Performance guarantee of two simple priority rules for production scheduling

Günter Schmidt

Information and Technology Management, Saarland University, P.O. Box 151150, D-66041 Saarbrücken, Germany

Received 1 April 1998; accepted 28 January 2000

Abstract

The paper presents analytical results concerning algorithms applied to off-line and on-line production scheduling problems. A problem is solved off-line if all the required information to solve it is known in advance; it is solved on-line if only incomplete knowledge about the entire problem input is available. Production planning is related to off-line scheduling and production control requires on-line scheduling. We present algorithms for both types of problems concentrating on machines with limited availability and show how the solution quality of two simple scheduling rules can be guaranteed in terms of a worst-case analysis. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Production scheduling; Priority rules; Performance guarantee

1. Introduction

Production scheduling is the task to assign jobs to machines over time. Solving these kinds of problems within a production planning and control environment requires off-line and on-line formulations of the questions to be answered. In [1] it is discussed how both formulations interrelate. Off-line production scheduling is the task of predictive scheduling where we make rough estimations about the problem parameters and we treat this kind of formulation as static for some planning horizon. We solve the corresponding scheduling problems and use the solution as a guideline for on-line production scheduling. This is the task of reactive scheduling and it is applied when actual data of jobs and machines are available from the shop floor. Now, a more detailed scheduling problem is formulated and solved trying to meet the constraints given by the off-line solution best possible. As there are unpredictable events which might occur during processing jobs on machines a dynamic formulation of the problem is required. Algorithms which can be applied to static problem formulations we will call off-line and those which can handle dynamic formulations we will call on-line.

Simple scheduling rules, sometimes also called priority rules, are used for off-line and on-line production scheduling. Before applying some rule there should exist information about its performance in terms of the solution quality which can be achieved. One way to analyze the performance of...
the rules is to carry out empirical tests. In this case problem data is used to perform simulation runs applying different rules and comparing the results from an empirical point of view by calculating expected values and variances of the solution values related to certain performance measures. Empirical analysis is an average oriented evaluation tool and cannot guarantee performance quality for all cases. If guarantees for all possible cases are required analytical analysis can help. Here the performance of a rule is sometimes analyzed for the worst case and this also guarantees the performance of the rule for all cases. In this paper we apply analytical worst-case analysis to two simple priority rules very often used in production scheduling environments which are arbitrary priority (AP) scheduling and earliest due date (EDD) scheduling. We will differ between off-line and on-line problems according to the available knowledge related to the set of jobs and to the set of machines.

The basic problem setting we investigate is that there is a set of $n$ jobs which have to be processed by a set of $m$ machines over time such that some performance measure is met. Each job $J_j$ is characterized by its release time $r_j$, its due date $d_j$, and its processing time $p_{ij}$ which is required on machine $P_j$; each machine is characterized by its ability to process the jobs. Most of the scheduling models assume that all machines are continuously available. This simplification does not apply in production scheduling environments; there are breakdowns, maintenance requirements, pre-assigned jobs to name just a few reasons for non-availability of machines [2,3]. In this paper we generalize the classical model and introduce intervals of non-availability of the machines. In such an environment AP scheduling assigns jobs in arbitrary order to the machines and EDD scheduling assigns them in order of non-decreasing due dates.

We will concentrate the analysis of the two priority rules on problems with a single machine ($m = 1$) but we will also report on results for parallel identical machines ($m$). We will assume that the machines are continuously available (C) or they are only available during given time intervals (NC). Jobs may be preempted on machines during processing (pmtn) or this may not be allowed (non-pmtn); jobs have release dates which might be the same for all the jobs ($r_j = r$) or they might be arbitrary ($r_j$); jobs have due dates which also might be the same for all the jobs ($d_j = d$) or they might be arbitrary ($d_j$). The performance measure we will investigate is maximum lateness ($L_{\text{max}}$) which is determined by this job for which the difference between its completion time ($C_j$) and its due date ($d_j$), i.e. $C_j - d_j$, is maximum. The objective is to minimize $L_{\text{max}}$.

