Online Non-Preemptive Story Scheduling in Web Advertising

Tie-Yan Liu
Microsoft Research
Beijing, China
tyliu@microsoft.com

Weidong Ma
Microsoft Research
Beijing, China
weima@microsoft.com

Tao Qin
Microsoft Research
Beijing, China
taoqin@microsoft.com

Pingzhong Tang
Tsinghua University
Beijing, China
kenshinping@gmail.com

Guang Yang
Tsinghua University &
Institute of Computing
Technology, CAS
Beijing, China
guang.research@gmail.com

Bo Zheng
Tsinghua University
Beijing, China
zhengb06science@gmail.com

ABSTRACT

This paper is concerned with online story scheduling, motivated by storyboarding in online advertising. In storyboarding, triggered by the browsing history of a user, advertisers arrive online and wish to present a sequence of ads (stories) on the website. The user ceases to browse with probability $1-\beta$ at each time step. Once the user finishes watching an ad, the advertiser derives a reward. The goal of the website is to determine a schedule that maximizes the expected total reward. This problem was first introduced by Dasgupta et al.(SODA'09) [7], and then improved by Alberts and Passen (ICALP'13) [4]. In this paper, we abandon the preemptive assumption in [7] and [4], and consider a more realistic setting: online non-preemptive story scheduling, i.e., a running job (correspond to advertiser' story) *cannot* be preempted even if another job leads to a higher reward.

Specifically, we study the setting where only 1-lengthed and k-lengthed ads are allowed. We first present a greedy algorithm which achieves a competitive ratio of β^{k-1}, and prove that this ratio is optimal for deterministic algorithms. Then, we propose a randomized algorithm with a competitive ratio of $\frac{1}{k+1}$ for general β, and then show that no randomized algorithm can achieve a competitive ratio better than $(1+\frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$.

1. INTRODUCTION

Online advertising [11, 13, 14, 22] is very popular in numerous industry sectors. It has even become the major source of revenue for several distinguished Internet companies [6]. As an advanced form of online ads, storyboarding was first launched in New York Times Digital [24] and has been gaining increasing attention recently.

Storyboarding is also referred to as sequence advertising or surround sessions [24]. In storyboarding, triggered by user's browsing history in a website, advertisers arrive online and wish to present a sequence of ads to the user. From one time to another, there is only one selected advertiser who owns the right to use a sequence of consecutive pages for advertising purpose. To convey his/her message, the advertiser can use these pages to create a story line of a given product.

The algorithmic approach to achieve storyboarding is referred to as story scheduling, which was first introduced by Dasgupta et al.(SODA'09) [7]. In their setting, stories (jobs) arrive online, and value is obtained for each scheduled unit. Jobs can be preempted, however, no further value can be derived from the remaining unscheduled units. Online algorithms were proposed in [7] and [4] for story scheduling, whose total rewards are competitive against that of the optimal offline scheduler with the knowledge of all jobs.

Different from the setting in [7] and [4], in this paper, we consider the more realistic non-preemption setting. The preemptive setting is not realistic for several reasons. First, user experience is bad if an ad (e.g., a video ad) is interrupted while it is being watched by a user. Second, it is troublesome to evaluate the reward of an interrupted ad, and thus is difficult for the website to charge fees for the advertiser. In our setting, once a job is scheduled, other arriving jobs must wait until that job is completed. No job (even with much higher value) can preempt an on-going job. For this new setting, we design two effective scheduling algorithms and conduct competitive analysis on them.

1.1 Model and Problem Formulation

Time is discretized throughout this paper. Consider a user who starts visiting a website at time $t=0$. At each time step, the user continues surfing with probability β, where $0<\beta<1$, and stops surfing with probability $1-\beta$. Jobs (stories) arrive online, and each job i is characterized by a vector $\langle a_i, l_i, v_i\rangle$, where a_i is the arrival time of the job, l_i is the length of the job (i.e., the length of story the advertiser would like to display), and v_i is the per-unit value[1] of the job. Note that the user may cease with probability $1-\beta$ at each step, so the real (expected) reward should be discounted by the show time of the job. For example, a job with length

[1]In this paper, we do not restrict the range of per-unit value, i.e., the per-unit value of a job can be arbitrary large.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

2 will get expected reward of $v_i + v_i\beta$ if shown at time 0 and of $v_i\beta^2 + v_i\beta^3$ if at time 2. As a result, the expected reward of a job that is scheduled at time t can be written as $v_i \sum_{j=t}^{t+l_i-1} \beta^j = v_i \cdot \beta^t \cdot \frac{1-\beta^{l_i}}{1-\beta}$.

