SIAM J. COMPUT.
Vol. 43, No. 1, pp. 126–143

ONLINE SCHEDULING WITH GENERAL COST FUNCTIONS*

SUNGJIN IM†, BENJAMIN MOSELEY‡, AND KIRK PRUHS§

Abstract. We consider a general online scheduling problem where the goal is to minimize $\sum_j w_j g(F_j)$, where w_j is the weight/importance of job J_j, F_j is the flow time of the job in the schedule, and g is an arbitrary nondecreasing cost function. Numerous natural scheduling objectives are special cases of this general framework. We show that the scheduling algorithm Highest Density First (*HDF*) is $(2+\epsilon)$-speed $O(1)$-competitive for all cost functions g *simultaneously.* We give lower bounds that show that the *HDF* algorithm and this analysis are essentially optimal. Finally, we show that scalable algorithms are achievable in some special cases.

Key words. online scheduling, simultaneous optimization, general cost functions, speed augmentation

AMS subject classification. 68W05

DOI. 10.1137/120902288

1. Introduction.

In online scheduling problems a collection of jobs $J_1, \ldots, J_n$ arrive over time to be scheduled by one or more servers. Job J_j arrives at a nonnegative real release time r_j and has a positive real size/work p_j. A client submitting a job would like the job completed as quickly as possible. In other words, the client desires the sever to minimize the flow time of the job. The flow time F_j of job J_j is defined as $C_j - r_j$, where C_j is the time when the job J_j completes. When there are multiple unsatisfied jobs, the server is required to make a scheduling decision of which job or jobs to prioritize. The order in which the jobs are completed depends on a global scheduling objective. For example, a global objective could be to minimize the total flow time of all the jobs. A scheduler for this objective optimizes the average performance. Another possible objective is to minimize the total squared flow time, i.e., $\sum_j (F_j)^2$. This objective naturally balances average performance and fairness. The scheduling literature has primarily focused on designing and analyzing algorithms separately for each objective.

In this paper, we study a framework for online single machine scheduling problems that generalizes many natural scheduling objectives. For our problem, we allow each job to have a positive real weight/importance w_j. For a job J_j with flow time F_j, a cost of $w_j g(F_j)$ is incurred for the job. The only restriction on the cost function $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is that it be nondecreasing, so that it is never cheaper to finish a job later. The cost of a schedule is $\sum_j w_j g(F_j)$. We assume that preemption is allowed without any penalty. This framework generalizes many scheduling problems that have been studied in the scheduling literature such as the objectives mentioned above and the following:

*Received by the editors December 12, 2012; accepted for publication (in revised form) September 17, 2013; published electronically January 30, 2014. Preliminary results from this work appeared previously in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2012. The work of the first and second authors was partially supported by NSF grants CCF-0728782 and CCF-1016684.
http://www.siam.org/journals/sicomp/43-1/90228.html

†Electrical Engineering and Computer Science, University of California, Merced, CA 95343 (sim3@ucmerced.edu).

‡Toyota Technological Institute at Chicago, Chicago, IL 606371 (moseley@ttic.edu).

§Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 (kirk@cs.pitt.edu). This author's work was partially supported by NSF grant CCF-0830558 and by an IBM Faculty Award.

- Weighted flow time: When $g(x) = x$, the objective becomes the total weighted flow time [6]. The total stretch is a special case of the total weighted flow time where $w_j = 1/p_j$ [7].
- Weighted flow time squared: If $g(x) = x^2$, then the scheduling objective is the sum of the weighted squares of the flows of the jobs [3].
- Weighted tardiness with equal spans: Assume that there is a deadline d_j for each job J_j that is equal to the release time of j plus a fixed span d. If $g(t) = 0$ for t not greater than the deadline d_j, and $g(t) = w_j(t - d_j)$ for t greater than the deadline $r_j + d$, then the objective is weighted tardiness.
- Weighted exponential flow: If $g(x) = a^x$ for some real value $a > 1$, then the scheduling objective is the sum of the exponentials of the flow, which has been suggested as an appropriate objective for scheduling problems related to air traffic control and to quality control in assembly lines [4, 5].

For the latter two objectives, no nontrivial results were previously known in the online setting. Note that our general problem formulation encompasses settings where the penalty for delay may be discontinuous, as is the penalty for late filing of taxes or late payment of parking fines. To the best of our knowledge, minimizing a discontinuous cost function has not been been previously studied in nonstochastic online scheduling.

Most commonly one seeks online algorithms that guarantee that the degradation in the scheduling objective relative to some benchmark is modest/minimal/bounded. The most natural benchmark is the optimal offline schedule. If every online algorithm performs poorly compared to the optimal solution, as is commonly the case, the most commonly used alternate benchmark is the optimal schedule on a slower processor [15]. The algorithm A is said to be s-speed c-competitive if A with an s-speed processor is guaranteed to produce a schedule with an objective value at most c times the optimal objective value obtainable on a unit-speed processor. The informal notion of an online scheduling algorithm A being "reasonable" is then generally formalized as A having constant competitiveness for some small constant speed augmentation s. Intuitively, an s-speed $O(1)$-competitive algorithm should be able to handle a load of $\frac{1}{s}$ of the server capacity [16]. Usually the ultimate goal is to find a scalable algorithm, one where the speed augmentation required to achieve $O(1)$-competitiveness is arbitrarily close to one. Our main result, given in section 2, is the following:

- The scheduling algorithm Highest Density First (*HDF*) is $(2 + \epsilon)$-speed $O(1)$-competitive for all cost functions g.

The density of a job J_j is $d_j = \frac{w_j}{p_j}$, the ratio of the weight of the job over the size of the job. The algorithm *HDF* always processes the job of highest density. Note that *HDF* is $(2 + \epsilon)$-speed $O(1)$-competitive *simultaneously* for all cost functions g. This is somewhat surprising since *HDF* is oblivious to the cost function g. Indeed, this implies that *HDF* performs reasonably for highly disparate scheduling objectives such as average flow time and exponential flows. In practice it is often not clear what the scheduling objective should be; for competing objectives, tailoring an algorithm for one can come at the cost of not optimizing the other. Our analysis shows that no single objective needs to be chosen. As long as the objective falls into the very general framework that we consider, *HDF* will optimize the objective. The main idea of this analysis of *HDF* is to show that at all times, and for all ages A, there must be $\Omega(1)$ times as many jobs of age A in the optimal (or an arbitrary) schedule as there are in *HDF*'s schedule. The bulk of the proof is a constructive method to identify the old jobs in the optimal schedule.

In section 3 we also show that it is not possible to significantly improve upon

HDF, or this analysis, along several axes:

- If each job J_j has a distinct cost function g_j, then there is no $O(1)$-speed $O(1)$-competitive algorithm for the objective $\sum_j g_j(F_j)$. Thus it is necessary that the cost functions for the jobs be uniform. Our lower bound instance is similar to and inspired by an instance given in Theorem 6.1 of [10].
- There is no online algorithm that is $(2-\epsilon)$-speed $O(1)$-competitive and oblivious to the cost function for any fixed $\epsilon > 0$. Hence *HDF* is essentially the optimal oblivious algorithm.
- No scalable algorithm exists. In other words, while there may be a non-oblivious algorithm that is $O(1)$-competitive with less than a factor of two in speed augmentation, some nontrivial speed augmentation is necessary.

All of these lower bounds hold even in the case where all jobs have unit weights. Hence, the intrinsic difficulty of the problem is unaffected by weights/priorities. All of these lower bounds hold even for randomized algorithms. Hence, randomization does not seem to be particularly useful to the online algorithm. In contrast, we show that in some special cases, scalable algorithms are achievable:

- In section 4 we show that the algorithm First-In-First-Out (*FIFO*) is scalable when jobs have unit sizes and weights.
- In section 5 we show that a variation of the algorithm Weighted Late Arrival Processor Sharing (*WLAPS*) is scalable when the cost function g is concave, continuous, and twice-differentiable; hence $g''(F) \leq 0$ for all $F \geq 0$. A concave cost function implies that the goal is to finish as many jobs as quickly as possible. The longer a job waits to be satisfied, the less urgent it is to complete the job. This objective can be viewed as making a few clients really happy rather than making all clients moderately happy. Although all of the scheduling literature that we are aware of focuses on convex cost functions, there are undoubtedly some applications where a concave cost function better models the scheduler's objectives. The algorithm *WLAPS* is oblivious to the cost function g as well as nonclairvoyant. A nonclairvoyant algorithm is oblivious to the sizes of the jobs.

