OR Spectrum manuscript No.
(will be inserted by the editor)

Worst-case analysis of the LPT algorithm for single processor scheduling with time restrictions

Oliver Braun · Fan Chung · Ron Graham

Received: date / Accepted: date

Abstract We consider the following scheduling problem. We are given a set S of jobs which are to be scheduled sequentially on a single processor. Each job has an associated processing time which is required for its processing. Given a particular permutation of the jobs in S, the jobs are processed in that order with each job started as soon as possible, subject only to the following constraint: For a fixed integer $B \geq 2$, no unit time interval $[x, x+1)$ is allowed to intersect more than B jobs for any real x. There are several real world situations for which this restriction is natural. For example, suppose in addition to the jobs being executed sequentially on a single main processor, each job also requires the use of one of B identical subprocessors during its execution. Each time a job is completed, the subprocessor it was using requires one unit of time in order to reset itself. In this way, it is never possible for more than B jobs to be worked on during any unit interval. In [1] it is shown that this problem is NP-hard when the value B is variable and a classical worst-case analysis of *List Scheduling* for this situation has been carried out. We prove a tighter bound for *List Scheduling* for $B \geq 3$ and we analyze the worst-case behaviour of the makespan $\tau_{LPT}(S)$ of *LPT* (longest processing time first) schedules (where we rearrange the set S of jobs into *non-increasing* order) in relation to the makespan $\tau_o(S)$ of optimal schedules. We show that *LPT* ordered jobs can be processed within a factor of $2-2/B$ of the optimum (plus 1) and that this factor is best possible.

Keywords scheduling · worst-case analysis · time restrictions · *LPT* (longest processing time first) algorithm

Oliver Braun
Trier University of Applied Sciences, Environmental Campus Birkenfeld, 55761 Birkenfeld, Germany
E-mail: o.braun@umwelt-campus.de

Fan Chung and Ron Graham
University of California, San Diego, USA
E-mail: fan@ucsd.edu, graham@ucsd.edu

1 Problem description

We are initially given a set $S = \{S_1, S_2, \ldots, S_n\}$ of jobs. Each job S_i has associated with it a length s_i. The jobs are all processed sequentially on a single processor. Only one job can be worked on at any point in time. A job S_i will be processed during a (semi-open) time interval $[\alpha, \alpha + s_i)$ for some $\alpha \geq 0$. A special type of job S_i, called a *zero-job*, has length $s_i = 0$. By convention, each zero-job is processed at some particular point in time. Furthermore, only one job can be worked on at any point in time except in the case of zero-jobs, where it is allowed for several zero-jobs to be processed at the same point in time, provided the constraint (1) below is not violated. Given some permutation π of S, say $\pi(S) = T = (T_1, T_2, \ldots, T_n)$ with corresponding job lengths $(t_1, t_2, \ldots, t_n)$, the jobs are placed sequentially on the real line as follows. The initial job T_1 in the list T begins at time 0 and finishes at time t_1. In general, T_{i+1} begins as soon as T_i is completed, provided the following constraint is always observed:

$$\text{For every real } x \geq 0, \text{ the unit interval } [x, x+1) \text{ can intersect at most } B \text{ jobs.} \quad (1)$$

Constraint (1) reflects the condition that each job needs one of B additional resources for being processed and that a resource has to be renewed after the processing of a job has been finished. The preceding procedure results in a unique placement (or *schedule*) of the jobs on the real line. We define the *finishing time* $\tau(T)$ to be the time at which the last job T_n is finished. A natural goal might be for a given job set S, to find those permutations $T = \pi(S)$ which minimize the finishing time $\tau(T)$.

In [1] it has been shown that the problem is NP-hard in general (the authors polynomially reduce the NP-hard problem PARTITION [3] to a special case of our scheduling problem).

Let us denote by $\tau_w(S)$ the largest possible finishing time for any permutation of S, and let $\tau_o(S)$ denote the optimal (i.e., the shortest possible) finishing time for any permutation of S. The following bounds from [1] are similar in spirit to some of the bounds in the very early *worst-case analysis* literature of scheduling algorithms [4, 5]: For $B = 2$ and any set S, $\tau_w(S) - \frac{4}{3}\tau_o(S) \leq 1$. For $B \geq 3$ and any set S, $\tau_w(S) - \left(2 - \frac{1}{B-1}\right)\tau_o(S) \leq 3$. The factors $\frac{4}{3}$ for $B = 2$ and $2 - \frac{1}{B-1}$ for $B \geq 3$ are best possible.

