A LOOK-AHEAD HEURISTIC FOR SCHEDULING JOBS WITH RELEASE DATES ON A SINGLE MACHINE

WEIZHEN MAO1† and REX K. KINCAID2‡§
1Department of Computer Science and 2Department of Mathematics, The College of William and Mary, Williamsburg, VA 23187-8795, U.S.A.

(Received September 1993; in revised form January 1994)

Scope and Purpose—For optimization problems which seek to minimize the cost of satisfying a sequence of requests, there are two types of heuristics: on-line and off-line. An on-line algorithm performs an immediate action in response to each request in the order it appears in the sequence, while an off-line algorithm receives the entire sequence in advance and takes corresponding actions. In this paper, we consider a third type of heuristic which is on-line in nature but allows limited look-ahead. In particular, we design one such algorithm for a single machine scheduling problem and prove that it outperforms most on-line and off-line algorithms.

Abstract—The paper explores how limited look-ahead improves the performance of on-line heuristics. In particular, we consider the NP-complete single machine scheduling of independent jobs with release dates to minimize the total completion time. We present an on-line with look-ahead algorithm which foresees the next incoming job. We study its worst-case behavior and prove that it outperforms most on-line and off-line heuristics.

1. INTRODUCTION

Given a sequence of requests, an on-line algorithm responds to each request in the order it appears in the sequence without any knowledge of the requests following it in the sequence, while an off-line algorithm receives the entire sequence before responding to any request. Since off-line algorithms know all requests in advance, in other words they know the future, the solutions obtained by these algorithms are usually optimal or near-optimal. However, knowing the future is costly and sometimes impossible. An alternative is on-line algorithms. On-line algorithms are more practical than off-line algorithms but may not provide as high of quality solutions as off-line algorithms.

In the related literature, the on-line approach is also referred to as real-time or reactive and off-line approach as a priori or predictive.

In many situations, it is too optimistic to assume that we know everything about the future as off-line algorithms do, and too pessimistic to assume that we know nothing about the future as on-line algorithms do. A reasonable and practical assumption is that we know something about the future. As an example, let us consider the situation in which a doctor responds to patients' requests for an office visit. The doctor is unable to know all such requests that will occur in the future, however, she has access to an appointment book which records all requests in the near future. Similar examples exist in workplaces where people make special effort (such as utilizing appointments or reservations) to find out future service requests and then schedule services in such a way that reduces customers' waiting times. This brings to our attention a new type of algorithm which is on-line in nature but, in addition, can foresee requests occurring in the near future. We say these algorithms are on-line with look-ahead. A fundamental problem is to examine how limited look-ahead can improve the performance of an on-line algorithm. In this paper, we design an

†Weizhen Mao received her Ph.D. in computer science from Princeton University. She is now Assistant Professor of Computer Science at the College of William and Mary. Her research interests are in design and analysis of algorithms and combinatorial optimization.
‡Rex K. Kincaid is Associate Professor of Mathematics at the College of William and Mary. He received both his Ph.D. in operations research and his M.S. in applied mathematics from Purdue University. His current research interests include network location theory and discrete optimization.
§To whom all correspondence should be addressed.
on-line with look-ahead algorithm for a single machine scheduling problem and show that the algorithm gives better solutions than most on-line and off-line algorithms available.

The scheduling problem we will consider is defined as follows. In the single machine environment, there are \( n \) independent jobs \( J_1, J_2, \ldots, J_n \). Job \( J_j \) has execution time \( p_j \) and is available at release date \( r_j \). Without loss of generality, we assume that in any given instance, there is at least one job which is released at time 0. The execution of a job is non-preemptive, i.e. the job can not be interrupted during the execution. The completion time \( C_j \) of job \( J_j \) is the time when the execution is completed. We wish to construct a schedule of the jobs such that the total completion time \( \sum C_j \) is minimized. Using the notations introduced by Lawler et al. [1], this single machine scheduling problem can be denoted as \( 1|r_j| \sum C_j \).

In the case that \( r_j = 0 \) for all \( j \)'s, the problem is solvable in time \( O(n \log n) \) using the shortest-job-first rule according to Smith [2]. However, when \( r_j \)'s are arbitrary, even if we know all parameters that define the problem instance in advance, the problem is strongly NP-hard, hence NP-complete according to Lenstra et al. [3]. So it is very unlikely that a polynomial-time algorithm for it will ever be found. Research has mainly focused on designing good heuristics for the problem.

We consider three types of heuristics for \( 1|r_j| \sum C_j \) : off-line, on-line, and on-line with look-ahead. In an off-line heuristic, the algorithm, also called a scheduler, knows at the beginning (at time 0) how many jobs will arrive, how long they are, and when they will arrive. An on-line heuristic knows the \( p_j \)'s and \( r_j \)'s of a job only after the job arrives. Finally, an on-line with look-ahead heuristic knows the \( p_j \)'s and \( r_j \)'s of the available jobs as well as those of the next several in-coming jobs.

To evaluate the worst-case behavior of a heuristic \( A \), we use the performance ratio formally introduced by Garey and Johnson [4]. Define the worst-case performance ratio \( R[A] \) to be the asymptotic maximum value of the ratio between the total completion time of the schedule generated by \( A \), denoted by \( \sum C_A(A) \), and the total completion time of the optimal schedule, denoted by \( \sum C_{OPT} \). That is, \( R[A] = \lim \max (\sum C_A(A)/\sum C_{OPT}) \to \infty \). For some heuristics, a worst-case bound may be overly pessimistic, but it provides the only overall performance guarantee available.