1.1. Related results. The online scheduling results that are probably most closely related to the results here are the results in [3], which considers the special case of our problem where the cost function is polynomial. The results in [3] are similar in spirit to the results here. They show that well-known priority scheduling algorithms have the best possible performance. In particular, [3] showed that *HDF* is $(1+\epsilon)$-speed $O(1/\epsilon^k)$-competitive, where k is the degree of the polynomial and $0 < \epsilon < 1$. [3] also showed similar results for the scheduling algorithms Shortest Job First and Shortest Remaining Processing Time, where jobs are of equal weight/importance. Notice that these results depend on the degree of the polynomial. Our work shows that *HDF* is $O(1)$-competitive independent of the rate of growth of the objective function when given $2 + \epsilon$ resource augmentation for a fixed $0 < \epsilon < 1$. [3] also showed that any online algorithm is $n^{\Omega(1)}$-competitive without resource augmentation. The analyses of *HDF* in [3] essentially showed that at all times, and for all ages A, there must be $\Omega(1)$ times as many jobs of age $\Omega(A)$ in the optimal (or an arbitrary) schedule as there are in *HDF*'s schedule. If the cost function g is arbitrary, such a statement is not sufficient to establish $O(1)$-competitiveness. In particular, if the cost function $g(F)$ grows exponentially quickly depending on F or has discontinuities, the previous analysis does not imply that *HDF* has bounded competitiveness. We show the stronger statement that there are $\Omega(1)$ times as many jobs in the optimal schedule that are of age at least A. This necessitates that our proof be quite different than that in [3].

It is well known that Shortest Remaining Processing Time is optimal for total flow time, when all jobs are of equal weight/importance and when $g(x) = x$. *HDF* was first shown to be scalable for weighted flow, when $g(x) = x$, in [6]. The nonclairvoyant algorithm Shortest Elapse Time First is scalable for total flow time [15]. The algorithm *LAPS* that round robins among recently arriving jobs is also nonclairvoyant and scalable for total flow time [12]. The nonclairvoyant algorithm *WLAPS*, a natural extension of *LAPS*, was shown to be scalable for weighted flow time [1], and later for weighted squares of flow time [11].

Recently, Bansal and Pruhs considered the offline version of this problem, where each job J_j has an individual cost function $g_j(x)$ [2]. The main result in [2] is a polynomial-time $O(\log \log nP)$-approximation algorithm, where P is the ratio of the size of the largest job to the size of the smallest job. This result is without speed augmentation. Obtaining a better approximation ratio, even in the special case of uniform linear cost functions, that is, when $g(x) = x$, is a well-known open problem. Thus it is fair to say that the problem that considers general cost functions is very challenging even in the offline setting.

1.2. Basic definitions and notation. Before describing our results, we formally define some notation. Let n denote the total number of jobs. Jobs are indexed as $J_1, J_2, \ldots, J_n$. Job J_i arrives at time r_i having weight/importance w_i and initial work/size p_i. For a certain schedule A, let C_i^A be the completion time of J_i under the schedule A. Let $F_i^A = C_i^A - r_i$ denote the flow time of job J_i. The cost function $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a nondecreasing function that takes a flow time and gives the cost for the flow time. That is, it incurs cost $g(F_i^A)$ for the unweighted objective and $w_i g(F_i^A)$ for the weighted objective. If the schedule is clear in context, the notation A may be omitted. Similarly, we let C_i^* and F_i^* denote the completion time and flow time of job J_i by a fixed optimal schedule. We will let $A(t)$ denote the set of jobs that are not satisfied at time t by the online algorithm A we consider. Likewise, $O(t)$ denotes the analogous set for a fixed optimal solution OPT. We will overload notation and allow A and OPT to denote the algorithms A and OPT as well as their final objectives. We will use $p_i^A(t)$ and $p_i^O(t)$ to denote the work remaining at time t for job J_i in A's schedule and OPT's schedule, respectively. Throughout the paper, for an interval I, we let $|I|$ denote the length of the interval I. For two intervals I and $I' \subseteq I$ we will let $I \setminus I'$ denote I with the subinterval I' removed.

2. Analysis of *HDF*. We show that Highest Density First (*HDF*) is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive, for any fixed $\epsilon > 0$, for the objective of $\sum_{i \in [n]} w_i g(F_i)$. We first appeal to the result in [6] that if *HDF* is s-speed c-competitive when jobs are unit-sized, then *HDF* is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon} \cdot c\right)$-competitive when jobs have varying sizes. Although in [6] this reduction is stated only for the objective of weighted flow, it can easily be extended to our general cost objective.

LEMMA 2.1 (see [6]). *If HDF is s-speed c-competitive for minimizing $\sum_{i \in [n]} w_i g(F_i)$ when all jobs have unit size and arbitrary weights, then HDF is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon} \cdot c\right)$-competitive for the same objective when jobs have varying sizes and arbitrary weights, where $\epsilon > 0$ is a constant.*

Before we prove this lemma, we show how to show that if some online algorithm A' is s-speed c-competitive when jobs are unit-sized, then there exists another online algorithm A that is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon} \cdot c\right)$-competitive when jobs have varying sizes.

After we show this, we will show how to replace A' and A with *HDF*.

LEMMA 2.2. *Given an online algorithm that is s-speed c-competitive for minimizing $\sum_{i\in[n]} w_i g(F_i)$ when all jobs have unit size and arbitrary weights, then there is an online algorithm that is $(1+\epsilon)s$-speed $\left(\frac{1+\epsilon}{\epsilon}\cdot c\right)$-competitive for the same objective when jobs have varying sizes and arbitrary weights where $\epsilon > 0$ is any constant.*

Proof. Let A' denote an algorithm that is s-speed c-competitive for minimizing $\sum_{i\in[n]} w_i g(F_i)$ when all jobs have unit size and arbitrary weights. Let $\epsilon > 0$ be a constant. Consider any sequence σ of n jobs with varying sizes and varying weights. From this instance, we construct a new instance σ' of unit sizes and varying weight jobs. Here we let Δ denote the unit size, and it is assumed that Δ is sufficiently small such that p_i/Δ and $\frac{\epsilon p_i}{(1+\epsilon)\Delta}$ are integers for all jobs J_i. For each job J_i of size p_i and weight w_i, replace this job with a set $\mathcal{U}_i$ of unit-sized jobs. There are $\frac{p_i}{\Delta}$ unit-sized jobs in $\mathcal{U}_i$; notice that this implies that the total size of the jobs in $\mathcal{U}_i$ is p_i. Each job in $\mathcal{U}_i$ has weight $\frac{\Delta w_i}{p_i}$. Each job in $\mathcal{U}_i$ arrives at time r_i, the same time when J_i arrived in σ. This completes the description of the instance σ'.

Let OPT denote the optimal solution for the sequence σ, and OPT$'$ denote the optimal solution for the sequence σ'. Note that

$$\mathrm{OPT}' \le \mathrm{OPT}. \tag{2.1}$$

This is because the most obvious schedule for σ' corresponding to OPT has cost no greater than OPT. From the above assumption we made on A', we know that with s speed, the cost of A' on σ' is at most $c\mathrm{OPT}'$. Let $\mathcal{U}_i(t)$ denote the jobs in $\mathcal{U}_i$ that have been released but are unsatisfied by time t in A''s schedule. Let β_i denote the first time that $|\mathcal{U}_i(\beta_i)| = \frac{\epsilon p_i}{(1+\epsilon)\Delta}$; recall that $|\mathcal{U}_i(r_i)| = \frac{p_i}{\Delta}$. Knowing that each of the jobs in $\mathcal{U}_i(\beta_i)$ is completed after time β_i in A''s schedule and that $g()$ is nondecreasing, we have

$$\sum_{i\in[n]} |\mathcal{U}_i(\beta_i)| \frac{\Delta w_i}{p_i} g(\beta_i) = \sum_{i\in[n]} \frac{\epsilon w_i}{1+\epsilon} g(\beta_i) \le A'. \tag{2.2}$$

Now consider constructing an algorithm A for the sequence σ based on A'. Whenever the algorithm A' schedules a job in $\mathcal{U}_i$, the algorithm A processes job J_i at a $(1+\epsilon)$ faster rate of speed (unless J_i is completed). We assume that, at any time, A has at most one unit-sized job $\mathcal{U}_i$ that has been partially processed. The algorithm A will complete the job J_i at time β_i. This is because A' completed $\frac{p_i}{\Delta} - \frac{\epsilon p_i}{(1+\epsilon)\Delta} = \frac{p_i}{(1+\epsilon)\Delta}$ jobs in $\mathcal{U}_i$ before β_i. This required A' spending at least $\frac{\Delta}{s}\cdot\frac{p_i}{(1+\epsilon)\Delta} = \frac{p_i}{(1+\epsilon)s}$ time units on jobs in $\mathcal{U}_i$ since A' has s speed and it takes $\frac{\Delta}{s}$ time units for A' to complete a unit-sized job. By the definition of A, the algorithm A with $(1+\epsilon)s$-speed did at least $\frac{p_i}{(1+\epsilon)s}\cdot(1+\epsilon)s = p_i$ volume of work for jobs in $\mathcal{U}_i$ by time β_i. Hence A completed each job J_i by time β_i. Knowing this and by (2.1) and (2.2), we have

$$\begin{aligned} A = \sum_{i\in[n]} w_i g(\beta_i) &= \frac{1+\epsilon}{\epsilon}\sum_{i\in[n]} \frac{\epsilon w_i}{1+\epsilon} g(\beta_i) \\ &\le \frac{1+\epsilon}{\epsilon} A' \le \frac{1+\epsilon}{\epsilon} c\mathrm{OPT}' \qquad \text{[by the definition of } A] \\ &\le \frac{1+\epsilon}{\epsilon} c\mathrm{OPT}. \end{aligned}$$

Knowing that A processes jobs at most $(1+\epsilon)$ times faster than A', we have that A is $(1+\epsilon)s$-speed $\frac{(1+\epsilon)}{\epsilon}c$-competitive for σ. $\square$

We now prove Lemma 2.1.

