ELSEVIER

Discrete Applied Mathematics 55 (1994) 145–162

DISCRETE APPLIED MATHEMATICS

Sensitivity analysis of list scheduling heuristics

A.W.J. Kolen[a], A.H.G. Rinnooy Kan[b], C.P.M. van Hoesel[b,**], A.P.M. Wagelmans[b,*]

[a] *University of Limburg, P.O. Box 616, NL-6200 MD Maastricht, Netherlands*
[b] *Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, Netherlands*

Received 14 November 1990; revised 12 March 1993

Abstract

When jobs have to be processed on a set of identical parallel machines so as to minimize the makespan of the schedule, list scheduling rules form a popular class of heuristics. The order in which jobs appear on the list is assumed here to be determined by the relative size of their processing times; well-known special cases are the LPT rule and the SPT rule, in which the jobs are ordered according to non-increasing and non-decreasing processing time respectively.

When all processing times are exactly known, a given list scheduling rule will generate a unique assignment of jobs to machines. However, when there exists a priori uncertainty with respect to one of the processing times, then there will be, in general, several possibilities for the assignment that will be generated once the processing time is known. This number of possible assignments may be viewed as a measure of the sensitivity of the list scheduling rule that is applied.

We derive bounds on the maximum number of possible assignments for several list scheduling heuristics, and we also study the makespan associated with these assignments. In this way we obtain analytical support for the intuitively plausible notion that the sensitivity of a list scheduling rule increases with the quality of the schedule produced.

Key words: Sensitivity analysis; List scheduling; Heuristics; Robustness; Scheduling

1. Introduction

Combinatorial problems whose computational complexity effectively rules out their optimal solution within a reasonable amount of time are frequently solved by *heuristics*, fast methods that produce a suboptimal, but hopefully reasonable, feasible solution. The analysis of their performance is a lively research area. In addition to *empirical* analysis, the emphasis has mostly been on *worst case* and *probabilistic*

* Corresponding author.
** Present address: Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

SSDI 0166-218X(93)E0089-H

analysis of the deviation between the heuristic solution value and the optimal one.

There is, however, an important feature of algorithmic behavior that has hardly received attention. It concerns the effect on algorithmic performance of *perturbations in the problem data.* This effect has been well studied for optimization methods, under the general heading of *sensitivity analysis* or *parametric optimization* (for a review see Wagelmans [5]). Typically, one finds that the optimal solutions to hard (i.e., NP-hard) optimization problems are highly unstable, in that a small change in the problem data can produce a large change in the value or structure of the optimal solution. This characteristic property provides an additional incentive to turn to heuristic methods in which case a more *robust* behavior could be hoped for. Indeed, it is plausible to conjecture an inverse relation between the quality of the solution produced by the heuristic and its robustness under changes in problem data. At one end of the spectrum, the optimal solution is very unstable; at the other end, very simplistic heuristics that extract little information from the data will produce very poor but very stable solutions. Most heuristics will be somewhere in between the two.

In this paper, we obtain some evidence supporting this general conjecture for the special case of the *minimization of makespan on parallel identical machines.* In this prototypical scheduling problem, n jobs with *processing times* $p_1, \ldots, p_n$ have to be distributed among m identical machines so as to minimize the time span of the resulting schedule. If the completion time of the jth job is denoted by C_j, then this criterion amounts to the minimization of $Z = \max_{j=1,\ldots,n}\{C_j\}$. Note that a solution corresponds to an *assignment of jobs to machines*, because the exact order in which the jobs are processed on the machines does not matter. Furthermore, since machines are identical we do not distinguish between them. Therefore, an assignment of jobs to machines is actually a *partition* of the jobs into m subsets.

The problem of minimizing makespan on parallel identical machines is NP-hard and it has been the subject of extensive research; many heuristics for its solution have been proposed (for a review see Lawler et al. [4]). We will concentrate on a class of heuristics known as *list scheduling rules.* Such rules are defined by a *priority list* on which the jobs appear in order of decreasing priority. Whenever a machine becomes idle, the unscheduled job with the highest priority is assigned to this machine. Depending on the way the priority list is constructed, the schedules produced by such heuristics may be quite poor or quite good. Thus, this class of heuristics provides a natural vehicle for the analysis of the relation between solution quality and robustness.

To arrive at an exact formulation, we restrict our attention to priority lists that are defined by the relative sizes of the processing times. To be more precise, it is assumed that the jobs are initially ordered according to non-decreasing processing time, and a list scheduling rule corresponds to a permutation π such that the jth job of this initial order is always put in the $\pi(j)$th position on the list. Hence, for given n, there exists a mapping from these *permutation list scheduling rules* to the elements of S_n, the set of n-permutations. For given n it is actually sufficient to consider only the

permutation, i.e., we do not need to know the exact rule by which it was generated. Therefore, we will often let the permutation represent the list scheduling rule. One should keep in mind, however, that the permutation that corresponds to a given permutation list scheduling rule in the case of n jobs, does, in general, not provide any information about the permutations that the same rule will generate when the number of jobs is unequal to n. In other words, a permutation list scheduling rule is only completely specified by a set of permutations, namely one permutation for every number of jobs.

Two well-known examples of permutation list scheduling rules are the SPT (*Shortest Processing Time*) and the LPT (*Longest Processing Time*) rules, defined by $\pi(j) = j$ and $\pi(j) = n - j + 1$, $j = 1, \ldots, n$, respectively. The quality of the solution produced by these rules is very different. The SPT rule yields schedules whose value can exceed the optimal one by a factor of $2 - 1/m$ (see Graham [2]); for the LPT rule, this factor is at most $\frac{4}{3} - 1/(3m)$ (see Graham [3]); both bounds are tight. A similar difference in quality emerges from a probabilistic analysis: if the processing times of the jobs are independent and uniformly distributed, then it can be shown that the expected absolute error of the LPT rule is $O(1/n)$, whereas the expected absolute error of the SPT rule is $\Omega(1)$ (see Frenk and Rinnooy Kan [1]).

From now on we assume that the processing times of jobs 2 to n are exactly known and that $p_2 \leqslant p_3 \leqslant \cdots \leqslant p_n$ holds. Suppose that, for the time being, there exists uncertainty with respect to the value of p_1. Because of this uncertainty, we do not yet know how a given list scheduling heuristic will eventually assign the jobs to the machines. In general, there will be several possibilities for that assignment and which one will actually be generated depends on the exact value of p_1. The total number of possible assignments can be taken as a measure of the sensitivity of the list scheduling rule with respect to the value of p_1. If this number is low, then this indicates that the rule is robust, because for many values of p_1 the same assignment will result. On the other hand, a high number of possible assignments means that the exact value of p_1 is obviously very important in constructing the solution. Therefore, in the latter case, the list scheduling rule can be considered sensitive to uncertainty with respect to p_1.