Note that the pairs given in Table 1 constitute special case and general case relationships. The first entry in each column is considered to be the special case and the second element the general case. The performance guarantee results we derive for the general cases hold by definition also for the special cases.

Table 1

<table>
<thead>
<tr>
<th>Relationships between problem parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special case</td>
</tr>
<tr>
<td>General case</td>
</tr>
</tbody>
</table>

Depending on the knowledge available about the problem parameters at the time the scheduling decision has to be taken we differ between off-line and on-line scheduling problems. Off-line scheduling requires off-line algorithms which generate a complete schedule before schedule execution starts. On-line scheduling requires on-line algorithms which do not generate a complete schedule but decide about the next step to be taken until the scheduling environment changes again due to external events. Off-line algorithms use information about all events happening now and in the future; on-line algorithms do not require information related to the future. To see the difference between both types of algorithms consider the priority rules AP scheduling and EDD scheduling. AP scheduling only uses information on jobs currently known; therefore, it is an on-line algorithm. EDD scheduling requires information about future due dates. In case it requires information about all future due dates it would be an off-line algorithm; if it requires only the due dates of the jobs currently known it would be an on-line algorithm (see also [2,3]). Note that on-line algorithms can be applied to off-line problems but on-line algorithms cannot be applied to on-line problems.
The performance guarantee $R$ we analyze is the ratio
\[ R_P(I) = \frac{F_P(I)}{F_{OPT}(I)} , \]
where $F_P(I)$ is the performance of some scheduling rule $P$ applied to some problem instance $I$ and $F_{OPT}(I)$ is the value of the corresponding optimal solution. In the following we will distinguish between off-line settings $I_{off}$ and on-line settings $I_{on}$. In case of an on-line setting we assume $F_{OPT}(I_{off})$ to be the optimal off-line solution. Minimizing $L_{max}$ we have an optimal performance guarantee if $R = 1$ and a $k$-approximate performance guarantee if $1 < R \leq k$. This means that the solution generated by a $k$-approximate scheduling rule is never $k$ times worse than the optimal off-line solution of the problem. Analyzing $L_{max}$ it seems that $R_P(I)$ is meaningless; if the optimal $L_{max}$ is negative then a $k$-approximate algorithm with $k > 1$ would have to generate a super-optimal schedule. To overcome these obstacles it is suggested in [4] to choose some sufficient large constant $K$ to be subtracted from each of the due dates such that after subtraction all due dates of all jobs are non-positive. Then the overall maximum lateness increases by exactly $K$. Consequently, from now on we will assume for the analysis of $L_{max}$ that $d_j \leq 0$ for all jobs $J_j$. Performance ratios for on-line scheduling $R_P(I_{on})$ are upper bounds for ratios for off-line scheduling; performance ratios for off-line scheduling $R_P(I_{off})$ are lower bound for ratios for on-line scheduling.

To see the difference between off-line and on-line planning situations assume that we want to generate a schedule at time $t = 0$. In off-line planning situations we assume that there exists complete knowledge about all data concerning the presence ($t = 0$) and the future ($t > 0$) before building a schedule. We know all the jobs to be scheduled, their processing times, release dates and due dates, all the machines, their availabilities for processing, and also all further parameters which are influencing the scheduling decision. All data is known and no data is changing. Off-line scheduling means static scheduling. In an on-line planning situation there exists complete knowledge about all data concerning the presence ($t = 0$) but only incomplete or no knowledge at all about the future ($t > 0$). If we have incomplete knowledge we might know that all machines will be available for processing but we may not know which jobs have to be processed at $t > 0$, or we might know all the jobs which are available now and in the future but we lack information about their precise processing times, or we might know all jobs and their processing times but we lack information about the availability of machines, etc. If we have no knowledge at all future occurrences are unknown and cannot be estimated. We do know the jobs and the machines which are available at time $t = 0$ but we do not have any information concerning jobs and machines available at time $t > 0$. Future data is not known or we cannot be sure that it is not changing. On-line scheduling means dynamic scheduling.