Proof of Lemma 2.1. Consider any sequence σ of jobs with varying sizes and weights. To prove this lemma consider the conversion of σ to σ' in Lemma 2.2, and consider setting the algorithm A' to *HDF*. Let A denote the algorithm which is generated from *HDF* in the proof of Lemma 2.2. To prove the lemma we prove a stronger statement by induction on time. We will show that at any time t, *HDF* on σ has worked on every job at least as much as A on σ. Here *HDF* and A are both given the same speed.

We prove this by induction on time t. When $t = 0$ the claim clearly holds. Now consider any time $t > 0$, and assume that *HDF* has worked on every job at least as much as A every time before t. Now consider time t. If A does not schedule a job at time t, then the claim follows. Hence, we can assume that A schedules some job J_i at time t. Notice that in the proof of Lemma 2.2, when generating a set of unit-sized jobs $\mathcal{U}_i$ from J_i, the density of the unit-sized jobs in $\mathcal{U}_i$ is the same as the density of job J_i. Knowing that *HDF* has worked at least as much as A on every job, and given the definition of *HDF*, this implies that if J_i is unsatisfied in *HDF*'s schedule at time t, then *HDF* will schedule job J_i. Otherwise J_i is finished in *HDF*'s schedule at time t. In either case, after time t, *HDF* scheduled each job at least as much as A on every job. Knowing that *HDF* processed every job at least as much as A at all times, Lemma 2.2 gives the claim. ∎

Lemma 2.1 implies that showing that *HDF* is 2-speed $O(1)$-competitive for unit-sized jobs is sufficient to prove that *HDF* is $(2 + \epsilon)$-speed $O(1/\epsilon)$-competitive for the varying size job case. Reducing a scheduling problem where jobs have varying sizes to one where jobs have unit size has become standard; e.g., see [6, 3, 9]. Thus we will make our analysis assuming that all jobs have unit size, which can be set to 1 without loss of generality by scaling the instance. We assume without loss of generality that weights are no smaller than 1. For the sake of analysis, we partition jobs into classes $\mathcal{W}_l,\ l \geq 0$, depending on their weight: $\mathcal{W}_l := \{J_i \mid 2^l \leq w_i < 2^{l+1}\}$. We let $\mathcal{W}_{\geq l} := \bigcup_{l' \geq l} \mathcal{W}_{l'}$. Consider any input sequence σ where all jobs have unit size. We consider the algorithm *HDF* with 2 speed-up. Note that *HDF* always schedules the job with the largest weight when jobs have unit size. We assume that *HDF* breaks ties in favor of the job that arrived the earliest. To prove the competitiveness of *HDF* on the sequence σ, we will recast our problem into a network flow where a feasible maximum flow maps flow times of the jobs in the algorithm's schedule and those in the optimal solution's schedule. The weight of each job in the algorithm's schedule will be charged to jobs in the optimal solution's schedule that have flow time at least as large. Moreover, the total weight of the algorithm's jobs mapped to a single job J_i in the optimal solution's schedule will be bounded by $O(w_i)$. Once this is established, the competitiveness of *HDF* follows.

Formally, the network flow graph $G = (V = \{s\} \cup X \cup Y\{t\}, E)$ is constructed as follows. We refer the reader to Figure 2.1. There are source and sink vertices s and t, respectively. There are two partite sets X and Y. There is a vertex $v_{x,i} \in X$ and a vertex $v_{y,i} \in Y$ corresponding to job J_i. Intuitively, the vertices in X correspond to jobs in the algorithm's schedule, and those in Y correspond to jobs in the optimal solution's schedule. There is an edge $(s, v_{x,i})$ with capacity w_i for all $i \in [n]$. There is an edge $(v_{x,i}, t)$ with capacity $8w_i$ for all $i \in [n]$. Making the capacity of edge $(v_{y,i}, t)$ equal to $8w_i$ ensures that job J_i in the optimal solution's schedule is not overcharged. There exists an edge $(v_{x,i}, v_{y,j})$ of capacity ∞ if $F_i \leq F_j^*$ and $w_i \leq w_j$. Recall that F_i and F_i^* denote the flow times of job J_i in the algorithm's and the optimal solution's schedules, respectively.

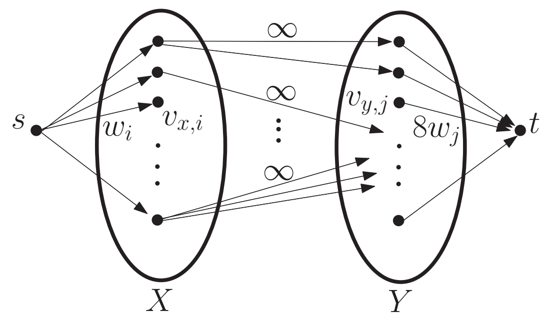

FIG. 2.1. *The graph G.*

Our main task remaining is to show the following lemma.

LEMMA 2.3. *The minimum cut in the graph G is $\sum_{i\in[n]} w_i$.*

Assuming that Lemma 2.3 holds, we can easily prove the competitiveness of *HDF* for unit-sized jobs.

THEOREM 2.4. *HDF is 2-speed 8-competitive for minimizing $\sum_{i\in[n]} w_i \cdot g(F_i)$ when all jobs are unit-sized.*

Proof. Lemma 2.3 implies that the maximum flow f is $\sum_{i\in[n]} w_i$. Let $f(u, v)$ denote the flow on the edge (u, v). Note that the maximum flow is achieved only when $f(s, v_{x,i}) = w_i$ for all jobs $i \in [n]$. We charge the cost of each job in the algorithm's schedule to the optimal cost in the most obvious way, as suggested by the maximum flow. That is, by charging $w_i g(F_i)$ to $\sum_j f(v_{x,i}, v_{y,j})g(F_j^*)$, we have

$$\begin{aligned}
\mathit{HDF} &= \sum_{i\in[n]} w_i g(F_i) \\
&= \sum_{i\in[n]}\sum_{j\in[n]} f(v_{x,i}, v_{y,j})g(F_i) \quad [\text{since } f \text{ is conserved at } v_{x,i}] \\
&\le \sum_{i\in[n]}\sum_{j\in[n]} f(v_{x,i}, v_{y,j})g(F_j^*) \quad [\text{since } (v_{x,i}, v_{y,i}) \in E \text{ only if } F_i \le F_j^*] \\
&= \sum_{j\in[n]} f(v_{y,j}, t)g(F_j^*) \quad [\text{since } f \text{ is conserved at } v_{y,j}] \\
&\le \sum_{j\in[n]} 8w_j g(F_j^*) \quad [8w_j \text{ is the capacity on } v_{y,j}] \\
&= 8\mathrm{OPT}. \qquad \square
\end{aligned}$$

By Lemma 2.1 and Theorem 2.4, we obtain the next result.

THEOREM 2.5. *HDF is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for minimizing $\sum_{i\in[n]} w_i g(F_i)$ when jobs have arbitrary sizes and weights.*

The remainder of this section is devoted to proving Lemma 2.3. Let (S, T) be a minimum s-t cut. For notational simplicity, for any pair of disjoint subsets of vertices A and B, we allow (A, B) to denote the set of edges from vertices in A to vertices in B. We let $c(e)$ denote the capacity of edge e, and $c(A, B)$ the total capacity of all edges in (A, B). Let $X_s = X \cap S$, $X_t = X \cap T$, $Y_s = Y \cap S$, and $Y_t = Y \cap T$. Note that all edges in $(\{s\}, X_t)$ are cut by the cut (S, T); i.e., $(\{s\}, X_t) \subseteq (S, T)$ and

$c(\{s\}, X_t) = \sum_{v_{x,i} \in X_t} w_i$. Knowing that $(Y_s, \{t\}) \subseteq (S, T)$, it suffices to show that

$$8 \sum_{v_{y,j} \in Y_s} w_j \geq \sum_{v_{x,i} \in X_s} w_i. \tag{2.3}$$

This suffices because if we assume that (2.3) is true, we have $c(S,T) \geq \sum_{v_{x,i} \in X_t} w_i + 8 \sum_{v_{y,j} \in Y_s} w_j \geq \sum_{v_{x,i} \in X_t} w_i + \sum_{v_{x,i} \in X_s} w_i = \sum_{i \in [n]} w_i$.