In practice, many websites impose certain constraints on the lengths of the stories. In this paper, we assume the website allows either 1-lengthed ad or k-lengthed ($k \geq 2$) ad, for simplicity and without loss of much generality. Actually, we can regard the 1-lengthed ad as a single shot ad, and the k-lengthed ad as a storyline of ads (or ad sequence) [14].

We study the *no-deadline* model, i.e., a job i can be scheduled at any time after its arrival (the time horizon is infinite). For a certain job scenario $\mathcal{S}$, a schedule $\mathcal{A}$ specifies which job to execute at each time $t \geq 0$. Note that preemption of jobs is not allowed. Therefore, the expected reward $R_{\mathcal{A}}(\mathcal{S})$ can be written $\sum_{t=0}^{\infty} \beta^t \cdot v(t)$, where $v(t)$ is the per-unit value of the job scheduled at time t. The goal of our algorithm design is to maximize the expected reward.

Similar to [7], we use competitive ratio [19] to evaluate the performance of an online scheduling algorithm. We say that an algorithm (scheduler) $\mathcal{A}$ is c-competitive (has a competitive ratio of c), if $R_{\mathcal{A}}(\mathcal{S}) \geq c \cdot R_{opt}(\mathcal{S})$ holds for any scenario $\mathcal{S}$, where $0 < c \leq 1$ and *opt* is the optimal offline schedule which has full information about all the future jobs of scenario $\mathcal{S}$.

1.2 Our Work

For deterministic algorithms, we observe that the best competitive ratio one can hope is β^{k-1}. We design a greedy algorithm $\mathcal{A}_1$ (higher per-unit value first) in Section 3 and prove that it achieves this optimal ratio (see Theorem 1).

Note that although the greedy algorithm is natural, it remains a great challenge to analyze its competitive ratio. The challenge comes from several aspects:

(1) Delayed Effect: It is possible that the scheduling time of a job under the greedy algorithm be much later[2] than that under the optimal algorithm. See Example 1 for some intuitive sense . Therefore, there does not exist such a simple "mapping", by which the optimal value obtained from any time unit can be "mapped" to either the same time unit or one of the next $k-1$ ones under the greedy algorithm.

EXAMPLE 1. *When a high-value job a_2 arrives, there is a low-value long job a_1 being scheduled. According to the non-preemption rule, job a_2 has to wait until a_1 is completed. Note that just after the time that a_1 is completed, jobs (denote as set C_1) with higher values (higher than that of a_2) might arrive one by one. Thus, under the greedy algorithm, a_2 would be delayed again and again. However, under the optimal offline algorithm, it is possible that a_2 is scheduled at the time it just arrives, as the optimal offline algorithm is clairvoyant to schedule the low-value long job a_1 somewhere else.*

(2) Nested Effect: The delayed effect may be nested. Take the job set C_1 mentioned above for example. It is possible that there is a job in C_1 encountering a similar situation as a_2. Formally, $a_{j+1} \in C_1$ has higher value than $a_j \in C_1$, but a_{j+1} arrives later than a_j. Then a_{j+1} may be delayed by a_j. Right after a_j is completed, jobs (denoted by C_2, where $C_2 \subset C_1$) with higher values (than a_{j+1}) could arrive one by one. Thus, the greedy algorithm would delay a_2 further, until all jobs in C_2 are accomplished. However, since the time horizon is infinite and the value of each job is unbounded, the adversary is able to design an unlimited number of such nesting. Clearly, the worst-case analysis would become significantly involved and hard to follow when the nesting is very deep.

(3) Chain Effect: Due to non-preemption rule, a job has to wait until the current job is completed. So a low-value long job may delay a high-value job. The chain effect happens when the delayed high-value job itself is a long job, so this long job may delay a subsequent higher-value job. Formally, let $a_1, \ldots, a_n$ be long jobs, such that for every $j = 1, \ldots, n-1$, a_j arrives just before a_{j+1} but the value of a_{j+1} is greater than that of a_j. As the time horizon is infinite and the value of a job is not upper bounded, this may cause a lot of delay and utility loss as beta goes down. It is not easy to estimate the accumulated delay cost compared with the optimal offline algorithm, let alone the chain effect may be deeply entwined with the nested effect.

Given this situation, a carefully designed competitive analysis is needed. Actually, we obtain a stronger result as we use the offline optimal preemptive allocation as the benchmark, which has a larger expected reward than the non-preemptive optimal allocation, and show the greedy algorithm can be β^{k-1} competitive regards to this benchmark.

