The Impact of Emerging Technologies on Computer Science and Operations Research
S. G. Nash and A. Sofer (ed.)
Kluwer Academic Publishers
1995

On-line Algorithms for a Single Machine Scheduling Problem

Weizhen Mao[1]

Department of Computer Science, College of William and Mary, Williamsburg, VA 23187

Rex K. Kincaid[2]

Department of Mathematics, College of William and Mary, Williamsburg, VA 23187

Adam Rifkin

Department of Computer Science, California Institute of Technology, Pasadena, CA 91125

Abstract

An increasingly significant branch of computer science is the study of on-line algorithms. In this paper, we apply the theory of on-line algorithms to job scheduling. In particular, we study the nonpreemptive single machine scheduling of independent jobs with arbitrary release dates to minimize the total completion time. We design and analyze two on-line algorithms which make scheduling decisions without knowing about jobs that will arrive in future.

Keywords: job scheduling, on-line algorithm, c-competitiveness

1 Introduction

Given a sequence of requests, an *on-line* algorithm is one that responds to each request in the order it appears in the sequence without the knowledge of any request following it in the sequence. For instance, in the bin packing problem, a list $L = (a_1, a_2, \ldots, a_n)$ of reals in $(0, 1]$ needs to be packed into the minimum number of unit-capacity bins. An on-line bin packing algorithm packs a_i, where i starts from 1, without knowing about $a_{i+1}, \ldots, a_n$.

[1]Supported in part by NSF grant CCR-9210372.
[2]Supported in part by a faculty research award from the College of William and Mary.

As pointed out by Karp [?], on-line algorithms are often contrasted with *off-line* algorithms, which receive the entire sequence of requests in advance. In other words, off-line algorithms know the future, while on-line algorithms do not. Therefore, given any objective function, the quality of the solution obtained with an on-line algorithm will be no better (and is typically worse) than the solution obtained with an off-line algorithm.

In many situations, we wish to know the quality of a solution obtained by an on-line algorithm (hereafter referred to as the *performance* of the algorithm). A commonly used method to evaluate the performance of an on-line algorithm is to define a stochastic model by assuming a certain probabilistic distribution for the problem instances. The expected performance of the on-line algorithm is then evaluated within the confines of the stochastic model. However, this approach is inconsistent with the nature of on-line algorithms since, as pointed out by Karp [?], the choice of a stochastic model requires data that may not be readily available about the request sequences that have been observed in the past, as well as faith that the future will resemble the past.

An alternative approach is to compare an on-line algorithm with the optimal off-line algorithm for the same problem in the worst case. Karlin, Manasse, Rudolph, and Sleator [?] defined the term *c-competitive* to refer to an on-line algorithm with performance that is within a factor of c (plus a constant) to optimum on any sequence of requests. More formally, given any instance I, assume the problem asks for the minimization of an objective function $\sigma(I)$. Let A be an on-line algorithm for the problem. Let $\sigma_A(I)$ and $\sigma^*(I)$ be the values of the objective function on I in the solution obtained by A and the solution obtained by the optimal off-line algorithm, respectively. Algorithm A is c-competitive (c is also called the *competitive ratio*) if there exists a constant a such that for any instance I,

$$\sigma_A(I) \quad \leq \quad c\sigma^*(I) + a.$$

For many optimization problems, on-line algorithms have been analyzed using this method. For example, Sleator and Tarjan [?] presented the 2-competitive Move-to-Front algorithm for list processing. Manasse, McGeoch, and Sleator [?] conjectured the existence of k-competitive algorithms for the k-server problem.

In the area of job scheduling, Graham [?] pioneered the on-line algorithm study by designing an on-line algorithm for the multiprocessor scheduling problem. In the problem, a set of n independent jobs are to be scheduled nonpreemptively on m parallel machines. The goal is to construct a schedule with the minimum makespan. Graham defined an on-line algorithm called List-Scheduling (LS), in which the jobs are kept in a list, and when a machine becomes idle the first job in the list is removed from the list and assigned to the machine. Graham proved that for any instance I, $\sigma_{LS}(I) \leq (2 - \frac{1}{m})\sigma^*(I)$.