To be able to say something about the number of possible assignments that can be generated by a given list scheduling rule when uncertainty exists with respect to one of the processing times, we will actually carry out a *parametric analysis* with respect to p_1. This means that we will study the SPT rule, the LPT rule and other permutation list scheduling rules when $p_1 = \lambda$, where λ gradually increases from zero to infinity. For a given list scheduling heuristic π and given n we define A_n^π to be the *worst case number of different assignments* of jobs to machines that occur when λ increases from zero to infinity. Besides the number of solutions generated by the list scheduling rule it is also natural to study the solution value $Z_n^\pi(\lambda)$ as a function of λ. As we will see in Section 2, Z_n^π is a *continuous piecewise linear* function. The values of λ in which this function is not differentiable are called *breakpoints*. Note that 0 is a breakpoint. We

define B_n^π to be the *worst case number of breakpoints* of Z_n^π. This number may serve as a first indication of how quickly the solution value adapts to changes in the problem data.

In Sections 3 and 4, we look at the SPT rule and the LPT rule, respectively. We establish that $A_n^{SPT} \leqslant n$ and $B_n^{SPT} \leqslant 2\lceil n/m \rceil$; both upper bounds are tight. In contrast $A_n^{LPT} \leqslant 2^{n-m}$ and $B_n^{LPT} \leqslant 2^{n-m+1}$; the first bound is tight, and there exists an example for which $B_n^{LPT} > 2^{(n-m)/2}$. These results nicely support the conjectured relationship between solution quality and robustness. In Section 5, we show that the LPT rule is almost an extreme case; for an arbitrary permutation π, $A_n^\pi \leqslant 2^{n-m+1}$ and $B_n^\pi \leqslant 2^{n-m+2}$. A summary of the results and some concluding remarks are collected in Section 6.

2. The function Z_n^π

In this section we show that for any list scheduling rule defined by a permutation π on the processing times $p_1 = \lambda$ and $p_2 \leqslant p_3 \leqslant \cdots \leqslant p_n$, Z_n^π is a continuous piecewise linear function of λ, $0 \leqslant \lambda < \infty$. Each of the linear parts of Z_n^π will be constant or have a slope of one.

To illustrate the result, we first present an example with two machines and four jobs having processing times $p_1 = \lambda$, $p_2 = 1$, $p_3 = 2$, $p_4 = 4$, that are scheduled according to the LPT rule. The different schedules and corresponding linear parts of Z_4^{LPT} are given in Fig. 1.

Let us now look at an arbitrary permutation list scheduling heuristic applied to an arbitrary problem instance. When λ is increased from zero to infinity the ordering of the jobs according to non-decreasing processing times will change. Hence the list will change too, although the relative order of the jobs $2, 3, \ldots, n$ will remain the same. If λ is equal to one of the processing times $p_2, \ldots, p_n$, then we may assume that in the order according to non-decreasing processing time job 1 follows directly after the job with largest index for which the processing time equals λ. Thus, a further increase of λ leaves the current ordering unchanged until λ becomes equal to the next larger processing time. It is easily verified that Z_n^π is continuous in $\lambda = p_j$ ($j = 2, \ldots, n$). To establish our result, it is sufficient to show that when λ varies between processing times p_j and p_{j+1} with $p_j < p_{j+1}$, Z_n^π is a continuous piecewise linear function.

Before analyzing $Z_n^\pi(\lambda)$ for $\lambda \in [p_j, p_{j+1})$ we will make one more assumption about the schedule produced by the heuristic. Consider a job which is after job 1 on the list, and is ready to be scheduled. If there exists a choice of machines to schedule this job on (i.e., at the same point in time more than one machine becomes available), we will always choose a machine not containing job 1. This assumption does not affect the distribution of total processing time among the machines and therefore it does not affect Z_n^π. Let us refer to this assumption as the *tie breaking assumption.*

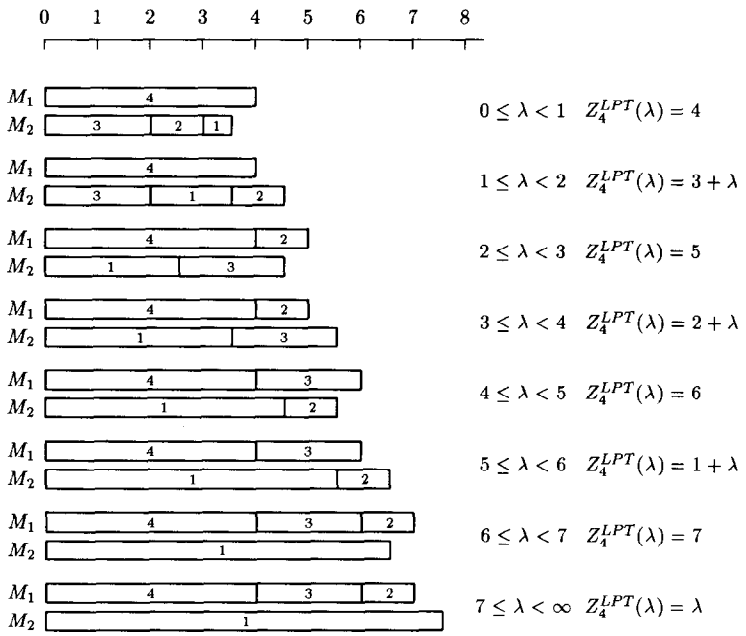

Fig. 1. Example LPT-schedules.

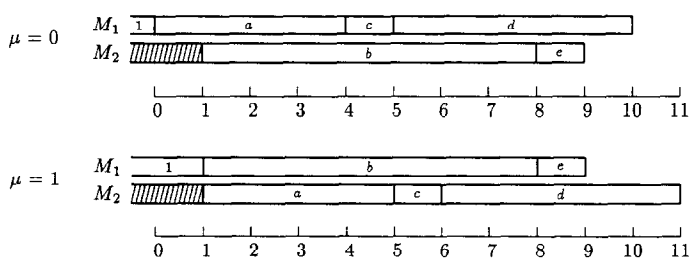

Fig. 2. A switch of tails.

Since we are analyzing Z_n^π over an interval in which the order of the jobs on the list remains unchanged, the only way the schedule can be affected is by changes in the times on which machines become available. This is illustrated in Fig. 2 for the case of two machines M_1 and M_2.

Assume that the jobs a, b, c, d, e occur on the list in alphabetical order. When the processing time of job 1 is increased by μ $(0 \leqslant \mu \leqslant 1)$ the schedule does not change and Z_n^π increases linearly with slope one. For $\mu = 1$, machines M_1 and M_2 become available at the same time, namely at the completion time of job 1. From our tie breaking assumption, jobs b, e are now scheduled on M_1 and jobs a, c, d on M_2. Note that Z_n^π remains the same but the *maximum machine*, i.e., the machine for which Z_n^π is attained, has changed. The motivation for the tie breaking assumption is that an

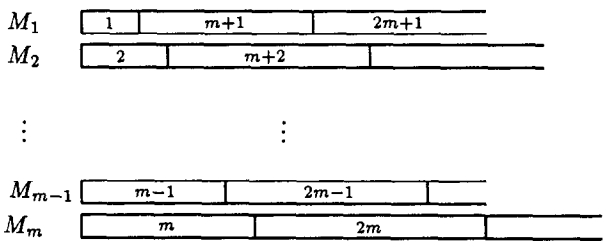

Fig. 3. SPT-schedule.

incremental increase of μ will now not affect the schedule. We refer to the transformation depicted in Fig. 2 as a *switch of tails*. If μ would increase to 4, then another switch of tails would occur; job e would switch to M_2. Note that Z_n^π is constant for $1 \leqslant \mu \leqslant 3$, whereas Z_n^π increases with slope one for $3 \leqslant \mu \leqslant 4$.