A few heuristics for \( 1|r_j| \sum C_j \) have been designed. Dessouky and Deogun [5] proposed a branch-and-bound algorithm. Deogun [6] presented a partitioning scheme. Gazmuri [7] gave a probabilistic analysis of the problem under the assumption that \( p_j \)'s and \( r_j \)'s are independently and identically distributed, and developed a heuristic whose relative error tends to 0 in probability. Rifkin [8] presented some simulation results of the stochastic behavior of a heuristic which foresees the in-coming events.

We notice that most of the previous research focused on designing heuristics and their simulated performance. Only recently has the analysis issue been considered. Chu [9] studied a few off-line heuristics. They are \( ECT \) (Earliest Completion Time), \( EST \) (Earliest Start Time), and two other algorithms, \( PRTF \) (Priority Rule for Total Flow time) and \( APRTF \) (Approximate \( PRTF \)), which both use a local optimal condition to assist in scheduling decisions. Each of these four algorithms can be implemented in \( O(n \log n) \) and is off-line. Chu proved that \( R[ECT] = R[EST] = \frac{1}{2}(n+1) \leq R[PRTF] \leq \frac{1}{2}(n+1) \), and \( R[APRTF] \leq n \). Mao et al. [10] studied two on-line heuristics. They are \( FCFS \) (First Come First Served) and \( SAJF \) (Shortest Available Job First). In both algorithms, a queue of jobs that have arrived but have not yet been served is maintained, and whenever the machine becomes idle, the first job in the queue is chosen to be executed on the machine. The difference between \( FCFS \) and \( SAJF \) is reflected in how the available queue is maintained. In \( FCFS \) the jobs in the queue are sorted by nondecreasing \( r_j \)'s, while in \( SAJF \) the jobs in the queue are sorted by nondecreasing \( p_j \)'s. \( FCFS \) and \( SAJF \) are on-line since in both cases the scheduler has to make decisions without knowing about any jobs arriving in the future. Mao et al. proved that \( R[FCFS] = R[SAJF] = n \).

In this paper, we define an on-line with look-ahead heuristic, which allows the scheduler to check the next job that will arrive when making scheduling decisions. The algorithm is called Look-1-Ahead (L1A), and two queues are maintained by the scheduler. \( Q_1 \), the available queue, contains all the jobs that have arrived but have not yet been served. The jobs in \( Q_1 \) are sorted by nondecreasing \( p_j \)'s. \( Q_2 \), the look-ahead queue, contains the next job that will arrive. When the machine becomes idle, the scheduler checks \( Q_1 \) and \( Q_2 \) and decides whether to wait until the job in \( Q_2 \) arrives, or to schedule the shortest job in \( Q_1 \). The scheduler always chooses the option that yields a shorter
total completion time based on the assumption that no more jobs will arrive except the one already in $Q_2$.

We organize the paper as follows. In Section 2 we examine how $L_1A$ works. In Section 3 we study the worst-case behavior of $L_1A$ and prove that $R[L_1A] = \frac{3}{2}(n+1)$. In Section 4 we extend $L_1A$ to $L_kA$ for $2 \leq k \leq n-1$, which allows the scheduler to check the next $k$ in-coming jobs while making a decision. Finally, we conclude in Section 5 with a summary of our contributions.

2. HOW $L_1A$ WORKS

We assume that at time $t$ the machine becomes idle, and that at $t$ the available queue $Q_1$ contains jobs $J_1, \ldots, J_i$ with $p_1 \leq \cdots \leq p_i$ and the look-ahead queue $Q_2$ contains job $J_{i+1}$ with execution time $p_{i+1}$ and release date $r_{i+1}$. Note that $J_{i+1}$ is the job that will be available in the nearest future, i.e. among those that have not arrived yet, $J_{i+1}$ has the earliest release date. If there is more than one job that will arrive at $r_{i+1}$, we put the shortest in $Q_2$. Let $\Delta = r_{i+1} - t$. For reasons which will be apparent later, we assume that each job, except those available at time 0, will have to pass through $Q_2$ before moving on to $Q_1$. This implies that $\Delta$ may be negative. The $L_1A$ scheduler considers the following situations. If $Q_2 = \emptyset$, i.e. $\Delta$ is undefined, no more jobs will arrive and the machine executes the shortest job in $Q_1$. If $\Delta \leq 0$, $J_{i+1}$ is inserted in $Q_1$ so that $Q_1$ is still sorted, and $Q_2$ is updated to contain the next job, possibly with the same release date as $J_{i+1}$, but with a longer execution time. If $\Delta > 0$ but $l = 0$, i.e. $Q_1$ is empty, the machine has no choice but to wait until time $r_{i+1}$. We now assume that $\Delta > 0$ and $l \geq 1$. According to the definition of $L_1A$, the scheduler assumes that no more jobs will arrive except $J_{i+1}$. Based on this assumption the scheduler chooses either to wait until $J_{i+1}$ arrives or to execute $J_i$, so that the total completion time of the jobs in $Q_1$ and $Q_2$ is minimized. Let us assume $\Delta > 0$ and $l \geq 1$ in Lemmas 1 and 2.

Lemma 1

If $(l+1)\Delta - p_{i} + p_{i+1}$, the machine waits until time $r_{i+1}$.

Proof. Since $\Delta > 0$, we have $p_{i} - p_{i+1} > (l+1)\Delta > 0$. So $p_{i+1} < p_{i}$. Since $l \geq 1$, we have $\Delta < (l+1) \Delta < p_{i} - p_{i+1} < p_{i}$. So $\Delta < p_{i}$. At time $t$, the machine has two options: wait until $r_{i+1}$ (waiting schedule), or execute $J_i$ in $Q_1$ (non-waiting schedule). We need to prove that the total completion time of jobs in $Q_1$ and $Q_2$ in the waiting schedule, $\sum C_j(W)$, is no greater than that in the non-waiting schedule, $\sum C_j(\bar{W})$.