We first introduce Lemma 2, which is universally applicable in the discounted time horizon scheduling. In brief, Lemma 2 presents a new algorithm $\mathcal{A}_d$, which is the same as Greedy Algorithm $\mathcal{A}_1$ except that it has to schedule jobs from time $k-1$ on. Lemma 2 says that the expected reward obtained by $\mathcal{A}_d$ is at least β^{k-1} times that obtained by $\mathcal{A}_1$.

At first sight, Lemma 2 may be misunderstood as trivial, because a careless thought may arise: *$\mathcal{A}_d$ schedules jobs in the same order as $\mathcal{A}_1$, except for that all jobs are scheduled $k-1$ time units later.*

This thought misses the important restriction that $\mathcal{A}_d$ itself is a greedy algorithm. For example, a short job 1 arrives at time 0 and a long job 2 (with higher per-unit value than job 1) arrives at time 1. Clearly, $\mathcal{A}_1$ will schedule job 1 at time 0 and schedule job 2 at time 1. Note that $\mathcal{A}_d$ will schedule job 2 at time $k-1$ and schedule job 1 at time unit $2k-1$, rather than *schedule job 1 at time $k-1$ and schedule job 2 at time k*. So, the order of scheduling jobs in $\mathcal{A}_d$ can be very different from that in $\mathcal{A}_1$. Therefore, the β^{k-1} approximation can not be derived directly. In fact, the proof is technical and involved.

Lemma 2 works as a useful tool to prove Theorem 1.

Then we prove the β^{k-1} upper bound for the competitive ratio of any online deterministic algorithm (see Theorem 2). In this way, we can come to the conclusion that the greedy algorithm is an optimal deterministic algorithm.

We then consider randomized algorithms. The motivation is that the deterministic greedy algorithm may perform poorly when β is extremely small or k is extremely large. To handle this issue, we design a randomized algorithm $\mathcal{A}_2$ in Section 4 and prove its competitive ratio to be $\frac{1}{k+1}$ (see Theorem 3). The key idea behind $\mathcal{A}_2$ is to ensure that each job has the chance of exactly $\frac{1}{k+1}$ to be scheduled at a unique time t, which estimates the earliest time of scheduling this job under *opt*.

[2]Later than $k-1$ time units.

We further derive an upper bound on the competitive ratio for any online randomized algorithms (see Theorem 4), which is $(1+\frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$, through classical adversary analysis. Note that we allow the adversary to use randomized strategy.[3]

The rest of this paper is organized as follows. In Section 2, we introduce several related works. In Section 3, we present the greedy algorithm and an upper bound for all deterministic algorithms. Our randomized algorithm and the upper bound for randomized algorithms are given in Section 4 and Section 5, respectively. Finally, we conclude our paper in Section 6.

2. RELATED WORK

Motivated by applications in web advertising, several variants and generalizations of online advertising have been studied recently [1, 2, 3, 11, 13, 14, 22]. As far as we know, storyboarding as an important form of advertising has only been studied by [4, 7]. Dasgupta et al. [7] give a deterministic $\frac{1}{7}$-competitive algorithm for the storyboarding problem, and show that no deterministic online algorithm can achieve a competitive ratio better than $\frac{1}{2}$, for general β. Alberts and Passen [4] improve the results by giving a $\frac{1}{1+\phi}$-competitive algorithm, where $\phi = (1+\sqrt{5})/2$ is the Golden Ratio. The main difference between their work and ours is that they consider a preemptive setting.

There is a long research line regarding of online scheduling problems (see [18, 21, 28] for a survey). However, the traditional scheduling problem does not consider a discount time horizon and usually specify a deadline for each job, which is the main difference from our work. Zhang et al. [29] study a model in which the job value discounts with the delay: the duration between its arrival time and schedule time. This model is most close to ours, but not identical, because in our model the value of every job discounts with the absolute time, regardless of what time the job arrives.

There is also a rich literature which is concerned with non-preemptive scheduling [20, 15, 16, 9, 8, 10]. In these works, all jobs have deadlines and the job values are not discounted with the time. By assuming the so called *proportional values* (i.e., the value of each job is proportional to the length), in [15], a tight upper and lower bound of 2 are given for the deterministic competitiveness when all jobs have equal lengths, and a $6(\lfloor \log_2 \kappa \rfloor + 1)$-competitive randomized algorithm is provided when jobs' values are restricted in $[1, k]$, matching the $\Omega(\log k)$ lower bound [20] within a constant factor.

Recently, the online scheduling problem has attracted the attention of computational economics community, and many results are obtained from a viewpoint of mechanism design [5, 12, 17, 19, 23, 25, 27]. These models differ from ours in multiple ways, most notably in not having a discounted time horizon.