In general, it is easy to see that before a switch of tails occurs Z_n^π is constant as long as job 1 is not on the maximum machine. If the machine containing job 1 becomes the maximum machine, then Z_n^π increases with slope one. A switch of tails does not affect the value of the makespan and after a switch Z_n^π will be constant or increase with slope one depending on whether or not job 1 is on the maximum machine. This establishes the desired result.

3. The SPT rule

According to the SPT rule jobs appear on the list in order of non-decreasing processing time. For n jobs with processing times $p_1 = \lambda < p_2 \leqslant \cdots \leqslant p_n$, the SPT-schedule is illustrated in Fig. 3. For $j = 1, 2, \ldots, m$ we may assume that job j is scheduled on machine j. Since M_1 is the first machine which becomes available again, job $m + 1$ is scheduled on M_1. Because $p_1 + p_{m+1} \geqslant p_m$ machine M_1 now is the maximum machine and job $m + 2$ is scheduled on M_2. Since $p_2 \geqslant p_1$ and $p_{m+2} \geqslant p_{m+1}$ machine M_2 has now become the maximum machine and job $m + 3$ is scheduled on M_3. Continuing this way we find that an SPT-schedule can always be assumed to have the following structure: job j is scheduled on machine M_i where i is given by $i = \lceil (j-1) \bmod m \rceil + 1$, $j = 1, \ldots, n$, and the maximum machine is the machine processing job n.

Let us now consider the case that the processing time $p_1 = \lambda$ is increased from zero to infinity. It follows from the structure of the SPT-schedule that there will be at most n different assignments of jobs to machines, and this bound is attained if $p_2 < p_3 < \cdots < p_n$. Hence, $A_n^{\mathrm{SPT}} = n$. On each machine in the SPT-schedule either $\lceil n/m \rceil$ or $\lceil n/m \rceil - 1$ jobs are scheduled. The maximum machine will always be machine $\lceil (n-1) \bmod m \rceil + 1$ and it always contains $\lceil n/m \rceil$ jobs. Therefore job 1 can be on the maximum machine at most $\lceil n/m \rceil$ times. Since Z_n^{SPT} can only have a slope of one if in the corresponding interval job 1 is on the maximum machine, it follows that Z_n^{SPT} has at

most $\lceil n/m \rceil$ linear parts with slope one. Therefore there can be no more than $2\lceil n/m \rceil$ breakpoints of Z_n^{SPT}. It is easy to see that if we take a problem instance with $n \bmod m \neq 1$ and $p_2 < p_3 < \cdots < p_n$, then this upper bound on the number of breakpoints is attained. Hence, $B_n^{\mathrm{SPT}} = 2\lceil n/m \rceil$. As Z_n^{SPT} is completely determined by its breakpoints and their function value, we conclude that Z_n^{SPT} can be computed in polynomial time.

4. The LPT rule

According to the LPT rule, jobs are assigned in order of decreasing processing times. Again, let us assume that job 1 has processing time λ, and that $p_2 \leqslant p_3 \leqslant \cdots \leqslant p_n$.

We first prove that the worst case number of different LPT-assignments, A_n^{LPT}, is at most equal to 2^{n-m} for all $n \geqslant m$. We do so by induction. For $n = m$, the statement is trivial since each machine is assigned exactly one job and we do not distinguish between machines. Assume that the statement holds for n ($n \geqslant m$), and consider an instance of the $(n+1)$-job problem. Compare this instance to the n-job instance obtained by deleting job 2. By induction, it is possible to partition the λ-axis $[0, \infty)$ into at most 2^{n-m} intervals, such that in the interior of each interval the assignment of jobs to machines for this n-job instance remains the same. We will investigate how the $(n+1)$-job assignments are related to these n-job ones.

To facilitate our analysis we define for the n-job instance the *minimum machine* to be the machine with minimal total processing time. Q_n^{LPT} will denote the completion time of the minimum machine as a function of λ. Analogously to Section 2 one can prove that Q_n^{LPT} is a continuous piecewise linear function.

Consider the $(n+1)$-job instance. First assume that $\lambda \geqslant p_2$. Thus, job 2 is the smallest one and the LPT rule assigns it (as the last job) to the minimum machine of the n-job instance. Now consider an arbitrary λ-interval in which the assignment of the n-job instance does not change. What can happen to the minimum machine in the interior of such an interval? Its index can change only once, namely when the initial minimum machine contains job 1 and loses its status due to the increase of λ (the slope of Q_n^{LPT} changes from one to zero). Hence, each such interval generates at most two intervals for the $(n+1)$-job instance and the corresponding assignments differ only in the assignment of job 2.

We now turn to the case that λ is less than p_2. Let $[0, a)$ be the first λ-interval for the n-job instance. Because the assignment of the n-job instance does not change as long as λ is less than the smallest processing time p_3, it holds that $a \geqslant p_3 \geqslant p_2$. It follows that $[0, p_2)$ is contained in $[0, a)$. Hence, $[0, a)$ is the only interval of the n-job instance for which we have not proven yet that it corresponds to at most two similar intervals of the $(n+1)$-job instance. To prove that the latter does hold, first note that we have already observed above that Q_n^{LPT} has at most one breakpoint in $(0, a)$. Furthermore, it is obvious that the assignment of the $(n+1)$-job instance can not change for $\lambda < p_2$. Hence, if Q_n^{LPT} has no breakpoint in (p_2, a), then there will be at most two different

