Online Scheduling with Lookahead: Multipass Assembly Lines

Rajeev Motwani [1]
Stanford University

Vijay Saraswat [2]
AT&T Research (formerly Xerox PARC)

Eric Torng[3]
Michigan State University

Abstract

This paper describes our use of competitive analysis and the on-line model of computation in a product development setting; specifically, we use competitive analysis to evaluate on-line scheduling strategies for controlling a new generation of networked reprographic machines (combination printer-copier-fax machines servicing a network) currently being developed by companies such as Xerox Corporation. We construct an abstract machine model, the multipass assembly line, which not only models networked reprographic machines but also models several common manufacturing environments such as a robotic assembly line or a mixed product assembly line. We consider on-line algorithms with finite lookahead because these machines typically have limited knowledge of the future. We first prove some lower bounds on the performance of any on-line algorithm with finite lookahead. We then show that simple greedy algorithms achieve competitive ratios that are close to these general lower bounds. In particular, we show that lookahead improves the competitive ratio of these simple greedy algorithms from approximately 2 (with no lookahead) to being arbitrarily close to 1 (for large lookahead). This implies these simple greedy algorithms are realistic candidates for field use in future reprographic products.

Key Words: on-line, non-omniscient, competitive analysis, lookahead, scheduling, product development, greedy

---

[1]Department of Computer Science, Building 460, Stanford University, Stanford, CA 94305-2140. E-mail: `motwani@cs.stanford.edu`.

This work supported by an IBM Faculty Development Award, NSF Grant CCR-9010517, Mitsubishi Corporation, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation and Xerox Corporation.

[2]AT&T Research. E-mail: `vj@research.att.com`. Much of this work was done when this author worked at Xerox Palo Alto Research Center.

[3]Department of Computer Science, 3115 Engineering Building, Michigan State University, East Lansing, MI 48824-1226. E-mail: `torng@cps.msu.edu`.

Some of this work was done while the author was a Stanford graduate student in computer science, and he was supported by a DOD NDSEG Fellowship, NSF Grant CCR-9010517, Mitsubishi Corporation, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation and Xerox Corporation. This work has since been supported by a Michigan State University All University Research Initiation Grant and NSF Grant CCR-9701679

1 Introduction

In this paper, we describe our use of competitive analysis and the on-line model of computation in a product development setting; specifically, we use competitive analysis to analyze on-line scheduling algorithms to control *networked reprographic machines*, a combination printer-copier-fax machine servicing a network currently being developed by companies such as Xerox Corporation. Because many practical systems must operate with limited knowledge of the future, one might hope that many of the theoretical results obtained using competitive analysis to analyze on-line algorithms have had significant impact on the development of actual systems. Unfortunately, the most practical ramification of most previous competitive analysis results [ABF93, BIRS91, FKL+91, IKP92, KPR92, MPT94, ST85] has been the development of some theoretical justification for the already observed superiority or inferiority of existing algorithms. One exception is the recent work of Fiat et al. [FR97] where they develop a new on-line paging algorithm which potentially may outperform the commonly-used LRU (least recently used) algorithm.

A fundamental reason why competitive analysis has not had a more significant impact on the development of real systems is that while designers are primarily concerned with performance on "typical" inputs, competitive analysis is a worst-case analysis technique. In particular, while a competitive ratio of 2 for an on-line scheduling algorithm may seem adequate from a theoretical perspective, this implies the schedules produced may be 100% longer than necessary (when schedule length is the metric under consideration) which is often unacceptable from a practical perspective.

In order to derive competitive analysis results which have significant impact on system design, we must either show that typical inputs, not just artificially constructed worst-case inputs, cause these algorithms to produce such inefficient schedules, or we must derive extremely strong performance guarantees for all inputs. For a few problems such as paging, there has been some success in deriving results for typical inputs [BIRS91, IKP92]. However, this is a difficult problem in general. For all previously considered on-line problems that we are aware of, no results with competitive ratios approaching 1 have been derived.

In this paper, we show that simple greedy algorithms achieve near-optimal performance with competitive ratios very close to 1. We are able to derive these strong results because we consider on-line algorithms with finite *lookahead* L. We focus on algorithms with finite lookahead L [Alb93, KMW96] because they realistically model systems which receive service requests via a buffer with capacity L that is typically full. For many such systems including networked reprographic machines, a theoretical model which does not incorporate lookahead runs the risk of being inapplicable to the real problem.

More specifically, we derive fairly tight results on the L-GREEDY, a natural greedy algorithm with lookahead L (Table 1). The term $M_{\text{L-GREEDY}}$ represents the competitive ratio of L-GREEDY with respect to the *makespan* performance metric which we define formally in Section 2. The term k is a machine parameter which intuitively represents the number of print cycles required for a sheet of paper to be turned over and returned to the printer when the back side of the sheet requires printing. We also describe this parameter in more detail in the next section. For the purposes of understanding this table, typical values of k range between 3 and 25 while typical values of L range from k to $100k$. Throughout the paper, we also show that randomization is not very helpful for this class of problems.

L/k	1	2	3	4	5	10	20	100
$M_{\text{L-GREEDY}}$	2	1.5	1.33	1.25	1.2	1.1	1.05	1.01
Lower Bound	1.1	1.07	1.06	1.05	1.04	1.02	1.01	1.002

Table 1: Competitive ratio of L-GREEDY

The rest of this paper is organized as follows. In Section 2 we lay the foundation for the remainder of the paper by presenting formal definitions of our problem, the algorithms we study, and the performance metrics we use. In Section 3 we present some preliminary results which are used in the later analysis. In Sections 4 and 5, we derive results for the null task metric and build on these results for the more practically relevant makespan metric for networked reprographic machines. We first show that no on-line algorithm with finite lookahead can be optimal. We then analyze the L-GREEDY family of algorithms and show that lookahead has a significant practical impact on performance. In particular, the competitive ratio of L-GREEDY improves from approximately 2 to approximately 1 as L increases. This stands in sharp contrast to many other on-line problems such as traditional paging [ST85] or on-line load balancing [Gra66] where it can be proven that lookahead cannot be used to improve the performance of on-line algorithms. This makes the L-GREEDY family of algorithms realistic candidates for field use in future reprographic products. In Section 6 we show that our results readily generalize to the case where there is no orientation constraint. In Section 7 we indicate how our results generalize to arbitrary multipass assembly line scheduling. We close by describing open problems in Section 8.

2 Problem Formulation

In this section, we lay the groundwork for the remainder of the paper. We first develop an abstract machine model, the *multipass assembly line machine*, which is a generalization of networked reprographic machines as well as a variety of manufacturing environments such as robotic assembly lines [KMW96, RL86] and multi-item assembly lines [EK81, Gas88, SSL89]. We define our algorithmic model to be on-line scheduling algorithms with finite lookahead because networked reprographic machines typically have limited knowledge of the future (the buffer size). In particular, we define the L-GREEDY family of algorithms. We then define two different performance measures, the null task metric and the makespan metric, and we argue that the makespan metric is more relevant in practice.

2.1 Multipass Assembly Line Scheduling Model

In this subsection we define a generic multipass assembly line. In a multipass manufacturing environment, each job is an unfinished product requiring between 1 to m equivalent processing tasks or passes (we use the two terms interchangeably). Between consecutive passes, the product needs to be physically reoriented so that its pose can facilitate the next pass. A simple assembly line architecture for maximizing throughput is an alternating pipeline of m processing and m orienting devices.

Processors and orienters can be expensive, so it is desirable to minimize their number while still achieving high efficiency and reasonable throughput. We consider the case where the manufacturer uses only one processor and one orienter. Normally, it is easy to simulate m processors using only one processor. However, these simulations typically ignore the problem of data storage and data movement. In our problem, data storage and movement are critical components which cannot be ignored. This means that the usual simulation techniques cannot be used in this case.

The problem we consider caters to *flexible* or *mixed product* assembly lines which process a variety of jobs rather than repeatedly handling the same job; the latter is essentially an offline problem. In Fig. 1, we present a schematic view of a multipass assembly line.