In fact, the constant 3 can be replaced by $\frac{B}{B-1}$ (and that bound is tight).

Theorem 1 *For $B \geq 3$ and any set S,*

$$\tau_w(S) \leq \left(2 - \frac{1}{B-1}\right)\tau_o(S) + \frac{B}{B-1}.$$

This bound is best possible.

We give the proof in Section 4. Note that the worst-case analysis of the finishing time of any permutation of S exactly corresponds to the classical worst-case analysis for the *List Scheduling* algorithm [4,5] where the jobs can be placed in any order on the machine. We need this result for the analysis of the *LPT* (longest processing time first) algorithm. For the *LPT* algorithm, we rearrange the job set S of jobs into *non-increasing* order to form the permutation $L = (L_1, L_2, \ldots, L_n)$ where l_i denotes

the length of L_i and $l_i \geq l_{i+1}$ for $1 \leq i \leq n-1$. We write $\tau_{LPT}(S)$ for the makespan of an *LPT* schedule. In what follows we assume that all the lengths l_i of the jobs L_i satisfy $l_i \leq 1$ for all i, since if any L_i had $l_i = 1+\varepsilon > 1$, then by decreasing l_i to 1, we decrease both $\tau_{LPT}(S)$ and $\tau_o(S)$ by ε, thereby increasing the upper bound.

In Sections 2 and 3, we show the following theorems:

Theorem 2 *For $B \geq 2$ and any set S,*

$$\tau_{LPT}(S) \leq \left(2 - \frac{2}{B}\right) \tau_o(S) + 1.$$

This bound is best possible.

Theorem 3 *For $B = 2$ and any set S,*

$$\tau_{LPT}(S) - \tau_o(S) \leq \begin{cases} \frac{1}{2}(1 + l_1 - l_{n-1} - l_n) & \text{if } n \text{ is odd,} \\ \frac{1}{2}(l_1 + l_2 - l_{n-1} - l_n) & \text{if } n \text{ is even.} \end{cases}$$

This bound is best possible.

2 Analysis of the *LPT* algorithm for $B \geq 2$

In this section, we are going to proof the general bound for the worst-case behaviour of the makespan of *LPT* schedules in relation to the makespan of optimal schedules for arbitrary $B \geq 2$ as stated in Theorem 2. First, we establish an upper bound for *LPT* schedules.

Lemma 1 *If the jobs are ordered according to the LPT rule and have only processing times $l_i \leq \frac{1}{B-1}, i = 1, \ldots, n$, then we have:*

$$\tau_{LPT}(S) \leq \frac{\sum_{i=1}^{n} l_i + n}{B}$$

Proof By (1) and because of $l_{i+1} \leq l_i$, we have

$$\begin{aligned} &(1)\ \tau_{LPT}(S) \leq 1 + l_B + 1 + l_{2B} + 1 + \ldots + l_n \\ &(2)\ \tau_{LPT}(S) \leq 1 + l_{B-1} + 1 + l_{2B-1} + 1 + \ldots + l_{n-1} \\ &\qquad \ldots \\ &(B)\ \tau_{LPT}(S) \leq 1 + l_1 + 1 + l_{B+1} + 1 + \ldots + l_{n-(B-1)} \end{aligned}$$

Consequently, this implies

$$B \cdot \tau_{LPT}(S) \leq \sum_{i=1}^{n} l_i + n$$

which in turn implies

$$\tau_{LPT}(S) \leq \frac{\sum_{i=1}^{n} l_i + n}{B}. \tag{2}$$

Note: This holds for $n = Bm, m \geq 1$. The cases $n = Bm+1, \ldots, n = Bm+(B-1)$ can be shown in a similar way. □

Now we are ready to prove Theorem 2. We separate the permutation $L = \{L_1, \ldots, L_n\}$ generated by the *LPT* rule into two sets $V = \{V_1, \ldots, V_v\}$ and $U = \{U_1, \ldots, U_u\}$, where $v_i = l_i \geq \frac{1}{B-1}$ denotes the length of V_i and $u_i = l_{i+v} < \frac{1}{B-1}$ denotes the length of U_i, $n = u+v$. In what follows we use the abbreviations $\beta := \sum_{i=1}^{v} v_i$ for the sum of the processing times of the job set V of "large" jobs, and $\gamma := \sum_{i=1}^{u} u_i$ for the sum of the processing times of the job set U of "small" jobs.