In this paper, we study a single machine scheduling problem, in which jobs are not all available at the beginning, but instead are given release dates. We define two on-line algorithms and study their c-competitiveness. It should be pointed out that since release dates are involved, we naturally define on-line scheduling algorithms to be ones that have no knowledge about the jobs that have not arrived yet and make scheduling decisions based on all the jobs available at any given time. This is clearly different from many on-line algorithms studied by the computer science community, in which the notion of time is not involved. It should also be pointed out that using competitive ratios to study heuristics is not new in the area of scheduling theory. However, scheduling theory has focused on off-line models, which assume that the entire sequence of jobs arriving at a service facility is known in advance. For example, Potts [**?**] and Hall and Shmoys [**?**] gave analysis of two off-line heuristics for a single machine scheduling problem that seeks to minimize the maximum lateness over all jobs.

We organize the paper as follows. In Section 2, we define the single machine scheduling problem and present two well known, but perhaps not well studied on-line algorithms, First-Come-First-Served ($FCFS$) and Shortest-Available-Job-First ($SAJF$). In Section 3, we prove that $FCFS$ and $SAJF$ are both n-competitive, where n is the number of jobs to be scheduled. In Section 4, we show that there is no on-line algorithm A for the problem such that A is c-competitive for any fixed constant c. In Section 5, we present some computational results. In Section 6, we give the conclusions.

2 A single machine scheduling problem

Given a set J of n independent jobs $J_1, J_2, \ldots, J_n$. Job J_j has *processing time* p_j and becomes available at *release date* r_j. A scheduling problem is defined to execute the jobs in J on a machine such that the total completion time $\sum C_j$, where C_j is the *completion time* of J_j in the schedule, is minimized. This problem can be denoted by $1|r_j|\sum C_j$ according to the $\alpha|\beta|\gamma$ classification used by Lawler, Lenstra, Rinnooy Kan, and Shmoys [**?**], and was proved to be strongly NP-hard by Lenstra, Rinnooy Kan, and Brucker [**?**] even if all parameters that define the problem instance are given in advance. The problem is solvable in polynomial time however if $r_j = 0$ for all j according to Smith [?].

The problem arises from process scheduling in operating system design. In the multi-user system, we have situations in which more than one process is waiting for CPU time, and the operating system must decide which one to run first. One of the goals that the operating system seeks to achieve is to minimize the average response time, i.e., $\frac{1}{n}\sum(s_j - r_j)$, where s_j is the starting time of the execution of a process (job J_j). Since $s_j = C_j - p_j$, the average response time is in fact $\frac{1}{n}(\sum C_j - \sum(p_j + r_j))$. Since $\frac{1}{n}\sum(p_j + r_j)$ can be considered as a constant for a given instance, the problem is therefore converted to $1|r_j|\sum C_j$.

Many algorithms have been designed and analyzed for the problem. Dessouky and Deogun [?] proposed a branch-and-bound algorithm. Deogun [?] presented a partitioning scheme. Chand, Traub, and Uzsoy [?] used a decomposition approach to improve branch-and-bound algorithms. Gazmuri [?] gave a probabilistic analysis. Posner [?] studied a greedy method and proved that it yields an optimal solution under certain conditions. Chu [?] defined a few algorithms which use a local optimal condition to make scheduling decisions.

All of the algorithms in the literature mentioned above are *off-line* in the sense that the algorithms know all the information about jobs even before they arrive in the system. This assumption becomes unrealistic in practice since jobs may arrive at any time and an algorithm is unable to know their p_j and r_j until their arrival. Furthermore, an algorithm may not even know n, the total number of jobs, until all of them arrive. We call this type of algorithm on-line. In other words, an on-line algorithm makes scheduling decisions without any knowledge about the future.

We present two well known on-line algorithms: First-Come-First-Served ($FCFS$) and Shortest-Available-Job-First ($SAJF$). In both algorithms, a queue is maintained to contain all the jobs that have arrived but have not been executed. In $FCFS$, jobs in the queue are listed according to nondecreasing r_j (jobs with the same r_j are ordered by nondecreasing p_j), while in $SAJF$, jobs in the queue are listed according to nondecreasing p_j. When the machine becomes idle after completing the execution of a job, the first job in the queue is assigned to the machine for execution. When a new job arrives, it is inserted into the correct position in the queue. Both $FCFS$ and $SAJF$ are on-line since they make scheduling decisions only based on the information in the available queue. Both $FCFS$ and $SAJF$ are also *conservative*, meaning that the machine is never left idle when there are jobs in the available queue. We notice that even though these two algorithms are commonly used, few analytic results are available.

We define the following notations. Let A be an algorithm for the scheduling problem. Let I be any problem instance. Then we use $\sigma_A(I)$ and $\sigma^*(I)$ to denote the total completion times of I in the schedule constructed by A and the optimal schedule, respectively. We note that the optimal schedule is off-line and yields the minimum total completion time.