In the waiting schedule, the machine waits until time $r_{i+1}$ and then executes all the available jobs using the shortest-job-first rule, i.e. in the order of $J_{i+1}, J_i, \ldots, J_1$. So $\sum C_j(W) = (t + \Delta + p_{i+1}) + \cdots + (t + \Delta + p_{i+1} + p_{i} + \cdots + p_l) = (l+1)(t + \Delta) + (l+1)p_{i+1} + p_{i} + (l-1)p_1 + \cdots + p_l$.

In the non-waiting schedule, the machine executes $J_i \in Q_1$ and then the remaining jobs using the shortest-job-first rule, i.e. in the order of $J_{i+1}, J_i, \ldots, J_1$. So $\sum C_j(\bar{W}) = (t + p_i) + (t + p_i + p_{i+1}) + \cdots + (t + p_i + p_{i+1} + \cdots + p_l) = (l+1)(t + p_i) + (l+1)p_{i+1} + (l-1)p_1 + \cdots + p_l$.

Since $(l+1)\Delta - p_{i} + p_{i+1}$, we have $\sum C_j(W) \leq \sum C_j(\bar{W})$. \hspace{1cm} $\square$

Lemma 2

If $(l+1)\Delta > p_{i} - p_{i+1}$, the machine executes $J_i$ in $Q_1$.

Proof. We consider two cases: $\Delta \leq p_{i}$, and $\Delta > p_{i}$, and prove that in both cases $\sum C_j(W) \geq \sum C_j(\bar{W})$.

We assume that $p_1, \ldots, p_h \leq p_{i+1} < p_{h+1}, \ldots, p_l$ for some $h$, where $0 \leq h \leq l$.

Case 1. $\Delta \leq p_{i}$.

In the waiting schedule, the machine waits until time $r_{i+1}$ and then executes all the available jobs using the shortest-job-first rule, i.e. in the order of $J_{i+1}, J_i, \ldots, J_1$. So $\sum C_j(W) = (l+1)(t + \Delta) + (l+1)p_{i+1} + \cdots + p_{i+1} + p_{i+1} + p_{i+1} + \cdots + p_l$.

In the non-waiting schedule, the machine executes $J_i \in Q_1$ and then the remaining jobs using the shortest-job-first rule. If $p_i \geq p_{i+1}$, i.e. $h = 1$, $\sum C_j(W) = (l+1)(t + p_i) + \cdots + (l-h+2)p_i + (l-h+1)p_{i+1} + \cdots + p_l$. If $p_{i+1} < p_i$, i.e. $h = 0$, $\sum C_j(\bar{W}) = (l+1)(t + p_i) + (l+1)p_{i+1} + p_{i+1} + \cdots + p_l$.

Since $(l+1)\Delta > 0$ and $(l+1)\Delta > p_{i} - p_{i+1}$, we have $\sum C_j(W) \geq \sum C_j(\bar{W})$.

Case 2. $\Delta > p_{i}$.
In the waiting schedule, the machine waits until time $r_{i+1}$ and then executes all the available jobs using the shortest-job-first rule. If we use $SJF(p_1, \ldots, p_n)$ to denote the total completion time of $J_1, \ldots, J_n$ in the $SJF$ (Shortest Job First) schedule, then
\[
\sum C_j(W) = (l+1)(t+\Delta) + SJF(p_1, \ldots, p_n, p_{i+1}).
\]
In the non-waiting schedule, the machine executes $J_1$ first. It is obvious that $\sum C_j(W) \leq t + p_1 + (t+\Delta) + SJF(p_2, \ldots, p_n, p_{i+1}).$

Since $\Delta > p_1$ and $SJF(p_1, \ldots, p_n, p_{i+1}) > SJF(p_2, \ldots, p_n, p_{i+1})$, we have $\sum C_j(W) \geq \sum C_j(W)$. \hfill \Box

To summarize, we can describe $L1A$ as follows. Obviously, the time complexity of $L1A$ is $O(n \log n)$.

1. Initialize the current clock $t$, the available queue $Q_1$, the look-ahead queue $Q_2$, and the waiting time $\Delta$. We assume that $Q_1$ contains $J_1, \ldots, J_i$ with $p_1 \leq \cdots \leq p_i$, and that $Q_2$ contains $J_{i+1}$.
2. If $Q_2 = \phi$, execute $J_1$, delete $J_1$ from $Q_1$, update $t$, and then go to step 7.
3. If $\Delta \leq 0$, insert $J_{i+1}$ in $Q_1$, so that $Q_1$ is still sorted, update $Q_2$ to contain the next job, update $\Delta$, and then go to step 7.
4. If $Q_2 = \phi$, wait until $r_{i+1}$, insert $J_{i+1}$ in $Q_1$, update $Q_2$ to contain the next job, update $t$ and $\Delta$, and then go to step 7.
5. If $\Delta > 0$, $l \geq 1$, and $(l+1)\Delta < p_1 - p_{i+1}$, wait until $r_{i+1}$, insert $J_{i+1}$ in $Q_1$, so that $Q_1$ is still sorted, update $Q_2$ to contain the next job, update $t$ and $\Delta$, and then go to step 7.
6. If $\Delta > 0$, $l \geq 1$, and $(l+1)\Delta > p_1 - p_{i+1}$, execute $J_1$, delete $J_1$ from $Q_1$, update $t$ and $\Delta$, and then go to step 7.
7. If $Q_1 \neq \phi$ or $Q_2 \neq \phi$, go to step 2.