Case 1: $u \leq \tau_o(S) \cdot (B-2) + B + \gamma(B-1)$
The jobs in the large job set V can be scheduled without any gap, so we have $\tau_{LPT}(V) = \beta$. Set U contains the small jobs with processing times $< \frac{1}{B-1}$. We know from Lemma 1 that in this case $\tau_{LPT}(U) \leq \frac{\gamma+u}{B}$. So we have for the makespan of the *LPT* ordered job set:

$$\begin{aligned}
\tau_{LPT}(S) &\leq \tau_{LPT}(V) + \tau_{LPT}(U) \\
&\leq \beta + \frac{\gamma+u}{B} \\
&\leq \beta + \gamma + \frac{\gamma+u}{B} - \gamma \\
&\leq \tau_o(S) + \frac{\tau_o(S)(B-2) + B + \gamma(B-1) + \gamma - B\gamma}{B} \\
&\leq \tau_o(S) + \tau_o(S) \cdot \frac{B-2}{B} + \frac{\gamma(B-1) - \gamma(B-1)}{B} + 1 \\
&\leq \tau_o(S) \left(1 + \frac{B-2}{B}\right) + 1 \\
&\leq \tau_o(S) \left(2 - \frac{2}{B}\right) + 1
\end{aligned}$$

Case 2: $u > \tau_o(S) \cdot (B-2) + B + \gamma(B-1)$
Again, we have for the makespan of the *LPT* ordered job set:

$$\begin{aligned}
\tau_{LPT}(S) &\leq \tau_{LPT}(V) + \tau_{LPT}(U) \\
&\leq \beta + \frac{\gamma+u}{B} \\
&\leq \beta + \frac{u}{B-1} \cdot \frac{B-1}{B} + \frac{\gamma}{B} \\
&\leq \beta + \frac{u}{B-1} \cdot \left(1 - \frac{1}{B}\right) + \frac{\gamma}{B} \\
&\leq \beta + \frac{u}{B-1} - \left(\frac{\frac{u}{B-1} - \gamma}{B}\right)
\end{aligned}$$

For the auxiliary job set T with job lengths $t_i = \frac{1}{B-1}$, if $s_i \leq \frac{1}{B-1}$, and $t_i = s_i$, if $s_i > \frac{1}{B-1}$, we have $\tau(T) = \beta + \gamma = \beta + \frac{u}{B-1}$. Together, we come so to

$$\tau_{LPT}(S) \leq \tau(T) - \left(\frac{\frac{u}{B-1} - \gamma}{B}\right)$$

and with (8) to

$$\begin{aligned}
\tau_{LPT}(S) &\leq \tau(T) - \frac{u}{B(B-1)} + \frac{\gamma}{B} \\
&\leq \tau_o(S)\left(2 - \frac{1}{B-1}\right) + \frac{B}{B-1} - \frac{u}{B(B-1)} + \frac{\gamma}{B} \\
&< \tau_o(S)\left(2 - \frac{1}{B-1}\right) + \frac{B}{B-1} - \frac{\tau_o(S)(B-2) + B + \gamma(B-1)}{B(B-1)} + \frac{\gamma}{B} \\
&\leq \tau_o(S)\frac{(2B-3)B-(B-2)}{B(B-1)} + 1 \\
&\leq \tau_o(S)\left(2 - \frac{2}{B}\right) + 1
\end{aligned}$$

□

We give the following example that shows that the bound is tight. Consider the set S consisting of t jobs of length 1 and $(B-2)t+B$ zero-jobs for some integer t. We denote a job of length 1 by $\mathbf{1}$ and a zero-job by $\mathbf{0}$. The optimal permutation $0^{B-1}[\mathbf{1}\ \mathbf{0}^{B-2}]^t\ \mathbf{0}$ has finishing time $\tau_o(S) = t$. On the other hand, the *LPT* order $\mathbf{1}^t\ \mathbf{0}^{t(B-2)+B}$ has a finishing time $\tau_{LPT}(S) = t + ((B-2)t+B)/B = (2-2/B)t+1$. Thus, $\tau_{LPT}(S) = (2-2/B)\tau_o(S) + 1$.