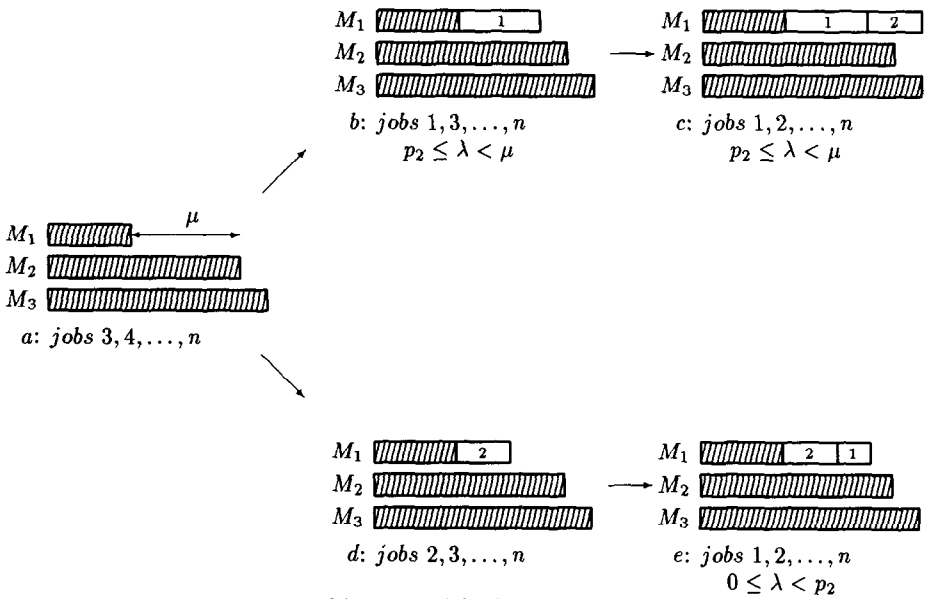

Fig. 4. Schedules for $\lambda \in [0, \mu)$.

assignments of the $(n+1)$-job instance in $[0, a)$, namely one in each of the intervals $[0, p_2)$ and $[p_2, a)$. So we are left with the case that Q_n^{LPT} has a breakpoint μ in (p_2, a). At first sight there may be three different assignments, corresponding to $[0, p_2)$, $[p_2, \mu)$, and $[\mu, a)$. However, we will show that the assignments on $[0, p_2)$ and $[p_2, \mu)$ are identical.

In Fig. 4(a) the LPT-schedule of the jobs $3, \ldots, n$ is given, where we have assumed that machines are numbered such that M_i has no more total processing time assigned to it than M_{i+1}, $i = 1, \ldots, m-1$. The fact that Q_n^{LPT} has a breakpoint $\mu \in (p_2, a)$ means that M_1 is the minimum machine as long as the processing time of job 1 is less than or equal to μ. Hence, the difference between the total processing time of M_1 and M_2 is exactly equal to μ. The schedule of the n-job instance on $[p_2, \mu)$ is given in Fig. 4(b). The schedule of the $(n+1)$-job instance on $[p_2, \mu)$ is obtained from the n-job instance by scheduling job 2 on the minimum machine M_1. The schedule is given in Fig. 4(c). On $[0, p_2)$ the schedule of the jobs $2, 3, \ldots, n$ is given by Fig. 4(d). This schedule is obtained from the schedule in Fig. 4(a) by scheduling job 2, which is smallest among the jobs we consider at this moment, on the minimum machine M_1. Note that, because $p_2 < \mu$, M_1 remains the minimum machine. Therefore, the schedule of the $(n+1)$-job instance on $[0, p_2)$ is obtained by scheduling job 1 also on M_1. This schedule is given in Fig. 4(e). By comparing the schedules in Figs. 4(c) and 4(e) we establish that there is only one assignment of jobs to machines on the interval $[0, \mu)$.

To summarize the discussion above, we have shown that every λ-interval of the n-job instance corresponds to at most two such intervals of the $(n+1)$-job instance. Therefore, $A_{n+1}^{\mathrm{LPT}} \leqslant 2A_n^{\mathrm{LPT}} \leqslant 2^{n-m+1}$, and this establishes the desired result.

Our final observation is that B_n^{LPT}, the worst case number of breakpoints, is at most equal to $2A_n^{\mathrm{LPT}}$ and, therefore, bounded by 2^{n-m+1}. The argument is simple: each assignment defines at most two linear segments of Z_n^{LPT}, the worst case being the one in which the index of the maximum machine changes as a result of the increase of λ.

We now turn to the question whether the derived upper bounds of 2^{n-m} on A_n^{LPT} and 2^{n-m+1} on B_n^{LPT} are tight. The following example shows that this is the case for $m = 2$.

Example 1. Suppose $m = 2$, the processing time of job j is $p_j = 2^{j-2}$ for $j = 2, \ldots, n$, and job 1 has processing time λ. It may be useful to note that for $n = 4$ this is actually the example of Fig. 1. We will show that for each $l \in \{0, 1, \ldots, 2^{n-2} - 1\}$ the following statements hold.

(a) The assignment of jobs to machines is constant and unique for $\lambda \in [2l, 2l + 2)$.

(b) $Z_n^{\mathrm{LPT}}(\lambda) = 2^{n-2} + l$ for $\lambda \in [2l, 2l + 1)$, and
$Z_n^{\mathrm{LPT}}(\lambda) = 2^{n-2} + l + \lambda - \lfloor \lambda \rfloor$ for $\lambda \in [2l + 1, 2l + 2)$.

Thus, for λ ranging over the interval $[0, 2^{n-1})$ there are 2^{n-2} assignments and 2^{n-1} breakpoints.

To prove claims (a) and (b), we will frequently make use of the fact that $2^k - 1 = \sum_{i=0}^{k-1} 2^i$ for every positive integer k. Let $l = \sum_{i=0}^{n-3} a_i 2^i$ with $a_i \in \{0, 1\}$, i.e., l is written in binary representation, and let $\lambda \in [2l, 2l + 2)$. Suppose that job n, the largest one, is placed on machine M_1. We will first show the following: job 1 is assigned to M_2 and for each $i \in \{0, 1, \ldots, n - 3\}$ the job with processing time 2^i (job $i + 2$) is placed on M_1 if and only if $a_i = 1$.

Clearly the statement holds if $l = 0$, because then $a_i = 0$ for all $i \in \{0, 1, \ldots, n - 3\}$ and, except for job n, all jobs are scheduled on M_2. So, suppose that l is positive and let the integer $k \leqslant n - 3$ be maximal such that $\lambda \geqslant 2^k$. Then, $a_{k+1} = \cdots = a_{n-3} = 0$, and it is obvious that all jobs with a processing time larger than λ, except job n, are scheduled on M_2. Because the total processing times of these jobs is easily seen to be less than p_n, job 1 is also assigned to M_2. We will now use induction to prove the statement for $i = 0, \ldots, k$. Assume that it holds for $i + 1, \ldots, n - 3$. Define PM_1 and PM_2 to be the total processing time on M_1 respectively M_2 after jobs $i + 3, \ldots, n$ and job 1 have been placed. It follows from the induction hypothesis that

$$PM_1 = 2^{n-2} + \sum_{j=i+1}^{n-3} s_j 2^j, \tag{1}$$

$$\begin{aligned} PM_2 &= \sum_{j=i+1}^{n-3} (1 - a_j) 2^j + \lambda \\ &= \sum_{j=i+1}^{n-3} (1 - a_j) 2^j + 2 \sum_{j=0}^{n-3} a_j 2^j + (\lambda - 2l). \end{aligned} \tag{2}$$

Thus,

$$PM_1 - PM_2 = 2^{i+1} - 2 \sum_{j=0}^{i} a_j 2^j + (2l - \lambda). \tag{3}$$

If $a_i = 0$, then $PM_1 - PM_2 \geqslant 2^{i+1} - 2(2^i - 1) + 2l - \lambda = 2 + 2l - \lambda > 0$. Hence, $PM_1 > PM_2$, which means that the job with processing time 2^i will be assigned to M_2. If $a_i = 1$, then $PM_1 - PM_2 \leqslant 2^{i+1} - 2(2^i) + 2l - \lambda \leqslant 0$. Now $PM_1 \leqslant PM_2$ and, because of the tie breaking assumption, the job with processing time 2^i will be assigned to M_1.