3 Analysis of $FCFS$ and $SAJF$

Phipps [?] showed that if the jobs arrive according to a Poisson process then $SAJF$ beats $FCFS$ for a variety of performance measures. Schrage [?] proved that with $SAJF$ the number of jobs in the queue at any point in time is less than or equal to the number of jobs in the queue for any other heuristic simultaneously acting on the same instance. We were unable to find extensions of these results for arbitrary arrival process and with total completion time as the performance measure. Consequently, we provide a proof that $SAJF$ always beats $FCFS$ with respect to total completion time, i.e., for all I, $\sigma_{SAJF}(I) \leq \sigma_{FCFS}(I)$. First, we need the following lemma.

Lemma 1 *Let $\mu_{SAJF}(I)$ and $\mu_{FCFS}(I)$ be the maximum completion times (also called makespans) of any instance I in SAJF and FCFS, respectively. Then $\mu_{SAJF}(I) = \mu_{FCFS}(I)$.*

Proof Consider any conservative scheduling heuristic A. For any instance I, the makespan $\mu_A(I)$ of the schedule obtained with A must be the sum of the total processing time $\sum p_j$ and the total idle time $\sum \phi_i$. That is,

$$\mu_A(I) \quad = \quad \sum p_j + \sum \phi_i.$$

Since in A the machine is left idle only when there are no jobs in the available queue, A obviously minimizes the total idle time, hence the makespan of the schedule. Because $SAJF$ and $FCFS$ are conservative, they both construct the schedules with the minimum makespan. So $\mu_{SAJF}(I) = \mu_{FCFS}(I)$. $\square$

We then have the following theorem.

Theorem 1 *$\sigma_{SAJF}(I) \leq \sigma_{FCFS}(I)$ for any instance I.*

Proof We prove by induction on n, the number of jobs in J. When $n = 1$, we denote the instance by I_1. $SAJF$ and $FCFS$ behave exactly the same.

So $\sigma_{SAJF}(I_1) = \sigma_{FCFS}(I_1)$. Assume for any instance I_{n-1} with $n-1$ jobs, $\sigma_{SAJF}(I_{n-1}) \leq \sigma_{FCFS}(I_{n-1})$.

Now consider any instance I_n with n jobs. Let J^*, with processing time p^* and release date r^*, be the job that is executed last in the $FCFS$ schedule for I_n. J^* must be the longest job among all jobs with the largest release date. Let $\phi \geq 0$ be the idle time just before J^* in the $FCFS$ schedule for I_n. Clearly, $\phi = \max\{0, r^* - \mu_{FCFS}(I_{n-1})\}$. Let I_{n-1} be the same instance as I_n with J^* removed. We have

$$\sigma_{FCFS}(I_n) \quad = \quad \sigma_{FCFS}(I_{n-1}) + (\mu_{FCFS}(I_{n-1}) + \phi + p^*). \tag{1}$$

Now consider the $SAJF$ schedule for I_n. Without loss of generality, we assume that if there are other jobs which have the same processing time and release date as J^* then J^* is executed last among them in the $SAJF$ schedule for I_n. Let $\psi \geq 0$ be the idle time just before J^* in the $SAJF$ schedule for I_n. Assume that there are $k \geq 0$ jobs following J^* in the $SAJF$ schedule for I_n, all no shorter than J^* and arriving earlier than J^*. If $\psi > 0$, J^* must be the last job in $SAJF$ and so $k = 0$, and if $\psi = 0$, J^* may be followed by some longer jobs in $SAJF$. Therefore,

$$k\psi \quad = \quad 0. \tag{2}$$

Furthermore, when $\psi > 0$, all jobs except J^* are scheduled earlier than J^* in the $SAJF$ schedule for I_n. So $\psi = \max\{0, r^* - \mu_{SAJF}(I_{n-1})\}$. Since $\mu_{SAJF}(I_{n-1}) = \mu_{FCFS}(I_{n-1})$ by Lemma 1, then

$$\psi \quad = \quad \phi. \tag{3}$$

Let C_l be the completion time of the job J_l that is scheduled right before J^* in the $SAJF$ schedule for I_n. Note that J_l and J^* may be separated by ψ. Then,

$$C_l + kp^* \quad \leq \quad \mu_{SAJF}(I_{n-1}). \tag{4}$$

Therefore, we have

$$\begin{aligned}
\sigma_{SAJF}(I_n) &= \sigma_{SAJF}(I_{n-1}) + (C_l + \psi + p^*) + k(\psi + p^*) \\
&\leq \sigma_{SAJF}(I_{n-1}) + \mu_{SAJF}(I_{n-1}) + \psi + p^* \quad \text{(By (2) and (4))} \\
&\leq \sigma_{FCFS}(I_{n-1}) + \mu_{SAJF}(I_{n-1}) + \psi + p^* \\
&\qquad \text{(By inductive hypothesis)} \\
&= \sigma_{FCFS}(I_{n-1}) + \mu_{FCFS}(I_{n-1}) + \phi + p^* \\
&\qquad \text{(By Lemma 1 and (3))} \\
&= \sigma_{FCFS}(I_n) \quad \text{(By (1))}. \quad \square
\end{aligned}$$

