_i$  are updated in lines 6 and 7. If a processor is completely filled,  $z$  decreases by one (line 8). Fig. 5 gives an example.

|       |   |   |   |   |   |   |   |   |   |   |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|
| $P_1$ | 4 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 6 | 6  |    |
| $P_2$ | 3 | 2 | 1 |   |   | 1 | 2 | 3 | 3 | 2 | 2  |    |
| $P_3$ | 2 | 1 |   |   |   |   | 1 | 2 | 2 | 1 | 1  |    |
| $P_4$ | 1 |   |   |   |   |   |   | 1 | 1 |   |    |    |
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Figure 5: *An example of applying the new algorithm*

Pictorially, our algorithm schedules the jobs around the non-availability intervals in the remaining free intervals. The algorithm is similar to McNaughton’s approach and analyzing the running time of *AlgB* shows that each job can be scheduled similar to McNaughton’s algorithm with the additional constraint that there are non-availability regions. As a result we come to a running time of  $O(n + m \log m)$  as with the algorithm *AlgA*.

Next we show that the algorithm works correctly. Let  $p_1 \geq p_2 \geq \dots \geq p_n$ . When the algorithm terminates, all jobs are scheduled, i.e. the scheduling process is complete. Because of the LPT ordering of the jobs, no job is scheduled on more than one processor at the same time. At any time during the scheduling process,  $RC_i$  denotes the remaining processing capacity of processors  $P_i, 1 \leq i \leq m$ . The algorithm always maintains the invariant  $RC_1 \geq RC_2 \geq \dots \geq RC_m$ , i.e. the *staircase* characteristic of the availability pattern of the processors is maintained. At the beginning we have  $RC_i = PC_i, i = 1, \dots, m$  and because of the *staircase* property of the processors

$$\forall k = 1, \dots, m - 1 : \sum_{j=1}^k p_j \leq \sum_{j=1}^k RC_j \leq \sum_{j=1}^n p_j \leq \sum_{j=1}^m RC_j$$

As  $p_{j+1} \leq p_j$  and as the remaining processing capacities of the processors decrease by the same amount as the scheduled job, the invariant is always maintained.

In the next lemma, we analyze the number of preemptions for the case of arbitrary availability patterns.

**Lemma 3** *AlgB produces at most  $(m - 1)Q$  preemptions on arbitrary availability patterns.*

**Proof:** As stated already in the proof of *AlgA* we have in the worst case  $(m - 1)Q$  preemptions induced by processor changings forced by the reordering process. *AlgB* preempts a job only in the case of an availability restriction. Therefore we have no additional preemptions.  $\square$

## 4 Computational results

We implemented the two algorithms in C and used for the computations an AMD 3200+ processor with 1024 MB main memory under Linux. The length of the jobs and the lengths of the non-availability intervals are integers drawn from the intervals  $[1, 50]$ ,  $[50, 100]$ , and  $[1, 100]$ . We tested the algorithms for problem instances with  $m = 10, 15, 20$  processors with  $m, 2m, 5m$  non-availability intervals, and  $n = 50, 100$  number of jobs. In the following tables, the results for *AlgA* are displayed in the left part of the corresponding columns and the results for the new algorithm *AlgB* are displayed in the right part of the corresponding columns.

As we can see in table 2, there is a small advantage of *AlgB* in comparison to *AlgA* with respect to the running time. This advantage results from the fact that *AlgB* has to check less conditions than *AlgA* and because the scheduling of the jobs according to *AlgB* follows only one fast scheduling rule. Table 3 shows that the number of induced preemptions of the new algorithm is smaller than that of *AlgA*. A detailed investigation of the scheduling pattern of *AlgA* shows that often on the first