In the remainder of this paper we will present performance guarantee results for off-line and on-line algorithms in multi and single machine environments. In Sections 2 and 3 we analyze the performance of AP scheduling and EDD scheduling, respectively. We will finish with some conclusions and proposals for future research.

2. Arbitrary priority scheduling

In this section we will analyze the performance of AP scheduling which can be applied for off-line and on-line problems. AP scheduling means that we generate a list of currently available jobs of arbitrary order. From this list we always take the top job whenever an appropriate machine becomes available and assign it to this machine for processing.

Algorithm (Arbitrary Priority Scheduling)

1. Order the jobs currently available in an arbitrary sequence.
2. Whenever a machine is ready for processing some job, schedule the first job of the list which is available for processing.
3. Delete the scheduled job from the list.

This algorithm works for off-line and for on-line problems because no information about future events is required. It can be executed in $O(n)$ time. A machine is ready at time $t$ for processing if it has finished an assigned job at $t$ and there is no interval of non-availability starting at $t$. Note that the algorithm does not require information about the due
dates and the processing times of the jobs. The availability of the job is related to its release date and the availability of the machines to the corresponding intervals. We assume that \( m \geq 1 \) machines are candidates for processing each job and that there is a single due date \( (d_j = d) \) for all jobs, i.e. \( L_{\text{max}} = \max\{C_j - d\} = \max\{C_j\} - d \). To minimize maximum lateness with a single due date is equivalent to minimize the maximum completion time of all jobs. We will now analyze the worst-case performance of AP scheduling. First, we review the case where all machines are continuously available. Then, we investigate the more general problem with machines having intervals of non-availability.

We start with a famous result which is due to [5].

**Theorem 1** (Graham [5]). The performance guarantee for AP on-line scheduling is \( R_{\text{AP}}(I_{\text{on}}) = 2 - 1/m \) if all machines are continuously available.

**Proof.** Assume there are \( n = m(m - 1) + 1 \) jobs to be scheduled on \( m \) machines. The processing time for each of the \( m(m - 1) \) jobs \( J_1, \ldots, J_{n-1} \) is 1 and the processing time of job \( J_n \) is \( m \). An arbitrary priority list \( L \) of the \( n \) jobs is a list where the job \( J_n \) is last in the sequence, i.e. \( L = (J_1, J_2, \ldots, J_n) \). Applying algorithm arbitrary priority scheduling we generate a schedule of length \( 2m - 1 \). The optimal list \( L^* = (J_n, J_1, J_2, \ldots, J_{n-1}) \) leads to a schedule length of \( m \). So we get \( R_{\text{AP}} = 2 - 1/m \) as a lower bound for this algorithm. The optimal and the worst-case schedule are shown in Fig. 1.

The ratio of \( 2 - 1/m \) is also an upper bound for AP scheduling. To prove this the following result can be used. Let \( m \) and \( m' \) be the number of machines available for processing the set of jobs. If we apply algorithm Arbitrary Priority Scheduling to the cases of \( m \) and \( m' \) machines it can be shown that we might reach a ratio of \( 1 + (m - 1)/m' \) for the length of the schedules for \( m' \) and for \( m \) machines, respectively. With \( m = m' \) we have \( 2 - 1/m \).

It follows from these considerations that an arbitrary priority scheduling algorithm generates off-line and on-line schedules at most twice as long as optimal ones. But this is only true if we assume that all the machines are continuously available for processing.

**Theorem 2.** The performance guarantee for AP off-line scheduling is \( R_{\text{AP}}(I_{\text{of}}) = \infty \) if machines have intervals of non-availability.

**Proof.** Assume the same problem formulation as used in the proof of Theorem 1. Additionally, we want to assume that from time \( m \) on all the machines are not available for some time \( T \). If we use the list \( L = (J_1, J_2, \ldots, J_n) \) for algorithm Arbitrary Priority Scheduling we generate a schedule where job \( J_n \) will only finish after some time \( 2m - 1 + T \) while the optimal schedule has length \( m \). As \( T \) approaches infinity \( R_{\text{AP}} \) also approaches infinity when machines have intervals of non-availability. The optimal and the worst-case schedule are shown in Fig. 2.

