SENSITIVITY ANALYSIS OF LIST SCHEDULING HEURISTICS

by

A.W.J. Kolen, A.H.G. Rinnooy Kan,
C.P.M. Van Hoesel, and
A.P.M. Wagelmans

OR 229-90 October 1990

SENSITIVITY ANALYSIS OF LIST SCHEDULING HEURISTICS

A.W.J. Kolen[1]
A.H.G. Rinnooy Kan[2]
C.P.M. Van Hoesel[2,3]
A.P.M. Wagelmans[2,4]

October 1990

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 one of the job processing times is gradually increased, the schedule produced by a list scheduling rule will be affected in a manner reflecting its sensitivity to data perturbations. We analyze this phenomenon and obtain analytical support for the intuitively plausible notion that the sensitivity of a list scheduling rule increases with the quality of the schedule produced.

Keywords: sensitivity analysis, list scheduling, heuristics, robustness, scheduling

1) Faculty of Economics, Limburg University, P.O. Box 616, 6200 MD Maastricht, The Netherlands
2) Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands
3) Supported by the Netherlands Organization for Scientific Research (NWO) under grant no. 611-304-017
4) On leave at the Operations Research Center, Room E 40-164, Massachusetts Institute of Technology, Cambridge, MA 02139

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* 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 (1990)). 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, 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, jobs with *processing times* $p_1,\dots,p_n$ 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 j-th job is denoted by C_j, then this criterion amounts to the minimization of $Z=\max_{j=1,\dots,n}\{C_j\}$.

This NP-hard problem has been the subject of extensive research ; many heuristics for its solution have been proposed (for a review see Lawler et al. (1989)). We will concentrate on a class of heuristics known as *list scheduling rules*. Such rules are defined by a *priority list* in 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 a more precise formulation, we restrict our attention to priority lists that are defined by the relative sizes of the processing times. Thus, when the jobs are initially ordered according to non-decreasing processing time, a list scheduling rule corresponds to a permutation π with the j-th job of this order appearing in the $\pi(j)$-th position in the list. Two well-known examples of such *permutation list scheduling rules* are the *SPT* (*Shortest Processing Time*) and the *LPT* (*Longest Processing Time*) rules, defined by $\pi(j)=j$ and $\pi(j)=n-j+1$, $j=1,...,n$, 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 (1966)); for the LPT rule, this factor is at most $4/3-1/(3m)$ (see Graham (1969)); both bounds are tight. A similar difference in quality emerges from a probabilistic analysis: under appropriate assumptions, 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 (1987)).

From now on we assume that the jobs are numbered such that initially $p_1 \leq p_2 \leq \cdots \leq p_n$ holds. In what follows, we will investigate the effect on the performance of the SPT rule, the LPT rule and the other rules in this class under a simple type of data perturbation: we allow p_1, the initially smallest processing time, to increase from zero to infinity. For a given heuristic H, given n and a given value $p_1 = \lambda$, it is then natural to study the solution value $Z_n^H(\lambda)$ as a function of λ, in the sense that this function provides information on the robustness of heuristic H. As we will see in Section 2, Z_n^H is a *continuous piecewise linear* function; the *worst case number of breakpoints* B_n^H will then serve as a first indication of how quickly the solution value adapts to changes in the problem data. Actually, there turns out to be a simple relationship between B_n^H and the *worst case number of different assignments* A_n^H of jobs to machines as λ increases from zero to infinity, i.e., the number of different partitions of the jobs into m subsets that can occur during this increase. Sometimes, it will turn out to be convenient to carry out the analysis in terms of A_n^H rather than B_n^H.