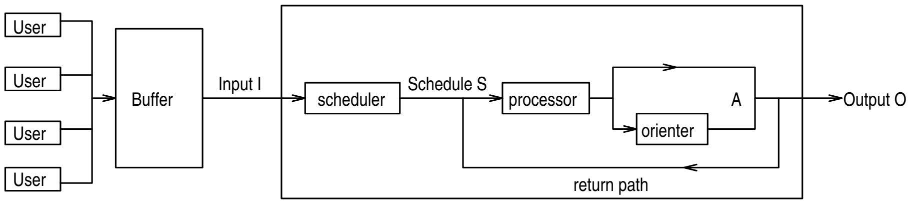

Figure 1: A multipass assembly line.

The problem may be formalized as follows. An input instance $\mathcal{I}$ is a sequence of n jobs. There are m different job types, $J_1, \ldots, J_m$, where job J_i requires i passes through the processor. A schedule $\mathcal{S}$ is a sequence of tasks, several of which may correspond to one job in $\mathcal{I}$. There are $\binom{m+1}{2}+1$ different task types, $\emptyset, J_1(1), J_2(1), J_2(2), \ldots, J_m(m-1), J_m(m)$, where $\emptyset$ is a null task denoting an idle cycle and $J_j(i)$ denotes the i^{th} pass for a type J_j job (for $i \leq j$). We denote the i^{th} task of $\mathcal{S}$ as $\mathcal{S}_i$. We call tasks $J_i(i)$ for $1 \leq i \leq m$ *final tasks*. Once a final task is executed, the corresponding job is completed. The following constraints on schedules indicate how each task of a feasible schedule $\mathcal{S}$ can be uniquely mapped to its corresponding job in $\mathcal{I}$. The goal is to devise a feasible schedule $\mathcal{S}$ of minimum makespan (i.e., length). The non-triviality in the problem arises due to the following constraints on $\mathcal{S}$.

FCFS Constraint. Jobs are to be output in the order of arrival. That is, the subsequence of $\mathcal{S}$ consisting only of final tasks should be isomorphic to $\mathcal{I}$.

Return Constraint. A job takes exactly k processor cycles to return to the processor. Therefore, the i^{th} pass of job J_j through the processor must be scheduled exactly k processor cycles before its $i+1^{st}$ pass through the processor. That is:

$$\mathcal{S}_l = J_j(1) \Leftrightarrow \mathcal{S}_{l+k} = J_j(2) \Leftrightarrow \cdots \Leftrightarrow \mathcal{S}_{l+(j-1)k} = J_j(j).$$

Orientation Constraint. As stated earlier in the text, a multipass job typically requires a reorientation after each pass so that its pose properly facilitates the next pass. In addition, because jobs are assumed to arrive in some canonical orientation, and because they must typically leave the system in some (usually the same) canonical orientation, multipass jobs require one final reorientation after the final pass whereas single pass jobs should not be reoriented after their single pass. Note, this exit orientation requirement is essential for finishing and packaging the product while the entrance orientation requirement can be relaxed at the cost of requiring the presence of an orienter prior to the processor stage. Reorientation is assumed to take one processor cycle which means collisions can occur at the output of the orienter (i.e., point A in Fig. 1) if a single pass job that requires no reorientation is processed immediately after any pass of a multipass job. Avoiding such collisions requires that for $j > 1$ and $i \leq j$,

$$S_l = J_1(1) \Rightarrow S_{l-1} \neq J_j(i).$$

The orientation constraint does not arise in all applications. For example, all jobs including J_1 jobs may require reorientation. Fortunately, our results are robust enough to apply essentially unchanged when the orientation constraint is removed (see Section 6).

2.2 Networked Reprographic Machine Scheduling

We developed the multipass assembly line model by abstracting the problem of scheduling networked reprographic machines. Documents (the jobs) arrive over the network and request the marker unit (the processor) to print an arbitrary sequence of single-sided pages (J_1 jobs) and double-sided pages (J_2 jobs). The marker unit can only print one side of a page at a time, so the double-sided pages need to be processed twice. After the front side of a double-sided page has been printed, the sheet of paper needs to be inverted (reoriented) and then returned to the marker to print the back side. Inverting a partially printed page and then returning it to the marker unit takes exactly k print cycles (the return constraint). It is required that the pages of a single document be output in order, and that the system operates in a first-come-first-served manner (the FCFS constraint). To avoid paper jams, neither pass of a double-sided page can be immediately followed by a single-sided page (the orientation constraint).

Thus, reprographic scheduling is the simplest non-trivial version of the multipass assembly line scheduling problem, i.e., the case $m = 2$. As a matter of fact, reprographic machines have other features such as color printing which may require the full generality of the multipass scheduling formulation. This is because some color printers require several passes on each side of the page to lay down the various tones and colors.

We develop our results by concentrating on the special case of black-and-white reprographic scheduling, i.e., the case $m = 2$. To make the notation more intuitive, we define $O = J_1$, $D = J_2$, $S = J_1(1)$, $F = J_2(1)$, and $B = J_2(2)$. We still use $\emptyset$ to represent a null task. Thus, O represents a single-sided or one-sided page (job) while D represents a double-sided page (job), and S represents the task of printing the single-sided page while while F and B represent the tasks of printing the front and back sides of a double-sided page, respectively. The input will be strings over the alphabet $\{O, D\}$ and the schedule will be strings over the alphabet $\{S, F, B, \emptyset\}$.

It is important to note that because we consider *networked* reprographic machines, as opposed to *stand-alone* copiers, printers or fax machines, we must handle a much richer set of input instances. In the stand-alone case, the machine processes documents one at a time, and documents typically consist of only one page type; therefore, input instances are typically restricted to strings in $O^* = \{O^n \mid n \geq 0\}$ or $D^* = \{D^n \mid n \geq 0\}$ (though there are exceptions). In the networked environment, the machine is shared by a network of users generating multiple documents simultaneously; therefore, input instances range over the space of all possible input instances, namely $(O + D)^*$. Handling the general case is far more interesting and challenging than handling the restricted case.

2.3 Algorithmic Model

We first define what information any algorithm may have about the input instance. The algorithm must lack some knowledge of the future because documents arrive dynamically. However, some lookahead is usually available because most documents have many pages and multiple documents may be ready for printing simultaneously. This lookahead is bounded by two factors. First, the scheduler only has knowledge of pages in the buffer (a disk storing compressed page images), so the lookahead is bounded by the buffer size. Second, users demand fast response time. In particular, users are not willing to wait minutes for the scheduler to compute an optimal print sequence which saves seconds of execution time, even if this extends the life of the machine significantly. The first restriction is the stringent one as any useful scheduler can schedule jobs far more quickly than they can be printed. We will assume that the buffer is full because otherwise the scheduler has access to the entire input sequence and can compute an optimal schedule. Thus, we only consider on-line algorithms with finite lookahead L (the buffer size). We assume that an algorithm A initially sees the first L jobs of the input sequence I, and every time A schedules a job, the next job in I arrives. Note that we consider a multipass job to be scheduled once its first pass (task) is scheduled since we know exactly when the remaining passes (tasks) associated with this job will be executed as well.

2.4 Greedy Algorithms

In order for an algorithm to be considered for implementation, it must compute schedules quickly in order to guarantee fast response time. This leads us to first consider a specific algorithm which we call 1-GREEDY utilized in some current reprographic products. This algorithm only sees 1 job of I at a time; specifically, the next job that needs to be scheduled. It schedules this job optimally, i.e., it schedules the job as early as possible. In trivial situations where the input sequence consists of only one page type, something that is fairly likely to occur in the stand-alone machine setting, this algorithm performs optimally. However, for input sequences which include a mixture of single-sided and double-sided pages, something that is likely to occur in the networked environment and does occur even in the stand-alone environment, the competitive ratio of this algorithm asymptotically approaches 2 which is unacceptable for this application.

We generalize 1-GREEDY which optimally schedules 1 job at a time to L-GREEDY which optimally schedules L jobs at a time. When scheduling a block of L jobs, L-GREEDY computes an optimal extension of the partial schedule computed for all earlier jobs to include these L jobs; it is known how to compute optimal *offline* extensions efficiently using dynamic programming techniques. That is, L-GREEDY computes the optimal schedule assuming the input instance consists only of the previously scheduled jobs with their previously computed schedule and the L jobs that are visible in the lookahead. We will show that the performance of L-GREEDY for reasonable values of L approaches 1.