Hence, we have proved that on the interval $[2l, 2l + 2)$ the assignment of the jobs can be determined from the binary representation of l. In particular this means that this assignment is constant and unique. Therefore, claim (a) holds.

Finally, if all jobs have been placed, then

$$PM_1 = 2^{n-2} + \sum_{j=0}^{n-3} a_j 2^j = 2^{n-2} + l, \tag{4}$$

$$\begin{aligned} PM_2 &= \sum_{j=0}^{n-3} (1 - a_j)2^j + \lambda \\ &= \sum_{j=0}^{n-3} (1 - a_j)2^j + \sum_{j=0}^{n-3} a_j 2^j - l + \lambda \\ &= \sum_{j=0}^{n-3} 2^j - l + \lambda = 2^{n-2} - 1 - l + \lambda \\ &= 2^{n-2} + l + \lambda - (2l + 1). \end{aligned} \tag{5}$$

This proves claim (b).

The following example shows that $A_n^{\mathrm{LPT}} = 2^{n-m}$ also holds for $m > 2$.

Example 2. Let $N = n - (m - 2)$, $p_j = 2^{j-2}$ for $j = 2, 3, \ldots, N$, $p_j = 2^{N-1}$ for $j = N + 1, \ldots, n$. From Example 1 it follows that the $m - 2$ jobs with processing times 2^{N-1} will always be the only jobs scheduled on their machine. This means that the number of different assignments of jobs to machines is determined by the jobs $1, 2, \ldots, N$ on two machines. Using the result of Example 1, we obtain $2^{N-2} = 2^{n-m}$ different assignments.

The upper bound of 2^{n-m+1} on the number of breakpoints cannot always be attained if $m > 2$. We have been able to show that for $n = 7$ and $m = 3$ there are at most 14 breakpoints. The proof is rather long and tedious and is therefore omitted. However, below we will give an example for which the number of breakpoints is at least $(\sqrt{2})^{n-m+2}$. This implies that in the worst case a complete description of Z_n^{LPT} is exponential in $n - m$.

Example 3. We first consider $2n$ jobs and 2 machines. The processing time of job 2 is a positive integer p_2, the processing times of jobs $3, 4, \ldots, 2n$ satisfy $p_{2j+1} = 2^{j+1} + p_2$,

$p_{2j+2} = 2^{j+1} + 2^j + p_2$ $(j = 1, 2, \ldots, n-1)$. We consider the processing time λ of job 1 with $\lambda \in [p_{2n} - 2^{n-1}, p_{2n} + 2^{n-1}]$. In this interval job 1 and job $2n$ are the two jobs with largest processing time and are therefore scheduled on different machines. Assume job 1 is scheduled on M_1. We claim that for $\lambda = p_{2n} - 2^{n-1} + 2l$ $(l = 0, \ldots, 2^{n-1} - 1)$ the LPT-schedule can be obtained by writing l in binary notation: if $l = \sum_{i=1}^{n-1} a_i 2^{i-1}$ $(a_i \in \{0, 1\})$, then the LPT-schedule is obtained by scheduling on M_1 the subset of jobs given by $\{1\} \cup \{2i \mid a_i = 1\} \cup \{2i + 1 \mid a_i = 0\}$.

The proof is again by induction. We claim that if $a_i = 1$, then job $2i + 1$ is scheduled on M_2 and job $2i$ is scheduled on M_1, else job $2i + 1$ is scheduled on M_1 and job $2i$ on M_2. Let $2 \leqslant k \leqslant n$ and suppose the claim holds for $a_k, a_{k+1}, \ldots, a_{n-1}$ ($k = n$ corresponds to the basis of induction). The total processing time of all jobs scheduled so far on M_1 is given by

$$PM_1 = p_{2n} - 2^{n-1} + 2\sum_{i=1}^{n-1} a_i 2^{i-1} \quad \text{(the processing time of job 1)}$$

$$+ \sum_{i=k}^{n-1} a_i (2^i + 2^{i-1} + p_2) \quad \text{(all even jobs scheduled on } M_1\text{)}$$

$$+ \sum_{i=k}^{n-1} (1 - a_i)(2^{i+1} + p_2) \quad \text{(all odd jobs scheduled on } M_1\text{)}.$$

And the total processing time of all jobs scheduled so far on M_2 is given by

$$PM_2 = p_{2n} + \sum_{i=k}^{n-1} (1 - a_i)(2^i + 2^{i-1} + p_2) + \sum_{i=k}^{n-1} a_i (2^{i+1} + p_2). \tag{6}$$

It follows that

$$PM_2 - PM_1 = 2^{k-1} - \sum_{i=1}^{n-1} a_i 2^i. \tag{7}$$

If $a_{k-1} = 1$, then $PM_2 \leqslant PM_1$ and job $2k - 1$ is scheduled on M_2. Since $-\sum_{i=1}^{k-2} a_i 2^i + p_{2k-1} = -\sum_{i=1}^{k-2} a_i 2^i + (2^k + p_2) > 0$, job $2k - 2$ is scheduled on M_1.

If $a_{k-1} = 0$, then $PM_2 > PM_1$ and job $2k - 1$ is scheduled on M_1. Since $\sum_{i=1}^{k-2} a_i 2^i - 2^{k-1} + p_{2k-1} \geqslant 0$, job $2k - 2$ is scheduled on M_2.

Hence, the claim also holds for $k - 1$, and this completes the proof.

Because (7) is also valid for $k = 1$, $PM_2 - PM_1 = 1$ if $\lambda = p_{2n} - 2^{n-1} + 2l$ $(l = 0, \ldots, 2^{n-1} - 1)$. Combining this with the tie breaking assumption and the fact that all data are integer, it follows that Z_{2n}^{LPT} is constant on $[p_{2n} - 2^{n-1} + 2l, p_{2n} - 2^{n-1} + 2l + 1]$. Because λ increases by 2 over the interval $[p_{2n} - 2^{n-1} + 2l, p_{2n} - 2^{n-1} + 2l + 2]$ and because in both endpoints the total processing time of M_2 exceeds the total processing time of M_1 by 1, it can easily be deduced that $Z_{2n}^{\mathrm{LPT}}(p_{2n} - 2^{n-1} + 2l + 2) - Z_{2n}^{\mathrm{LPT}}(p_{2n} - 2^{n-1} + 2l) = 1$. Hence, Z_{2n}^{LPT} must be strictly increasing on $[p_{2n} - 2^{n-1} + 2l + 1, p_{2n} - 2^{n-1} + 2l + 2]$.