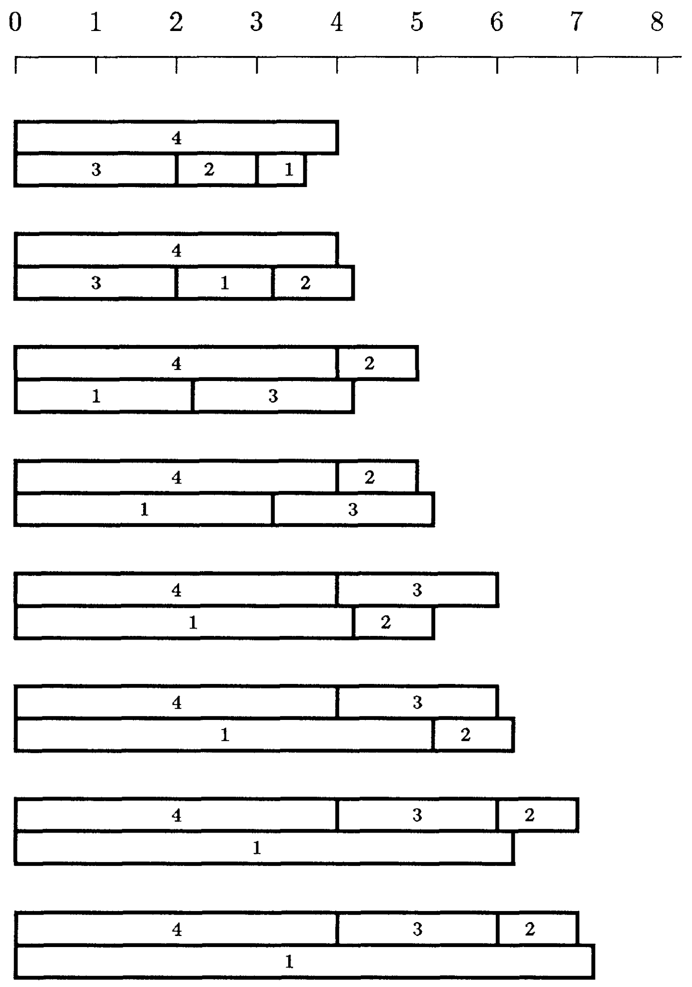
In Sections 3 and 4, we look at the SPT rule and the LPT rule, respectively. We establish that $A_n^{SPT} \leq n$ and $B_n^{SPT} \leq 2\lceil n/m \rceil$; both upper bounds are tight. In contrast $A_n^{LPT} \leq 2^{n-m}$ and $B_n^{LPT} \leq 2^{n-m+1}$; the first bound is tight, and there exists an example 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, $A_n^{\pi} \leq 2^{n-m+1}$ and $B_n^{\pi} \leq 2^{n-m+2}$. Some concluding remarks are collected in Section 6.

2. The function Z_n^{π}

In this section we will show that, for any list scheduling rule defined by a permutation π on the processing time λ for job 1 and processing time p_j for job j, j=2,3,....,n, with $p_2 \leq p_3 \leq \ldots \leq p_n$, Z_n^{π} is a continuous piecewise linear function of λ, $0 \leq \lambda \leq \infty$. 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 on two machines and four jobs with 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 Figure 1.



$0 \leq \lambda < 1, \quad Z_4^{LPT}(\lambda) = 4$

$1 \leq \lambda < 2, \quad Z_4^{LPT}(\lambda) = 3+\lambda$

$2 \leq \lambda < 3, \quad Z_4^{LPT}(\lambda) = 5$

$3 \leq \lambda < 4, \quad Z_4^{LPT}(\lambda) = 2+\lambda$

$4 \leq \lambda < 5, \quad Z_4^{LPT}(\lambda) = 6$

$5 \leq \lambda < 6, \quad Z_4^{LPT}(\lambda) = 1+\lambda$

$6 \leq \lambda < 7, \quad Z_4^{LPT}(\lambda) = 7$

$7 \leq \lambda < \infty, \quad Z_4^{LPT}(\lambda) = \lambda$

Figure 1: *Example LPT – schedules*

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 and hence the list will change, although the relative order of the jobs 2,3,...,n will remain the same. If λ is equal to one of the processing times $p_2, \dots, p_n$, 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. To establish our result, it is sufficient to show that when λ varies between two existing 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. Whenever a job which is below job 1 on the list is ready to be scheduled and there exists a choice of machines to schedule this job on, we will always choose a machine not containing job 1. This assumption does not effect the distribution of total processing time among the machines and therefore does not effect Z_n^π. Let us refer to this assumption as the *tie breaking assumption.*

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 Figure 2 for the case of two machines M_1 and M_2.

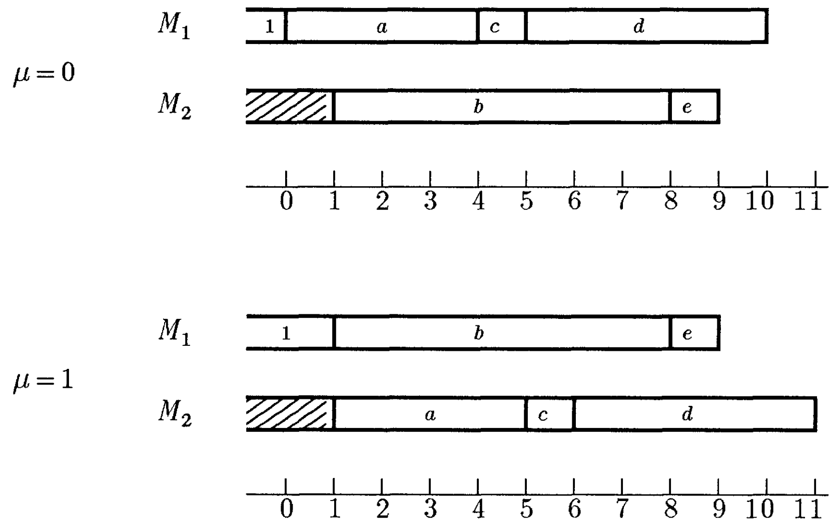


Figure 2: A switch of tails

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 $(0 \leq \mu \leq 1)$ 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 a further increase of μ will now not affect the schedule. We refer to the transformation depicted in Figure 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 $1 \leq \mu \leq 3$, whereas Z_n^π increases with slope one for $3 \leq \mu \leq 4$.