3. THE TIGHT BOUND OF $L1A$

3.1. A lower bound

To obtain a lower bound for $L1A$, all we need to do is define an instance, compute its ratio, and declare that $R[L1A]$, the largest possible ratio, is no smaller than the ratio for the instance.

Theorem 1

\[ R[L1A] \geq \frac{1}{2}(n+1). \]

Proof. Consider the problem instance in Table 1, where $M$ is an arbitrarily large positive number, and $\varepsilon$ is an arbitrarily small positive number.

<table>
<thead>
<tr>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$\cdots$</th>
<th>$J_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_j$</td>
<td>$M$</td>
<td>$M$</td>
<td>$1$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$r_j$</td>
<td>$0$</td>
<td>$\varepsilon$</td>
<td>$2\varepsilon$</td>
<td>$\cdots$</td>
</tr>
</tbody>
</table>

Fig. 1. (a) The optimal schedule; (b) the $L1A$ schedule.
In \( OPT \), the machine intentionally waits \( 2s \) time units until \( J_3, \ldots, J_n \) arrive, then executes them followed by \( J_1 \) and \( J_2 \). See Fig. 1(a). Therefore, \( \sum C_i(OPT) = [(1+2s) + (2+2s) + \cdots + (n-2) + 2s] + (M + n - 2 + 2s) + (2M + n - 2 + 2s) = 3M + f_1(n, \varepsilon) \), where \( f_1 \) is a function of \( n \) and \( \varepsilon \).

In \( L1A \), the machine executes \( J_1 \), followed by \( J_3, \ldots, J_n \) and then \( J_2 \). See Fig. 1(b). Therefore, \( \sum C_i(L1A) = M + [(M+1) + (M+2) + \cdots + (M+n-2)] + (2M + n - 2) = (n+1)M + f_2(n, \varepsilon) \), where \( f_2 \) is a function of \( n \) and \( \varepsilon \).

By the definition of \( R[L1A] \), we have

\[
R[L1A] \geq \lim_{M \to \infty} \frac{(n+1)M + f_2(n, \varepsilon)}{3M + f_1(n, \varepsilon)} = \frac{1}{3}(n+1).
\]

\[
= \frac{1}{3}(n+1).
\]

### 3.2. An upper bound

We already know from Theorem 1 that \( R[L1A] \geq \frac{1}{3}(n+1) \). In this subsection, we prove that \( R[L1A] \leq \frac{1}{3}(n+1) \). The method we use is an unconventional top-down scheme, in which the correctness and completeness of a proof rely on those of the next proof, and so on. This kind of presentation clearly shows the train of thought and is easier to understand in this particular case than the conventional method. Let \( L1A_n \) and \( OPT_n \) denote the look-ahead schedule and the optimal schedule of an arbitrary problem instance with \( n \) jobs, respectively.

**Theorem 2**

\[ R[L1A] \leq \frac{1}{3}(n+1). \]

**Proof.** To prove this theorem, all we need to do is prove \( \sum C_j(L1A_n) \leq \frac{1}{3}(n+1) \sum C_j(OPT_n) \) for \( n \geq 2 \). We proceed by induction on \( n \). If \( n = 2 \), \( L1A \) is in fact \( OPT \). So \( \sum C_j(L1A_n) = \sum C_j(OPT_n) \leq \frac{1}{3}(n+1) \sum C_j(OPT_n) \).

Now assume that there are \( n \) jobs that need to be executed. Let \( J_n \) be the job that comes last. If there are more than one such job, pick the job with the largest \( p_n \) i.e. the one which last appears as a member of \( Q_2 \) in \( L1A_n \). Let \( p_n \) be its execution time and \( r_n \) be its release date. First consider \( OPT_n \). Let \( s_n \) be the starting time of \( J_n \) and \( m \) be the number of jobs scheduled after \( J_n \) (they are all no shorter than \( J_n \)). Let \( OPT_n-1 \) stand for the optimal schedule with \( J_n \) being removed. We see that when \( J_n \) is removed, each of the \( m \) jobs scheduled after \( J_n \) can then be executed \( p_n \), time units earlier, yielding a feasible schedule (not necessarily optimal) for the remaining \( n-1 \) jobs. Therefore we have

\[
\sum C_j(OPT_n-1) \leq \sum C_j(OPT_n) - mp_n - (p_n + s_n).
\]

Now consider \( L1A_n \). Let \( s'_n \) be the starting time of \( J_n \), \( m' \) be the number of jobs scheduled after \( J_n \) (they are all no shorter than \( J_n \)), and \( \Delta' \) be the idle period, if any, during which the machine is waiting for only \( J_n \). Let \( L1A_n-1 \) stand for the look-ahead schedule with \( J_n \) being removed. We see that when \( J_n \) is removed, each of the \( m' \) jobs scheduled after \( J_n \) is then executed \( \Delta' + p_n \) time units earlier in \( L1A_n-1 \). Therefore we have

\[
\sum C_j(L1A_n) = \sum C_j(L1A_n-1) + m'(\Delta' + p_n) + (p_n + s'_n)
\]

\[
\leq \frac{1}{3}n \sum C_j(OPT_n-1) + (m' + 1)p_n + m'\Delta' + s'_n \quad \text{(by inductive hypothesis)}
\]

\[
\leq \frac{1}{3}n \sum C_j(OPT_n) - \frac{1}{3}n[(m+1)p_n + s_n] + (m' + 1)p_n + m'\Delta' + s'_n \quad \text{[by condition (1)]}
\]

All that is left is prove \(-\frac{1}{3}n[(m+1)p_n + s_n] + (m' + 1)p_n + m'\Delta' + s'_n \leq \frac{1}{3} \sum C_j(OPT_n). \)

The completeness of the above proof relies on Claim 1.