From these results we conclude that for problems with a single due date for all jobs and machines which are continuously available we always can find off-line and on-line schedules which are optimal for one machine problems and not worse than twice as bad as the optimal schedule for multi-machine problems. If machines are not continuously available we do not have a bounded worst-case performance ratio for \( m > 1 \) even if preemption of
jobs is allowed. We will now show that for $m = 1$ and preemption of jobs is not allowed the worst-case performance ratio is also not bounded.

**Theorem 3.** The performance guarantee for AP offline scheduling for $m = 1$ is $R_{AP}(I_{off}) = \infty$ if the machine has two or more intervals of non-availability.

**Proof.** Let us assume there are three jobs. One job has processing time $n$ and the two others have processing time $(n + 1)/2$. The machine is available in the intervals $[0, n]$ and $[n + 2, 2n + 2]$. After $t = 2n + 2$ the machine is not available for processing for some time $T$. An arbitrary list schedule would generate a sequence where the job with processing time $n$ would be scheduled in interval $[0, n + 1]$ leaving one unit of idle time. As preemption is not allowed the two remaining jobs have to be scheduled in the interval $[n + 2, 2n + 2]$ and one of them could only finish after time $2n + 3 + T$. In an optimal schedule the two jobs with the shorter processing time would be scheduled in interval $[0, n + 1]$ and the third job would be scheduled in interval $[n + 2, 2n + 2]$ finishing all three jobs within the given intervals at time $2n + 2$. If $T$ approaches infinity also $R_{AP}$ approaches infinity for $m = 1$ (cf. Fig. 3). □

From Theorem 3 we know that AP scheduling generates schedules which can be arbitrarily bad if there are intervals of non-availability of the machines and preemption of jobs is not allowed. Let us
now investigate what we can achieve if we allow preemptions of jobs.

**Theorem 4.** The performance guarantee for AP on-line scheduling for \( m = 1 \) is \( R_{\text{AP}}(I_{\text{on}}) = 1 \) if machines have intervals of non-availability and preemptions of jobs are allowed.

**Proof.** The minimum schedule length is given by the sum of all processing times and the sum of all forced idle times. This can be achieved by any sequence of jobs creating a schedule without unforced idle time of the machine, i.e. a machine is processing a job whenever one is available. \( \square \)

This argument can also be extended to the case where jobs have different release dates. It is easy to see that any feasible schedule is optimal. We can conclude that AP scheduling is optimal for off-line and on-line settings if \( m = 1 \) and preemption of jobs is allowed; if preemption is not allowed we still can guarantee optimality if machines are continuously available.

3. **Earliest due-date scheduling**

If we have different due dates for individual jobs it is easy to see that AP scheduling can be arbitrarily bad. In this case we should apply a rule which generates a schedule taking due date information into account. Such a rule is used for earliest due-date (EDD) scheduling. To apply this rule the due dates of the jobs have to be known in an ordinal sense; their processing times do not have to be known. The basic version of EDD scheduling builds a list where all jobs available are ordered according to non-decreasing due dates. This list defines the sequence in which available jobs are assigned to the machines over time. EDD scheduling can be executed in \( O(n \log n) \) time if job preemption is not allowed. Again we start our analysis with a review of a result which is due to [6].

**Theorem 5** (Jackson [6]). The performance guarantee for EDD on-line scheduling for \( m = 1 \) is \( R_{\text{EDD}}(I_{\text{on}}) = 1 \) if the machines are continuously available and all jobs have a common release date.