To summarize, we have proved that exactly every integer value in $[p_{2n} - 2^{n-1}, p_{2n} + 2^{n-1} - 1]$ is a breakpoint of Z_{2n}^{LPT}. Therefore the number of breakpoints of Z_{2n}^{LPT} for this example is at least 2^n.

For the case $m > 2$, take n jobs such that $N = n - (m-2)$ is even, take $p_2 > 2^{(N/2)-1}$ and $p_3, \ldots, p_N$ as defined above for the 2-machine case. Furthermore, take $p_{N+1}, \ldots, p_n$ equal to the makespan for the 2-machine case with N jobs when the processing time λ of job 1 equals $p_N - 2^{(N/2)-1}$. The similarity to the 2-machine example will be clear. When $\lambda = p_N - 2^{(N/2)-1}$, jobs $N+1, \ldots, n$ are all scheduled as the only jobs on their machine and jobs $1, 2, \ldots, N$ are scheduled on the two remaining machines, say M_1 and M_2, in the same way as in the 2-machine case. For $\lambda \in (p_N - 2^{(N/2)-1}, p_N + 2^{(N/2)-1}]$ the jobs $1, 2, \ldots, N$ will also be scheduled on M_1 and M_2. To see this, recall that we have already derived for two machines that $Z_N^{\mathrm{LPT}}(p_N + 2^{(N/2)-1}) - Z_N^{\mathrm{LPT}}(p_N - 2^{(N/2)-1}) = 2^{(N/2)-1}$. Since the jobs $1, \ldots, N$ all have processing time greater than $2^{(N/2)-1}$, it follows that these jobs always start before $Z_N^{\mathrm{LPT}}(p_N - 2^{(N/2)-1})$, which is the completion time of jobs $N+1, \ldots, n$. Hence, jobs $1, \ldots, N$ are never assigned to a machine that is already processing one of the jobs $N+1, \ldots, n$. Applying now the result for the 2-machine case we conclude that there are at least $(\sqrt{2})^N = (\sqrt{2})^{n-m+2}$ breakpoints of Z_n^{LPT}.

5. Permutation list scheduling rules

The upper bounds derived in the previous section are not valid for arbitrary list scheduling heuristics. A counter-example is given by four jobs with processing times $p_1 = \lambda$, $p_2 = 2$, $p_3 = 3$ and $p_4 = 4$ to be scheduled on three machines using the permutation $\pi(1) = 1$, $\pi(2) = 2$, $\pi(3) = 4$ and $\pi(4) = 3$. For $n = 4$ and $m = 3$ the previously derived bounds on the number of assignments and breakpoints are equal to 2 and 4 respectively. However, this example has 4 different assignments and 6 breakpoints of Z_4^{π}. In this section we show that when an arbitrary permutation list scheduling rule is used to schedule n on m machines while the processing time of one job changes, a valid upper bound on the number of different assignments of jobs to machines is given by 2^{n-m+1}.

As in Section 4 the result will be proved by induction on n ($n \geqslant m$). Although the analysis in this section is very much of the same flavor as that presented in Section 4, there is one complicating factor which has led us to prove the upper bound for a more general class of problems.

To motivate this class of problems consider the problem instance defined by a permutation $\pi \in S_{n+1}$, $\pi(q) = n + 1 (q \neq n+1)$, job 1 with processing time λ and job j with processing time p_j, $j = 2, \ldots, n+1$, with $p_2 \leqslant p_3 \leqslant \cdots \leqslant p_{n+1}$. When λ, the processing time of job 1, varies, three different situations occur as follows.

(a) For $0 \leqslant \lambda < p_q$, job q is the last job scheduled and the remaining jobs are scheduled according to $\sigma \in S_n$ defined by $\sigma(j) = \pi(j)$, $j = 1, \ldots, q-1$, and

$\sigma(j) = \pi(j+1), j = q, \ldots, n$. The processing times of the n-job instance corresponding to the remaining jobs are $\lambda, p_2, \ldots, p_{q-1}, p_{q+1}, p_{q+2}, \ldots, p_{n+1}$. To avoid confusion, we should point out here that if a particular heuristic prescribes permutation π for $(n+1)$-job instances, then that heuristic does in general not correspond to permutation σ for n-job instances.

(b) For $p_q \leqslant \lambda < p_{q+1}$, job 1 is the last job scheduled.

(c) For $p_{q+1} \leqslant \lambda < \infty$, job $q+1$ is the last job scheduled and the remaining jobs are scheduled according to $\sigma \in S_n$ defined under (a). The processing times of the n-job instance corresponding to the remaining jobs are $\lambda, p_2, \ldots, p_{q-1}, p_q, p_{q+2}, \ldots, p_{n+1}$.

If we want to prove our result by induction on n, then it is obvious that we should take an n-job instance with permutation σ, but it is not clear which processing times to use since the processing times in (a) and (c) differ with respect to p_q and p_{q+1}. Hence, we can not use induction in exactly the same way as in the previous section, because it is not clear to which n-job instance the induction hypothesis should be applied.

To be able to use induction, we introduce a more general class of problems, which is defined such that the data in (a) and (c) belong to the same problem instance. Note that the data in (c) differ from the data in (a) in only one aspect: a job with processing time p_{q+1} is replaced by a job with processing time p_q. We may view this as if we are considering the same job, which is crashed, i.e., it has its processing time reduced. Therefore, we will analyze the number of different assignments of jobs to machines for the following class of problems:

> Given are $\lambda = p_1$, $p_2 \leqslant p_3 \leqslant \cdots \leqslant p_n$, $\pi \in S_n$ and $0 \leqslant t_1 < t_2 < \cdots < t_s$ (s arbitrary). The list scheduling rule corresponding to π is applied while increasing λ from zero to infinity, and whenever $\lambda = t_i$ for some i the current data changes as follows: the lowest indexed job with a processing time greater than t_i has its processing time reduced to t_i.

To see that this is a useful problem class, first note that the problem instance defined by $p_1 = \lambda, p_2, \ldots, p_{q-1}, p_{q+1}, p_{q+2}, \ldots, p_{n+1}, \sigma, s = 1$ and $t_1 = p_q$, yields for $0 \leqslant \lambda < p_q$ and $p_{q+1} \leqslant \lambda < \infty$ the n-job instances described in (a) and (c) respectively. Moreover, the subset of this problem class obtained by defining no t-values ($s = 0$) is the set of permutation list scheduling problems we are ultimately interested in.

Finally, note that at most $n-1$ t-values are *effective*, i.e., there is a job such that the processing time of that job is decreased to that t-value. This means that we may assume that $s < n$ holds and that all t-values are effective.

We define the set of *reassignment points* to consist of 0 and

- all values of λ for which a reassignment of jobs to machines actually occurs,
- all t-values, except those for which job 1 and the job that is crashed are both the only jobs scheduled on their machines at the moment that λ becomes equal to the t-value.