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. Given processing times p_j for job j, $j=1,2,...,n$, with $p_1 \leq p_2 \leq \ldots \leq p_n$, the SPT-schedule is illustrated in Figure 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. Since $p_1 + p_{m+1} \geq p_m$ machine M_1 now is the maximum machine and job $m+2$ is scheduled on M_2. Since $p_2 \geq p_1$ and $p_{m+2} \geq p_{m+1}$ machine M_2 now is 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 $i = [(j-1) \bmod m] + 1$, $j=1,...,n$, and the maximum machine is the machine processing job n.

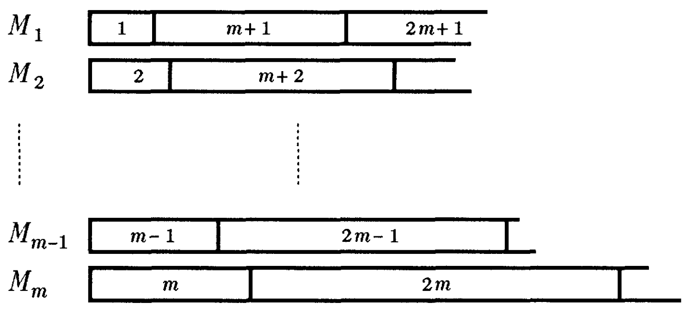


Figure 3: *SPT – schedule*

Let us now consider the case that the processing time λ of job 1 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. This bound is attained if $p_2 < p_3 < \ldots < p_n$. 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 $[(n-1)\text{mod}m]$+1 and always contains $\lceil n/m \rceil$ jobs. Therefore job 1 can be on the maximum machine at most $\lceil n/m \rceil$ times. Since A_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}, i.e., $B_n^{SPT} \leq 2\lceil n/m \rceil$. It is easy to see that if we take a problem instance with $n\text{mod}m \neq 1$ and $p_2 < p_3 < \ldots < p_n$, then the upper bound on the number of breakpoints is attained. 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 $p_2 \leq p_3 \leq \ldots \leq p_n$.

We first prove that the number of different LPT-assignments, A_n^{LPT}, is at most equal to 2^{n-m}, $n \geq m$. We do so by induction. For $n = m$, the statement is trivial. Assume that the statement holds for n ($n \geq m$), 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 within each interval the assignment of jobs to machines for this n-job instance remains the same. How are the (n+1)-job assignments related to the n-job ones?

To facilitate our analysis we define 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.

First, assume that $\lambda \geq p_2$. Thus, job 2 is the smallest one and hence assigned last to the minimum machine of the n-job instance. Consider a particular λ-interval in which the assignment of the n-job instance does not vary. What can happen to the minimum machine in such an interval? Its index can only change once, namely when the initial minimum machine 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.

Now, consider the λ-interval $[0,p_2)$ for the (n+1)-job instance and 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 $\lambda < p_3$, it holds that $[0,p_2)$ is contained in $[0,a)$. Analogously the assignment of the (n+1)-job instance cannot change for $\lambda < p_2$. Hence, there will be at most two different assignments in $[0,a)$ if Q_n^{LPT} has no breakpoint in (p_2,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 can 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.