**Proof.** The optimality of the EDD rule can be proved by a simple interchange argument. Let \( S \) be any schedule and \( S^* \) be an EDD schedule. If \( S \) is different from \( S^* \) then there exist two jobs \( J_i \) and \( J_j \) with due dates \( d_i \leq d_j \), such that \( J_j \) immediately precedes \( J_i \) in \( S \), but \( J_i \) precedes \( J_j \) in \( S^* \). Since \( d_i \leq d_j \) interchanging the positions of \( J_i \) and \( J_j \) in \( S \) cannot increase the value of \( L_{\text{max}} \). A finite number of these changes transforms \( S \) into \( S^* \), showing that \( S^* \) is optimal. \( \square \)

Note that the EDD rule of Moore and Hodgson’s algorithm [7] can be used to minimize the number of late jobs when the conditions of Theorem 5 are met. EDD scheduling works also well for on-line scheduling if the machine is not continuously available and job preemption is allowed.

**Theorem 6.** The performance guarantee for EDD on-line scheduling for \( m = 1 \) is \( R_{\text{EDD}}(I_{\text{on}}) = 1 \) if there are different release dates for all jobs, preemption of jobs is allowed and the machine may have intervals of non-availability.

**Proof.** Assume a modification of EDD scheduling (MEDD) which preempts a current job whenever a new job is available for processing or an interval of non-availability occurs. If a new job becomes available we sort the list of all available unfinished jobs according to EDD scheduling. The following algorithm is related to the one in [8] and solves the problem in \( O(n^2) \) time.

**Algorithm** *(Modified EDD scheduling)*

\[
\begin{align*}
& \text{begin} \\
& \quad t = 0 \\
& \quad \text{repeat} \\
& \quad \quad \text{choose job } J_j \text{ which is available at time } t \text{ and has the earliest due date of all jobs available at } t; \\
& \quad \quad \text{process job } J_j \text{ until time } t + z \text{ where } J_j \text{ is finished or a new job becomes available for processing}; \\
& \quad \quad \text{if } J_j \text{ is finished} \\
& \quad \quad \quad \text{then delete } J_j \text{ from the set of jobs} \\
& \quad \quad \quad \text{else } p_i := p_i - z \\
& \quad \quad \text{until all jobs are finished} \\
& \quad \text{end};
\end{align*}
\]
Note that intervals of non-availability of the machines do not influence the sequence of the list; they only lengthen the schedule. □

If we do not allow job preemption the problem becomes NP-hard in the strong sense [9]. So we cannot expect that EDD scheduling finds some optimal schedule. If all machines are continuously available Potts [10] was able to prove \( R_{\text{EDD}}(I_{\text{off}}) \leq \frac{3}{2} \) for an iterative version of EDD (IEDD) scheduling. A more efficient algorithm reaching the same bound is given by Nowicki and Smutnicki in [11].

**Theorem 7** (Nowicki and Smutnicki [11]). The performance guarantee for IEDD off-line scheduling for \( m = 1 \) is \( R_{\text{EDD}}(I_{\text{off}}) = \frac{3}{2} \) if there are different release dates for all jobs.

Let us assume that the release dates of the jobs are not known in advance. Whenever some jobs become available for processing they are sequenced according to EDD scheduling. It is shown in [12] that in this case we get \( R_{\text{EDD}}(I_{\text{on}}) < 2 - 2/p \) where \( p \) is the sum of processing times of all jobs. In case we would not use the due date information of the released jobs and we schedule them in arbitrary order a bound of \( R_{\text{AP}}(I_{\text{on}}) < 2 \) can be achieved. The algorithm schedules an arbitrary job from the list of all waiting jobs whenever the machine becomes available [4]. In the same reference, it is also proved that EDD scheduling cannot improve this bound.

**Theorem 8** (Hall [4]). The performance guarantee for MEDD on-line scheduling for \( m = 1 \) is \( R_{\text{MEDD}}(I_{\text{on}}) = 2 \) if there are different release dates for each job.

Let us finally investigate what can be achieved with EDD scheduling when machines may have intervals of non-availability.

**Theorem 9.** The performance guarantee for EDD off-line scheduling for \( m = 1 \) is \( R_{\text{EDD}}(I_{\text{off}}) = \infty \) if preemption of jobs is not allowed and machines may have intervals of non-availability.