For the t-values that are excluded above it is clear that the assignment of jobs to machines will not change. Of course, the set of reassignment points may still contain t-values for which this is also the case. However, the property that we are going to use

is that the assignment of jobs does not change when λ is varied in the interior of the interval defined by two consecutive reassignment points. Furthermore, because we included the t-values, it is easy to verify that on the interior of such intervals the completion time of the minimum machine is a non-decreasing, piecewise linear, continuous function of λ.

Let us define a_n to be the maximum number of reassignment points over all n-job instances. Note that this definition is with respect to all permutation list scheduling heuristics and all sequences of effective t-values. In particular $a_m = 1$, since by definition all t-values are not reassignment points in this case. Considering again all n-job instances, we define b_n to be the maximum number of breakpoints of Q_n^π (the completion time of the minimum machine) which occur in the interior of intervals defined by consecutive reassignment points. Note that Q_n^π is continuous on each of these intervals and that each open interval contains at most one breakpoint of this function. This implies that $b_m = 1$.

Proposition. *It holds that* $a_{n+1} \leqslant a_n + b_n + 2$ *and* $a_{n+1} + b_{n+1} \leqslant 2(a_n + b_n) + 2$ *for all* $n \geqslant m$.

Proof. Consider the problem instance defined by $p_1 = \lambda$, $p_2 \leqslant p_3 \leqslant \cdots \leqslant p_{n+1}$, $\pi \in S_{n+1}$, $\pi(q) = n + 1$, and $t_1, \ldots, t_s$. Define c and d to be the processing times of job q respectively job $q + 1$ at the end of the procedure in which λ is varied from zero to infinity. Because jobs q and $q + 1$ may be crashed jobs, it holds that $c \leqslant p_q$, $d \leqslant p_{q+1}$. Furthermore, it is easily seen that $c \leqslant d$. Let us first assume that $q \neq n + 1$ and $c < d$. Three different situations occur when λ is varied.

(i) For $0 \leqslant \lambda < c$ job q is the last job scheduled. Job q and $q + 1$ have processing times p_q and p_{q+1} respectively.

(ii) For $c \leqslant \lambda < d$ job 1 is the last job scheduled.

(iii) For $d \leqslant \lambda < \infty$ job $q + 1$ is the last job scheduled. Job q and $q + 1$ have processing times c and d respectively.

We will consider the n-job instance defined by $p_1 = \lambda$, $p_2 \leqslant \cdots \leqslant p_{q-1} \leqslant p_{q+1} \leqslant \cdots \leqslant p_{n+1}$, $\sigma \in S_n$ with $\sigma(j) = \pi(j)$, $j = 1, \ldots, q - 1$, $\sigma(j) = \pi(j + 1)$, $j = q, \ldots, n$, and t-values defined to ensure that for $0 \leqslant \lambda \leqslant c$ and $d \leqslant \lambda \leqslant \infty$ the schedules of the n-job instance are identical to the partial schedules of the $(n + 1)$-job instance if the last scheduled job (job q respectively job $q + 1$) is ignored. All other jobs are reduced to the same value as in the $(n + 1)$-job instance. This can be achieved by first deleting d from the set of t-values of the $(n + 1)$-job instance (if present) and then adding c to the remaining set (if not yet present).

First we focus on situations (i) and (ii). Consider an open interval defined by two consecutive reassignment points of the n-job instance. If the interval does not contain a breakpoint of Q_n^σ, then there is a machine which is the minimum machine for the whole interval. So in this case adding a job to the minimum machine will still lead to only one assignment of jobs to machines in such an interval. By definition there are at most b_n intervals in which Q_n^σ does have a breakpoint. So we will have at most b_n additional reassignment points. Note that this statement is true irrespective of the

processing time of the job that is added. Therefore, if we forget about situation (ii), the total number of reassignment points of the $(n+1)$-job instance can be bounded by $a_n + b_n$. Let us now look at how the interval $[c,d)$ fits into this picture. First note that since job 1 is scheduled last, there is only one assignment in $[c,d)$. Hence, this interval introduces at most two additional reassignment points, namely c and d. Therefore, the number of reassignment points of the $(n+1)$-job instance is bounded by $a_n + b_n + 2$. This proves the first inequality of the proposition.

We are now going to prove the second inequality. Q^{π}_{n+1} can have at most one breakpoint in the open interval defined by consecutive reassignment points of the $(n+1)$-job instance. We have just proved that there are at most $a_n + b_n$ of such intervals if we forget about the interval $[c,d)$. In the interior of $[c,d)$ there can be at most one breakpoint of Q^{π}_{n+1}. Therefore, an upper bound on the total number of reassignment points plus breakpoints of Q^{π}_{n+1} would be $2(a_n + b_n) + 3$, where the last term comes from the possible reassignment points c and d, and the possible breakpoint of Q^{π}_{n+1} in (c,d). However, we will show that the constant 3 can be reduced to 2. Note that the worst case of 3 additional points only occurs if $[c,d)$ does not contain a reassignment point of the n-job instance or a breakpoint of Q^{σ}_n, because otherwise the bound $a_n + b_n$ on the total number of different reassignment points, which was derived ignoring the interval $[c,d)$, could be lowered by at least 1. Therefore, we may assume in the sequel that $[c,d)$ does not contain a breakpoint of Q^{σ}_n and that this interval is contained in an interval (e,f), with e and f two consecutive reassignment points of the n-job instance. It suffices to show that in that case Q^{π}_{n+1} does not have a breakpoint in (c,d).

Since c is a t-value but not a reassignment point of the n-job instance, it follows from the definition of reassignment points that when λ equals c, both job 1 (with processing time λ) and the job that is crashed (from p_{q+1} to c) are scheduled as the only jobs on their machine, say machine M_1 and M_2 respectively. Suppose Q^{σ}_n is strictly increasing over the interval (e,c), i.e., M_1 is the minimum machine. Then there is a breakpoint at $\lambda = c$, because M_2 becomes the minimum machine. This is a contradiction with our assumption that $[c,d)$ does not contain a breakpoint of Q^{σ}_n. Therefore, we only have to consider the cases that Q^{σ}_n is constant on $[e,c)$ or that it has a breakpoint in the interior of this interval.

First assume that Q^{σ}_n is constant on $[e,c)$. Consider the n-job instance for $\lambda \in [e,f)$. There is a minimum machine M_3 with total processing time less than or equal to e. This follows from the fact that machine M_1 containing only job 1 has a completion time of e at $\lambda = e$ and is increasing as λ increases. When $\lambda = c$ the schedule of the $(n+1)$-job instance can be obtained from the schedule of the n-job instance by replacing job 1 by job q, and placing job 1 on minimum machine M_3 (see Fig. 5(a)). Note that for the $(n+1)$-job instance job q has been crashed to c. However, job $q+1$ has not been crashed (yet). This explains the difference between the n-job and $(n+1)$-job instance with respect to the processing time of the job on M_2. Since M_1 (which only contains job q) has a smaller completion time than M_3 (which contains job 1), Q^{π}_{n+1} is constant on $[c,d)$. Hence, no breakpoint occurs in this interval.