3 Refined analysis of the *LPT* algorithm for $B = 2$

Our goal in this section will be to prove Theorem 3. A first version of this result has been presented by the authors in [2]. As usual (see [1]), we let $s(l_i)$ denote the *starting* time of L_i when L is scheduled, and we let $f(L_i)$ denote the corresponding *finishing* time. We start with the following observation.

Lemma 2 *If the jobs are ordered according to the LPT rule and have only processing times $l_i \leq \frac{1}{B-1}, i = 1,\ldots,n$, then we have:*

$$s(L_i) = f(L_{i-B}) + 1,\ i \geq B+1$$

Proof It follows from (1) that

$$s(L_i) \geq f(L_{i-B}) + 1,\ i = B+1,\ldots,n. \tag{3}$$

In fact, we claim that (3) holds with *equality* for all i if the jobs are given in *LPT* order. This is certainly true for $i = B+1$ (since $l_1,\ldots,l_B \leq \frac{1}{B-1}$). The only reason that we could have $s(L_i) > 1 + f(L_{i-B})$ (for some value $i \geq B+2$) is if

$$f(L_{i-1}) > f(L_{i-B}) + 1. \tag{4}$$

But we know by induction that $f(L_{i-1}) = s(L_{i-1}) + l_{i-1} = f(L_{i-(B+1)}) + 1 + l_{i-1}$. Hence, by (4), $f(L_{i-(B+1)}) + 1 + l_{i-1} > f(L_{i-B}) + 1$, i.e., $f(L_{i-(B+1)}) + l_{i-1} > f(L_{i-B}) = s(L_{i-B}) + l_{i-B}$. However, $s(L_{i-B}) \geq f(L_{i-(B+1)})$. Therefore, $s(L_{i-B}) + l_{i-1} > s(L_{i-B}) + l_{i-B}$, or $l_{i-1} > l_{i-B}$, which is a contradiction. Thus we have $s(L_i) = 1 + f(L_{i-B})$ for all i. □

We know from [1] that the optimal schedule for S satisfies

$$\begin{aligned}\tau_o(S) &\geq \frac{1}{2}\big(s_1+s_n+\sum_{i=1}^{n} s_i+n-2\big)\\ &\geq \frac{1}{2}\big(l_{n-1}+l_n+\sum_{i=1}^{n} l_i+n-2\big)\end{aligned} \tag{5}$$

(note that l_{n-1} and l_n are the lengths of the two smallest jobs in the *LPT* order).

Now we consider two cases:

(i) $n=2m+1$. From Lemma 2 we see that

$$\tau_{LPT}(S) = l_1+1+l_3+1+\ldots+l_{2m-1}+1+l_{2m+1} = \sum_{i=1}^{m+1} l_{2i-1}+m.$$

Because of $l_{i+1}\leq l_i$ for $1\leq i\leq n-1$, we have consequently

$$\tau_{LPT}(S) \leq l_1+1+l_2+1+\ldots+l_{2m-2}+1+l_{2m} = l_1+\sum_{i=1}^{m} l_{2i}+m.$$

Thus,

$$\begin{aligned}\tau_{LPT}(S) &\leq \frac{1}{2}\big(l_1+\sum_{i=1}^{2m+1} l_i+2m\big)\\ &= \frac{1}{2}\big(l_1+\sum_{i=1}^{n} l_i+n-1\big).\end{aligned}$$