Let us next consider the c-competitiveness of $FCFS$ and $SAJF$.

Theorem 2 *$\sigma_{FCFS}(I) \le n\sigma^*(I)$ for any instance I of n jobs.*

Proof The $FCFS$ schedule for any instance I with n jobs has a block structure: $B_1, B_2, \ldots, B_l$, where in each block there is no idle time, and between two consecutive blocks there is an idle period. Let $s(B_i)$ be the starting time of block B_i. Obviously, $r_j \ge s(B_i)$ for any $J_j \in B_i$.

Define another instance I' so that it contains $J'_1, \ldots, J'_n$, where the processing time of J'_j is the same as J_j, and the release date of J'_j is $s(B_i)$ if $J_j \in B_i$ in the $FCFS$ schedule for I. The optimal schedule for I' has the same block structure as the $FCFS$ schedule for I. Each block has the same jobs as in the corresponding block in the $FCFS$ schedule for I. Furthermore, based on Smith [?], the jobs in each block in the optimal schedule of I' are executed according to the shortest-job-first rule.

We have $\sigma^*(I) \ge \sigma^*(I')$ because I and I' have the same processing time for each job and the release dates in I' are all at least as early as those in I. Assume B_i has jobs $J_{i1}, \ldots, J_{ik_i}$ with $p_{i1} \le \cdots \le p_{ik_i}$. Let $\sigma^*(I', B_i)$ and $\sigma_{FCFS}(I, B_i)$ be the total completion times of jobs in B_i in the optimal schedule for I' and in the $FCFS$ schedule for I, respectively. Then

$$\sigma^*(I', B_i) \quad = \quad k_i s(B_i) + k_i p_{i1} + (k_i - 1)p_{i2} + \cdots + p_{ik_i}$$

and

$$\begin{aligned} \sigma_{FCFS}(I, B_i) &\le k_i s(B_i) + p_{i1} + 2p_{i2} + \cdots + k_i p_{ik_i} \\ &\le k_i \sigma^*(I', B_i). \end{aligned}$$

Therefore,

$$\begin{aligned} \sigma_{FCFS}(I) &= \sigma_{FCFS}(I, B_1) + \cdots + \sigma_{FCFS}(I, B_l) \\ &\le k_1\sigma^*(I', B_1) + \cdots + k_l\sigma^*(I', B_l) \\ &\le n(\sigma^*(I', B_1) + \cdots + \sigma^*(I', B_l)) \\ &= n\sigma^*(I') \\ &\le n\sigma^*(I). \quad \Box \end{aligned}$$

Theorem 3 *$\sigma_{SAJF}(I) \le n\sigma^*(I)$ for any instance I of n jobs.*

Proof Straightforward using Theorems 1 and 2. □

We can show that the competitive ratio n in Theorems 2 and 3 is tight in the sense that it is achievable by some instance. Consider the following instance I with n jobs. Let $p_1 = M$ and $r_1 = 0$, where M is an arbitrarily large positive number. Let $p_j = 1$ and $r_j = \epsilon$ for $j = 2, \ldots, n$, where ϵ is an arbitrarily small positive number.

In the optimal schedule, the machine waits intentionally for ϵ time units until jobs $J_2, \ldots, J_n$ are released, then executes $J_2, \ldots, J_n$ sequentially, and finally executes the long job J_1. Therefore, $\sigma^*(I) = (1+\epsilon) + (2+\epsilon) + \cdots + (n-1+\epsilon) + (M+n-1+\epsilon) = M + \frac{1}{2}n(n+1) - 1 + n\epsilon$.

In the schedules constructed by $FCFS$ and $SAJF$, the machine executes J_1 and then $J_2, \ldots, J_n$. Therefore, $\sigma_{FCFS}(I) = \sigma_{SAJF}(I) = M + (M+1) + \cdots + (M+n-1) = nM + \frac{1}{2}n(n-1)$.