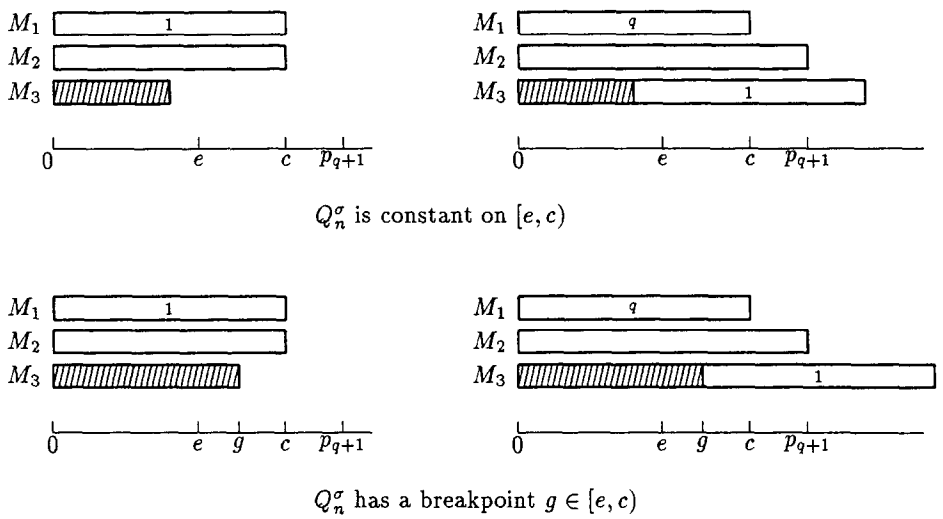

Fig. 5. Partial schedules at $\lambda = c$; on the left the n-job instance, on the right the $(n+1)$-job instance.

We are left with the case that Q_n^σ has a breakpoint $g \in (e, c)$. Then there is a minimum machine M_3 which has a total completion time equal to g. This follows from the fact that for $\lambda \geqslant g$ the minimum completion time remains constant. We are now in the same situation as in the previous case (see Fig. 5(b)). Hence, no breakpoint occurs.

This completes our proof for the case that $\pi(n+1) \neq n+1$ and $c \neq d$. The other cases are easier to analyze and lead to the same result. □

We are now able to establish the upper bound 2^{n-m+1} on the number of different assignments when an arbitrary permutation list scheduling heuristic is used. As we have pointed out before, we can take $a_m = b_m = 1$. Using the proposition it can easily be shown by induction that $a_n \leqslant 2^{n-m+1}$ and $a_n + b_n \leqslant 2^{n-m+2} - 2$ for all $n \geqslant m$. Hence, $A_n^\pi \leqslant a_n \leqslant 2^{n-m+1}$ and $B_n^\pi \leqslant 2A_n^\pi \leqslant 2^{n-m+2}$ for all permutation list scheduling rules π.

For $m = 2$ we can derive a tighter upper bound for B_n^π. This bound is valid for all scheduling rules R for which Z_n^R is a continuous piecewise linear function of the processing time of job 1 with linear parts that are constant or have slope one. The constant term of the function describing a linear part of Z_n^R always equals the sum of processing times of a subset of the jobs $2, 3, \ldots, n$. It also follows from the shape of Z_n^R that each constant term can occur at most once. Since there are only 2^{n-1} subsets of $\{2, 3, \ldots, n\}$ this leads to an upper bound of 2^{n-1} on the number of breakpoints. Note that this bound is valid for all values of $m \geqslant 2$, but it constitutes only an improvement on the bound derived in this section for $m = 2$.

Table 1
Summary of worst case analysis results

	Assignments		Breakpoints	
	Lower bound	Upper bound	Lower bound	Upper bound
SPT	n	n	n/m	n/m
LPT	2^{n-m}	2^{n-m}	$(\sqrt{2})^{n-m}$	2^{n-m}
Permutation	n^*	2^{n-m}	n/m^*	2^{n-m}
Optimal	$(\sqrt{2})^{n-m}$	2^n	$(\sqrt{2})^{n-m}$	2^n

6. Summary and concluding remarks

Our analysis of the number of different assignments of jobs to machines and the number of breakpoints of Z_n has resulted in intervals containing their worst case value. We have summarized the result in Table 1. "Permutation" represents an arbitrary but fixed permutation list scheduling rule. When a lower bound is given, e.g. 2^{n-m} for the number of assignments, then this means that the worst case number of assignments is $\Omega(2^{n-m})$. An upper bound of 2^{n-m} on the number of assignments means that this number is $O(2^{n-m})$. An asterisk indicates a conjectured result.

The lower bound for the number of assignments for the LPT rule follows from Example 1. The lower bound on the number of assignments and breakpoints for any optimal algorithm follows from the fact that the LPT schedule is optimal for Example 3. The upper bounds for optimal algorithms follow from the observation in the last paragraph of Section 5.

We conjecture that for all permutation list scheduling heuristics n is a lower bound on the worst case number of different assignments and n/m is a lower bound on the worst case number breakpoints. Note that if the conjecture is true, the SPT rule and LPT rule are extreme cases of permutation list scheduling heuristics in the sense that the worst case number of assignments corresponds to the lower bound respectively the upper bound for the overall class.

If we look at schedules instead of assignments, i.e., we also distinguish between the order in which jobs are processed on the machines, the SPT rule is again an extreme case. The SPT schedule changes only when the order of the processing time changes, i.e., there are only n different schedules. Because permutation list scheduling heuristics are defined with respect to the non-decreasing order of the processing time, n is a trivial lower bound on the number of schedules.

Thus, we have found supporting evidence for the intuitively plausible notion that the performance of an algorithm and its sensitivity are correlated in that a good performance is identical with a high degree of sensitivity. An important research question is how to formalize this relationship between sensitivity and performance. To

answer this question we need a proper index to measure algorithmic sensitivity. The functions A_n^π and B_n^π used in this paper are a first step in that direction.

Acknowledgements

We would like to thank three anonymous referees for their comments on an earlier draft of this paper. The third author was supported by the Netherlands Organization for Scientific Research (NWO) under grant no. 611-304-017. Part of this research was carried out while the fourth author was visiting the Operations Research Center at the Massachusetts Institute of Technology with financial support of NWO.

References

[1] J.B.G. Frenk and A.H.G. Rinnooy Kan, The asymptotic optimality of the lpt rule, Math. Oper. Res. 12 (1987) 241–254.
[2] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System Tech. J. 45 (1966) 1563–1581.
[3] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17 (1969) 263–269.
[4] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: S.C. Graves, A.H.G. Rinnooy Kan and P.H. Zipkin, eds., Logistics of Production and Inventory, Handbooks in Operations Research and Management Science 4 (North-Holland, Amsterdam, 1993) Ch. 9, 445–522.
[5] A.P.M. Wagelmans, Sensitivity analysis in combinatorial optimization, Ph.D. Thesis, Econometric Institute, Erasmus University, Rotterdam, Netherlands (1990).