Our attention is focused on showing (2.3). For any $V' \subseteq V$, let $N(V')$ denote the set of out-neighbors of V'; i.e., $N(V') = \{z \mid (v, z) \in E, v \in V'\}$. Since (S, T) is a minimum s-t cut, (S, T) does not contain an edge connecting a vertex in X to a vertex in Y; recall that such an edge has infinite capacity. Therefore $N(X_s) \subseteq Y_s$, where $N(X_s)$ is the out-neighborhood of the vertices in X_s. For any positive integer l, define $\mathcal{W}_l(X_s) := \{v_{x,i} \mid v_{x,i} \in X_s, J_i \in \mathcal{W}_l\}$; recall that J_i is in class $\mathcal{W}_l$ if $2^l \leq w_i < 2^{l+1}$. We show the following key lemma. Here it is shown that the neighborhood of $\mathcal{W}_l(X_s)$ is large compared to $|\mathcal{W}_l(X_s)|$.

LEMMA 2.6. *The vertices in $\mathcal{W}_l(X_s)$ have at least $\frac{1}{2}|\mathcal{W}_l(X_s)|$ neighbors in Y, i.e., $|N(\mathcal{W}_l(X_s)) \cap Y| \geq \frac{1}{2}|\mathcal{W}_l(X_s)|$.*

Proof. Consider each maximal busy time interval I where *HDF* is always scheduling jobs in $\mathcal{W}_{\geq l}$. Let $C(I, l)$ be the set of jobs in $\mathcal{W}_l(X_s)$ which are completed by *HDF* during the interval I. Let J_k be the job that is in $\mathcal{W}_l(X_s)$ which is completed during the interval I and has the highest priority in *HDF*'s schedule (if such a job exists). This implies that the job J_k has the shortest flow time of any job in $\mathcal{W}_l(X_s)$ that is completed during the interval I. We will show that $v_{x,k}$ has at least $\frac{1}{2}|C(I,l)|$ neighbors in Y, i.e.,

$$|N(\{v_{x,k}\}) \cap Y| \geq \frac{1}{2}|C(I, l)|, \tag{2.4}$$

and that all jobs corresponding to these neighbors were completed by *HDF* during I. Taking a union over all maximal busy intervals will complete the proof.

We now focus on proving (2.4). Recall that $F_k = C_k - r_k$ is the flow time of job J_k. Since J_k has the highest priority among all jobs in $C(I, l)$, J_k is not preempted during $[r_k, C_k]$ by any job in $C(I, l)$ (but could be by higher priority jobs not in $C(I, l)$). Hence J_k is the only job in $C(I, l)$ that is completed during $[r_k, C_k]$. Now we count the number of jobs in $C(I, l)$ that are completed during $I \setminus [r_k, C_k]$. Since *HDF* is 2-speed, *HDF* can complete at most $2|I| - 2F_k$ volume of work during $I \setminus [r_k, C_k]$. Since we assumed that all jobs have unit size, the number of such jobs is at most $\lfloor 2|I| - 2F_k \rfloor$. Hence, using this and by including J_k itself, we obtain

$$\lfloor 2|I| - 2F_k \rfloor + 1 \geq |C(I, l)|. \tag{2.5}$$

We now lower-bound $|N(\{v_{x,k}\}) \cap Y|$ to show (2.4). Roughly speaking, we want to show that OPT has many jobs of flow time at least F_k. Let $\mathcal{J}_{HDF}(I)$ be the set of jobs that are completed by *HDF* during I. Note that all jobs in $\mathcal{J}_{HDF}(I)$ must arrive during the interval I. For the sake of contradiction, suppose that this is not true, i.e., that there is a job J_j that arrives before the start of I and completes during I. Then *HDF* must be busy processing jobs of weight as high as J_j during $[r_j, C_j]$, contradicting the definition of the interval I being maximal. Consider the time at $e(I) + F_k$, where $e(I)$ is the ending time of the interval I. Since the volume of jobs in $\mathcal{J}_{HDF}(I)$ is $2|I|$ (recall that *HDF* has 2 speed) and OPT can process at most $|I| + F_k$ volume of work during $I \cup [e(I), e(I) + F_k]$, OPT must have at least $2|I| - (|I| + F_k) = |I| - F_k$ volume

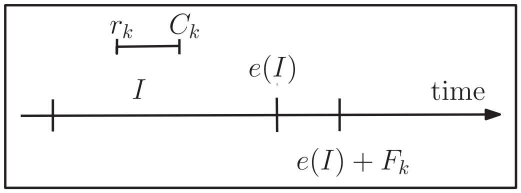

FIG. 2.2. *The interval I in HDF's schedule.*

of jobs in $\mathcal{J}_{HDF}(I)$ left at time $e(I) + F_k$; see Figure 2.2. Therefore if $|I| - F_k$ is an integer, OPT has at least $|I| - F_k + 1$ jobs in $\mathcal{J}_{HDF}(I)$ that have flow time at least F_k; here one extra job that is completed by OPT exactly at time $e(I) + F_k$ is counted. If $|I| - F_k$ is not integral, then OPT has at least $\lceil |I| - F_k \rceil$ jobs in $\mathcal{J}_{HDF}(I)$ that have flow time at least F_k. In both cases, we conclude that OPT has at least $\lfloor |I| - F_k \rfloor + 1$ jobs in $\mathcal{J}_{HDF}(I)$ that have flow time at least F_k. All such jobs have weight at least 2^l, since they are in $\mathcal{J}_{HDF}(I)$. Hence the vertices in Y corresponding to such jobs are neighbors of $v_{x,k}$, and we have

$$|N(\{v_{x,k}\}) \cap Y| \geq \lfloor |I| - F_k \rfloor + 1. \tag{2.6}$$

The inequalities (2.5) and (2.6) prove (2.4), and the lemma follows. ∎

Now we are ready to complete the proof of Lemma 2.3. For a subset $S \subseteq X$ let $N(S)$ denote the out-neighborhood S, and let $N(S, l) := N(S) \cap \mathcal{W}_l$. By Lemma 2.6 we have

$$\begin{aligned}
\sum_{v_{x,i} \in \mathcal{W}_l(X_s)} w_i &\leq |\mathcal{W}_l(X_s)| 2^{l+1} \leq 2|N(\mathcal{W}_l(X_s))| 2^{l+1} \quad \text{[by Lemma 2.6]} \\
&= 2 \sum_{h \geq l} |N(\mathcal{W}_l(X_s), h)| 2^{l+1} = 2 \sum_{h \geq l} \sum_{v_{y,j} \in N(\mathcal{W}_l(X_s), h)} 2^{l+1} \\
&= 4 \sum_{h \geq l} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{W}_l(X_s), h)} 2^h = 4 \sum_{h \geq l} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{W}_l(X_s)), h)} w_j.
\end{aligned}$$

Using this, we have that

$$\begin{aligned}
\sum_{v_{x,i} \in X_s} w_i &= \sum_l \sum_{v_{x,i} \in \mathcal{W}_l(X_s)} w_i \leq \sum_l 4 \sum_{h \geq l} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{W}_l(X_s), h)} w_j \\
&\leq 4 \sum_h \sum_{l \leq h} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(\mathcal{W}_l(X_s), h)} w_j \leq 4 \sum_h \sum_{l \leq h} \frac{1}{2^{h-l}} \sum_{v_{y,j} \in N(X_s, h)} w_j \\
&\leq 8 \sum_h \sum_{v_{y,j} \in N(X_s, h)} w_j \leq 8 \sum_{v_{y,j} \in Y_s} w_j.
\end{aligned}$$

This completes the proof of (2.3) and thus of Lemma 2.3.

3. Lower bounds. In this section we show that there is no scalable algorithm, there is no oblivious algorithm better than *HDF*, and the uniform cost functions are necessary to obtain $O(1)$-speed $O(1)$-competitiveness. All these lower bounds hold even for randomized algorithms.

THEOREM 3.1. *For any $\epsilon > 0$, no randomized online algorithm is constant competitive with speed $7/6 - \epsilon$ for the objective of $\sum_j g(F_j)$.*

Proof. We will rely on Yao's min-max principle to prove a lower bound on the competitive ratio of any randomized online algorithm [8]. The randomized instance is constructed as follows. Consider the cost function $g(F) = 2c$ for $F > 15$ and $g(F) = 0$ for $0 \leq F \leq 15$, where $c \geq 1$ is an arbitrary constant. The job instance is as follows:

- J_b: one big job of size 15 that arrives at time 0.
- S_1: a set of small jobs that arrive at time 10. Each job has size $\frac{35-30s}{c}$, and the total size of jobs in S_1 is 10.
- S_2: a set of small jobs that arrive at time 15. Each job has size $\frac{35-30s}{c}$, and the total size of jobs in S_2 is 10.