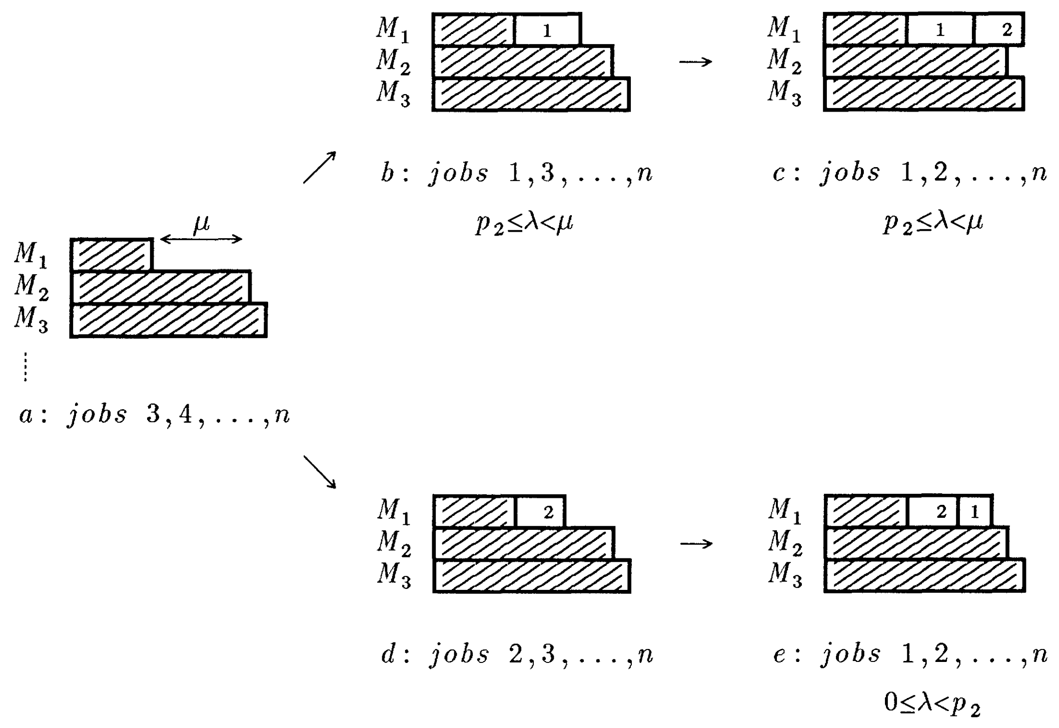


Figure 4: Schedules for $\lambda \in [0,\mu)$.

In Figure 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 difference between the total processing time of M_1 and M_2 follows from the fact that Q_n^{LPT} has a breakpoint $\mu \in [0,a)$, i.e., M_1 is the minimum machine as long as the processing time of job 1 is less than or equal to μ. The schedule of the n-job instance on $[p_2,\mu)$ is given in Figure 4.b. On $[p_2,\mu)$ the schedule of the $(n+1)$-job instance is obtained from the n-job instance by scheduling job 2 on the minimum machine (in this case M_1). This schedule is given in Figure 4.c. On $[0,p_2)$ the schedule of the jobs $2,3,\ldots,n$ is given by Figure 4.d. This schedule is obtained from the schedule in Figure 4.a by scheduling the smallest job (job 2) on the minimum machine. Note that $p_2 < \mu$. On $[0,p_2)$ the schedule of the $(n+1)$-job instance is obtained by scheduling job 1 on the minimum machine. This schedule is given in Figure 4.e. By comparing the schedules in Figures 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 $A_{n+1}^{LPT} \leq 2A_n^{LPT} \leq 2^{n-m+1}$ and this establishes the desired result.

Our final observation is that B_n^{LPT}, the number of breakpoints, is at most equal to $2A_n^{LPT}$ and hence bounded by 2^{n-m+1}. The argument is simple: each assignment defines at most two breakpoints for 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 the number of different assignments and 2^{n-m+1} on the number of breakpoints are tight. The following example shows that this is the case for $m=2$.

Example 1

Suppose $m=2$ and the processing time of job j is $p_j = 2^{j-2}$, $j=2,\ldots,n$. Note that $p_j = \sum_{k=2}^{j-1} p_k + 1$, $j=3,\ldots,n$. This implies that $Z_n^{LPT}(0) = 2^{n-2}$. If job 1 has a processing time $\lambda \geq \sum_{j=2}^{n} p_j = 2^{n-1}-1$, then job 1 is the only job scheduled on the maximum machine. Therefore all breakpoints must lie in the interval $[0, 2^{n-1}-1]$. We will prove that all subintervals on which Z_n^{LPT} is increasing have a length of one. Since $Z_n^{LPT}(2^{n-1}-1) - Z_n^{LPT}(0) = 2^{n-1}-1-2^{n-2} = 2^{n-2}-1$, this implies that there are 2^{n-1} breakpoints (including zero).

Assume that job 1 is scheduled on M_1 and the total processing time on M_1 is given by $C_1(\lambda)$; C_2 denotes the total processing time of M_2. It is sufficient to prove that $C_1(\lambda) - C_2 \leq 1$ for all $\lambda \in [0, 2^{n-1}-1]$. To prove this, we will use the property that for the LPT-schedule the starting time of the last job scheduled on M_1 is less than or equal to the starting time of all jobs with smaller processing time scheduled on M_2. We distinguish between two cases.

(i) Assume job 1 is the last job scheduled on M_1. Let the processing time λ of job 1 satisfy $p_k \leq \lambda \leq p_{k+1}$ for some k $(1 \leq k \leq n)$, where $p_1 \equiv 0$ and $p_{n+1} \equiv 2^{n-1}-1$. The property mentioned above yields $C_1(\lambda) - \lambda \leq C_2 - \sum_{j=2}^{k} p_j$. Hence,

$$C_1(\lambda) - C_2 \leq \lambda - \sum_{j=2}^{k} p_j = \lambda - p_{k+1} + 1 \leq 1$$

where the last inequality follows from $\lambda \leq p_{k+1}$.