**Claim 1**

\[
-\frac{1}{3}n[(m+1)p_n + s_n] + \frac{3}{3}[(m' + 1)p_n + m'\Delta' + s'_n] \leq \sum C_j(OPT_n) \quad \text{for } n \geq 3.
\]
Proof. Let \( w'_n \) be the waiting time of \( J_n \) in \( L1A_n \), i.e. \( w'_n = s'_n - r_n \). Then \( 3s'_n - ns_n \leq 3(w'_n + r_n) - nr_n \leq 3(w'_n + r_n) - 3r_n = 3w'_n \). The claim is true if we can prove that \( 3(m' + 1)p_n + 3m'\Delta' + 3w'_n \leq \sum C_j(OPT_n) + n(m + 1)p_n \).

At any time during the \( L1A_n \) schedule, the machine is either executing a job or waiting for a job. We define a turning point to be the time at which the machine changes status from executing to waiting, from waiting to executing, from waiting for a job to waiting for another job, or from executing a job to executing another job. It is easy to see that there are at least \( n - 1 \) turning points in the interval \( (0, C_{\max}) \), where \( C_{\max} \) is the maximum completion time of all jobs in the \( L1A_n \) schedule. We also use the time at which a turning point occurs to represent the turning point. Consider the interval \([t_1, t_2]\), where \( t_1 = \max\{t | t \text{ is a turning point and } t < r_n\} \), and \( t_2 = \min\{t | t \text{ is a turning point and } t \geq r_n\} \). We study the cases in which \([t_1, t_2]\) is an idle period, or \([t_1, t_2]\) is a job, and prove that in both cases the claim is true.

The completeness of the above proof relies on the following two claims.

Claim 2

If \([t_1, t_2]\) is an idle period, then \( 3(m' + 1)p_n + 3m'\Delta' + 3w'_n \leq \sum C_j(OPT_n) + n(m + 1)p_n \).

Proof. Since \([t_1, t_2]\) is an idle period, we must have \( t_1 \geq r_{n-1} \), where \( r_{n-1} \) is the second largest release date, and \( t_2 = r_n \). We shall use \( C_j \) to denote the completion time of \( J_j \) in \( OPT_n \). Obviously, \( C_j \geq r_j + p_j \) and \( C_j \geq \sum p_j + p_j \), where \( \sum p_j \) is the total execution time of the jobs scheduled before \( J_j \) in \( OPT_n \). Consider the two cases of \( L1A_n \) shown in Fig. 2.

Case 1. \( J_n \) is the only job arriving at \( r_n \).

Assume that at time \( t_1 \), \( Q_1 \) contains jobs \( J_1, \ldots, J_i \) with processing times satisfying \( p_1 \leq \cdots \leq p_i \), and \( Q_2 \) contains \( J_i \). Since the machine waits until \( r_n \) we have either \( l = 0 \) or \( (l + 1)(r_n - t_i) \leq p_i - p_n \) from Lemma 1. Moreover, \( J_n \) is not longer than any job in \( Q_1 \). It is clear that \( w'_n = s'_n - r_n = 0 \), \( \Delta' = t_2 - t_1 = r_n - t_i \), \( m' = l \) and \( n \geq m' + 1 \).

Subcase 1.1. \( m' = 0 \).

\[
3(m' + 1)p_n + 3m'\Delta' + 3w'_n = 3p_n \\
\leq \sum C_j(OPT_n) + n(m + 1)p_n
\]

Subcase 1.2. \( m' = 1 \).

We have \( 2\Delta' \leq p_1 - p_n \) by Lemma 1.

If \( m = 0 \), then \( C_n \geq p_1 + p_n \) since \( J_n \) is the last job in \( OPT_n \), and

\[
3(m' + 1)p_n + 3m'\Delta' + 3w'_n = 6p_n + 3\Delta' \\
\leq 6p_n + 4\Delta' \\
\leq 6p_n + 2(p_1 - p_n) \\
= p_1 + (p_1 + p_n) + 3p_n \\
\leq C_1 + C_n + n(m + 1)p_n \\
\leq \sum C_j(OPT_n) + n(m + 1)p_n
\]

If \( m \geq 1 \), then

\[
3(m' + 1)p_n + 3m'\Delta' + 3w'_n = 6p_n + 3\Delta' \\
\leq 6p_n + \Delta' + (p_1 - p_n) \\
= 6p_n + (r_n - t_1) + (p_1 - p_n) \\
\leq (p_1 + r_n) + 6p_n \\
\leq p_1 + r_n + n(m + 1)p_n \\
\leq C_1 + C_n + n(m + 1)p_n \\
\leq \sum C_j(OPT_n) + n(m + 1)p_n
\]
A look-ahead heuristic

\[ \begin{array}{c|c|c}
\cdots & \cdots & p_n \\
\hline
r_n & m' \text{ jobs} & \ldots \\
\end{array} \]

(a)

\[ \begin{array}{c|c|c}
\cdots & A & p_n \\
\hline
r_n & m' \text{ jobs} & \ldots \\
\end{array} \]

(b)

Fig. 2. \([t_1, t_2]\) is an idle period. (a) Only one job arrives at \(r_n\). (b) More than one job arrives at \(r_n\).

Subcase 1.3. \(m' \geq 2\).