3. DETERMINISTIC ALGORITHM

The main challenge of designing online algorithms for the non-preemptive scheduling problem is the competition of a current low-value job and a possible future high-value job: As the online algorithm has no information about future jobs, when it is executing a long job (with length k), it is possible that a job with much higher value arrives. Thus the more valuable job has to be delayed and incurs a loss. The loss may be extremely large when the value of a job is not upper bounded.

In this section we propose a greedy algorithm and analyze its competitive ratio. We prove that the algorithm is an optimal deterministic algorithm in terms of competitive ratio by charactering an upper bound on the competitive ratio of any deterministic algorithm.

3.1 The Greedy Algorithm

The greedy algorithm is described in Algorithm 1. Here, a time point t is called *available* if no job is being processed at t.

Algorithm 1 The Greedy Algorithm $\mathcal{A}_1$

1: **for** each available time $t \geq 0$ **do**
2: Schedule the job with the highest per-unit value.
3: **end for**

From now on, we will prove the competitive ratio of $\mathcal{A}_1$, which is shown in Theorem 1.

THEOREM 1. *The greedy algorithm $\mathcal{A}_1$ has a competitive ratio of β^{k-1}.*

As aforementioned, let *opt* denote the optimal offline scheduling when the full information of future jobs is known in advance. Furthermore, let *opt** denote the optimal offline scheduling when the full information of future jobs is known in advance and preemption is allowed. Obviously, *opt** always leads to a reward no less than *opt*. To analyze the competitive ratio of $\mathcal{A}_1$, in principle, we need to compare with *opt*. Here, instead of comparing with *opt* directly, we achieve a slightly stronger result since we compare with a stronger benchmark *opt**. In the proof, we will show that $\mathcal{A}_1$ obtains a competitive ratio of at least β^{k-1} even if the benchmark is *opt** instead of *opt*.

To make the proof more readable, we introduce the following lemma.

LEMMA 2. *For any job set $\mathcal{S}$, suppose w.l.o.g. that the earliest arrival time of jobs in $\mathcal{S}$ is 0. We define a new algorithm $\mathcal{A}_d$ which is the same as Greedy Algorithm $\mathcal{A}_1$ except that it has to schedule the jobs from time $k-1$ on. We claim that the expected reward obtained by $\mathcal{A}_d$ is at least β^{k-1} times that obtained by $\mathcal{A}_1$.*

PROOF. We use induction to prove this lemma. If there is only one job, the theorem holds trivially. Now we only need to prove the case of n jobs given that the theorem holds for less than n jobs. First, we introduce the definition of *conformity*.

DEFINITION 1. *We say a job sequence scheduled by $\mathcal{A}_d$ during the period $[t_1+k-1, t_2+k-1]$ is in* conformity *with the job sequence scheduled by $\mathcal{A}_1$ during the period $[t_1, t_2]$, if the job scheduled by $\mathcal{A}_d$ at time $t+k-1$ is the same as the job scheduled by $\mathcal{A}_1$ at time t, for any time $t \in [t_1, t_2]$.*

If the job sequence scheduled by $\mathcal{A}_d$ is in conformity with $\mathcal{A}_1$ all the time, then the reward obtained by $\mathcal{A}_d$ is β^{k-1}

[3]There is still a gap between the competitive ratio of $\mathcal{A}_2$ and the upper bound, since $(1+\frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$ is always greater than $\frac{1}{2}$ when $k \geq 2$ and $0 < \beta < 1$.