So $\frac{\sigma_{FCFS}(I)}{\sigma^*(I)} = \frac{\sigma_{SAJF}(I)}{\sigma^*(I)} = \frac{nM+\frac{1}{2}n(n-1)}{M+\frac{1}{2}n(n+1)-1+n\epsilon} \to n$ for $M \to \infty$ and $\epsilon \to 0$.

Now let us compare $FCFS$ and $SAJF$ with other algorithms. As a matter of fact, very few algorithms for the problem have been analyzed using the competitive ratio. Among those that have been analyzed, there are Earliest-Completion-Time (ECT), Earliest-Start-Time(EST), and Priority-Rule-for-Total-Flow-time ($PRTF$). In his recently published paper, Chu [**?**] proved that the tight competitive ratio for ECT and EST is n, and the competitive ratio for $PRTF$ is between $\frac{1}{3}(n+1)$ and $\frac{1}{2}(n+1)$. ECT, EST and $PRTF$ are all off-line. The study of $FCFS$ and $SAJF$ tells us that an algorithm does not have to be off-line to achieve the same competitive ratio of some off-line algorithms. Just knowing the available jobs is adequate.

In many settings, ignorance of the future is a great disadvantage, yet knowing the future is costly and sometimes impossible. How much is it worth to know the future? This becomes a very interesting question.

4 A general lower bound

From the discussion in the last section, we have found that in the worst case both $FCFS$ and $SAJF$ behave badly since their competitive ratios are n, and n can be arbitrarily large depending upon the size of the instance. We are interested to know whether there is any on-line algorithm for $1|r_j|\sum C_j$ whose competitive ratio is bounded by a constant instead of an instance parameter. The answer to this question is in the following theorem.

Theorem 4 *For any on-line algorithm A for $1|r_j|\sum C_j$, there are no constants c and a such that $\sigma_A(I) \leq c\sigma^*(I) + a$ for any instance I.*

Proof We prove by contradiction. Assume that there is an on-line algorithm A such that $\sigma_A(I) \leq c\sigma^*(I) + a$ for some constants c, a, and for any instance I.

Using the adversary argument, we assume that the input instance is provided by an adversary. A good adversary forces the algorithm to make bad scheduling decisions. Suppose that job J_1 is the only job that arrives at time 0, i.e., $r_1 = 0$, and J_1 has processing time $p_1 > a$. Since A is an on-line algorithm, it only sees J_1 in the queue and makes a scheduling decision. There are two possibilities to consider.

Case 1. A decides to execute J_1. The adversary then chooses n, the number of jobs in the instance, to be larger than $c+3$, and assumes that for $j = 2, 3, \ldots, n$, $r_j = \delta$, where $\delta < \frac{p_1}{cn}$, and $p_j = \epsilon$, where $\epsilon < \frac{2p_1}{c(n-1)(n+2)}$. We call this instance I.

In the schedule constructed by A, after J_1 is completed, all of the $n-1$ remaining jobs are available. Therefore, $\sigma_A(I) \geq np_1 + (n-1)\epsilon + (n-2)\epsilon + \cdots + 2\epsilon + \epsilon = np_1 + \frac{1}{2}n(n-1)\epsilon$.

In the optimal schedule, the machine waits until all the short jobs arrive. Therefore, $\sigma^*(I) = n\delta + n\epsilon + (n-1)\epsilon + \cdots + 2\epsilon + p_1 = n\delta + \frac{1}{2}(n-1)(n+2)\epsilon + p_1$.

So we have

$$\begin{aligned} c\sigma^*(I) + a &= cn\delta + \frac{c}{2}(n-1)(n+2)\epsilon + cp_1 + a \\ &< p_1 + p_1 + cp_1 + p_1 \\ &= (c+3)p_1 \\ &< np_1 + \frac{1}{2}n(n-1)\epsilon \\ &= \sigma_A(I). \end{aligned}$$

This is a contradiction to the assumption that $\sigma_A(I) \leq c\sigma^*(I) + a$.