Hence, by (5),

$$\tau_{LPT}(S)-\tau_o(S) \leq \frac{1}{2}\big(1+l_1-l_{n-1}-l_n\big).$$

(ii) $n=2m$. From Lemma 2 we see that

$$\tau_{LPT}(S) = l_1+l_2+1+l_4+1\ldots+l_{2m-2}+1+l_{2m} = l_1+\sum_{i=1}^{m} l_{2i}+m-1.$$

Because of $l_{i+1}\leq l_i$ for $1\leq i\leq n-1$, we have consequently

$$\tau_{LPT}(S) \leq l_1+l_2+1+l_3+1+\ldots+l_{2m-3}+1+l_{2m-1} = l_2+\sum_{i=1}^{m} l_{2i-1}+m-1.$$

Thus,

$$\begin{aligned}\tau_{LPT}(S) &\leq \frac{1}{2}\big(l_1+l_2+\sum_{i=1}^{2m} l_i+2m-2\big)\\ &= \frac{1}{2}\big(l_1+l_2+\sum_{i=1}^{n} l_i+n-2\big).\end{aligned}$$

Hence, by (5),

$$\tau_{LPT}(S) - \tau_o(S) \leq \frac{1}{2}\big(l_1 + l_2 - l_{n-1} - l_n\big).$$

□

It follows immediately that the makespan of *LPT* schedules is at most one unit interval longer than the makespan of optimal schedules. We give the following example that shows that the bound is tight. Consider the set S consisting of $t = 2m$ jobs of length 1 and t zero-jobs for some integer t. We denote a job of length 1 by **1** and a zero-job by **0**. The optimal permutation $\mathbf{0[1]^t[0]^{t-1}}$ has finishing time $\tau_o(S) = (3t-2)/2$. On the other hand, the *LPT* order $\mathbf{1^t 0^t}$ has a finishing time $\tau_{LPT}(S) = 3t/2$. Thus, $\tau_{LPT}(S) - \tau_o(S) = 1$. For the case $t = 2m+1$ one has to add one more zero-job to achieve the given worst-case bound.

4 Refined analysis of the *List Scheduling* algorithm for $B \geq 3$

In this section, we analyze the worst-case behaviour of the makespan $\tau_w(S)$ of any permutations of S in relation to the makespan $\tau_o(S)$ of optimal schedules as stated in Theorem 1.

In [1] the authors assume that the permutation $(S_1, S_2, \ldots, S_n)$ is the optimal permutation for the set of jobs $S = \{S_1, S_2, \ldots, S_n\}$, so that this permutation has finishing time $\tau_o(S)$. Then the authors create an auxiliary job set $T = \{T_1, T_2, \ldots, T_n\}$ as follows. The size of $t_i = |T_i|$ is defined by:

$$t_i = \begin{cases} \frac{1}{B-1} & \text{if } s_i \leq \frac{1}{B-1}, \\ s_i & \text{if } s_i > \frac{1}{B-1}. \end{cases} \tag{6}$$

Observe, that for *any* permutation T' of T,

$$\tau(T') = \tau(T) = \sum_{i=1}^{n} t_i \geq \tau_w(S). \tag{7}$$

The authors next examine the schedule for S. They replace each job S_i that has $s_i \leq \frac{1}{B-1}$ by a zero-job S_i' placed on the time axis at the starting time of S_i (the lengths of the jobs with $s_i > \frac{1}{B-1}$ remain unchanged). This certainly causes no violations of the B-constraint (1). Then they assign a *weight* of size $\frac{1}{B-1}$ to each zero-job in this modified job set S'. Thus, the schedule for S' consists of the original "large" jobs S_i of length $s_i > \frac{1}{B-1}$ and a number of "point masses" of weight $\frac{1}{B-1}$, all placed so that condition (1) still holds, i.e., no unit interval intersects more than B of these jobs. The sum of all the weights of the jobs in S' is just equal to the sum of all the lengths of the jobs in T, which by (7) is an upper bound on $\tau_w(S)$. Also (since some of the jobs have been replaced by zero-jobs), all of the jobs in S' still fit in the interval $[0, \tau_w(S))$.

In that way the problem has been reduced to that of finding an upper bound W on the total weight of an arbitrary assignment of (semi-open) intervals of length $s_i > \frac{1}{B-1}$ and point masses of weight $\frac{1}{B-1}$ so that condition (1) is satisfied (and, of course, so that positive length intervals are disjoint, and point masses can only intersect a

positive interval at its starting point). In particular, we can conclude that $\tau_w(S) \leq \tau(T) \leq W$.