**Proof.** Assume the instance shown in Fig. 3 and introduce due dates \( d_1 \leq d_2 \leq d_3 \). EDD scheduling would generate the same schedule as AP scheduling would do. From this we conclude \( R_{\text{EDD}}(I_{\text{off}}) = R_{\text{AP}}(I_{\text{on}}) = \infty \). □

4. Conclusions

We have investigated the question which performance guarantee ratio can be achieved in off-line and in on-line environments for single machine scheduling. The objective is to minimize the maximum lateness of all jobs. It could be shown that for this kind of problem performance guarantees for off-line and on-line scheduling are nearly the same considering the two simple rules AP and EDD scheduling. There is only one difference related to the case where jobs have multiple release dates, machines are continuously available, and preemption of jobs is not allowed. Here EDD scheduling reaches in an on-line setting a performance guarantee of 2 while the guarantee for the corresponding off-line setting is \( \frac{3}{2} \).

Tables 2 and 3 summarize all results for one machine due-date scheduling. The results related to intervals of non-availability and stated in Theorems 2–4, 6, and 9 are new results. The entries refer to problems with a single release date for all jobs.

| Table 2
| | |
|---|---|---|---|---|
| \( d_i = d \) offline/online | C | non-pmtn | C | pmtn | NC | non-pmtn | NC | pmtn |
| \( r_i = r \) | \( R_{\text{AP}} = 1 \) | \( R_{\text{AP}} = \infty \) | \( R_{\text{AP}} = 1 \) |
| \( r_i \) | Theorem 1 | Theorem 3 | Theorem 4 |
Table 3
Off-line and on-line EDD scheduling with multiple due dates

<table>
<thead>
<tr>
<th>d_j offline/online</th>
<th>C</th>
<th>non-pmnt</th>
<th>C</th>
<th>pmtn</th>
<th>NC</th>
<th>non-pmnt</th>
<th>NC</th>
<th>pmtn</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_j = r</td>
<td>R_{EDD} = 1 Theorem 5</td>
<td></td>
<td></td>
<td></td>
<td>R_{EDD} = \infty Theorem 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r_j</td>
<td>R_{EDD}(I_{at}) = \frac{3}{2} Theorem 7</td>
<td></td>
<td></td>
<td></td>
<td>R_{EDD} = 1 Theorem 6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(r_j = r), multiple release dates (r_j), a single due date for all jobs (d_j = d), multiple due dates (d_j), machines being continuously available (C), machines having intervals of non-availability (NC), job preemption is allowed (pmtn), and job preemption is not allowed (non-pmnt).

Off-line scheduling assumes that all job and machine data related to times t > 0 are already known at time t = 0 before the scheduling decision has to be taken; on-line scheduling assumes that no or only incomplete data for jobs and machines with respect to times t > 0 are known, i.e. we assume that at every time t only the following information is given: jobs currently available at t including their release dates and due dates, and the availability status of the machine at t. No information is given on future jobs, their release dates and due dates, their processing times and future availability of the machine. Note that the investigated scheduling rules do not require information about the jobs processing times.

If there is only a single due date for all jobs (cf. Table 2) and machines are continuously available or job preemption is allowed maximum lateness L_{max} is minimized by AP off-line and on-line scheduling. If there are intervals of non-availability of the machine and job preemption is not allowed the approximation ratio of AP scheduling has no finite value and the corresponding results from the single due-date case hold. If there are multiple release dates EDD on-line scheduling achieves an approximation ratio of 2 and an iterative application of EDD off-line scheduling results in an approximation ratio of \frac{3}{2}.

There are still some open questions. Is it possible to improve the performance guarantees of \frac{3}{2} and 2 for EDD scheduling for the corresponding problem settings? What is the best rule from an empirical point of view to solve off-line and on-line single machine scheduling problems if the machine is not continuously available and job preemption is not allowed?

Acknowledgements

Thanks go to an anonymous referee for his detailed and helpful comments to improve the quality of the paper.

References