Case 2. A decides to wait for the next job. The adversary then chooses n, the number of jobs, to be 1, i.e., no more jobs will arrive. This forces A to wait forever. Therefore, $\sigma_A(I) = \infty$. In the optimal schedule, $\sigma^*(I) = p_1$. So $\sigma_A(I) > c\sigma^*(I) + a$. This is again a contradiction. □

5 Computational results

The purpose of this section is to examine the performance of the bound given in Theorem 1 and the c-competitiveness ratios given in Theorems 2 and 3. We begin with a set of four, 1000 job simulation experiments that provide insight into the quality of the bound given in Theorem 1 for total completion time, as well as the differences between $FCFS$ and $SAJF$ for several other performance measures. We have conducted many other simulation experiments, but these four suffice to illustrate the key conclusions. We note that $SAJF$ has already been shown by Conway, Maxwell, and Miller [**?**] to be a robust queue discipline under a variety of conditions and we make no attempt to provide an exhaustive computational analysis here. Next, we investigate the conclusions of Theorems 2 and 3 for a 10 job, 30 job, and 50 job problem. Since this scheduling problem is NP-hard we were unable to produce a guaranteed optimum for the 30 and 50 job problems. A tabu search heuristic was used to generate what appear to be high quality feasible solutions. Tabu search has enjoyed many recent successes with a variety of scheduling problems (cf. Glover and Laguna [**?**]) For completeness we give a brief description of the tabu search procedure in Section 5.2.

5.1 Simulation experiments

The simulation experiments were implemented using a SLAM II (see Pritsker [**?**]) discrete event simulation model of a single server queue. Each simulation run begins with the queue empty and the server idle. 1000 jobs are created and processed. Table 1 characterizes each of the simulation models. Column 2 lists the distribution of the time between job arrivals. Column 3 gives the percentage of jobs generated from two job classes (small processing times and large processing times). The distribution from which the job processing times are sampled is given in column 4. Column 5 is the traffic intensity. When $\rho > 0.9$ we consider the system to be congested. Table 2 lists the values of several performance measures for the $FCFS$ and $SAJF$ queue disciplines for each of the four simulation models of Table 1. Column 2 lists the average job completion time and column 3 lists the average waiting time. The average and maximum number of jobs in the queue is given in columns 4 and 5, respectively. Column 6 lists the value of the makespan which are identical for $SAJF$ and $FCFS$ (Lemma 1).

Table 1. Simulation Model Descriptions—1000 Jobs

#	Interarrival	%Sm/Lg	p_j Distribution	ρ
1	expon(4)	95/5	triag(1,2,4)/(10,12,14)	0.69
2	expon(4)	80/20	triag(1,2,4)/(10,12,14)	0.97
3	expon(3)	100/0	triag (1,2,4)	0.78
4	unfrm(2,4)	100/0	triag (1,2,4)	0.79

Table 2. Simulation Results $FCFS/SAJF$—1000 Jobs

#	$\sum C_j/n$	W_q	L_q	max in Q	$\max C_j$
1	2087/2085	4.9/3.6	1.2/0.9	13/9	4072.4
2	2170/2110	84.2/29.8	20/7.0	41/18	4218.3
3	1567/1566	5.3/4.2	1.7/1.4	15/12	3054.7
4	1492/1492	0.26/0.25	.087/.085	2/2	2989.6

We note that the average number in the queue, L_q, is smaller for $SAJF$ than for $FCFS$. Little's formula (see Gross and Harris [?]), $L_q = \lambda W_q$, states that the average number of jobs in the queue equals the product of the arrival rate to the queue, λ, and the average waiting time in the queue, W_q. It can be shown that Little's formula applies to our single server queue with either $SAJF$ or $FCFS$ queue discipline. In Section 3 we showed that $\sigma_{SAJF}(I) \leq \sigma_{FCFS}(I)$ for any instance I. This result implies that $W_q(SAJF) \leq W_q(FCFS)$ and, since λ is a constant, we have $L_q(SAFJ) \leq L_q(FCFS)$.

As we expected, the uniform interarrival distribution smoothed out the arrivals and decreased the size of the queue. When $\rho = 0.79$ (model 4) this resulted in nearly identical performance of $SAJF$ and $FCFS$. A more telling comparison of the performance of $SAJF$ versus $FCFS$ lies in the waiting times. Since the processing times and release dates are included in the computation of the completion time the differences between the waiting times is obscured. A final observation is that the maximum number of jobs in the queue under $SAJF$ was never larger than the maximum number of jobs in the queue for $FCFS$. This follows from Schrage's [?] result for $SAJF$.