We observe that since \(p_1 \leq p_1 \leq \cdots \leq p_1 (l = m')\), \(J_a\) must be scheduled before \(J_1, \ldots, J_l\) in \(SJF_n\), which is the shortest-job-first schedule of all \(n\) jobs with the release date constraint removed. So \[ \sum C(OPT_n) \geq \sum C(SJF_n) \geq (m'+1)p_a + m'p_1 + \cdots + p_l \geq (m'+1)p_a + (m'+1)p_1 = (m'+1)p_a + (m'-2)p_1 + 3p_1 \geq (m'+1)p_a + (m'-2)p_1 + 3p_1 = (2m'-1)p_a + 3p_1. \]

Furthermore, since \((l+1)\Delta' \leq p_1 - p_n \leq (l+1)\Delta'\) by Lemma 1, \(3m'\Delta' = 3l\Delta' \leq 3(p_1 - p_n - \Delta')\).

\[ \begin{align*}
3(m'+1)p_a + 3m'\Delta' + 3w_n & \leq 3(m'+1)p_a + 3(p_1 - p_n - \Delta') \\
& \leq 3m'p_a + 3p_1 \\
& = (2m'-1)p_a + 3p_1 + (m'+1)p_n \\
& \leq (2m'-1)p_a + 3p_1 + np_n \\
& \leq \sum C(OPT_n) + n(m+1)p_n.
\end{align*} \]

Case 2. \(J_a\) is not the only job arriving at \(r_n\).

Let \(A\) be the set of jobs scheduled after \(r_n\) but before \(s'_n\). The jobs in \(A\) are all not longer than \(J_n\). Without confusion, we also use \(A\) to denote the sum of the execution time of the jobs in \(A\). It is clear that \(w_n = A, \Delta' = 0, \text{ and } n \geq m'+2\).

Subcase 2.1. \(m' \geq 1\) and \(m \geq 1\).

We observe that \(J_a\) must not be the longest and that in \(SJF_n\) the jobs in \(A\) are scheduled before \(J_a\) and \(J_n\) is scheduled before at least \(\max\{1, m'\}\) longer jobs. So \[ \sum C(OPT_n) \geq \sum C(SJF_n) \geq (\max\{1, m'\} + 2)A + (\max\{1, m'\} + 1)p_n + \max\{1, m'\}p_a \geq 3A + (m'+1)p_a + m'p_n = 3A + (2m'+1)p_n. \]

\[ \begin{align*}
3(m'+1)p_a + 3m'\Delta' + 3w_n &= 3(m'+1)p_a + 3A \\
& = (2m'+1)p_a + 3A + (m'+2)p_n \\
& \leq \sum C(OPT_n) + np_n \\
& \leq \sum C(OPT_n) + n(m+1)p_n.
\end{align*} \]

Subcase 2.2. \(m' = m = 0\).

We observe that in \(SJF_n\) the jobs in \(A\) must be scheduled before \(J_n\). Let \(J_i\) be the longest job in \(A\), then \[ \sum C(OPT_n) \geq \sum C(SJF_n) \geq (A - p_i) + 2p_1 + p_n \geq 3(A - p_i) = 3A. \]

\[ \begin{align*}
3(m'+1)p_a + 3m'\Delta' + 3w_n &= 3p_a + 3A \\
& \leq \sum C(OPT_n) + np_n \\
& \leq \sum C(OPT_n) + n(m+1)p_n.
\end{align*} \]
Claim 3

If \([t_1, t_2]\) is a job, then \(3(m' + 1)p_n + 3m'\Delta' + 3w'_w \leq \sum C(J)OPT_n + n(m + 1)p_n\).

Proof. It is obvious that \(\Delta' = 0\). Assume \([t_1, t_2]\) is job \(J\) with execution time \(p\). Let \(A\) be the sum of execution time of the jobs scheduled after \(J\) but before \(J_n\) in \(L1A_n\) and \(B\) be the sum of execution time of the \(m'\) jobs scheduled after \(J_n\) in \(L1A_n\). See Fig. 3. Obviously, each job is \(A\) is no longer than \(J_n\) and each job in \(B\) is no shorter than \(J_n\). At time \(t_1\), the shortest job in \(Q_1\) must be \(J_n\). Assume that at \(t_1\) the look-ahead queue \(Q_2\) contains job \(J_i\) with execution time \(p_i\) and release date \(r_i\). \(J_i\) is then the shortest job arriving at \(r_i\) and \(r_i \leq r_w\). Note that \(J_i\) may be \(J_n\). Since the machine executes \(J\), by Lemma 2 we have \((l + 1)(r_i - t_1) > p - p_n\), where \(l \geq 1\) is the number of jobs in \(Q_1\). Let \(\delta = t_2 - r_w\), then \(w_w = \delta + A\) and \(\delta + (r_i - t_1) \leq p_n\). Moreover, \(n \geq m' + 2\). Consider two cases.

Case 1. \(J\) is not the longest job.

Assume there exists \(J_n\) with \(p_n \geq p\).

Subcase 1.1. \(m' = 0\).