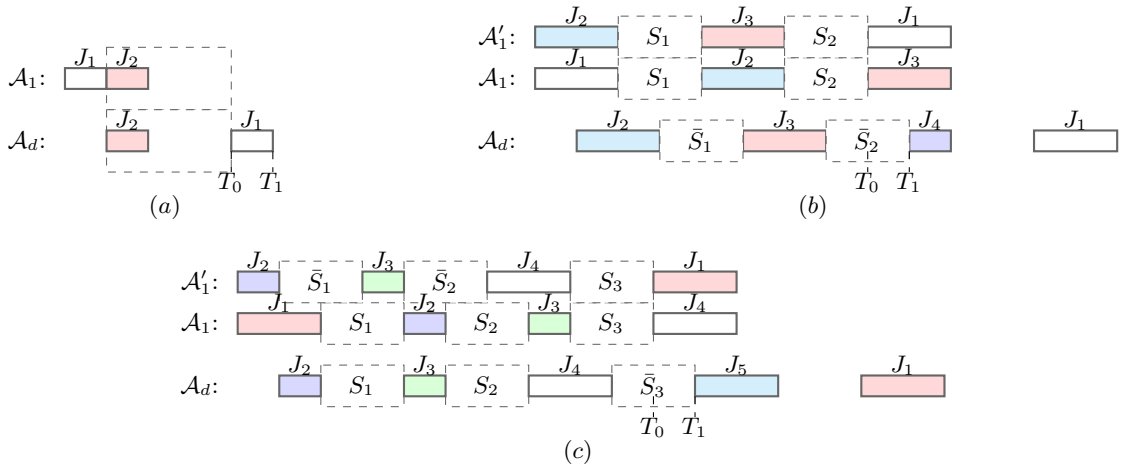

Figure 1: Compare the scheduling under $\mathcal{A}_1$ and $\mathcal{A}_d$

times that obtained by $\mathcal{A}_1$, clearly. Therefore, it suffices to compare the sequences at the time when unconformity occurs. There are three possible cases that may happen, as shown in Figure 1. For a job J_i scheduled by $\mathcal{A}_1$ (resp. $\mathcal{A}_d$), we denote its starting time as s_i (resp. s_i^d) and its finishing time as f_i (resp. f_i^d).

(a) Fig 1 (a) describes such a case: when the unconformity occurs, $\mathcal{A}_1$ has just finished a short job J_1. At this time (i.e., time f_1), $\mathcal{A}_d$ schedules a job J_2 instead of J_1.[4] In fact, J_2 is the job with highest per-unit value at that time, so it is also scheduled by $\mathcal{A}_1$ at the same time. We claim that, before the time $\mathcal{A}_d$ schedules J_1, the sequences (in the dashed boxes) under $\mathcal{A}_1$ and $\mathcal{A}_d$ must be identical. In particular, these are the jobs with higher per-unit value than J_1 and arrival time earlier than $T_0 = s_1^d$. Obviously, the reward obtained by $\mathcal{A}_d$ before $T_1 = f_1^d$ is at least β ($\geq \beta^{k-1}$) times that obtained by $\mathcal{A}_1$ before T_0. Thus the problem is reduced to analyze the remaining part of $\mathcal{A}_d$ from T_1 on (comparing with $\mathcal{A}_1$ from T_0 on), which is already covered by the induction hypothesis.

(b) Fig 1 (b) describes such a case: when the unconformity occurs, $\mathcal{A}_1$ schedules a long job J_1, but $\mathcal{A}_d$ schedules another long job J_2 instead. It is obvious that J_2 must have a higher per-unit value than J_1. Therefore, $s_2 \leq s_1^d - 1$. Now, we consider the job scheduled by $\mathcal{A}_d$ at time $f_2 - 1$. There are three cases: (1) it is just the job J_1; (2) it is a short job; (3) it is a long job other than J_1. For simplicity, we combine these cases in the example in Figure 1 (b). In this example, a long job J_3 is scheduled by $\mathcal{A}_d$ at time $f_2 - 1$. Then similarly, we have $s_3 \leq s_1^d - 1$ and continue to discuss the job scheduled by $\mathcal{A}_d$ at time $f_3 - 1$. In this example, a short job J_4 is scheduled by $\mathcal{A}_d$ at time $f_3 - 1$. We now consider a subsidiary allocation[5] $\mathcal{A}'_1$ which is the same as $\mathcal{A}_1$ except the allocation of J_1, J_2, J_3 (see the figure). Since the per-unit value of J_1 is less than J_2 and J_3, the reward of $\mathcal{A}'_1$ is greater than that of $\mathcal{A}_1$. By comparing $\mathcal{A}_d$ with $\mathcal{A}_1$, we notice that S and $\bar{S}_1$ share the same constitution. That is, the same set of jobs, though the orders may differ. Therefore, using the induction hypothesis, $\bar{S}_1$ has reward at least β^{k-1} times that of S_1. Thus, the reward of $\mathcal{A}_d$ before time $T_1 = f_3 - 1$ is at least β^{k-1} times that of $\mathcal{A}'_1$ before $T_0 = s_3$. The remaining part of this problem (the subsequence of $\mathcal{A}_d$ from T_1 on) reduces to case (c).