2.5 Competitive Analysis

We use competitive analysis to evaluate the performance of the various on-line algorithms under consideration. Let A be an on-line algorithm with finite lookahead L. For a job sequence $\mathcal{I}$, let $A(\mathcal{I})$ denote A's schedule for $\mathcal{I}$, and let $OPT(\mathcal{I})$ denote the optimal schedule for $\mathcal{I}$. Let $|\mathcal{S}|$ denote the length or *makespan* of a schedule $\mathcal{S}$ for $\mathcal{I}$, and let $M_A(\mathcal{I}) = \frac{|A(\mathcal{I})|}{|OPT(\mathcal{I})|}$. The competitive ratio of A, M_A, is then defined as

$$M_A = \sup_{\mathcal{I} \neq \emptyset} M_A(\mathcal{I}),$$

and the expected competitive ratio $E[M_R]$ of a randomized algorithm R is defined as

$$E[M_R] = \sup_{\mathcal{I} \neq \emptyset} E\left[M_R(\mathcal{I})\right].$$

We focus on the makespan of a schedule because it is closely related to the utilization efficiency of the machine (the percentage of time the marker unit is actually printing a page when the machine is in use), an important practical measure. More specifically, the lifespan of the machine is directly related to the amount of time it is in use, even when performing null tasks. Thus, achieving a high utilization efficiency and minimizing the number of null tasks performed is critical to building machines which will last.

We will also consider the null task performance metric which measures the number of null tasks performed. This performance metric is of interest since null tasks wear down the machine while producing no benefit to the users. Let $U(\mathcal{S}(\mathcal{I}))$ denote the number of null tasks in $\mathcal{S}(\mathcal{I})$ for any input $\mathcal{I}$. The null task competitive ratios N_A and $E[N_R]$ are defined using the number of null tasks $U(\mathcal{S}(\mathcal{I}))$ as the performance measure instead of the makespan $|\mathcal{S}(\mathcal{I})|$.

Given these two performance measures, the question arises, which measure has more practical value? We will show that although both measures are equivalent when considering optimal schedules, they are vastly different when considering approximate schedules. In particular, the makespan performance measure results reflect the influence of lookahead whereas the null task performance measure results are insensitive to lookahead, and the makespan measure results are stable with respect to slight variations in the problem formulation whereas the null task measure results are not. Thus we use our makespan measure results to guide our search for good algorithms in practice.

3 General Observations

In this section, we prove some general results about the networked reprographic machine scheduling problem (i.e., $m = 2$). We first show that this is a trivial problem when $k = 1$ or $k = 2$. Thus, we only consider the case $k \geq 3$ until Section 6 where we remove the orientation constraint (and the case $k = 2$ becomes non-trivial).

Lemma 3.1 *For $k = 1$ or $k = 2$, the algorithm* 1-GREEDY *is optimal.*

Proof: When $k \leq 2$, no S task may lie in between the F and B tasks of a D job because of the orientation constraint. Therefore, each block of O and D jobs must be scheduled independently. 1-GREEDY does this in an optimal fashion. ∎

We now describe some patterns in the input sequence which can be used to decompose the problem into independent pieces. This will help us to derive lower bounds on the best possible competitive ratio of any on-line algorithm with finite lookahead. The first pattern is the string O^{k-2} which allows us to partition the input into two independent sub-problems.

Lemma 3.2 *If $\mathcal{I} = wxy$ where $x = O^{k-2}$, then $OPT(\mathcal{I}) = OPT(w) \circ OPT(xy)$ or $OPT(\mathcal{I}) = OPT(w) \circ \emptyset \circ OPT(xy)$. The null task is needed if and only if w ends with a D job and $OPT(xy)$ begins with an S task.*

Proof: The only way for this not to be true is if some F task of a D job in xy can be scheduled before some S task of an O job in w or some B task of a D job in w. Otherwise, there is no interaction between the jobs in w and xy, so they can be independently scheduled.

Consider the first D job in xy. This job cannot occur before the first job of y as one of our assumptions is that $x = O^{k-2}$. If the F task for this D job were to precede some task of a job in w, it must precede all S tasks of the $k-2$ O jobs in x. By the orientation constraint, some null task must lie between this F task and the first S task of the first O job in x. This means that $k-1$ tasks must lie in between the F and B tasks of the first D job in xy. No other tasks can fit in between these F and B tasks which means that no tasks from w can lie in between these F and B tasks of this first D job in xy. Therefore, w and xy can be scheduled independently. The only condition that needs to be checked is whether or not w ends with a D job and xy begins with an O job. If so, then a null task must be inserted between the two partial schedules $OPT(w)$ and $OPT(xy)$. ∎

Lemma 3.3 *Let job sequence $\mathcal{J} = D^k$. Then for any job sequence $\mathcal{I}$, $U(OPT(\mathcal{I})) = U(OPT(\mathcal{I}\mathcal{J}))$.*

Proof: Showing $U(OPT(\mathcal{I}\mathcal{J})) \leq U(OPT(\mathcal{I}))$ is easy. Take $OPT(\mathcal{I})$ and append to it F^kB^k. The number of null tasks in this legal schedule is still $U(OPT(\mathcal{I}))$ and is thus an upper bound on $U(OPT(\mathcal{I}\mathcal{J}))$.

Showing $U(OPT(\mathcal{I})) \leq U(OPT(\mathcal{I}\mathcal{J}))$ requires a bit more effort. Let $|OPT(\mathcal{I}\mathcal{J})| = n$. While there may be many optimal schedules for input $\mathcal{I}\mathcal{J}$, we shall restrict our attention to optimal *tail loaded* schedules where $\mathcal{I}\mathcal{J}$ ends with the longest string of B tasks possible. Let q be the length of this string of B tasks.

We first observe that $1 \leq q \leq k$. The lower bound of 1 follows from the FCFS constraint and the fact that $\mathcal{I}\mathcal{J}$ ends with a D job. The upper bound of k follows from the return constraint where the F task of the last B task will occur in position $n-k$.

If $OPT(\mathcal{I}\mathcal{J})$ ends with k B tasks ($q = k$), then it must end with F^kB^k. Simply remove these $2k$ tasks and the lemma is proven. We now prove that all optimal tail loaded schedules $OPT(\mathcal{I}\mathcal{J})$ must end with k B tasks.

Assume $OPT(\mathcal{I}\mathcal{J})$ is an optimal tail loaded schedule with $q < k$. Consider $OPT(\mathcal{I}\mathcal{J})_{n-q}$, the last task in $OPT(\mathcal{I}\mathcal{J})$ which is not a B task. Because of the return constraint, we know that $OPT(\mathcal{IJ})_{n-q} \neq F$. Because of the FCFS constraint, we know that $OPT(\mathcal{IJ})_{n-q} \neq S$. Thus, we conclude that $OPT(\mathcal{IJ})_{n-q}$ must be a null task.

Now consider $OPT(\mathcal{IJ})_{n-q-k}$ to $OPT(\mathcal{IJ})_{n-q-1}$. Because $OPT(\mathcal{IJ})_j = B$ for $n-q+1 \leq j \leq n$, the return constraint implies that $OPT(\mathcal{IJ})_i = F$ for $n-q+1-k \leq i \leq n-k$. Because the final k jobs of $\mathcal{IJ}$ are D jobs, the FCFS constraint and the return constraint imply that $OPT(\mathcal{IJ})_j = \emptyset$ or $OPT(\mathcal{IJ})_j = B$ for $n-k+1 \leq j \leq n-q-1$. For similar reasons, $OPT(\mathcal{IJ})_{n-q-k} \in \{\emptyset, B\}$.

Consider all the D jobs which have B tasks scheduled in positions $n-q-k$ and $n-k+1$ to $q-1$. There must be at least one, since otherwise we could reduce the number of null tasks in $OPT(\mathcal{IJ})$ by eliminating the null tasks that lie between the last B task before position $n-q-k$ and the F task that lies in position $n-q+1-k$. Shift the F and B tasks of each of these D jobs back in the schedule as follows. The F and B tasks of the last D job move into positions $n-q-k$ and $n-q$. The F and B tasks of the second to last D job move into the positions just vacated. This pattern continues for the remaining D jobs.