We first prove \(\sum C(J)OPT_n \geq 3A + 3p\). We observe that \(J\) is always scheduled before \(J_n\) in \(SJF_n\). When there is no job in \(A\), i.e. \(A = 0\), \(\sum C(J)OPT_n \geq \sum C(SJF_n) \geq 2p + p_n \geq 3p = 3A + 3p\). Now we assume that there is at least one job in \(A\). Let \(J_n\) be the longest job in \(A\). In \(SJF_n\) the jobs in \(A\) must be all scheduled before \(J_n\). Now consider the position of \(J\) in \(SJF_n\). If \(J\) is scheduled after \(J_n\) in \(SJF_n\), then \(\sum C(J)OPT_n \geq \sum C(SJF_n) \geq 3A + 2p + p_n \geq 3A + 3p\). If \(J\) is scheduled before \(J_n\) in \(SJF_n\), then \(\sum C(J)OPT_n \geq \sum C(SJF_n) \geq 3A - p_n + p_n \geq 3A + 3p\).

\[3(m' + 1)p_n + 3m'\Delta' + 3w'_w = 3(m' + 1)p_n + 3(\delta + A)\]

\[\leq 3(m' + 1)p_n + 3(p + A)\]

\[= 3p_n + (3A + 3p)\]

\[\leq \sum C(J)OPT_n + n(m + 1)p_n.\]

Subcase 1.2. \(m' > 1\).

We first prove that \(\sum C(J)OPT_n \geq 3A + 3p + (2m' + 1)p_n\). We observe that all jobs in \(B\) are no shorter than \(J_n\). So in \(SJF_n\), the jobs in \(A\) are scheduled before \(J_n\) and the jobs in \(B\) are scheduled after \(J_n\). Now consider the position of \(J\) in \(SJF_n\). If \(J\) is scheduled before \(J_n\) in \(SJF_n\), then \(\sum C(J)OPT_n \geq \sum C(SJF_n) \geq 3A + p + (m' + 1)p_n + p_n = 3A + 3p + (2m' + 1)p_n\). If \(J\) is scheduled after \(J_n\) in \(SJF_n\), then \(\sum C(J)OPT_n \geq \sum C(SJF_n) \geq 3A + (m' + 2)p_n + (m' - 1)p_n + 2p_n + p_n \geq 3A + 3p + (2m' + 1)p_n\).

\[3(m' + 1)p_n + 3m'\Delta' + 3w'_w = 3(m' + 1)p_n + 3(\delta + A)\]

\[\leq 3(m' + 1)p_n + 3(p + A)\]

\[= (2m' + 1)p_n + 3A + 3p + (m' + 2)p_n\]

\[\leq \sum C(J)OPT_n + n(m + 1)p_n.\]

Case 2. \(J\) is the longest job.

Obviously \(l = 1\) and \(2(r_i - t_1) > p - p_n\) by Lemma 2. We then have \(3\delta = 3\delta + 2\delta \leq 2A - 3\delta + 2(r_i - t_1) = 2r_i - 2t_1 = \delta + 3p_n + 3p_n + 2r_i - 2t_1 \leq p + 2p + r_w\). We also have \(\delta \leq 2\delta + 2\delta \leq 2\delta + 2(r_i - t_1) + p_i \leq 2p + p_i\).

Subcase 2.1. \(m' = 0\).

\(J_n\) is the last job in \(OPT_n\).

If \(m' = 0\), then \(p_i \leq p_n\) and \(\sum C(J)OPT \geq 3A + 2p + p_n \geq 3A + 2p + p_n\).

\[3(m' + 1)p_n + 3m'\Delta' + 3w'_w = 3(m' + 1)p_n + 3(\delta + A)\]

\[\leq 3p_n + (3A + 2p + p_i)\]

\[\leq \sum C(J)OPT_n + n(m + 1)p_n.\]

\[\begin{array}{c|c|c|c|c}
\text{t}_1 & \cdots & \text{t}_2 & p & A & p_n & B \\
\hline
& & & & & m' \text{ jobs} \\
\text{r}_i & & & & & \\
\end{array}\]

Fig. 3. \([t_1, t_2]\) is a job.
If \( m' \geq 1 \), then \( B \geq p_s \) and \( B \geq m'p_n \) (\( B \) has \( m' \) jobs, each no shorter than \( p_n \)), and \( \sum C_j(OPT_n) \geq 3(A + B) + 2p + p_n \geq 3A + 2m'p_n + p_1 + 2p + p_n = 3A + (2m' + 1)p_n + 2p + p_r \).

\[
3(m' + 1)p_n + 3m'\Delta' + 3w' \geq 3(m' + 1)p_n + 3(6 + A) \\
\leq 3(m' + 1)p_n + 3A + 2p + p_r \\
\leq (2m' + 1)p_n + 3A + 2p + p_r + (m' + 2)p_n \\
\leq \sum C_j(OPT_n) + n(m + 1)p_n
\]

Subcase 2.2. \( m \geq 1 \).

\( J_s \) is followed by at least one longer job in \( OPT_n \). Assume \( J_s \) is the last job in \( OPT_n \). We also have \( B \geq (m' - 1)p_n + p_i \). We first prove that \( \sum C_j(OPT_n) \geq 3A + (m' - 1)p_n + 2p_1 + p + r_n \). If \( A = 0 \), then \( \sum C_j(OPT) \geq C_n + \sum_{j \neq n} p_j + s_j \geq (r_n + p_n) + (B + p) + p_r \geq 3A + (m' - 1)p_n + 2p_1 + p + r_n \). If \( A > 0 \), then assuming \( J_g \) is the job in \( A \) that is executed last in \( OPT_n \), \( \sum C_j(OPT_n) \geq C_n + \sum_{j \neq n} p_j + s_j + s_n \geq (r_n + p_n) + (A + B + p) + (A - p_g) + (A + p_1) + (A - p) + (A + p) = 3A + (m' - 1)p_n + 2p_1 + p + r_n \).