For simplicity, we assume that $\frac{10c}{35-30s}$ is an integer. The job J_b and the set S_1 of jobs arrive with probability 1, while the set S_2 of jobs arrives with probability $\frac{1}{2c}$. Let $\mathcal{E}$ denote the event that the set S_2 of jobs arrives.

Consider any deterministic algorithm A. We will consider two cases depending on whether A finishes J_b by time 15 or not. Note that A's scheduling decision concerning whether A completes J_b by time 15 or not does not depend on the jobs in S_2, since jobs in S_2 arrive at time 15. We first consider the case where A did not finish the big job J_b by time 15. Conditioned on $\neg\mathcal{E}$, A's cost is at least $2c$. Hence A has an expected cost at least $2c(1-\frac{1}{2c}) \geq c$. Now consider the case where A completes J_b by time 15. For this case, say the event $\mathcal{E}$ occurred. Let $V(S,t) := \sum_{j\in S} p^A(t)$ denote the remaining volume, under A's schedule, of all jobs in some set S at time t. Let $s = 7/6 - \epsilon$ be the speed that A is given, where $\epsilon > 0$ is a fixed constant. Since A spent $\frac{15}{s}$ amount of time during $[0, 15]$ working on J_b, A could have spent at most $15 - \frac{15}{s}$ time on jobs in S_1. Hence $V(S_1, 15) \geq 10 - s(15 - \frac{15}{s}) = 25 - 15s$ and $V(S_2, 15) = 10$. Since A can process at most $15s$ volume of work during $[15, 30]$, we have $V(S_1 \cup S_2, 30) \geq 35 - 30s = 30\epsilon$. Since each job in $S_1 \cup S_2$ has size $\frac{35-30s}{c}$, the number of jobs left is at least c. Since at time 30, each job has flow time at least 15, the algorithm A has total cost no smaller than $2c^2$. Recalling that $\Pr[\mathcal{E}] = \frac{1}{2c}$, A's expected cost is at least c.

Now let us look at the adversary's schedule. Conditioned on $\neg\mathcal{E}$, the adversary completes J_b first and all jobs in S_1 by time 25, thereby having no cost. Conditioned on $\mathcal{E}$, the adversary delays the big job J_b until it completes all jobs in S_1 and S_2 by time 20 and 30, respectively. Note in this schedule that each job in $S_1 \cup S_2$ has flow time at most 15. The adversary has cost $2c$ only for the big job. Hence the expected cost of the adversary is $\frac{1}{2c}(2c) = 1$. This, together with the above argument that A's expected cost is at least c, shows that the competitive ratio of any online algorithm is at least c. Since this holds for any constant c, the theorem follows. ∎

THEOREM 3.2. *For any $\epsilon > 0$, there is no oblivious randomized online algorithm that is $O(1)$-competitive for the objective of $\sum_j g(F_j)$ with speed augmentation $2 - \epsilon$.*

Proof. We appeal to Yao's min-max principle [8]. Let A be any deterministic online algorithm. Consider the cost function g such that $g(F) = 2c$ for $F > D$ and $g(F) = 0$ for $0 \leq F \leq D$. The constant D is hidden to A and is set to 1 with probability $\frac{1}{2c}$ and to $n + 1$ with probability $1 - \frac{1}{2c}$. Let $\mathcal{E}$ denote the event that $D = 1$. At time 0, one big job J_b of size $n + 1$ is released. At each integer time $1 \leq t \leq n$, one unit-sized job J_t is released. Here n is assumed to be sufficiently large such that $\epsilon(n + 1) - 1 > c$. Note that the event $\mathcal{E}$ has no effect on A's scheduling decision, since A is ignorant of the cost function.

Suppose that the online algorithm A finished the big job J_b by time $n+1$. Further, say that the event $\mathcal{E}$ occurs, that is, $D = 1$. Since $2n + 1$ volume of jobs in total were released and A can process at most $(2 - \epsilon)(n + 1)$ amount of work during $[0, n+1]$, A has at least $2n + 1 - (2 - \epsilon)(n + 1)$ volume of unit-sized jobs unfinished at time $n + 1$.

Each such unit-sized job has flow time greater than 1; hence A has total cost at least $2c(\epsilon(n+1)-1)) > 2c^2$. Knowing that $\Pr[\mathcal{E}] = \frac{1}{2c}$, A has an expected cost greater than c. Now suppose that A did not finish J_b by time $n+1$. Conditioned on $\neg\mathcal{E}$, A has cost at least $2c$. Hence A's expected cost is at least $2c(1-\frac{1}{2c}) > c$.

We now consider the adversary's schedule. Conditioned on $\mathcal{E}$ $(D=1)$, the adversary completes each unit-sized job within one unit time, hence has a nonzero cost only for J_b, and so has total cost $2c$. Conditioned on $\neg\mathcal{E}$ $(D=n+1)$, the adversary schedules jobs in a first-in-first-out fashion, thereby having cost 0. Hence the adversary's expected cost is $\frac{1}{2c}(2c) = 1$. The claim follows since A has cost greater than c in expectation. ∎

THEOREM 3.3. *There is no randomized online algorithm that is $O(1)$-speed $O(1)$-competitive for the objective of $\sum_j g_j(F_j)$.*

Proof. To show a lower bound on the competitive ratio of any randomized algorithm, we appeal to Yao's min-max principle [8] and construct a distribution on job instances for which any deterministic algorithm performs poorly compared to the optimal schedule. All cost functions g_i have a common structure. That is, each job J_i is completely defined by two quantities d_i and λ_i, which we call J_i's relative deadline and cost, respectively: $g_i(F_i) = 0$ for $0 \le F_i \le d_i$, and $g_i(F_i) = \lambda_i$ for $F_i > d_i$. Hence J_i incurs no cost if completed by time $r_i + d_i$, and cost λ_i otherwise. Recall that r_i and p_i are J_i's arrival time and size, respectively. For this reason, we will say that J_i has deadline $r_i + d_i$. For notational convenience, let us use a compact notation $(r_i, r_i + d_i, p_i, \lambda_i)$ to characterize all properties of each job J_i where p_i is J_i's size.

Let h, T, L be integers such that $h \ge 2s$, $T = 2^h$, $L > 2cT^2$. For each integer $0 \le l \le h = 2s$, there is a set $\mathcal{C}_l$ of jobs. (According to a distribution we will define soon, some jobs in $\mathcal{C}_l$ may or may not arrive.) All jobs have deadlines no greater than T. We first describe the set $\mathcal{C}_0$. In $\mathcal{C}_0$, all jobs have size 1 and relative deadline 1, and there is exactly one job that arrives at each unit time. The job with deadline t has cost L^t. More concretely, $\mathcal{C}_0 = \{(t-1, t, 1, L^t) \mid t \text{ is an integer such that } 1 \le t \le T\}$. Note that $|\mathcal{C}_0| = T$. We now describe the other sets of jobs $\mathcal{C}_l$ for each integer $1 \le l \le h$. All jobs in $\mathcal{C}_l$ have size 2^{l-1} and relative deadline 2^l, and at every 2^l time steps, exactly one job in $\mathcal{C}_l$ arrives. The job with deadline t has cost L^t. Formally, $\mathcal{C}_l = \{(2^l(j-1), 2^l j, 2^{l-1}, L^{2^l j}) \mid j \text{ is an integer such that } 1 \le j \le 2^{h-l}\}$. Note that $|\mathcal{C}_l| = 2^{h-l}$. Let $\mathcal{C} = \bigcup_{0 \le l \le h} \mathcal{C}_l$. Notice that all jobs with deadline t have cost L^t.

As we mentioned above, jobs in $\mathcal{C}$ do not arrive according to a probability distribution. To formally define such a distribution on job instances, let us group jobs depending on their arrival time. Let $\mathcal{R}_t$ denote the set of jobs in $\mathcal{C}$ that arrive at time t. Let $\mathcal{R}_{\le t} := \bigcup_{0 \le t' \le t} \mathcal{R}_{t'}$. We let $\mathcal{E}_t$, $0 \le t \le T-1$, denote the event that all jobs in $\mathcal{R}_{\le t}$ arrive and these are the only jobs that arrive. Let $\mathtt{Pr}[\mathcal{E}_t] = \frac{1}{L^t \theta}$, where $\theta = \sum_{0 \le j \le T-1} \frac{1}{L^j}$ is a normalization factor to ensure that $\sum_{0 \le t \le T-1} \mathtt{Pr}[\mathcal{E}_t] = 1$.

The following lemma will reveal a nice structure of the instance we created. Let $\mathcal{D}_t$ denote the set of jobs in $\mathcal{C}$ that have deadline t. Let $\mathcal{D}_{>t} := \bigcup_{t < t' \le T} \mathcal{D}_{t'}$.