| processors | jobs      | lengths | number of non-availability intervals |       |       |       |       |       |
|------------|-----------|---------|--------------------------------------|-------|-------|-------|-------|-------|
|            |           |         | 10                                   |       | 20    |       | 50    |       |
| $m = 10$   | $n = 50$  | 1- 50   | 0.015                                | 0.013 | 0.019 | 0.015 | 0.024 | 0.024 |
|            |           | 50-100  | 0.013                                | 0.013 | 0.017 | 0.014 | 0.021 | 0.021 |
|            |           | 1-100   | 0.015                                | 0.014 | 0.017 | 0.015 | 0.020 | 0.020 |
|            | $n = 100$ | 1- 50   | 0.030                                | 0.020 | 0.030 | 0.026 | 0.040 | 0.039 |
|            |           | 50-100  | 0.023                                | 0.025 | 0.030 | 0.024 | 0.039 | 0.038 |
|            |           | 1-100   | 0.028                                | 0.022 | 0.030 | 0.026 | 0.041 | 0.038 |
| $m = 15$   | $n = 50$  | 1- 50   | 0.022                                | 0.023 | 0.031 | 0.026 | 0.041 | 0.041 |
|            |           | 50-100  | 0.022                                | 0.019 | 0.030 | 0.024 | 0.035 | 0.036 |
|            |           | 1-100   | 0.020                                | 0.021 | 0.029 | 0.028 | 0.038 | 0.041 |
|            | $n = 100$ | 1- 50   | 0.040                                | 0.038 | 0.051 | 0.047 | 0.069 | 0.065 |
|            |           | 50-100  | 0.034                                | 0.036 | 0.041 | 0.040 | 0.065 | 0.060 |
|            |           | 1-100   | 0.039                                | 0.035 | 0.049 | 0.044 | 0.068 | 0.065 |
| $m = 20$   | $n = 50$  | 1- 50   | 0.031                                | 0.030 | 0.041 | 0.043 | 0.058 | 0.060 |
|            |           | 50-100  | 0.028                                | 0.031 | 0.037 | 0.038 | 0.056 | 0.059 |
|            |           | 1-100   | 0.033                                | 0.033 | 0.042 | 0.043 | 0.056 | 0.059 |
|            | $n = 100$ | 1- 50   | 0.056                                | 0.052 | 0.068 | 0.067 | 0.096 | 0.095 |
|            |           | 50-100  | 0.058                                | 0.051 | 0.071 | 0.067 | 0.095 | 0.095 |
|            |           | 1-100   | 0.053                                | 0.049 | 0.068 | 0.064 | 0.099 | 0.097 |

Table 2: Running time (in Seconds)

| processors | jobs      | lengths | number of non-availability intervals |       |       |       |       |       |
|------------|-----------|---------|--------------------------------------|-------|-------|-------|-------|-------|
|            |           |         | 10                                   |       | 20    |       | 50    |       |
| $m = 10$   | $n = 50$  | 1- 50   | 74.4                                 | 73.0  | 133.3 | 128.3 | 174.8 | 167.4 |
|            |           | 50-100  | 80.7                                 | 80.0  | 126.6 | 125.2 | 162.5 | 158.6 |
|            |           | 1-100   | 78.5                                 | 77.4  | 136.5 | 131.8 | 165.2 | 159.7 |
|            | $n = 100$ | 1- 50   | 74.4                                 | 73.0  | 133.3 | 128.3 | 174.8 | 167.4 |
|            |           | 50-100  | 80.7                                 | 80.0  | 126.6 | 125.2 | 162.5 | 158.6 |
|            |           | 1-100   | 78.5                                 | 77.4  | 136.5 | 131.8 | 165.2 | 159.7 |
| $m = 15$   | $n = 50$  | 1- 50   | 181.4                                | 170.4 | 231.3 | 207.8 | 331.4 | 295.6 |
|            |           | 50-100  | 169.7                                | 163.4 | 234.2 | 221.8 | 285.6 | 272.1 |
|            |           | 1-100   | 153.5                                | 143.1 | 231.8 | 211.0 | 314.5 | 280.7 |
|            | $n = 100$ | 1- 50   | 192.1                                | 191.2 | 294.5 | 290.3 | 397.4 | 388.1 |
|            |           | 50-100  | 195.1                                | 194.9 | 304.0 | 300.2 | 350.6 | 345.9 |
|            |           | 1-100   | 201.5                                | 200.3 | 290.5 | 283.4 | 406.5 | 430.6 |
| $m = 20$   | $n = 50$  | 1- 50   | 252.8                                | 215.4 | 379.1 | 315.0 | 467.7 | 406.4 |
|            |           | 50-100  | 286.4                                | 263.8 | 368.1 | 335.1 | 493.4 | 448.7 |
|            |           | 1-100   | 300.9                                | 258.0 | 345.2 | 297.7 | 419.9 | 356.4 |
|            | $n = 100$ | 1- 50   | 333.5                                | 326.2 | 497.3 | 476.8 | 654.7 | 625.0 |
|            |           | 50-100  | 325.7                                | 322.3 | 496.8 | 485.5 | 610.5 | 592.5 |
|            |           | 1-100   | 311.6                                | 311.0 | 518.9 | 502.2 | 655.0 | 620.8 |

Table 3: Number of preemptions

processors (in the availability pattern) the largest jobs are scheduled. With the five rules of *AlgA* and reordering the processors to the former availability pattern there arise additional preemptions.