(c) Fig 1 (c) describes such a case: $\mathcal{A}_d$ schedules a short job J_2 instead of J_1 when the unconformity occurs. Similar to our argument for case (b), we discuss the behavior of $\mathcal{A}_d$ at s_2 and so forth. As depicted in the figure, a short job J_3 is scheduled at s_2, and a long job J_4 is scheduled at s_3. Through the subsidiary allocation $\mathcal{A}'_1$, which differs from $\mathcal{A}_1$ in the allocations of J_1, J_2, J_3, J_4 (see the figure), we will get: on the one hand, the reward of $\mathcal{A}'_1$ is greater than $\mathcal{A}_1$; on the other hand, the reward of $\mathcal{A}_d$ before $T_1 = f_4 - 1$ is at least β^{k-1} times the reward of $\mathcal{A}'_1$ before $T_0 = s_4$. It remains to discuss the sequence scheduled by $\mathcal{A}_d$ starting from T_1. This can be reduced to case (c) if J_5 is a short job, to case (b) if J_5 is a long job other than J_1, and to the induction hypothesis if it is job J_1.

□

3.2 Proof of Theorem 1

In this subsection, we apply Lemma 2 to prove the competitive ratio of $\mathcal{A}_1$. To make it easier to follow the technique, we first prove the special case, i.e., $k = 2$. The proof for general k is similar, and we will introduce it afterwards.

Proof. Consider the job sequence arranged by *opt**. If there does not exist preemption-resume, then $\mathcal{A}_1$ must arrange the same sequence as *opt** does, and the ratio is 1 obviously. So in the following, we assume there exists preemption-resume. We denote the first preempted long job as job a, and its two segments as a_1 and a_2 (refer to Fig 2 (a)). Clearly, a_1 and a_2 are both of length 1 and a_1 is scheduled earlier than a_2. Fig 2 (a) depicts a part of job sequence under *opt**,

[4] In fact, it does not matter whether J_1 is a short job or a long job. Without loss of generality, we depict it as a short one in the figure.

[5] Note, $\mathcal{A}'_1$ is not a legal scheduling of jobs, since J_2 has not arrived at time 0 and hence cannot be scheduled there.

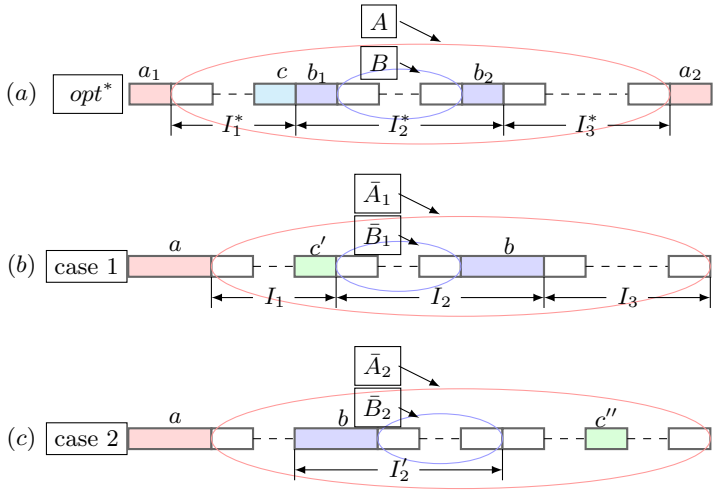

Figure 2: Compare the scheduling under opt^* and Greedy Algorithm

which includes a_1, A and a_2. Here, we use A to represent the job sequence scheduled between a_1 and a_2. First, we give a useful observation below.

OBSERVATION 1. *Each time unit between a_1 and a_2 are occupied and every job scheduled between a_1 and a_2 has a higher per-unit value than job a.*

Fig 2 (b) and Fig 2 (c) depict two possible sequences under $\mathcal{A}_1$, and in these two cases we denote the job sequence after a as $\bar{A}_1$ and $\bar{A}_2$ respectively. By taking Observation 1 into consideration, we have another two observations as follows.

OBSERVATION 2. *The constitutions of A, $\bar{A}_1$ and $\bar{A}_2$ are the same, i.e., they include the same jobs.*

OBSERVATION 3. *There may exist jobs whose preemption and resume both occur between a_1 and a_2 (or occur after a_2), but cannot exist such situation: preemption occurs between a_1 and a_2, but resume occurs after a_2.*

Observation 2 and Observation 3 are non-negligible, since then we can focus on sequence $\langle a_1, A, a_2\rangle$ without loss of generality.

We look into the job sequence A. If there does not exist preemption-resume in A, then the jobs in A are scheduled greedily under opt^*. Because $\mathcal{A}_1$ has to schedule the jobs in A after completing a, which leads to one time-unit delay, we can apply Lemma 2 here, and derive our desired result directly.