Note this shifted schedule still obeys the FCFS constraint, the orientation constraint, and the return constraint. Note also no position is assigned more than one task. Thus this shifted schedule is legal, the length of this shifted schedule is identical to $|OPT(\mathcal{IJ})|$, and this shifted schedule has a longer tail. This contradicts the fact that $OPT(\mathcal{IJ})$ is an optimal tail loaded schedule with $q < k$. Thus it must be the case that all optimal tail loaded schedules have $q = k$ and we are done. ∎

Corollary 3.1 *Let $\mathcal{I} = xOD^{jk+m}$ where $0 \leq m \leq k-1$. Then $OPT(\mathcal{I}) = OPT(xOD^m) \circ OPT(D^{jk})$.*

Finally, we prove a simple lower bound on $U(OPT(\mathcal{I}))$ for any input sequence $\mathcal{I}$.

Lemma 3.4 *For any job sequence $\mathcal{I}$, $U(OPT(\mathcal{I}))$ is no less than the number of alternations in $\mathcal{I}$ between sequences of D jobs and sequences of O jobs.*

Proof: Consider an O job which immediately follows a D job in any input instance $\mathcal{I}$. In any feasible schedule $\mathcal{S}$ for $\mathcal{I}$, the S task for this O job cannot be preceded by an S task (FCFS constraint) or an F or B task (orientation constraint); hence, it must be preceded in $\mathcal{S}$ by $\emptyset$. ∎

4 The Null Task Measure

In this section we consider only the null task measure of performance. We will prove strong lower bounds on the competitive ratios of deterministic or randomized algorithms with finite lookahead. We also analyze the L-GREEDY algorithm and, surprisingly, discover that increasing the lookahead L does not substantially improve the performance of the algorithm. This arises from measuring only the number of null passes scheduled without regard to the total schedule length, and it provides an indication that the null task measure of performance results are not meaningful to practitioners.

4.1 Lower Bounds on Competitive Ratio

We first prove lower bounds on the performance of any deterministic algorithm with finite lookahead L. As a general rule, we assume that the lookahead L exceeds $k-2$. Clearly, all derived lower bounds extend to algorithms with less lookahead since an algorithm with lookahead L could choose to not fully utilize its lookahead.

Theorem 4.1 *For $k \geq 3$, any deterministic algorithm A with lookahead L has $N_A \geq \lceil \frac{k+1}{2} \rceil$.*

Proof: The adversary initializes the input job sequence $\mathcal{I}$ to be $O^{k-2}D^{L-k+2}$. Each time A schedules the S task of an O job, the adversary adds another D job to the end of $\mathcal{I}$. Eventually, A must schedule the F task of the first D job. Let q be the number of O jobs scheduled before this first D job is scheduled. Let $j = \lceil \frac{L-(k-2)+(q-1)}{k} \rceil$. The adversary now chooses between revealing the entire job sequence $\mathcal{I}$ to be $O^{k-2}D^{jk+q+2}$ (for small q) or $O^{k-2}D^{jk+1}$ (for large q). Because of Corollary 3.1, we ignore the final jk D jobs in $\mathcal{I}$ for the remainder of this proof.

In both cases, $U(OPT(\mathcal{I})) = 1$. In the first case, the optimal schedule $OPT(\mathcal{I}) = S^{q+1}F^{q+2}\emptyset S^{k-q-3}B^{q+2}$. In the second case, the optimal schedule $OPT(\mathcal{I}) = F\emptyset S^{k-2}B$.

In the first case (chosen by the adversary when q is small), A has scheduled the first F task too early. Now it is impossible to schedule all $q+2$ F tasks of the $q+2$ D jobs before the $k-q-2$ S tasks of the remaining $k-q-2$ O jobs because this would put at least $q+1$ F tasks, $k-q-2$ S tasks, and 1 null task (a total of k tasks) in between the F task and the B task of the first D job violating the return constraint. Therefore, the $q+2^{nd}$ F task must lie to the right of the first B task in $A(\mathcal{I})$. Given this, the best A can do is set

$$A(\mathcal{I}) = S^qF\emptyset S^{k-q-2}F^qBF\emptyset^{k-q-2}B^q\emptyset B$$

since this puts the middle q F tasks in between the first F and the first B tasks while putting the middle q B tasks in between the $q+2^{nd}$ F and $q+2^{nd}$ B tasks. The net result is $U(A(\mathcal{I})) = k-q$.

In the second case (chosen by the adversary when q is large), A has waited too long to schedule the F task of the first D job. It can no longer use the q S tasks it has already scheduled to occupy the space between the F task and the B task of the one D job. Given this, the best A can do is set $A(\mathcal{I}) = S^qF\emptyset^{q+1}S^{k-q-2}B$. The net result is that $U(A(\mathcal{I})) = q+1$.

Since $U(OPT(\mathcal{I})) = 1$ in either case, the adversary will choose the first option when $k-q \geq q+1$, and the adversary will choose the second option when $q+1 \geq k-q$. Thus A's optimal strategy is to set $q = \lceil \frac{k-1}{2} \rceil$ and the result follows. ∎

Note that the lower bound sequence only uses one alternation of O and D jobs. Having only one alternation between O and D jobs may represent an artificial situation in which lookahead is less helpful as there is only one event in the input sequence that needs to be identified. When we consider the case where sequences contain an arbitrary number of alternations between O and D jobs, the lower bound on the lower order term of N_A for any on-line algorithm A is halved.

Corollary 4.1 *For $k \geq 3$, any deterministic algorithm A with lookahead L has $N_A \geq \frac{k+3}{4}$ when there are an unbounded number of alternations between O and D jobs.*

Proof: Let $\mathcal{I} = \mathcal{I}_1\mathcal{I}_2\cdots\mathcal{I}_n$ where $\mathcal{I}_i$ is generated by the adversary of Theorem 4.1 for $1 \le i \le n$. Note $|\mathcal{I}_i|$ is completely independent of $|\mathcal{I}_j|$ for $i \neq j$. From Lemma 3.2, we know $|OPT(\mathcal{I})| \le |(OPT(\mathcal{I}_1))\emptyset(OPT(\mathcal{I}_2))\emptyset\cdots\emptyset(OPT(\mathcal{I}_n))|$; the null tasks may be unnecessary. Furthermore, by the proof of Lemma 3.2, we know A cannot interleave tasks of jobs from $\mathcal{I}_i$ and $\mathcal{I}_j$ for $i \neq j$, so $A(\mathcal{I}) = (A(\mathcal{I}_1))\emptyset(A(\mathcal{I}_2))\emptyset\cdots\emptyset(A(\mathcal{I}_n))$. Therefore, from Theorem 4.1, the best A can do is make $U(A(\mathcal{I})) = \lceil\frac{k+1}{2}\rceil n + n - 1$ while $U(OPT(\mathcal{I})) \le n + n - 1$. Therefore, for large n, $N_A = \frac{\lceil\frac{k+3}{2}\rceil}{2} \ge \frac{k+3}{4}$. ■

Randomization does not substantially change the lower bound result.

Theorem 4.2 *For $k \ge 3$, any randomized algorithm R with lookahead L has $E[N_R] \ge k/4$.*

Proof: This proof is very similar to the proof of Theorem 4.1. We use Yao's method [Yao77] (a variant of the von-Neumann minimax principle) of proving a lower bound on the expected performance ratio of any deterministic algorithm A on problem instances chosen from a specific probability distribution.

The input job sequence $\mathcal{I}$ will be chosen uniformly from the 2 job sequences $O^{k-2}D^{jk+m}$ where $m \in \{1, \lfloor k/2\rfloor + 1\}$ and $j = \lceil\frac{L-(k-2)+\lfloor\frac{k}{2}\rfloor-1}{k}\rceil$. Since there are at least $jk+1 \ge L-(k-2)+\lfloor\frac{k}{2}\rfloor$ D jobs, no deterministic algorithm A can know $\mathcal{I}$ in its entirety until either the F task of the first D job has been scheduled or $\lfloor k/2\rfloor$ O jobs have been scheduled. Let q be the number of O jobs scheduled when either of the above two conditions is met. We assume at this point that A now knows the entire job sequence $\mathcal{I}$ and can proceed optimally. We can ignore the final jk D jobs because of Corollary 3.1.

If $q = m-1$, then A can optimally schedule $\mathcal{I}$ by setting

$$A(\mathcal{I}) = S^qF^{q+1}\emptyset S^{k-2-q}B^{q+1}$$

which makes $U(A(\mathcal{I})) = 1$.