\[
3(m' + 1)p_n + 3m'\Delta' + 3w' \geq 3(m' + 1)p_n + 3(6 + A) \\
\leq 3(m' + 1)p_n + 3A + 2p + p_r \\
\leq (m' + 1)p_n + 3A + 2p + 2p_1 + r_n + 2(m' + 2)p_n \\
\leq \sum C_j(OPT_n) + n(m + 1)p_n
\]

The correctness of Claim 2 and Claim 3 guarantees the correctness of Claim 1, and hence that of Theorem 2. Combining Theorem 1 and Theorem 2, we have the following result.

\[
R[L1A] = \frac{3}{8}(n + 1).
\]

4. A GENERAL LOOK-AHEAD ALGORITHM \( LKA \)

In \( L1A \), the size of the look-ahead queue \( Q_2 \) is 1, which implies that the scheduler can only check the next one job in addition to the jobs in \( Q_1 \) to make a scheduling decision. Let us extend the definition of \( L1A \) to \( LKA \), which allows the scheduler to check the next \( k \) jobs. We notice that when \( k = 0 \), \( L0A \) is in fact the on-line \( SACF \). Therefore, \( R[L0A] = R[SACF] = n \). When \( k \geq n \), the behavior of \( LKA \) is the same as that of \( L(n - 1)A \). Let us assume \( 2 \leq k \leq n - 1 \). We also notice that \( L(n - 1)A \) becomes a off-line algorithm.

\( LKA \) works as follows: when the machine becomes idle, the scheduler checks \( Q_1 \) and \( Q_2 \) and makes the best possible decision of whether to wait until the first job in \( Q_2 \) arrives or to schedule the first job in \( Q_1 \) on the machine. Let us be more specific. Assume at time \( t \), the machine becomes idle. The scheduler assumes that no more jobs will arrive except those already in \( Q_2 \), and then computes the total completion times of the jobs in \( Q_1 \) and \( Q_2 \) for all possible schedules. There are two kinds of schedules. The schedules in which the machine begins waiting at time \( t \) are called the \( waiting \ schedules \), and the schedules in which the machine begins executing at time \( t \) are called the \( non-waiting \ schedules \). Finally, the scheduler decides to wait if the optimal schedule (with the shortest total completion time) is a waiting schedule and decides to execute if the optimal schedule is a non-waiting schedule.

By a simple generalization of the instance in the proof of Theorem 1, we can show \( R[LKA] \geq \frac{3}{8}(n + 1) + \frac{1}{2}k(k + 1) \). The instance is shown in Table 2.

We conjecture that \( R[LKA] = \frac{n + 1}{8} + \frac{1}{8}k(k + 1) \) for \( 2 \leq k \leq n - 1 \), even though it is not clear whether a similar method of analysis and inductive proof can be applied to the case of \( 2 \leq k \leq n - 1 \). The conjecture, if proved to be correct, suggests that when \( k \) is large enough, such that

\[
k \geq \sqrt{\frac{2n}{c - 1}}
\]
Table 2. An instance for LkA

<table>
<thead>
<tr>
<th>J_1</th>
<th>J_2</th>
<th>...</th>
<th>J_{k+1}</th>
<th>J_{k+2}</th>
<th>...</th>
<th>J_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_j</td>
<td>M</td>
<td>M</td>
<td>...</td>
<td>M</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>r_j</td>
<td>0</td>
<td>ε</td>
<td>...</td>
<td>ε</td>
<td>2ε</td>
<td>...</td>
</tr>
</tbody>
</table>

for some constant \( c > 1 \), the performance ratio \( R[LkA] \) is bounded by \( c \) since

\[
\frac{2}{(k+1)(k+2)} \left( n + \frac{1}{2} k(k+1) \right) \leq \frac{2}{(k+1)(k+2)} \left( \frac{1}{2} (c-1)k^2 + \frac{1}{2} k(k+1) \right) \leq c.
\]

However, \( k \) is no longer a fixed constant, but instead a function of \( n \). The time complexity of \( LkA \) for

\[ k \geq \sqrt{\frac{2n}{c-1}} \]

may not be polynomial since when the machine becomes idle, we may have to check all permutations of the \( k \) jobs in the look-ahead queue to determine the best schedule at the moment.

5. CONCLUSION

In this paper, we studied how adding limited look-ahead to an on-line algorithm improves its performance ratio. We defined a look-ahead heuristic \( L1A \) for the NP-complete scheduling problem \( 1|p_j| \sum C_j \) and proved that its worst-case performance ratio \( R[L1A] = \frac{1}{2} n + 1 \). We noticed that \( L1A \) has advantages against most on-line and off-line heuristics available. First, \( L1A \) can easily be implemented in time \( O(n \log n) \) as the others. Secondly, the limited look-ahead capability of \( L1A \) provides a significant improvement over its on-line counterpart \( SAJF \). \( L1A \) gives solutions about 67% closer to the optimal value than \( SAJF \). Thirdly, \( L1A \) even outperforms some off-line heuristics, such as \( ECT \) and \( EST \), and is at least as good as \( PRTF \), which is considered among the best of the off-line algorithms for the problem. We also generalized the idea of limited look-ahead and defined a class of look-ahead algorithms \( LkA \) for \( 0 \leq k \leq n - 1 \). We established a lower bound \( R[LkA] \geq \frac{k+1}{k+2} (n + \frac{1}{2} k(k+1)) \). Since this lower bound is tight for \( L0A \) and \( L1A \), we conjectured that it is also tight for \( LkA \), where \( 2 \leq k \leq n - 1 \). The most obvious open problem is to prove or disprove this conjecture.

Acknowledgements—Supported in part by NSF grant CCR-9210372 (W.M.) and by a faculty research award from the College of William and Mary (R. K. K.).

REFERENCES