(*ii*) Assume job k $(k>2)$ is the last job scheduled on M_1.
Then

$$C_1(\lambda) - p_k \leq C_2 - \sum_{j=2}^{k-1} p_j$$

or

$$C_1(\lambda) - C_2 \leq p_k - \sum_{j=2}^{k-1} p_j = 1.$$

We leave it to the reader to prove that there are 2^{n-2} different assignments to jobs to machines (see also Example 3). The assignment of jobs with processing time $\lambda \in [2l, 2l+2]$, $l=0,\dots,2^{n-2}-1$, for job 1 can be computed as follows: if $2^{n-2}+l = \sum_{i=2}^{n} a_i 2^{i-2}$ with $a_i \in \{0,1\}$ (i.e., $2^{n-2}+l$ is written in binary representation), then the set of jobs scheduled on M_2 is defined by $\{j \mid a_j = 1\}$. An example with $n=4$ was given in Section 2.

The upper bound of 2^{n-m} on the number of different assignments can also be attained for $m>2$ as is shown in Example 2.

Example 2
Let $N = n-(m-2)$, $p_j = 2^{j-2}$ $(j=2,3,\dots,N)$, $p_j = 2^{N-1}$ $(j=N+1,\dots,n)$. From Example 1 it follows that the $m-2$ largest jobs 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 $1,2,\dots,N$ 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 be attained for $m>2$. 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 $3,4,\dots,2n$ satisfy $p_{2j+1} = 2^{j+1} - 3 + p_2$, $p_{2j+2} = 2^{j+1} + 2^j - 3 + p_2$ $(j=1,2,\dots,n-1)$. We consider the

processing time λ of job 1 with $\lambda \in [p_{2n}-2^{n-1}, p_{2n}+2^{n-1}-1]$. 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 $\lambda = p_{2n}-2^{n-1}+2l$, $(l=0,\dots,2^{n-1}-1)$ 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 as follows. We claim that if $a_i=1$, then job $2i$ is scheduled on M_1 and job $2i+1$ is scheduled on M_2, else job $2i$ is scheduled on M_2 and job $2i+1$ on M_1. If we define $a_n=0$ and job $2n+1$ to be equal to job 1, then the claim holds for a_n. Suppose the claim holds for $a_k, a_{k+1}, \dots, a_n$, $2 \le k \le n$. The total processing time on M_1 of all jobs scheduled so far is given by C_1 where

$$\begin{aligned} C_1 &= p_{2n} - 2^{n-1} + 2\sum_{i=1}^{n-1} a_i 2^{i-1} && \text{(the processing time of job 1)} \\ &+ \sum_{i=k}^{n-1} a_i(2^i + 2^{i-1} - 3 + p_2) && \text{(all even jobs scheduled on } M_1\text{)} \\ &+ \sum_{i=k}^{n-1} (1-a_i)(2^{i+1} - 3 + p_2) && \text{(all odd jobs scheduled on } M_1\text{)} \end{aligned}$$

The total processing time C_2 on M_2 of all jobs scheduled so far is given by

$$C_2 = p_{2n} + \sum_{i=k}^{n-1} (1-a_i)(2^i + 2^{i-1} - 3 + p_2) + \sum_{i=k}^{n-1} a_i(2^{i+1} - 3 + p_2) \tag{1}$$

Calculating $C_2 - C_1$ we find

$$C_2 - C_1 = 2^{k-1} - \sum_{i=1}^{k-1} a_i 2^i \tag{2}$$

If $a_{k-1}=1$, then $C_2 \le C_1$ and job $2k-1$ can be scheduled on M_2. Since $-\sum_{i=1}^{k-2} a_i 2^i + p_{2k-1} = -\sum_{i=1}^{k-2} a_i 2^i + (2^k - 3 + p_2) > 0$, job $2k-2$ is scheduled on M_1.

If $a_{k-1}=0$, then $C_2 > C_1$ and job $2k-1$ is scheduled on M_1. Since $\sum_{i=1}^{k-2} a_i 2^i - 2^{k-1} + p_{2k-1} \ge 0$, job $2k-2$ is scheduled on M_2.

This proves the claim. Also note that the total processing time of M_2 is one more than that of M_1, because (2) is also valid for $k=1$.