Next, in [1] the authors define *blocks* $U_i = [i, i+1)$ for $0 \leq i \leq N-1$, where $N = \lceil \tau_o(S) \rceil$. They define the *content* $c(U_i)$ to be the sum of all the weight (or "mass") in U_i. In other words, all the contributions of the point masses in U_i together with all the portions of those S'_k that happen to lie within U_i are added up.

In what follows, the authors analyze various possibilities for the contents of the blocks U_i. The interval U_i is named *bad* if $c(U_i) > 2-\theta$, otherwise U_i is *good*. By analyzing the possible numbers of zero-jobs in a block U_i the authors conclude that the total gain of the sum of the contents of the blocks U_i over the average value $2-\frac{1}{B-1}$ is at most $B \cdot \frac{1}{B-1}$ (for details we refer the reader to [1, p. 402]). That is, the total weight $\sum_{i=0}^{N-1} c(U_i)$ of the blocks of S' satisfies

$$\begin{aligned}\tau_w(S) \leq \tau(T) &= \sum_{i=1}^{n} t_i = \sum_{i=0}^{N-1} c(U_i) \\ &\leq N\left(2-\frac{1}{B-1}\right) + B \cdot \frac{1}{B-1}\end{aligned}$$

We observe that in order to achieve this upper bound the last block U_{N-1} in the optimal schedule S must be a *bad* block, i.e., the last block U_{N-1} must have content 2. This means that there has to be a job of size 1 in the last block U_{N-1}. And that implies that N actually equals $\tau_o(S)$. So we can follow that

$$\tau_w(S) \leq \tau(T) \leq \tau_o(S)\left(2-\frac{1}{B-1}\right) + \frac{B}{B-1} \tag{8}$$

which shows that the worst-case example for $B \geq 3$ as given in [1] is a tight worst-case example. (In this worst-case example the job set S consists of $(B-1)t+1$ jobs of length 1 and $(B-2)((B-1)t+1)+B$ jobs of length 0 for a positive integer t, t must be large enough.) □

5 Concluding remarks

For the single processor scheduling problem with time restrictions it is never possible for more than B jobs to be worked on during any unit interval. Another view on the problem is to consider B parallel processors that need one additional resource (tool) for processing. After a processor has used this resource that processor needs one timeunit to reset itself (e.g. for cleaning, cooling down, etc.) whereas the resource is immediately available again. The single processor scheduling problem with time restrictions was studied for the first time by Braun, Chung and Graham in [1] where the authors show that this problem is NP-hard when the value B is variable and where they provide a detailed worst-case analysis of *List Scheduling*. We analyze the worst-case behaviour of *LPT* schedules (where we rearrange the set S of jobs into *non-increasing* order) and prove a tight bound for *List Scheduling* for $B \geq 3$. We show that *LPT* ordered jobs can be processed within a factor of $2-2/B$ of the optimum (plus 1) and that this factor is best possible. There might be algorithms other than

LPT that achieve a better worst-case behaviour than *LPT*. An easy improvement for the case $B = 2$ is a heuristic H where we start the schedule with the smallest job and then perform *LPT*. In that way the bound $\tau_H(S) - \tau_o(S) \leq 1/2$ can be achieved. It would be interesting to know if our problem remains NP-hard if $B \geq 2$ is fixed. The development of a mathematical programming model to find the optimal solutions would also be an interesting topic to investigate.

References

1. Braun O, Chung F, Graham RL (2014a) Single processor scheduling with time restrictions. Journal of Scheduling 17: 399–403
2. Braun O, Chung F, Graham RL (2014b) Bounds on single processor scheduling with time restrictions. In: Fliedner T, Kolisch R, Naber A (eds) Proceedings of the 14th International Conference on Project Management and Scheduling, pp 48–51
3. Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness W.H. Freeman, New York
4. Graham RL (1966) Bounds on certain multiprocessing anomalies. Bell System Technical Journal 45: 1563–1581
5. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17: 416–429