5.2 Tabu search

Tabu Search (TS) incorporates conditions for strategically constraining and freeing the search process and memory functions of varying time spans to intensify and diversify the search. The search proceeds from one solution to another via a move function and attempts to avoid entrapment in local optima by constructing a *tabu list* which records critical attributes of moves selected during a window of recent iterations. These attributes identify elements of the solutions that change when progressing from one solution to another, and those from the elected window are declared tabu (forbidden). Current moves are then chosen from a restricted set that excludes the inclusion of tabu attributes (or of a specified number or combination of these attributes), thus insuring that solutions with these attributes (including solutions from the specified window) will not be visited. This restriction may be modified by including aspiration criteria that allow an otherwise tabu move to be made if it leads to a solution that is sufficiently attractive as measured by these criteria—as, for example, a solution better than any previously discovered solutions. Together, these tabu restrictions and aspiration criteria form the short term memory function of tabu search, which can also be augmented by longer term memory functions to achieve goals of intensification and diversification. We briefly outline the overall structure of a TS solution approach, as a modification of an outline suggested by Skorin-Kapov [?].

1. CONSTRUCTION PHASE: Generate a feasible solution.
2. IMPROVEMENT PHASE: Perform the short term memory TS improvement phase *maxit* times, and then execute one of the following longer term memory functions:
    - INTENSIFY the search by choosing a previously unselected member of a recorded set of best solutions to restart the improvement phase (retaining memory to choose a new successor), or by choosing a member of the best solutions uncovered but not yet visited. Repeat step 2.
    - DIVERSIFY the search by driving it to explore new regions. This may be done with either a frequency based memory that favors the inclusion of rarely incorporated attributes, or a recency based memory that may use distance measures that favor high quality solutions with attributes as unlike as possible from previously observed solutions. Repeat step 2.
    - STOP. Display the best solution found.

We implemented a plain vanilla version of TS. Instead of a construction phase we use the $FCFS$ schedule as an initial feasible solution. The improvement phase is a simple greedy local improvement scheme. All pairwise interchanges of the jobs in the schedule are considered. The pair that improves the objective function the most (or degrades it the least if all improving interchanges are tabu) is selected at each iteration of the improvement phase. A tabu interchange is allowed only if it results in the best objective function (total completion time) value yet generated. This is called the *aspiration criterion.* For further information on TS interested readers are referred to Glover [**?**], Glover and Laguna [**?**] and Kincaid [**?**].

5.3 Experiments with $c-$competitiveness

The second set of experiments compares the performance of $SAJF$ and $FCFS$ to the optimal schedule, with respect to total completion time, for a 10 job example and to the best schedules found by a tabu search heuristic for a 30 job and a 50 job problem. The examples are the first 10, 30 and 50 jobs, respectively, generated via model 2 of Table 1. In the example with 10 jobs, $\sigma_{SAJF}(I) = 342.0$, $\sigma_{FCFS}(I) = 342.3$, and $\sigma^*(I) = 269.0$. The optimal schedule was found by enumerating all of the 10! schedules and computing σ for each one.

It was computationally infeasible to calculate the optimal schedule for the number of jobs greater than 10 (We used a 33Mhz 486 class micro-computer). For the 30 and 50 job examples a tabu search heuristic was used to generate good solutions. The tabu search we use, as described in Section 5.2, is a plain vanilla approach. Instead of a construction phase the $FCFS$ schedule was used as the initial starting solution. No intensification or diversification was used. Table 3 lists the parameters selected for our tabu search ($maxit$ and $tabusize$) as well as three performance features (columns 4–6). In column 2, $maxit$ is the maximum number of neighborhood searches allowed. Column 4 lists the iteration when the observed best total completion time was found. The number of times the aspiration criterion was satisfied is given in column 5. Column 6 lists the total number of moves that were declared tabu.

Table 4 summarizes the performance of $FCFS$, $SAJF$ and TS for three job sequences taken from the job data generated in simulation model 2 of Table 1. Column 2 gives the best solution found by TS. When $n = 10$ we have verified that this is also the optimal value. The next two pairs of columns (3-4 and 5-6) list the average completion times for $FCFS$ and $SAJF$ and the ratios

$FCFS/TS$ and $SAJF/TS$, respectively. These ratios show that, at least for these examples, the worst-case analysis of Theorems 2 and 3 may be overly pessimistic for the average case.

Table 3. Tabu Search Characteristics

n	$maxit$	$tabusize$	itr. best	# Asp.	# tabu
10	50	10	19	0	221
30	150	80	68	8	4,350
50	300	80	110	12	10,401

Table 4. Comparison of Average Completion Times

n	TS	$FCFS$	$ratio$	$SAJF$	$ratio$
10	26.9*	34.2	1.27	34.2	1.27
30	75.8	80.8	1.07	80.5	1.06
50	126.3	134.0	1.06	132.0	1.04

6 Conclusions

In this paper, we applied the theory of on-line algorithms to the scheduling problem $1|r_j|\sum C_j$, and studied the c-competitiveness of two on-line algorithms—$FCFS$ and $SAJF$. Furthermore, we proved that there is no on-line algorithm with constant competitive ratio for this problem. We also presented some computational results that illustrated the dominance of $SAJF$ and the overly pessimistic nature of the c-competitiveness worst-case results.