Otherwise, if there exists preemption-resume in A, then we denote the first preempted long job as b, and its two segments b_1 and b_2 respectively. We may have two possible cases:

(1) Job b is scheduled at the time that b_2 is scheduled by opt^*, please refer to Fig 2 (b). Actually, $\bar{B}_1$ is identical to B in this case. Fig 2 (a) shows that the job sequence of A is divided into three parts: I_1^*, I_2^* and I_3^*. Fig 2 (b) shows that the job sequence of $\bar{A}_1$ is also divided into three parts I_1, I_2 and I_3. Then applying Lemma 2 to I_1, I_2 and I_3 respectively, we can get that the reward obtained by $\mathcal{A}_1$ during I_1, I_2 or I_3 is at least β times of that obtained by opt^* during I_1^*, I_2^* or I_3^* respectively. So in this case the sequence in $\mathcal{A}_1$ must have reward at least β times of that in opt^*.

(2) Job b is scheduled at the time that b_1 is scheduled by opt^*, please refer to Fig 2 (c). Clearly, job sequence I_2' is in accord with I_2^*, and we will discuss them later. We first consider the remaining parts in A and $\bar{A}_2$, and denote them as $A - I_2^*$ and $\bar{A}_2 - I_2'$ respectively. We can apply Lemma 2 to job sequence $\bar{A}_2 - I_2'$, and derive that the reward obtained by $\mathcal{A}_1$ during $\bar{A}_2 - I_2'$ is at least β times of that obtained by opt^* during $A - I_2^*$. We now turn to discuss I_2' and I_2^*. Since jobs in sequence $\bar{B}_2$ are "delayed" compared to the jobs in sequence B, we reduce to analyze the job sequence B, and the analysis is the same as that of A.

Then, we can apply reduction until the proof is completed.

As for the case of general k, the proof is similar. We can still refer to Fig 2, though some notations in Fig 2 have different meanings now: a_1 (resp. b_1) and a_2 (resp. b_2) denote the first and last segments of job a (resp. b) in opt^*. Different from the case of $k = 2$, when $k > 2$, a_1 (resp. b_1) and a_2 (resp. b_2) can be integers other than 1, and A (resp. B) can include other segments of a (resp. b). We denote the segments of a (resp. b) in A (resp. B) as A' (resp. A'), and denote $\bar{A} = A - A'$ (resp. $\bar{B} = B - B'$). Clearly, all jobs in $\bar{A}$ (resp. $\bar{B}$) have higher per-unit value than a (resp. b).

If there does not exist any preemption-resume in A, then the jobs in A are scheduled greedily under opt^*. Since $\mathcal{A}_1$ has to schedule the jobs in A after completing a, which induces $|a| - |a_1|$ time units (less than $k - 1$) delay, we can apply Lemma 2 here and derive our desired result directly.

If there does exist preemption-resumes in A, then we discuss the time that b is scheduled under $\mathcal{A}_1$. Case 1 denotes the case in which job b is scheduled after the jobs in B, and case 2 denotes the case in which job b is scheduled before the jobs in B. Refer to Fig 2 (b) and Fig 2 (c) respectively.

OBSERVATION 4. *In both case 1 and case 2, the jobs scheduled in I_3^* under opt^* must be scheduled after job b under $\mathcal{A}_1$.*

In Fig 2 (b), I_1 contains part (all) of the jobs which are scheduled in I_1^* under opt^*, $\bar{B}_1$ contains all the jobs in $\bar{B}$

and part of jobs scheduled in I_1^*, and I_3 contains all the jobs scheduled in I_3^* and part of jobs scheduled in I_1^*.

In Fig 2 (c), $\bar{B}_2$ only contains all the jobs in $\bar{B}$, as all jobs in $\bar{B}$ (thus $\bar{B}_2$) have higher per-unit values than job b.

We now discuss the reward obtained by $\mathcal{A}_1$ under the above two cases.

(1) In case 1, we construct a subsidiary allocation, in which, "all" the jobs in I_1^* (resp. $\bar{B}$ and I_3^*) are scheduled greedily in I_1 (resp. $\bar{B}_1$ and I_3). On one hand, similar to the $k = 2$ case, we can get that the reward obtained by the subsidiary allocation during I_1, I_2 or I_3 is at least β^{k-1} times of that obtained by opt^* during I_1^*, I_2^* or I_3^* respectively. On the other hand, the reward of $\mathcal{A}_1$ must be no less than that of the subsidiary allocation, due to the greedy property. In fact, the time b scheduled under $\mathcal{A}_1$ might be no later than that under the subsidiary allocation. Therefore, in this case the sequence in $\mathcal{A}_1$ must have reward at least β^{k-1} times of that in opt^*.