From the tie breaking assumption and the fact that all data are integer it follows that Z_{2n}^{LPT} is constant on $[p_{2n}-2^{n-1}+2l, p_{2n}+2^{n-1}+2l+1]$. On $[p_{2n}-2^{n-1}+2l+1, p_{2n}+2^{n-1}+2l+2]$ Z_{2n}^{LPT} must be increasing with slope one because $Z_{2n}^{LPT}(p_{2n}+2^{n-1}+2l+2)-Z_{2n}^{LPT}(p_{2n}+2^{n-1}+2l)=1$. The proof of this fact is as follows: if l is even then $a_i=0$ and $l+1=\sum_{i=1}^{n-1}\bar{a}_i 2^{i-1}$ with $\bar{a}_1=1$ and $\bar{a}_i=a_i$ for $i=2,\dots,n-1$; if l is odd then there is a k such that $a_k=0$, $a_i=1$ for $i=1,\dots,k-1$ and $l+1=\sum_{i=1}^{n-1}\bar{a}_i 2^{i-1}$ with $\bar{a}_k=1$, $\bar{a}_i=0$ for $i=1,\dots,k-1$ and $\bar{a}_i=a_i$ for $i=k+1,\dots,n-1$. In both cases we can calculate the difference between $Z_{2n}^{LPT}(p_{2n}+2^{n-1}+2l+2)$ and $Z_{2n}^{LPT}(p_{2n}+2^{n-1}+2l)$ by using expressions analogous to (1), because we have already shown that the makespans are determined by the completion time of M_2. Then the result follows directly.

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 $p_2 \geq 2^{N/2-1}$, $p_3,\dots,p_N$ as defined above for the 2-machine case and $p_{N+1},\dots,p_n$ equal to the makespan for the 2-machine case 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 λ equals $p_N-2^{N/2-1}$ jobs $N+1,\dots,n$ are all scheduled as the only jobs on their machine and jobs $1,2,\dots,N$ are scheduled on the two remaining machines, say M_1 and M_2, in the same way as in the 2-machine case. For $\lambda \in [p_N-2^{N/2-1}, p_N+2^{N/2-1}-1]$ the jobs $1,2,\dots,n$ will be scheduled on M_1 and M_2. This follows from the earlier result that for two machines $Z_N^{LPT}(p_N+2^{N/2-1}-1)-Z_N^{LPT}(p_N-2^{N/2-1})=2^{N/2-1}-1$. Since p_2 is the smallest job and $p_2 \geq 2^{N/2-1}$ the result follows. Using 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 $2^{n-m}=2$ and $2^{n-m+1}=4$ respectively. However, this example has four different assignments and six breakpoints of Z_4^{π}. Our main result of this section is an upper bound of 2^{n-m+1} on the number of different assignments of jobs to machines in a permutation list scheduling heuristic for n jobs on m machines whenever the processing time of one job is varied.

As in Section 4 the result will be proved by induction on n ($n \geq m$). 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 $p_2 \leq p_3 \leq \ldots \leq p_{n+1}$. When the processing time of job 1, i.e. λ, varies, three different situations occur:

(a) for $0 \leq \lambda \leq p_q$ job q is the last job scheduled and the remaining jobs are scheduled according to $\sigma \in S_n$ defined by $\sigma(j)=\pi(j)$, $j=1,\ldots,q-1$, $\sigma(j)=\pi(j+1)$, $j=q,\ldots,n$. The processing times of the n-job instance corresponding to the remaining jobs are λ, $p_2,\ldots,$ p_{q-1}, p_{q+1}, $p_{q+2},\ldots,$ p_{n+1},

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

(c) for $p_{q+1} \leq \lambda < \infty$ 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 λ, $p_2,\ldots,$ p_{q-1}, p_q, p_{q+2}, p_{n+1}.

If we want to prove our result by induction on n, then it is obvious that we have to use 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}.

The general class of problems is defined such that the data in (a) and (c) belong to the same problem instance. The class of problems is defined as follows:

Given $p_1 = \lambda$, $p_2 \le p_3 \ldots \le p_n$, $\pi \in S_n$ and $0 \le t_1 < t_2 < \ldots < t_s$ (s arbitrary), we study the number of different assignments of jobs to machines when λ is increased from zero to infinity given the rule that whenever $\lambda = t_i$ for some i the current data changes in the sense that for the lowest indexed job with a processing time greater than t_i, its processing time is permanently reduced to t_i.

Note that at most $n-1$ t-values are *effective*, i.e., such that 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.

The problem instance defined by $p_1 = \lambda$, $p_2, \ldots, p_{q-1}, p_{q+1}, p_{q+2}, \ldots, p_{n+1}$, σ, $s = 1$ and $t_1 = p_q$, corresponds to (a) and (c) above for values of λ given by $0 \le \lambda < p_q$ and $p_{q+1} \le \lambda < \infty$ respectively. 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.