If $q < m-1$ (i.e., $q < \lfloor k/2\rfloor$ when $m = \lfloor k/2\rfloor + 1$), then the best A can do is set

$$A(\mathcal{I}) = S^qF\emptyset S^{k-2-q}F^qBF^{m-(q+1)}\emptyset^{k-m}B^q\emptyset B^{m-q-1}$$

making $U(A(\mathcal{I})) = k-m+2$. When $m = \lfloor k/2+1\rfloor$, $U(A(\mathcal{I})) = k - \lfloor k/2\rfloor + 1 = \lceil k/2\rceil + 1$ (this case is similar to the case where $m = q+2$ in Theorem 4.1).

If $q > m-1$ (i.e., $q > 0$ and $m = 1$), then the best A can do is set

$$A(\mathcal{I}) = S^qF^m\emptyset^{q-m-2}S^{k-q-2}B^m$$

making $U(A(\mathcal{I})) = q-m+2$. When $m = 1$, $U(A(\mathcal{I})) = q+1$ (this case is similar to the case where $m = 1$ in Theorem 4.1). $U(OPT(\mathcal{I})) = 1$ for any value of q.

These results imply that if $q < \lfloor k/2\rfloor$, $E[U(A(\mathcal{I}))] = \frac{(\lceil k/2\rceil+1)+q+1}{2}$. This value is minimized to be $\frac{(\lceil k/2\rceil+1)+1}{2}$ when $q = 0$. If $q = \lfloor k/2\rfloor$, then $E[U(A(\mathcal{I}))] = \frac{(\lfloor k/2\rfloor+1)+1}{2}$. Therefore, by a small margin, the best choice of q for A is $\lfloor k/2\rfloor$ making $E[U(A(\mathcal{I}))] = \frac{\lfloor k/2\rfloor}{2} + 1$. Yao's argument then implies that for any randomized algorithm R, there exists an instance $\mathcal{I}$ chosen from the above distribution such that $E[U(R(\mathcal{I}))] \ge \frac{\lfloor k/2\rfloor}{2} + 1$. Since $U(OPT(\mathcal{I})) = 1$ for both instances, the result follows. ■

Again, the lower bound on the expected competitive ratio of a randomized algorithm only uses a sequence with one alternation of O and D jobs. We can again extend this lower bound to hold for sequences with unbounded numbers of alternations between O and D jobs by repeating subsequences as before.

Corollary 4.2 *Let $k \geq 3$ and R be any randomized algorithm with lookahead L. Then, $E[N_R] > \frac{k+4}{8}$ when there are an unbounded number of alternations of O and D jobs in the input sequence.*

Proof: As in Corollary 4.1, let $\mathcal{I} = \mathcal{I}_1\mathcal{I}_2\cdots\mathcal{I}_n$ where $\mathcal{I}_i$ is drawn uniformly from the two sequence distribution of Theorem 4.2. Note $\mathcal{I}_i$ is selected independently from $\mathcal{I}_j$ for $i \neq j$.

From Lemma 3.2, $OPT(\mathcal{I}) = OPT(\mathcal{I}_1)OPT(\mathcal{I}_2)\cdots OPT(\mathcal{I}_n)$ with a null task inserted in between sub-schedules when necessary. Note that if $\mathcal{I}_i = O^{k-2}D^{jk+1}$, no null task will be necessary. We conservatively assume that $OPT(\mathcal{I})$ will always require a null task in between sub-schedules making $U(OPT(\mathcal{I})) = 2n-1$. Meanwhile, Theorem 4.2 implies $E[U(R(\mathcal{I}))] \geq (k/4+1)n$ since each $\mathcal{I}_i$ must be scheduled independently. Therefore,

$$E[N_R] \geq E\left[\frac{(\frac{k}{4}+1)n}{2n-1}\right] > \frac{k+4}{8}.$$

■

4.2 The Performance of Greedy Algorithms

The following results show that greedy algorithms are nearly optimal and that lookahead makes little difference if we choose to work with the null task measure of performance.

Lemma 4.1 *For all $L \geq 1$ and $k \geq 3$, the competitive ratio $N_{\text{L-GREEDY}} \leq k$.*

Proof: Let m be the number of blocks of D jobs. Consider the first block of D jobs. Let the number of D jobs in this block be q. L-GREEDY schedules it using $k-i$ null tasks, where $i = q \bmod k$. If $i = 0$, then L-GREEDY uses no null tasks. If $i \neq 0$, then OPT must use at least one null task before the final D task of this block. Therefore, the two worst possible scenarios are either L-GREEDY uses $k-1$ null tasks while OPT uses only 1 or both algorithms use 0 null tasks. After the last B task of the first block of D jobs has been scheduled, L-GREEDY uses at most $k-1$ null tasks in scheduling each of the last $m-1$ blocks of D jobs and one null task after each of the first $m-1$ blocks of D jobs for a total of at most $(m-1)k$ null tasks. OPT must also schedule at least one null task following each of the first $m-1$ blocks of D jobs for a total of $m-1$ null tasks. Note L-GREEDY uses k times as many null tasks as OPT for all but the first block. To maximize this difference, L-GREEDY and OPT should use 0 null tasks to schedule the first block of D jobs; otherwise, the ratio will lessen slightly. In this case, $U(\text{L-GREEDY}(\mathcal{I})) \leq (m-1)k$ and $U(OPT(\mathcal{I})) \geq m-1$. Therefore, $N_{\text{L-GREEDY}} \leq k$. ■

Theorem 4.3 *For $k \geq 3$, the competitive ratio $N_{\text{1-GREEDY}} = k$.*

Proof: By Lemma 4.1, $N_{\text{1-GREEDY}} \leq k$. This bound is tight as demonstrated by the sequence $\mathcal{I} = D^k O^{k-2} D$. ■

Theorem 4.4 *Let $L \geq 2$ and $k \geq 3$. Then, $k - 1 \leq N_{\text{L-GREEDY}} \leq k$.*

Proof: By Lemma 4.1, $N_{\text{L-GREEDY}} \leq k$. The lower bound follows from the sequence $\mathcal{I} = O^{kL} D$. ■

5 The Makespan Measure

In this section we consider only the makespan measure of performance. We show that there exists a strong correlation between the amount of lookahead an algorithm is allowed to utilize and its competitive ratio. In particular, the L-GREEDY algorithm has a competitive ratio of approximately 2 for small L but has a competitive ratio approaching 1 for reasonable L. These makespan measure of performance results are closely related to the efficiency of the machine, so these results provide a good guide towards finding good algorithms.

5.1 Lower Bounds on Competitive Ratio

We first prove that any deterministic on-line algorithm A with finite lookahead L is at best $(1+\epsilon)$-competitive, for some $\epsilon > 0$, with respect to the makespan measure of performance. We then extend this result to randomized algorithms with finite lookahead L.

Theorem 5.1 *For any deterministic algorithm A with lookahead L, $M_A \geq 1 + \frac{k-1}{4L+6k+2}$. If L is a multiple of k, then $M_A \geq 1 + \frac{k-1}{4L+4k+4}$.*

Proof: This proof is based on the proof of Theorem 4.1. The adversary initializes the input job sequence $\mathcal{I}$ to be $O^{k-2} D^{L-(k-2)}$. If A schedules an S task of an O job, then the adversary adds another D job to the end of $\mathcal{I}$. Eventually, A must schedule the F task of the first D job. Let q be the number of O jobs scheduled before this first D job is scheduled. Let $j = \lceil \frac{L-(k-2)+(q-1)}{k} \rceil$. Note that $jk \leq L - (k-2) + (q-1) + (k-1) = L + q$. The adversary now chooses between revealing $\mathcal{I}$ to be $\mathcal{I}_1 = O^{k-2} D^{jk+q+2}$ (for small q) or $\mathcal{I}_2 = O^{k-2} D^{jk+1}$ (for large q). From Corollary 3.1, we can temporarily ignore the final jk D jobs.

For $\mathcal{I}_1$, we showed in the proof of Theorem 4.1 that the best A can do is make $U(A(\mathcal{I}_1)) = k - q$ while $U(OPT(\mathcal{I}_1)) = 1$. This means $|A(\mathcal{I}_1)| = 2jk + 2k + q + 2$ while $|OPT(\mathcal{I}_1)| = 2jk + k + 2q + 3$ which implies $M_A(\mathcal{I}_1) = 1 + \frac{k-q-1}{2jk+k+2q+3}$. For $\mathcal{I}_2$, we showed in the proof of Theorem 4.1 that the best A can do is make $U(A(\mathcal{I}_2)) = q + 1$ while $U(OPT(\mathcal{I}_2)) = 1$. This means $|A(\mathcal{I}_2)| = 2jk + k + q + 1$ while $|OPT(\mathcal{I}_2)| = 2jk + k + 1$ which implies $M_A(\mathcal{I}_2) = 1 + \frac{q}{2jk+k+1}$.