LEMMA 3.4. *Consider the occurrence of event $\mathcal{E}_t$, $0 \le t \le T-1$. There exists a schedule with speed 1 that completes all jobs in $\mathcal{R}_{\le t} \cap \mathcal{D}_{>t}$ before their deadline. Further, such a schedule has cost at most $2TL^t$.*

Proof. We first argue that all jobs in $\mathcal{R}_{\le t} \cap \mathcal{D}_{>t}$ can be completed before their deadlines. Observe that there exists exactly one job in $\mathcal{C}_l \cap \mathcal{R}_{\le t} \cap \mathcal{D}_{>t}$ for each l. This is because the set of intervals $\{[2^l(j-1), 2^l j] \mid j \text{ is an integer s.t. } 1 \le j \le 2^{h-l}\}$ defined by the arrival time and deadline of jobs in $\mathcal{C}_l$ forms a partition of the time interval $[0, T]$. We schedule jobs in $\mathcal{R}_{\le t} \cap \mathcal{D}_{>t}$ in increasing sizes. Hence the first job we schedule is the job in $\mathcal{C}_0 \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$, and it has no choice other than being scheduled exactly during $[t, t+1]$. Now consider each job J_i in $\mathcal{C}_l \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$. It is not difficult to see that either $[2^l(j-1), 2^l(j-1) + 2^{l-1}]$ or $[2^l(j-1) + 2^{l-1}, 2^l j]$ is empty and therefore is ready to schedule the job J_i of size 2^{l-1} in $\mathcal{C}_l \cap \mathcal{R}_{\leq t} \cap \mathcal{D}_{>t}$. Finally, we upper-bound the cost of the above schedule. Since all jobs with deadlines greater than t are completed before their deadline under the schedule, each job can incur cost at most L^t. Knowing that there are at most $2T$ jobs, the total cost is at most $2TL^t$. ∎

COROLLARY 3.5. $\mathbb{E}[\mathrm{OPT}] \leq \frac{2T^2}{\theta}$.

Proof. Recall that $\Pr[\mathcal{E}_t] = \frac{1}{L^t\theta}$. By Lemma 3.4, we know, in case of the occurrence of event $\mathcal{E}$, that there exists a feasible schedule with speed 1 that results in cost at most $2TL^t$. Hence we have $\mathbb{E}[\mathrm{OPT}] \leq \sum_{0 \leq t < T} 2TL^t \frac{1}{L^t\theta} = \frac{2T^2}{\theta}$. ∎

We now show that any deterministic algorithm A performs much worse in expectation than the optimal schedule OPT.

LEMMA 3.6. *Any deterministic algorithm A given speed less than s has cost at least $\frac{L}{\theta}$ in expectation.*

Proof. Note that the total size of jobs in $\mathcal{C}_0$ is T, and the total size of jobs in each $\mathcal{C}_l$, $1 \leq l \leq h$, is $T/2$. Hence the total size of all jobs in $\mathcal{C}$ is at least $(h/2+1)T \geq (s+1)T$. The algorithm A, with speed s, cannot complete all jobs in $\mathcal{C}$ before their deadlines, since all jobs have arrival times and deadlines during $[0, T]$. Let J_i be a job in $\mathcal{D}_{t+1}$ that A fails to complete before its deadline for an integer $0 \leq t \leq T-1$. Note that J_i arrives no later than t since all jobs have size at least 1. Further, the decision concerning whether A completes J_i before its deadline or not has nothing to do with jobs in $\mathcal{R}_{t+1}$. Hence it must be the case that, for at least one of the events $\mathcal{E}_0, \ldots, \mathcal{E}_t$, A does not complete J_i by time $t+1$, which incurs an expected cost of at least $\frac{1}{L^t\theta} L^{t+1} \geq \frac{L}{\theta}$. ∎

By Yao's min-max principle, Corollary 3.5 and Lemma 3.6 show that the competitive ratio of any randomized algorithm is at least $\frac{L}{\theta} / \frac{2T^2}{\theta} = \frac{L}{2T^2} > c$. ∎

4. Analysis of *FIFO* for unit-size jobs. In this section we will show that *FIFO* is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing $\sum_{i\in[n]} g(F_i)$ when jobs have uniform sizes and unit weights. Without loss of generality, we can assume that all jobs have size 1, since jobs are allowed to arrive at arbitrary times. The proof follows similarly as in the case where jobs have unit size and arbitrary weight. Recall that in the previous section we charged the flow time of a job in the algorithm's schedule to jobs in the optimal solution's schedule that have larger flow time. In this case we can get a tighter bound on the number of jobs in the optimal solution's schedule that a job in *FIFO*'s schedule can charge to, which allows us to reduce the resource augmentation.

Consider an input sequence σ, and fix a constant $0 < \epsilon \leq \frac{1}{2}$. Let F_i denote the flow time of job J_i in *FIFO*'s schedule, and F_i^* be the flow time of J_i in OPT's schedule. Let $G = (V, E)$ be a flow network. There are source and sink vertices s and t, respectively. As before, there are two partite sets X and Y. There is a vertex $v_{x,i} \in X$ and a vertex $v_{y,i} \in Y$ corresponding to job J_i for all $i \in [n]$. There is an edge $(s, v_{x,i})$ with capacity 1 for all $i \in [n]$. There is an edge $(v_{y,i}, t)$ with capacity $\frac{4}{\epsilon^2}$ for all $i \in [n]$. There exists an edge $(v_{x,i}, v_{y,j})$ of capacity ∞ if $F_i \leq F_j^*$. The focus of this section is showing the following lemma.

LEMMA 4.1. *The maximum flow in G is n.*

Assuming that this lemma is true, then the following theorem can be shown.

THEOREM 4.2. *FIFO is $(1+\epsilon)$-speed $\frac{4}{\epsilon^2}$-competitive for minimizing $\sum_{i\in[n]} g(F_i)$*

when all jobs are unit sized.

Proof. Lemma 4.1 states that the maximum flow in G is n. Let f denote a maximum flow in G, and let $f(u,v)$ be the flow on an edge (u,v). Note that the maximum flow is achieved only when $f(s,v_{x,i}) = 1$ for all $i \in [n]$. We have that

$$\begin{aligned}
\mathit{FIFO} &= \sum_{i\in[n]} g(F_i) = \sum_{i\in[n]} f(s,v_{x,i}) g(F_i) \\
&= \sum_{i\in[n]}\sum_{j\in[n]} f(v_{x,i},v_{y,j}) g(F_i) \quad [f \text{ is conserved at } v_{x,i}] \\
&\le \sum_{i\in[n]}\sum_{j\in[n]} f(v_{x,i},v_{y,j}) g(F_j^*) \quad [v_{x,i},v_{y,j} \in E \text{ only if } F_i \le F_j^*] \\
&\le \sum_{j\in[n]} \frac{4}{\epsilon^2} g(F_j^*) \quad [f \text{ is conserved at } v_{y,j}, \text{ and the capacity of } v_{y,j}t \text{ is } \tfrac{4}{\epsilon^2}] \\
&= \frac{4}{\epsilon^2}\mathrm{OPT}. \qquad \square
\end{aligned}$$

Thus it only remains to prove Lemma 4.1. Clearly the min-cut value is at most n; thus we focus on lower-bounding the min-cut value. Let (S,T) be a minimum cut such that S contains the source s and T contains the sink t. To simplify the notation let $X_s = X \cap S$, $X_t = X \cap T$, $Y_s = Y \cap S$, and $Y_t = Y \cap T$. By definition each edge connecting s to a vertex in X_t is in (S,T), and the total capacity of these cut edges is $\sum_{v_{x,i}\in X_t} 1$. Knowing that each edge from a vertex in Y_s to t is in (S,T), it suffices to show that

$$\sum_{v_{y,j}\in Y_s} \frac{4}{\epsilon^2} \ge \sum_{v_{x,i}\in X_s} 1. \tag{4.1}$$

As in the proof of Lemma 2.3, Y_s is a subset of the out-neighborhood of the vertices in X_s since the edges connecting vertices in X and Y have capacity ∞. We now show a lemma similar to Lemma 2.6.

Lemma 4.3. *The vertices in X_s have at least $\frac{\epsilon^2}{4}|X_s|$ neighbors in Y; i.e., $|N(X_s)\cap Y| \ge \frac{\epsilon^2}{4}|X_s|$.*

Proof. Consider a maximal time interval I where *FIFO* is always busy scheduling jobs. Let J_k be the job (if it exists) in X_s that has arrived the earliest (and thus has highest priority in *FIFO*) out of all the jobs in X_s scheduled during I. Let $C(I)$ be the jobs in X_s scheduled by *FIFO* during I. We will show that $v_{x,k}$ has at least $\frac{\epsilon^2}{4}|C(I)|$ neighbors in Y such that for each such neighbor $v_{y,j}$, *FIFO* completed the corresponding job J_j during the interval I. By taking a union over all possible intervals I, we will have that the neighborhood of X_s has size at least $\frac{\epsilon^2}{4}|X_s|$.