For a given instance we define the set of *reassignment points* by the union of

- the set containing zero,
- the set of values of λ not equal to a t-value for which a reassignment of jobs to machines occurs and
- the set of effective t-values with the exception of those for which job 1 and the job for which the processing time is reduced to this t-value are the only jobs scheduled on their machines. (In general a reassignment occurs if λ becomes equal to a t-value, because job 1 and the job that has its processing time reduced swap positions in the permutation. However, for the t-values that are excluded here the assignment will clearly not change.)

Let us define a_n to be the maximum cardinality of a set of reassignment points if we consider all n-job instances, all permutation list scheduling heuristics and all values of $s < n$. Note that $a_m = 1$, since by definition all t-values are not reassignment points in this case. Let us define b_n to be the maximum number of breakpoints of Q_n^π (the completion time of the minimum machine) which occur strictly within the intervals defined by the reassignment points if we consider all n-job instances and all permutation list scheduling heuristics defined by a permutation $\pi \in S_n$. Note that within each interval Q_n^π is continuous and has at most one breakpoint. This implies that $b_m = 1$.

Proposition 1

For $n \geq m$ and a_n, b_n, a_{n+1}, b_{n+1} defined as above the following holds:

$$a_{n+1} \leq a_n + b_n + 2 \qquad \text{and} \qquad a_{n+1} + b_{n+1} \leq 2(a_n + b_n) + 2$$

Proof

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

(*i*) for $0 \leq \lambda < c$ job q is the last job scheduled,

(*ii*) for $c \leq \lambda < d$ job 1 is the last job scheduled,

(*iii*) for $d \leq \lambda < \infty$ job $q+1$ is the last job scheduled.

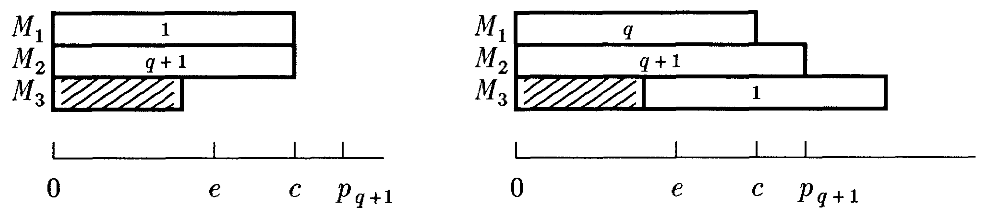
We will consider the n-job instance defined by $p_1 = \lambda$, $p_2 \leq \cdots \leq p_{q-1} \leq$ $\leq p_{q+1} \leq \cdots \leq p_{n+1}$, $\sigma \in S_n$ with $\sigma(j) = \pi(j)$, $j = 1, \ldots, q-1$, $\sigma(j) = \pi(j+1)$, $j = q, \ldots, n$, and t-values defined to ensure that the schedules obtained in (*i*) and (*iii*) correspond to the first n jobs scheduled from the $(n+1)$-job instance. These t-values must be such that the processing time of job $q+1$ has to be reduced to c and 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).

Consider the intervals defined by the set of reassignment points for the n-job instance. If such an interval does not contain a breakpoint of Q_n^σ, there is a machine which is the minimum machine during the whole interval. So adding a job to the minimum machine will lead to one assignment of jobs to machines in such an interval. There are at most b_n intervals in which Q_n^σ has a breakpoint. So we have at most b_n additional reassignment points. Therefore the total number of different reassignment points of the $(n+1)$-job instance can be bounded by $a_n + b_n$ if we forget about the interval $[c, d)$. Let us see how this last interval fits into this picture. First, it introduces at most two different reassignment points, namely c and d. Note that there is only one assignment in the interval $[c,d)$ since job 1 is scheduled last. Therefore the number of different reassignment points for the $(n+1)$-job instance is bounded by a_n+b_n+2. This proves the first part of the proposition.

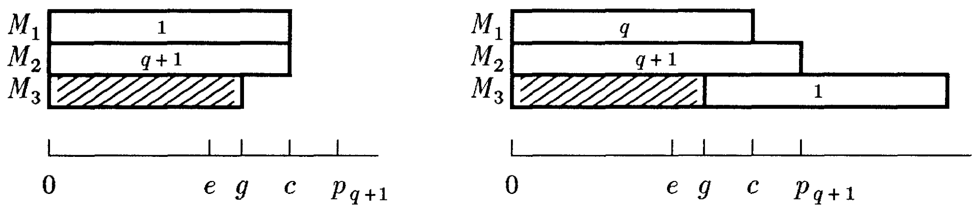
As we have seen before, Q^{π}_{n+1} can have at most one breakpoint within an interval defined by the set of reassignment points. Therefore, there can be at most a_n+b_n different breakpoints of Q^{π}_{n+1} outside of the interval $[c,d)$. In $[c,d)$ there can be at most one breakpoint of Q^{π}_{n+1}, so that an upper bound on the total number of reassignment points and breakpoints of Q^{π}_{n+1} would be $2(a_n+b_n)+3$, where the third term comes from c, d and the breakpoint of Q^{π}_{n+1} in $[c,d)$. We will show that the constant 3 can be reduced to 2. Note that the worst case of 3 additional points only occurs when $[c,d)$ does not contain a reassignment point or 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 $[c,d)$ is strictly contained in an interval $[e,f)$ for which the n-job instance does not have a reassignment. 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 for the n-job instance and not a reassignment point, it follows from the definition of reassignment points that job 1 with processing time $\lambda=c$ and job $q+1$ (the job reduced to c) are scheduled each as the only jobs on their machine, say machine M_1 and M_2 respectively. We distinguish between two cases:

(i) Q^{σ}_{n} is constant on $[e,c)$.
There is a machine M_3 which has a 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 $\lambda=c$ we know that the schedule obtained for the $(n+1)$-job instance is identical to the schedule of the n-job instance with job 1 replaced by job q and job 1 placed on the minimum machine (in this case: M_3). (See Figure 5.a). 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.



a: Q_n^σ *is constant on* $[e,c)$



b: Q_n^σ *has a breakpoint* $g \in (e,c)$

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

(*ii*) Q_n^σ has a breakpoint $g \in (e,c)$.

A breakpoint at g implies that there is a machine M_3 which has a total completion time of g. This follows from the fact that for $\lambda \geq g$ the minimum completion time remains constant. We are now in the same situation as in (*i*). When $\lambda = c$ the schedule obtained for the $(n+1)$-job instance is identical to the schedule of the n-job instance with job 1 replaced by job q and job 1 placed on the minimum machine (in this case: M_3). Since M_3 is not the minimum machine for the $(n+1)$-job instance (M_1 with completion time c is smaller) there is again no breakpoint of Q_{n+1}^π in $[c,d)$. (See Figure 5.b.)

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.

$\square$

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 $a_m = b_m = 1$. For $n > m$ we deduce that

$$\begin{aligned} a_n &\leq a_{n-1} + b_{n-1} + 2 \\ &\leq [2(a_{n-2} + b_{n-2}) + 2] + 2 = 2(a_{n-2} + b_{n-2}) + 4 \\ &\leq 2[2(a_{n-3} + b_{n-3}) + 2] + 4 = 4(a_{n-3} + b_{n-3}) + 8 \\ &\leq 2^{n-m-1}(a_m + b_m) + 2^{n-m} = 2^{n-m+1} \qquad (3) \end{aligned}$$

Of course, (3) implies the upper bound 2^{n-m+2} on the total number of breakpoints of Z_n^π for all permutation list scheduling rules π.

For $m = 2$ we can derive a tighter upper bound. 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 $m \geq 2$, but it constitutes only an improvement on the bound derived in this section for $m = 2$.

6. Concluding remarks

Our worst case analysis on the number of different assignments of jobs on machines and the number of breakpoints of Z_n has resulted in an interval containing their maximal value. We have summarized the results in Table 1. This table should be interpreted as follows. When a lower bound is given, e.g. 2^{n-m} for the number of assignments this means that the maximum number of assignments is $\Omega(2^{n-m})$. An upper bound of 2^{n-m} on the number of assignments means that the number of assignments is $O(2^{n-m})$. A $*$ indicates a conjectured result.

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

Table 1: Summary of worst case analysis results

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 with respect to worst case analysis n is a lower bound on the number of assignments and n/m a lower bound on the number breakpoints for all permutation list scheduling heuristics. 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 number of assignments corresponds to the lower bound respectively the upper bound with respect to the overall class.

If we look at schedules instead of assignments, i.e., we also distinguish between the order in which jobs are executed on a machine, the SPT-rule again is 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^H and B^H used in this paper are a first step in that direction.

References

Frenk, J.B.G., and Rinnooy Kan, A.H.G. (1987), "The asymptotic optimality of the LPT rule", *Mathematics of Operations Research* 12, 241 – 254

Graham, R.L. (1966), "Bounds for certain multiprocessing anomalies", *Bell System Technichal Journal* **45**, 1563 – 1581

Graham, R.L. (1969), "Bounds on Multiprocessing Timing Anomalies", *SIAM Journal on Applied Mathematics* **17**, 263 – 269

Lawler, E.L., Lenstra J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (1989), "Sequencing and scheduling: algorithms and complexity", Report 8934/A, Erasmus University Rotterdam, Rotterdam, The Netherlands

Wagelmans, A.P.M., "Sensitivity Analysis in Combinatorial Optimization" (1990), Ph.D. dissertation, Erasmus University Rotterdam, Rotterdam, The Netherlands