The adversary will choose to set $\mathcal{I}$ equal to $\mathcal{I}_1$ if q is small or $\mathcal{I}_2$ if q is large. The best A can do is to choose q so that the two options are equivalent, and this is accomplished by choosing $q \approx \frac{k-1}{2}$ which makes $M_A(\mathcal{I}) \geq 1 + \frac{\frac{k-1}{2}}{2jk+2k+2} \geq 1 + \frac{k-1}{4L+6k+2}$. Note that in converting from jk to L, we use the fact that $jk \leq L + q$. If L is a multiple of k, then we simply use $jk = L$ which improves the lower bound on M_A to $1 + \frac{k-1}{4L+4k+4}$. ■

Theorem 5.2 *For any randomized algorithm R with lookahead L, $E[M_R] \geq 1 + \frac{k}{8L+12k}$ provided $k \geq 3$. If L is a multiple of k, then $E[M_R] \geq 1 + \frac{k}{8L+8k}$.*

Proof: This proof is based on the proof of Theorem 4.2, and we again use Yao's method [Yao77]. The input job sequence $\mathcal{I}$ will be chosen uniformly from the 2 job sequences $O^{k-2}D^{jk+m}$ where $m \in \{1, \lfloor k/2 \rfloor + 1\}$ and $j = \lceil \frac{L-(k-2)+\lfloor \frac{k}{2} \rfloor - 1}{k} \rceil$. (Note $jk \leq L-(k-2)+(\lfloor \frac{k}{2} \rfloor - 1)+(k-1) = L + \lfloor \frac{k}{2} \rfloor$. Since there are at least $jk+1$ D jobs, no deterministic algorithm A can know I in its entirety until the F task of the first D job has been scheduled or at least $\lfloor k/2 \rfloor$ O jobs have been scheduled. Once the first D job has been scheduled or the first $\lfloor k/2 \rfloor$ O jobs have been scheduled, we assume A knows the entire sequence and can proceed optimally. Let q be the number of O jobs scheduled before the first D job has been scheduled. We can temporarily ignore the final jk D jobs because of Corollary 3.1.

In the proof of Theorem 4.2, we showed that the best A can do is set $q = \lfloor k/2 \rfloor$ followed closely by setting $q = 0$. Here, setting $q = 0$ is better because then A's poor performance occurs when $m = \lfloor \frac{k}{2} \rfloor + 1$. This value of m increases $|OPT(\mathcal{I})|$ which in turn decreases $\frac{|A(\mathcal{I})|}{|OPT(\mathcal{I})|}$. If $m = 1$, then $A(\mathcal{I})$ is optimal which means $\frac{|A(\mathcal{I})|}{|OPT(\mathcal{I})|} = 1$. If $m = \lfloor k/2 \rfloor + 1$, then we showed $U(A(\mathcal{I})) = \lceil k/2 \rceil + 1$ which means $|A(\mathcal{I})| = 2jk + k + 2\lfloor k/2 \rfloor + \lceil k/2 \rceil + 1$ while $|OPT(\mathcal{I})| = 2jk + k + 2\lfloor k/2 \rfloor + 1$. Therefore, $E[M_A(\mathcal{I})] = \frac{1}{2}1 + \frac{1}{2}(1 + \frac{\lceil \frac{k}{2} \rceil}{2jk+k+2\lfloor \frac{k}{2} \rfloor + 1}) \geq 1 + \frac{\lceil \frac{k}{2} \rceil}{4jk+4k+2} \geq 1 + \frac{k}{8L+12k+4}$. Note that in converting from jk to L, we use the fact that $jk \leq L + \lfloor \frac{k}{2} \rfloor$. If L is a multiple of k, then we simply use $jk = L$ which improves the lower bound on $E[M_A(\mathcal{I})]$ to $1 + \frac{k}{8L+8k+4}$. ■

5.2 The Performance of Greedy Algorithms

The following results show that lookahead significantly improves the effectiveness of the L-GREEDY algorithm when we work with the makespan measure of performance. This is in marked contrast to the situation with the null task measure of performance. We first show that L-GREEDY is essentially a 2-competitive algorithm when $L < k$. We then show that the competitive ratio of the L-GREEDY algorithm approaches 1 as the ratio $\frac{L}{k}$ increases. In particular, for reasonable values of L, it can be proven that L-GREEDY has good performance.

Theorem 5.3 *For $k \geq 3$ and L a perfect divisor of $k-1$, the competitive ratio $M_{\text{L-GREEDY}} \geq 2 - \frac{2}{k+1}$. For $k \geq 3$ and L not a perfect divisor of $k-1$, let $L' = L\lceil \frac{k-1}{L} \rceil$. Then the competitive ratio $M_{\text{L-GREEDY}} \geq 1 + \frac{k-2}{L'+3}$.*

Proof: Consider the input sequence $\mathcal{I} = (DO^{k-2})^n$ when $k \geq 3$. It is easy to see that L-GREEDY$(\mathcal{I}) = (F\emptyset^{k-1}B\emptyset S^{k-2})^n$ when L is a perfect divisor of $k-1$. Thus $|\text{L-GREEDY}(\mathcal{I})| = 2kn$. Meanwhile, $OPT(\mathcal{I}) = F\emptyset^{k-1}B(F\emptyset S^{k-2}B)^{n-1}\emptyset S^{k-2}$. Thus, $|OPT(\mathcal{I})| = n(k+1) + k - 1$. This implies the competitive ratio of $M_{\text{L-GREEDY}}$ is lower bounded by $\frac{2kn}{(k+1)n+k-1}$ which asymptotically approaches $2 - \frac{2}{k+1}$ as n approaches infinity.

When L is not a perfect divisor of $k-1$, define $L' = L\lceil \frac{k-1}{L} \rceil$. Note $L' \leq 2k-4$. In this case, consider the input sequence $\mathcal{I} = (DO^{L'-1})^n$. It is easy to see that L-GREEDY$(\mathcal{I}) =$

$(F\emptyset^{k-1}B\emptyset S^{L'-1})^n$, so $|\text{L-GREEDY}(\mathcal{I})| = n(L'+k+1)$. Meanwhile, $|OPT(\mathcal{I})| = n(L'+3)+k-1$ because $OPT(\mathcal{I}) = F\emptyset^{k-1}B(\emptyset S^{L'-k+1}F\emptyset S^{k-2}B)^{n-1}\emptyset S^{L'-1}$. This implies the competitive ratio of $M_{\text{L-GREEDY}}$ is lower bounded by $\frac{n(L'+k+1)}{n(L'+3)+k-1}$ which asymptotically approaches $1+\frac{k-2}{L'+3}$ as n approaches infinity. ∎

Thus, when $L \le k-1$, the competitive ratio of L-GREEDY ranges between approximately $\frac{3}{2}$ and 2. In particular, when $L=1$, the competitive ratio of L-GREEDY is $2-\frac{2}{k+1}$. We now show that this lower bound on the competitive ratio approaches 1 when L increases.

Theorem 5.4 *For $k \ge 3$ and $L \ge k$, the competitive ratio $M_{\text{L-GREEDY}} \ge 1+\frac{k-2}{L+3}$.*

Proof: Consider the input sequence $\mathcal{I} = (DO^{L-1})^n$. $\text{L-GREEDY}(\mathcal{I}) = (F\emptyset^{k-1}B\emptyset S^{L-1})^n$, so $|\text{L-GREEDY}(\mathcal{I})| = n(L+k+1)$. $OPT(\mathcal{I}) = F\emptyset^{k-1}B(\emptyset S^{(L-1)-(k-2)}F\emptyset S^{k-2}B)^{n-1}\emptyset S^{L-1}$ (when $L \ge k$ and $k \ge 3$), so $|OPT(\mathcal{I})| = n(L+3)+k-2$. Thus, $M_{\text{L-GREEDY}}(\mathcal{I})$ is lower bounded by $\frac{n(L+k+1)}{n(L+3)+k-2}$ which asymptotically approaches $1+\frac{k-2}{L+3}$ as n increases to infinity. ∎

We now prove an almost matching upper bound on the performance of L-GREEDY for $L \ge k$. We will actually analyze a slightly different algorithm which we call L-CHUNK to prove the upper bound. The algorithm L-CHUNK schedules each block of L jobs independently and simply concatenates the sub-schedules together to form a schedule for I. This algorithm clearly is no better than L-GREEDY because it fails to take advantage of gaps in the resulting schedule of the previously scheduled block of L jobs; that is, it ignores these gaps and simply schedules each block of L jobs independently. However, L-CHUNK is easier to analyze. Furthermore, for worst-case instances as in the proof of Theorem 5.4, the two algorithms are identical.