Notice that *FIFO* does a $(1+\epsilon)(|I| - F_k)$ volume of work during $I \setminus [r_k, C_k]$ since *FIFO* is given $(1+\epsilon)$ speed and is busy during this interval. Knowing that jobs are unit-sized, *FIFO* completes at most $\lfloor (1+\epsilon)(|I| - F_k)\rfloor$ jobs during $I \setminus [r_k, C_k]$. The job J_k is the only job in $C(I)$ scheduled during $[r_k, C_k]$ because J_k has the highest priority in *FIFO*'s schedule of the jobs in $C(I)$. This implies that $|C(I)| \le \lfloor (1+\epsilon)(|I| - F_k)\rfloor + 1$. *FIFO* completes a volume of $(1+\epsilon)|I|$ work during I. Further, every job *FIFO* completes during I arrived during I since *FIFO* was not busy before I and *FIFO* scheduled these jobs during I. Let $e(I)$ denote the ending time point of I, and $\mathcal{J}_{FIFO}(I)$ be the jobs completed by *FIFO* during I. The previous argument implies

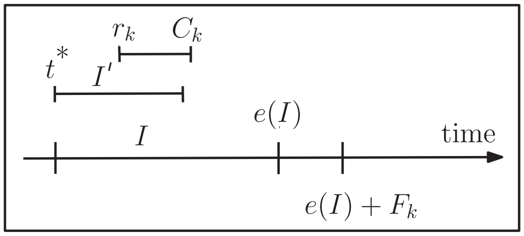

FIG. 4.1. *The intervals I and I' in FIFO's schedule.*

that at least a $(1+\epsilon)|I| - |I| - F_k = \epsilon|I| - F_k$ volume of work corresponding to jobs in $\mathcal{J}_{FIFO}(I)$ remains in OPT's queue at time $e(I) + F_k$ since OPT has 1 speed. If $\epsilon|I| - F_k$ is integral, then at least $\epsilon|I| - F_k + 1$ jobs in $\mathcal{J}_{FIFO}(I)$ have flow time at least F_k in OPT's schedule; here there is one job that could be completed exactly at time $e(I) + F_k$ that is counted. Otherwise, OPT has $\lceil \epsilon|I| - F_k \rceil = \lfloor \epsilon|I| - F_k \rfloor + 1$ jobs in $\mathcal{J}_{FIFO}(I)$ that have flow time at least F_k. In either case, at least $\lfloor \epsilon|I| - F_k \rfloor + 1$ jobs in $\mathcal{J}_{FIFO}(I)$ have flow time at least F_k in OPT's schedule.

First consider the case where $F_k \le \frac{\epsilon}{2}|I|$. In this case at least $\lfloor \epsilon|I| - F_k \rfloor + 1 \ge \lfloor \frac{\epsilon}{2}|I| \rfloor + 1$ jobs wait at least F_k time in OPT. Knowing that $|C(I)| \le \lfloor (1+\epsilon)(|I| - F_k) \rfloor + 1 \le \lfloor (1+\epsilon)|I| \rfloor + 1$, the neighborhood of $v_{x,k}$ contains at least $\lfloor \frac{\epsilon}{2}|I| \rfloor + 1 \ge \frac{\epsilon}{2(1+\epsilon)} \lfloor (1+\epsilon)|I| \rfloor - \frac{\epsilon}{2(1+\epsilon)} + 1 \ge \frac{\epsilon}{2(1+\epsilon)}(\lfloor (1+\epsilon)|I| \rfloor + 1) \ge \frac{\epsilon}{2(1+\epsilon)}|C(I)| \ge \frac{\epsilon^2}{2}|C(I)|$ nodes. The last inequality follows from $\epsilon < 1/2$.

Let us consider the other case that $F_k > \frac{\epsilon}{2}|I|$. Let t^* be the earliest time before C_k such that *FIFO* only schedules jobs that arrived no later than r_k during $[t^*, C_k]$. Equivalently, t^* is the beginning of the interval I by definition of *FIFO*. Notice that $t^* \le r_k$. Let $I' = [t^*, C_k]$. We know that *FIFO* completes a $(1+\epsilon)|I'|$ volume of work during $|I'|$. Let $\mathcal{J}_{FIFO}(I')$ denote the jobs completed by *FIFO* during I'. Any job in $\mathcal{J}_{FIFO}(I')$ arrives after t^* because *FIFO* was not scheduling a job before t^* by definition of I'. Note that any job in $\mathcal{J}_{FIFO}(I')$ will have flow time at least F_k if it is not satisfied until time C_k, since the jobs arrived no later than r_k. See Figure 4.1. Therefore, at least a $(1+\epsilon)|I'| - |I'| = \epsilon|I'| \ge \epsilon F_k > \frac{\epsilon^2}{2}|I|$ volume of work corresponding to jobs in $\mathcal{J}_{FIFO}(I')$ remains unsatisfied in OPT's schedule at time C_k because OPT has unit speed and it was assumed that $F_k > \frac{\epsilon}{2}|I|$. Thus at least $\lceil \frac{\epsilon^2}{2}|I| \rceil$ jobs in $\mathcal{J}_{FIFO}(I')$ have flow time at least F_k in OPT. We also know that $|C(I)| \le \lfloor (1+\epsilon)(|I| - F_k) \rfloor + 1 \le (1+\epsilon)|I|$ since $F_k \ge \frac{1}{1+\epsilon}$. Together this shows that $v_{x,k}$ has at least $\frac{\epsilon^2}{2(1+\epsilon)}|C(I)| \ge \frac{\epsilon^2}{4}|C(I)|$ neighbors in Y knowing that $\epsilon \le \frac{1}{2}$. ∎

Using Lemma 4.3, we can complete the proof of Lemma 4.1. By Lemma 4.3 we have $|X_s| \le \frac{4}{\epsilon^2}|N(X_s) \cap Y| \le \frac{4}{\epsilon^2}|Y_s|$, which implies (4.1) and Lemma 4.1.

5. Analysis of *WLAPS* for concave functions.

In this section we consider the objective function $\sum_{i \in [n]} w_i g(F_i)$, where w_i is a positive weight corresponding to job J_i and $g : \mathbb{R}_{\ge 0} \to \mathbb{R}_{\ge 0}$ is a twice-differentiable, nondecreasing, concave function. We let g' and g'' denote the derivative of g and the second derivative function of g, respectively. For this objective we will show a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm that is *nonclairvoyant*. The algorithm that we consider is a generalization of the algorithm *WLAPS* [12, 13, 11].

Consider any job sequence σ, and let $0 < \epsilon \le 1/3$ be fixed. Without loss of generality it is assumed that each job has a distinct arrival time. We assume that

WLAPS is given $(1+3\epsilon)$-speed. At any time t, let $A(t)$ denote the jobs in *WLAPS*'s queue. The algorithm at each time t finds the set of the most recently arriving jobs $A'(t) \subseteq A(t)$ such that $\sum_{J_i \in A'(t)} w_i g'(t-r_i) = \epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. The algorithm *WLAPS* distributes the processing power among the jobs in $A'(t)$ according to their current increase in the objective. That is, $J_j \in A'(t)$ receives processing power $(1+3\epsilon) w_j g'(t-r_j)/(\epsilon \sum_{J_i \in A'(t)} w_i g'(t-r_i))$.

In case where there does not exist such a set $A'(t)$ such that the sum of $w_i g'(t-r_i)$ over all jobs $A'(t)$ is exactly $\epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$, we make the following small change. Let $A'(t)$ be the smallest set of the most recently arriving jobs such that the sum of $w_i g'(t-r_i)$ over all jobs in $A'(t)$ is no smaller than $\epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. We let the job J_k, which arrives the earliest in $A'(t)$, receive processing power $\sum_{J_i \in A'(t)} w_i g'(t-r_i) - \epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. For simplicity, throughout the analysis, we will assume that there exists such a set $A'(t)$ of the most recently arriving jobs such that $\sum_{J_i \in A'(t)} w_i g'(t-r_i) = \epsilon \sum_{J_i \in A(t)} w_i g'(t-r_i)$. This is done to make the analysis more readable and the main ideas transparent.

To prove the competitiveness of *WLAPS* we define the following potential function. (For a survey on potential functions for scheduling problems, see [14].) For a job J_i let $p_i^A(t)$ be the remaining size of job J_i in the *WLAPS* schedule at time t, and let $p_i^O(t)$ be the remaining size of job J_i in OPT's schedule at time t. Let $z_i(t) = \max\{p_i^A(t) - p_i^O(t), 0\}$. The potential function is

$$\Phi(t) = \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t-a_i) \sum_{J_j \in A(t), r_j \geq r_i} z_j(t).$$

We will look into noncontinuous changes of $\Phi(t)$ that occur due to job arrivals and completions, and continuous changes of $\Phi(t)$ that occur due to *WLAPS*'s processing, OPT's processing, and time elapsed. We will aggregate all these changes later.