Our new algorithm *AlgB* schedules the jobs already on the staircase pattern with preemptions. The reordering does not induce additional preemptions. A typical example of schedules generated by *AlgA* and by our new algorithm *AlgB* are given in figures 6 and 7. In the upper part of the figures the arbitrary availability pattern is displayed and in the lower parts the staircase pattern is displayed.

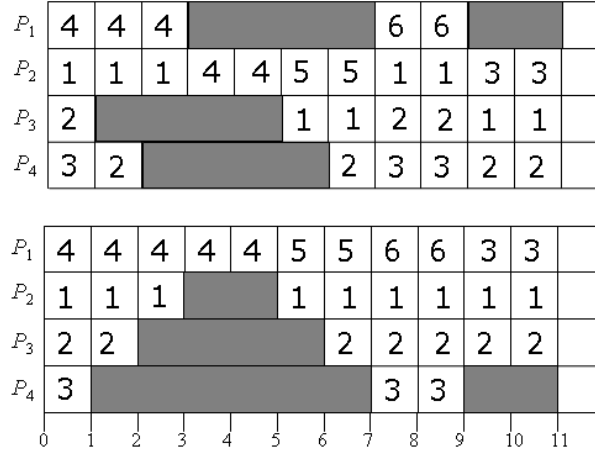


Figure 6: *Typical example of a schedule generated by AlgA*

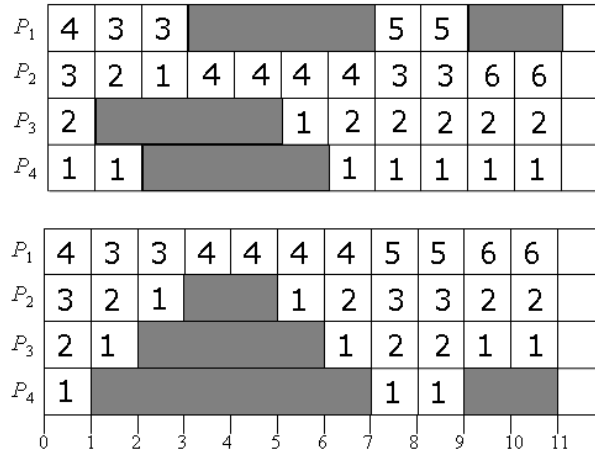


Figure 7: *Typical example of a schedule generated by AlgB*

## 5 Summary

We presented a new algorithm for the problem  $P, NC_{OFF} \mid pmtn \mid C_{max}$  and compared the running time and the induced number of preemptions asymptotically and by experiments to an existing algorithm. Table 4 shows the asymptotical results.

|             | Stair [Sch84]     | New               |
|-------------|-------------------|-------------------|
| time        | $O(n + m \log m)$ | $O(n + m \log m)$ |
| preemptions | $(m - 1)(Q + 1)$  | $(m - 1)Q$        |

Table 4: Results

By computational experiments these results could be confirmed. It would be interesting to investigate the question whether or nor there exists an algorithm with linear running time (in the number  $n$  of jobs) to solve the parallel processor scheduling problem with offline known limited machine availability.

## References

- [AS01] S. Albers, G. Schmidt, *Scheduling with unexpected machine breakdowns*, Discrete Applied Mathematics, 110:85–99, 2001.
- [BBS04] O. Braun, J. Breit, G. Schmidt, *Deterministic machine scheduling with limited machine availability*, Discussion Paper B0403, Department of Economics, Saarland University, Saarbrücken, 2004
- [LS95] Z. Liu, E. Sanlaville, *Profile scheduling by list algorithms*, in P. Chretienne et al. (Eds.), Scheduling Theory and its Applications, Wiley, pages 91-110, 1995
- [McN59] R. McNaughton, *Scheduling with deadlines and loss functions*, Management Science, 6:1-12, 1959
- [Sch84] G. Schmidt, *Scheduling on semi-identical processors*, Z. Oper. Res., 28:153–162, 1984
- [SS98] E. Sanlaville, G. Schmidt, *Machine scheduling with availability constraints*, Acta Informatica, 35:795–811, 1998