Theorem 5.5 *For $k \ge 3$ and $L \ge k$, $M_{\text{L-CHUNK}} \le 1+\frac{k-1}{L+1}$.*

Proof: Let $|\mathcal{I}| = n$. Let $q = \lceil \frac{n}{L} \rceil$. Let $\mathcal{I}(i)$ be the L jobs $\mathcal{I}_{(i-1)L+1}$ to $\mathcal{I}_{iL}$ for $1 \le i < q$. Let $\mathcal{I}(q)$ be the jobs $\mathcal{I}_{(q-1)L+1}$ to $\mathcal{I}_n$. Take $OPT(\mathcal{I})$ and break it into q different subschedules $OPT[\mathcal{I}(i)]$ for $1 \le i \le q$ where the i^{th} subschedule contains the minimum length portion of $OPT(\mathcal{I})$ in which tasks associated with jobs in $\mathcal{I}(i)$ occur. Let q' denote the number of input chunks $\mathcal{I}(i)$ for $2 \le i \le q$ which begin with a D job.

By definition of L-CHUNK, $|\text{L-CHUNK}(\mathcal{I}(i))| \le |OPT[\mathcal{I}(i)]|$ for $1 \le i \le q$. Thus $\sum_{i=1}^{q} |\text{L-CHUNK}(\mathcal{I}(i))| \le \sum_{i=1}^{q} |OPT[\mathcal{I}(i)]|$. Of course, we wish to compare $|\text{L-CHUNK}(\mathcal{I})|$ with $|OPT(\mathcal{I})|$, not $\sum_{i=1}^{q} |\text{L-CHUNK}(\mathcal{I}(i))|$ with $\sum_{i=1}^{q} |OPT[\mathcal{I}(i)]|$.

To move towards this comparison, we first observe that $\sum_{i=1}^{q} |OPT[\mathcal{I}(i)]|$ overestimates $|OPT(\mathcal{I})|$ because a given task which appears only once in $OPT(\mathcal{I})$ may appear in $OPT[\mathcal{I}(i)]$ for more than one value of i. However, because $L \ge k-1$, it is easy to see that a given task can appear in at most two subschedules, $OPT[\mathcal{I}(i)]$ and $OPT[\mathcal{I}(i+1)]$ for some i between 1 and $q-1$. Furthermore, it is easy to see that $OPT[\mathcal{I}(i)]$ and $OPT[\mathcal{I}(i+1)]$ share at most $k-1$ tasks, and this can happen only if the first job of $\mathcal{I}(i)$ is a D job. Otherwise, if the first job of $\mathcal{I}(i)$ is an O job, the maximum sharing between $OPT[\mathcal{I}(i)]$ and $OPT[\mathcal{I}(i+1)]$ is $k-2$ jobs since the S task of the first O job of $\mathcal{I}(i+1)$ will not appear in

$OPT[\mathcal{I}(i)]$. Thus

$$|OPT(\mathcal{I})| \geq \sum_{i=1}^{q} |OPT[\mathcal{I}(i)]| - (q-1)(k-2) - q' \tag{1}$$

or

$$|OPT(\mathcal{I})| + (q-1)(k-2) + q' \geq \sum_{i=1}^{q} |OPT[\mathcal{I}(i)]|. \tag{2}$$

We now show that

$$|\text{L-CHUNK}(\mathcal{I})| \leq \sum_{i=1}^{q} |\text{L-CHUNK}(\mathcal{I}(i))| + (q-1-q') \tag{3}$$

or

$$|\text{L-CHUNK}(\mathcal{I})| - (q-1-q') \leq \sum_{i=1}^{q} |\text{L-CHUNK}(\mathcal{I}(i))|. \tag{4}$$

The $q-1-q'$ term in equation 4 arises due to the possible need for a null task between L-CHUNK$(\mathcal{I}(i))$ and L-CHUNK$(\mathcal{I}(i+1))$ for $1 \leq i \leq q-1$. This null task might be needed only if $\mathcal{I}(i+1)$ begins with an O job, and thus the $q-q-q'$ term.

Putting equations 2 and 4 together, we derive the fact that $|OPT(\mathcal{I})| + (q-1)(k-1) \geq |\text{L-CHUNK}(\mathcal{I})|$. Thus $M_{\text{L-CHUNK}}(\mathcal{I}) \leq \frac{|OPT(\mathcal{I})|+(q-1)(k-1)}{|OPT(\mathcal{I})|}$. Furthermore, in order for this to happen, there must be a D job in each $\mathcal{I}(i)$ for $2 \leq i \leq q$, so $|OPT(\mathcal{I})| \geq n+q-1$ (since each D job will require at least 2 tasks in $OPT(\mathcal{I})$). Clearly, $M_{\text{L-CHUNK}}(\mathcal{I})$ is maximized when $|OPT(\mathcal{I})|$ is minimized, so we derive that $M_{\text{L-CHUNK}}(\mathcal{I}) \leq \frac{n+(q-1)+(q-1)(k-1)}{n+(q-1)} = 1 + \frac{(q-1)(k-1)}{n+q-1} \leq 1 + \frac{(k-1)(q-1)}{L(q-1)+q-1} = 1 + \frac{k-1}{L+1}$. ■

Corollary 5.1 *For $k \geq 3$ and $L \geq k$, $1 + \frac{k-2}{L+3} \leq M_{\text{L-GREEDY}} \leq 1 + \frac{k-1}{L+1}$.*

6 Removing the Orientation Constraint

While most reprographic machines place the inverter (orienter) prior to the return path, there are machines with the inverter in the return path itself. This means that the orientation constraint can be removed if the back side of a double-sided sheet is printed first. In this section, we consider this modified problem.

This modified problem differs from the original problem in one important way. In the original problem, the number of null passes in the optimal schedule for a particular input sequence could not be upper bounded by any constant as it is lower bounded by the number of transitions between D and O jobs (Lemma 3.4). However, without the inversion constraint, this lower bound no longer applies. In fact we can prove that there is an upper bound of $k-1$ on the number of null tasks in the optimal schedule for any input sequence; that is, for any input sequence $\mathcal{I}$, $U(OPT(\mathcal{I})) \leq k-1$.

Our proof of this fact uses an algorithm A developed by the Machine Control Group at Xerox PARC which is confidential information. The algorithm A has the property such that there are no null tasks to the right of any S or B tasks in $A(\mathcal{I})$ for any input $\mathcal{I}$. Therefore, the only null tasks that are in the schedule must be to the left of the leftmost B task. There can be at most $k-1$ null tasks to the left of the leftmost B task, and the result follows.

Clearly we cannot hope to do better than this as the input sequence D must be scheduled with $k-1$ null tasks. We now examine what effects this property has on our makespan measure of performance results.

Theorem 6.1 *For $L \geq k$ and $k \geq 2$,* L-GREEDY *is exactly $\left(1+\frac{k-1}{L+1}\right)$-competitive.*

Proof: The lower bound follows from considering the input sequence $\mathcal{I} = O^L(DO^{L-1})^n D$. $OPT(\mathcal{I}) = S^{L-k-1}FS^{k-1}B(S^{L-k-2}FS^{k-1}B)^n$ which means $|OPT(\mathcal{I})| = L+2+n(L+1)$. Meanwhile, L-GREEDY$(\mathcal{I}) = S^L F\emptyset^{k-1}B(S^{L-1}F\emptyset^{k-1}B)^n$ which means $|$L-GREEDY$(\mathcal{I})| = L+k+1+n(L+k)$. Therefore, $M_{\text{L-GREEDY}}(\mathcal{I})$ asymptotically approaches $\frac{L+k}{L+1} = 1+\frac{k-1}{L+1}$ as n increases.