Job arrival. Consider when job J_k arrives at time t. There the change in the potential function is $\frac{1}{\epsilon} w_k g'(0) z_k(r_k) + \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g'(t-a_i) z_k(r_k)$. When job J_i arrives, $z_i(r_i) = 0$, so there is no change in the potential function.

Job completion. The optimal solution completing a job has no effect on the potential function. When the algorithm completes a job J_i at time t, some terms may disappear from $\Phi(t)$. In this case, the potential function can only decrease, since all terms in $\Phi(t)$ are nonnegative.

Continuous change. We now consider the continuous changes in the potential function at time t. These include changes due to time elapsed and changes in the z variable due to OPT and *WLAPS* processing jobs. First consider the change due to time. This is equal to

$$\frac{\mathrm{d}}{\mathrm{dt}} \Phi(t) = \frac{1}{\epsilon} \sum_{J_i \in A(t)} w_i g''(t-a_i) \sum_{J_j \in A(t), r_j \geq r_i} z_j(t).$$

We know that w_i and $z_i(t)$ are positive for all jobs $J_i \in A(t)$. Further, g'' is always nonpositive since g is concave. Therefore, time changing can only decrease the potential.

Now consider the change due to OPT's processing. It can be seen that the most OPT can increase the potential function is by working exclusively on the job which has the latest arrival time. In this case, for any job $J_i \in A(t)$ the variable $\sum_{J_j \in A(t), r_j \geq r_i} z_j(t)$ changes at rate 1 because OPT has speed 1. The increase in the potential due to OPT's processing is at most

$$\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \le \frac{1}{\epsilon}\sum_{J_i\in A(t)} w_i g'(t-a_i).$$

Now consider the change in the potential function due to the algorithm's processing. The algorithm decreases the z variable and therefore can only decrease the potential function. Recall that a job $J_j \in A'(t)$ is processed by *WLAPS* at a rate of $(1+3\epsilon)w_j g'(t-r_j)/(\sum_{J_i\in A'(t)} w_i g'(t-r_i))$ because *WLAPS* is given $(1+3\epsilon)$-speed. Therefore, for each job $J_j \in A'(t)\setminus O(t)$ the variable z_j decreases at a rate of $(1+3\epsilon)w_j g'(t-r_j)/(\sum_{J_i\in A'(t)} w_i g'(t-r_i))$. Hence we can bound the change in the potential as

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) &\le -\frac{1}{\epsilon}\sum_{J_i\in A(t)\setminus A'(t)} w_i g'(t-a_i) \sum_{J_j\in A'(t)\setminus O(t)} \frac{(1+3\epsilon)w_j g'(t-r_j)}{\sum_{J_k\in A'(t)} w_k g'(t-r_k)}\\
&\le -\frac{1-\epsilon}{\epsilon}\sum_{J_i\in A(t)} w_i g'(t-a_i) \sum_{J_j\in A'(t)\setminus O(t)} \frac{(1+3\epsilon)w_j g'(t-r_j)}{\sum_{J_k\in A'(t)} w_k g'(t-r_k)}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\text{by definition of } A'(t)]\\
&= -\frac{1-\epsilon}{\epsilon^2}\sum_{J_j\in A'(t)\setminus O(t)} (1+3\epsilon)w_j g'(t-r_j)\\
&\le -\frac{1+\epsilon}{\epsilon^2}\sum_{J_j\in A'(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2}\sum_{J_j\in O(t)} w_j g'(t-r_j) \qquad [\text{since } 0<\epsilon\le 1/3]\\
&\le -\frac{1+\epsilon}{\epsilon}\sum_{J_j\in A(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2}\sum_{J_j\in O(t)} w_j g'(t-r_j). \qquad [\text{by definition of } A'(t)]
\end{aligned}$$

By combining the changes due to OPT and the algorithm's processing and the change due to time, we determine the continuous change in the potential function to be at most

$$\begin{aligned}
&\frac{1}{\epsilon}\sum_{J_i\in A(t)} w_i g'(t-a_i) - \frac{1+\epsilon}{\epsilon}\sum_{J_j\in A(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2}\sum_{J_j\in O(t)} w_j g'(t-r_j)\\
&\quad = -\sum_{J_j\in A(t)} w_j g'(t-r_j) + \frac{2}{\epsilon^2}\sum_{J_j\in O(t)} w_j g'(t-r_j).
\end{aligned}$$

Completing the analysis. At this point we are ready to complete the analysis. We know that $\Phi(0)=\Phi(\infty)=0$ by definition of Φ, which implies that the total sum of noncontinuous changes and continuous changes of $\Phi(t)$ is 0. Further, there are no increases in Φ for noncontinuous changes. Hence, we have $\int_{t=0}^{\infty}\frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \ge 0$. Let *WLAPS* denote the algorithm's final objective, and OPT denote the optimal solution's final objective. Let $\frac{\mathrm{d}}{\mathrm{dt}}\mathit{WLAPS}(t) = \sum_{J_j\in A(t)} w_j g'(t-r_j)$ denote the increase in *WLAPS* objective at time t, and let $\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) = \sum_{J_j\in O(t)} w_j g'(t-r_j)$ denote the increase in OPT's objective at time t. We have that

$$
\begin{aligned}
WLAPS &= \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}} WLAPS(t) \\
&\leq \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}} WLAPS(t) + \frac{\mathrm{d}}{\mathrm{dt}}\Phi(t) \\
&\leq \int_{t=0}^{\infty} \frac{\mathrm{d}}{\mathrm{dt}} WLAPS(t) - \frac{\mathrm{d}}{\mathrm{dt}} WLAPS(t) + \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{OPT}(t) \\
&\leq \frac{2}{\epsilon^2}\mathrm{OPT}.
\end{aligned}
$$

This proves the following theorem.

THEOREM 5.1. *The algorithm WLAPS is* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^2})$*-competitive for minimizing* $\sum_{i\in[n]} w_i g(F_i)$ *when* $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *is a concave nondecreasing positive function that is twice differentiable.*

6. Conclusions and discussion. One obvious question is whether there exists an online algorithm that is $O(1)$-competitive with speed less than two. To obtain such an algorithm (if one exists), one must exploit the structure of cost functions. Our analysis can be extended to show that there exists an $O(1)$-speed $O(1)$-competitive algorithm on identical parallel machines.

Acknowledgment. We thank Ravishankar Krishnaswamy for extensive and valuable discussions throughout the development of these results.

REFERENCES

[1] N. Bansal, R. Krishnaswamy, and V. Nagarajan, *Better scalable algorithms for broadcast scheduling*, in Proceedings of the 37th International Colloquium on Automata, Languages, and Programming (ICALP), Bordeaux, France, 2010, pp. 324–335.

[2] N. Bansal and K. Pruhs, *The geometry of scheduling*, in Proceedings of the IEEE Symposium on the Foundations of Computer Science, 2010, pp. 407–414.

[3] N. Bansal and K. R. Pruhs, *Server scheduling to balance priorities, fairness, and average quality of service*, SIAM J. Comput., 39 (2010), pp. 3311–3335.

[4] C. Barnhart, D. Bertsimas, C. Caramanis, and D. Fearing, *Equitable and efficient coordination in traffic flow management*, Transport. Sci., 46 (2012).

[5] E. Bayraktar and S. Dayanik, *Poisson disorder problem with exponential penalty for delay*, Math. Oper. Res., 31 (2006), pp. 217–233.

[6] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, *Online weighted flow time and deadline scheduling*, J. Discrete Algorithms, 4 (2006), pp. 339–352.

[7] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, *Approximation algorithms for average stretch scheduling*, J. Scheduling, 7 (2004), pp. 195–222.

[8] A. Borodin and R. El-Yaniv, *On randomization in online computation*, in Proceedings of the IEEE Conference on Computational Complexity, 1997, pp. 226–238.

[9] J. S. Chadha, N. Garg, A. Kumar, and V. N. Muralidhara, *A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation*, in Proceedings of the 41st ACM Symposium on Theory of Computing, 2009, pp. 679–684.

[10] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania, *Preemptive scheduling in overloaded systems*, J. Comput. System Sci., 67 (2003), pp. 183–197.

[11] J. Edmonds, S. Im, and B. Moseley, *Online scalable scheduling for the ℓ_k-norms of flow time without conservation of work*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2011, pp. 109–119.

[12] J. Edmonds and K. Pruhs, *Scalably scheduling processes with arbitrary speedup curves*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2009, pp. 685–692.

[13] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs, *Scheduling jobs with varying parallelizability to reduce variance*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, 2010, pp. 11–20.

[14] S. Im, B. Moseley, and K. Pruhs, *A tutorial on amortized local competitiveness in online scheduling*, SIGACT News, 42 (2011), pp. 83–97.

[15] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, J. ACM, 47 (2000), pp. 617–643.

[16] K. Pruhs, J. Sgall, and E. Torng, *Online scheduling*, in Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, Boca Raton, FL, 2004, Chapter 15.