As for the direction of future research, we are currently working on algorithms with look-ahead allowed. We are interested in applying the theory of c-competitiveness to other job scheduling problems. We would also like to generalize the algorithms for $1|r_j|\sum C_j$ to $P|r_j|\sum C_j$ and $R|r_j|\sum C_j$ in the multi-machine environment.

Acknowledgement

We wish to thank two anonymous referees for their helpful comments.

References

[1] S. Chand, R. Traub, and R. Uzsoy, 1993. Single machine scheduling with dynamic arrivals: Decomposition results and a forward algorithm. technical Report 93-10, School of Industrial Engineering, Purdue University, West Lafayette, IN.

[2] C. Chu, 1992. Efficient heuristics to minimize total flow time with release dates, *Oper. Res. Lett. 12*, 321–330.

[3] R. W. Conway, W. L. Maxwell, and L. W. Miller, 1967. *Theory of Scheduling*, Addison-Wesley, Reading, MA.

[4] J. S. Deogun, 1983. On scheduling with ready times to minimize mean flow time, *Comput. J. 26*, 320–328.

[5] M. I. Dessouky and J. S. Deogun, 1981. Sequencing jobs with unequal ready times to minimize mean flow time, *SIAM J. Comput. 10*, 192–202.

[6] P. G. Gazmuri, 1985. Probabilistic analysis of a machine scheduling problem, *Math. Oper. Res. 10*, 328–339.

[7] F. Glover, 1990. Tabu Search: A Tutorial, *Interfaces 20*, 74–94.

[8] F. Glover and M. Laguna, 1993. Tabu Search in *Modern Heuristic Techniques for Combinatorial Problems*, C. R. Reeves, ed., Blackwell Scientific Publishing, 70–150.

[9] R. L. Graham, 1969. Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math. 17*, 416–429.

[10] D. Gross and C. M. Harris, 1974. *Fundamentals of Queueing Theory*, John Wiley and Sons, New York.

[11] L. Hall and D. Shmoys, 1992. Jackson's Rule for One-Machine Scheduling: Making a Good Heuristic Better, *Math. Oper. Res. 17*, 22–35.

[12] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, 1988. Competitive snoopy caching, *Algorithmica 3*, 79–119.

[13] R. M. Karp, 1992. On-line Algorithms Versus Off-line Algorithms: How Much is it Worth to Know the Future?, International Computer Science Institute Technical Report TR-92-044, Berkeley, CA.

[14] R. Kincaid, 1992. Good Solutions to Discrete Noxious Location Problems, *Ann. of Oper. Res. 40*, 265–281.

[15] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, 1990. Sequencing and Scheduling: Algorithms and Complexity, in *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan and P. Zipkin, ed., North-Holland.

[16] J. K. Lenstra, A. H. G. Rinnooy Kan and P. Brucker, 1977. Complexity of Machine Scheduling Problems, *Ann. Discrete Math. 1*, 343–362.

[17] M. S. Manasse, L. A. McGeoch and D. D. Sleator, 1990. Competitive Algorithms for Server Problems, *J. of Algorithms 11*, 208–230.

[18] T. E. Phipps, 1956. Machine Repair as a Priority Waiting-line Problem, *Oper. Res. 4*, 45–61.

[19] M. E. Posner, 1988. The Deadline Constrained Weighted Completion Time Problem: Analysis of a Heuristic, *Oper. Res. 36*, 742–746.

[20] C. N. Potts, 1980. Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Times, *Oper. Res. 28*, 1436–1441.

[21] A. A. B. Pritsker, 1986. *An Introduction to Simulation and SLAM II*, John Wiley and Sons, New York.

[22] L. Schrage, 1969. A Proof of the Optimality of the Shortest Remaining Service Time Discipline, *Oper. Res. 16*, 687–690.

[23] J. Skorin-Kapov, 1990. Tabu Search Applied to the Quadratic Assignment Problem, *ORSA J. on Comput. 2*, 33-a-45.

[24] D. D. Sleator and R. E. Tarjan, 1985. Amortized Efficiency of List Update and Paging Rules, *Comm. ACM 28*, 202–208.

[25] W. E. Smith, 1956. Various Optimizers for Single-Stage Production, *Naval Res. Logist. Quart. 3*, 56–66.