The upper bound follows immediately from the proof of Theorem 5.5. ■

Theorem 6.2 *For $k \geq 2$ and L a perfect divisor of k, the competitive ratio of* L-GREEDY *is lower bounded by $2-\frac{2}{k+1}$. For $k \geq 2$ and $L \leq k-1$ and not a perfect divisor of k, let $L' = L\lceil \frac{k}{L} \rceil$. Then the competitive ratio of* L-GREEDY *is lower bounded by $1+\frac{k-1}{L'+1}$.*

Proof: When $k \geq 2$ and L is a perfect divisor of k, consider the input sequence $\mathcal{I} = (DO^{k-1})^n$. L-GREEDY$(\mathcal{I}) = F\emptyset^{k-1}BS^{k-1}$, so $|$L-GREEDY$(\mathcal{I})| = 2nk$. Meanwhile, $OPT(\mathcal{I}) = F\emptyset^{k-1}B(FS^{k-1}B)^{n-1}S^{k-1}$, so $|OPT(\mathcal{I})| = n(k+1)+k-1$. Thus, the competitive ratio of L-GREEDY is lower bounded by $\frac{2nk}{n(k+1)+k-1}$ which asymptotically approaches $2-\frac{2}{k+1}$ as n approaches infinity.

When $k \geq 2$ and $L \leq k-1$ is not a perfect divisor of k, let $L' = L\lceil \frac{k}{L} \rceil$. Consider the input sequence $\mathcal{I} = (DO^{L'-1})^n$. L-GREEDY$(\mathcal{I}) = F\emptyset^{k-1}BS^{L'-1}$, so $|$L-GREEDY$(\mathcal{I})| = n(L'+k)$. Meanwhile, $OPT(\mathcal{I}) = F\emptyset^{k-1}B(S^{L'-k}FS^{k-1}B)^{n-1}S^{L'-1}$, so $|OPT(\mathcal{I})| = n(L'+1)+k-1$. Thus, the competitive ratio of L-GREEDY is lower bounded by $\frac{n(L'+k)}{n(L'+1)+k-1}$ which asymptotically approaches $1+\frac{k-1}{L'+1}$ as n approaches infinity. ■

Theorem 6.3 *Let $k \geq 2$.*

1. *Any deterministic algorithm A with lookahead L has competitive ratio $M_A \geq 1+\frac{k-1}{4L+6k}$.*
2. *Any randomized algorithm R with lookahead L has expected competitive ratio $E[M_R] \geq 1+\frac{k}{8L+12k}$.*

The proof of this theorem is almost identical to the proofs of Theorem 5.1 and Theorem 5.2, so we do not repeat these proofs here.

We now examine what effect this property has on our null task measure of performance results. We find, in contrast to our makespan measure of performance results, our results with respect to the null task measure of performance change dramatically.

Theorem 6.4 *For $k \geq 2$, N_A is unbounded for deterministic algorithms A with finite lookahead.*

Proof: This proof will be almost identical to that of Corollary 4.1. The only change is that now $U(OPT(\mathcal{I})) = 0$. Since we can pump $U(A(\mathcal{I}))$ to an arbitrarily large value for any on-line algorithm A with finite lookahead, N_A for any A is unbounded. ∎

Theorem 6.5 *For $k \geq 2$, $E[N_R]$ is unbounded for randomized algorithms R with finite lookahead.*

Proof: This proof is almost identical to the proof of Corollary 4.2. Note that $U(OPT(\mathcal{I})) = 0$ and $E[U(A(\mathcal{I}))]$ can be made arbitrarily large for any deterministic algorithm on this input distribution. Therefore, $E[N_R]$ is unbounded for any randomized algorithm R with finite lookahead. ∎

The stability of our results with respect to the makespan measure of performance and the instability of our results with respect to the null task measure of performance provide further evidence that the makespan measure of performance results are more meaningful than our null task measure of performance results.

7 Multipass Assembly Line Scheduling

Finally, we consider the general multipass assembly line scheduling problem (i.e., with arbitrary m), and we only consider the more practical makespan measure of performance. We again assume the existence of the orientation constraint.

With respect to lower bounds, we currently are unable to derive stronger results than we were for the case $m = 2$. The problem with our techniques is that while we can force any on-line algorithm to utilize $\Omega(mk)$ extra jobs per $J_1^{mk} J_m^L$ substring instead of only $\Omega(k)$ extra jobs, we can only accomplish this by also increasing the optimal makespan for this substring from $\Omega(L)$ to $\Omega(mL)$ which means that the competitive ratio will be $1 + \frac{mk}{mL} = 1 + \frac{k}{L}$. Therefore, Theorem 5.1 and Theorem 5.2 provide the best lower bounds on the competitive ratio of any on-line algorithm with finite lookahead that we can prove.

Unfortunately, our upper bound result for L-GREEDY does not generalize as well because L-GREEDY can be forced to use $(m-1)(k-2)$ extra null tasks on sequences which can be optimally scheduled with roughly $L + m$ tasks.

Lemma 7.1 *If $\mathcal{I} = wxy$ where $x = J_1(1)^{(m-1)(k-2)}$, then $OPT(\mathcal{I}) = OPT(w) \circ OPT(xy)$ or $OPT(\mathcal{I}) = OPT(w) \circ \emptyset \circ OPT(xy)$.*

Proof: This proof generalizes naturally from the proof of Lemma 3.2. We now need $(m-1)(k-2)$ jobs to insure that no tasks from y can interleave with tasks from w. ∎

Theorem 7.1 *Let $k \geq 3$ and $L \geq (m-1)(k-2) + 1$. Then,*

$$1 + \frac{(k-2)(m-1) - 1}{L + 2m - 1} \leq M_{\text{L-GREEDY}} \leq 1 + \frac{(m-1)(k-1)}{L + m - 1}.$$

Proof: The lower bound proof is a generalization of the proof of Theorem 5.4 with job J_m used in place of job D. The upper bound proof is a generalization of the proof for Theorem 5.5 with job J_m used in place of job D. ∎

8 Further Work

Three main open problems remain. First, can we close the gap between the competitive ratio of L-GREEDY and our lower bounds on the optimal competitive ratio of any on-line algorithm, particularly in the general case? For example, the greedy algorithm which commits to scheduling only the first job of the optimal extension rather than the entire extension might do slightly better. Second, is L-GREEDY still effective when we expand our model to include other features of networked reprographic machines such as handling multiple page sizes and printing colored pages? Finally, how does the problem change if we focus on client-oriented performance measures such as average response time rather than machine-oriented performance measures such as makespan?

References

[ABF93] B. Awerbuch, Y. Bartal, and A Fiat. Competitive distributed file allocation. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 164–173, 1993.

[Alb93] S. Albers. The influence of lookahead in competitive on-line algorithms. Technical Report MPI-I-92-143, Max Planck Institute, Germany, August 1993.

[BIRS91] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proceedings of 23rd ACM Symp. on Theory of Computing*, pages 249–259, 1991.

[EK81] D. Ermer and S. Kromodihardjo. Optimization of multipass turning with constraints. *Transactions of the ASME*, 103:462–468, 1981.

[FKL+91] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[FR97] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the lru standard. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[Gas88] A. Gascon. The lookahead heuristic for multi-item single machine production scheduling with dynamic, stochastic demands. *INFOR*, 26:114–126, 1988.

[Gra66] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[IKP92] S.S. Irani, A.R. Karlin, and S.J. Phillips. Strongly competitive algorithms for paging with locality of reference. In *Third ACM-SIAM Symposium on Discrete Algorithms*, pages 228–236, 1992.

[KMW96] S. Khanna, R. Motwani, and R. Wilson. On certificates and lookahead in dynamic graph problems. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 222–231, 1996.

[KPR92] A.R. Karlin, S.J. Phillips, and P. Raghavan. Markov paging. In *Proc. of 33rd IEEE Symp. on Foundations of Computer Science*, pages 208–217, 1992.

[MPT94] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[RL86] A. Redford and E. Lo. *Robots in assembly.* Halsted Press, New York, 1986.

[SSL89] C. Sriskandarajah, S. Sethi, and P. Ladet. Scheduling methods for a class of flexible manufacturing systems. *Annals of Operations Research*, 17:139–162, 1989.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28:202–208, 1985.

[Yao77] A.C.-C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. of 18th IEEE Symp. on Foundations of Computer Science*, pages 222–227, 1977.