(2) In case 2, similar to the $k = 2$ case, we first consider $A - I_2^*$ and $\bar{A}_2 - I_2'$. We can apply Lemma 2 to job sequence $\bar{A}_2 - I_2'$, and derive that the reward obtained by $\mathcal{A}_1$ during $\bar{A}_2 - I_2'$ is at least β^{k-1} times of that obtained by opt^* during $A - I_2^*$. We then discuss I_2' and I_2^*. It is easy to know that jobs in sequence $\bar{B}_2$ are delayed (no more than $k - 1$ time units) compared to the jobs in sequence B. We reduce to analyze the job sequence B, and the analysis is the same as that of A.

Similarly, we apply reduction until the proof is completed. □

3.3 Upper Bound for the Deterministic Algorithms

One may wonder whether there exists a deterministic algorithm which can achieve a competitive ratio better than β^{k-1}. The following theorem gives a negative answer, and thus bounds the power of deterministic algorithms.

THEOREM 2. *No deterministic algorithm can achieve a competitive ratio better than β^{k-1}.*

PROOF. We use an adversary argument. Consider the following scenario: at time 0, a long job (with length k) with per-unit value v_a (thus total value $v_a' = v_a \cdot \frac{1-\beta^k}{1-\beta}$) arrives. If a deterministic algorithm $\mathcal{A}$ was to schedule it at time t, then the adversary would release a dominant short job with per-unit value v_b (v_b is sufficiently large) at time $t + 1$. In such case, $\mathcal{A}$ obtains a reward $\beta^t \cdot (v_a' + \beta^k \cdot v_b)$, while the optimal offline algorithm can obtain $\beta^t \cdot (\beta \cdot v_b + \beta^2 \cdot v_a')$. Therefore, the competitive ratio is at best $\frac{\beta^t \cdot (v_a' + \beta^k \cdot v_b)}{\beta^t \cdot (\beta \cdot v_b + \beta^2 \cdot v_a')} \to \beta^{k-1}$, when $\frac{v_b}{v_a'} \to \infty$. □

In this way, we come to the conclusion that the greedy algorithm is among the optimal deterministic algorithms.

4. A $\frac{1}{K+1}$-COMPETITIVE RANDOMIZED ALGORITHM

In the previous section, we study the performance of deterministic algorithms. Although the performance of greedy algorithm matches the upper bound for any deterministic algorithms, it may perform poorly when β is extremely small or k is extremely large. Therefore, in the following two sections, we focus on randomized algorithms, and consider whether randomization will help improve the performance of algorithms. We introduce a randomized algorithm which can ensure a competitive ratio of $\frac{1}{k+1}$ in this section.

The algorithm $\mathcal{A}_2$ is described in Algorithm 2. In $\mathcal{A}_2$, $\mu(\cdot)$ is an indicator function: if input is a non-zero set, then $\mu(\cdot) = 1$, otherwise $\mu(\cdot) = 0$.

$\mathcal{A}_2$ operates as follows: for each available time[6] t, $\mathcal{A}_2$ selects a long job l^* with highest per-unit value (Line 2) and schedules it at t according to a well-defined probability (Line 6). If there exist other long jobs, we delay them to time $t + k$ as if their arrival time was $t + k$ (Line 3). This process is based on the fact that in *opt* the reasonable earliest scheduling time of these jobs is not earlier than $t + k$, since job l^* has higher per-unit value than them. If there exist short jobs, we denote the one with highest per-unit value as s^* (Line 2), and delay the others to time $t + 1$ as if their arrival time was $t + 1$ (Line 3).

Intuitively, the probability of scheduling l^* at time t depends on how many long jobs arrive before t whose execution may interfere with the execution of l^* at t. In precise, we only need to check the $k - 1$ time points: $\{t - k + 1, \ldots, t - 2, t - 1\}$. Taking the interference into consideration, we intend to ensure the scheduling of l^* at time t with probability $\frac{1}{k+1}$ finally.

Algorithm 2 The Randomized Algorithm $\mathcal{A}_2$

1: **for** each time $t \geq 0$ **do**
2: Denote the set of long (resp. short) jobs arriving at t as $L(t)$ (resp. $S(t)$), the job with highest per-unit value in $L(t)$ (resp. $S(t)$) as l^* (resp. s^*).
3: Delay all jobs in $L(t) \setminus l^*$ (resp. $S(t) \setminus s^*$) to the time $t + k$ (resp. $t + 1$), as if their arrival time was $t + k$ (resp. $t + 1$).
4: **if** time t is available **then**
5: **if** $L(t) \neq \emptyset$ **then**
6: Schedule the job l^* with probability

$$\sigma(t) = \frac{1}{k + 1 - \sum_{i=1}^{k-1} \mu(L(t-i))}.$$

7: **if** job l^* is scheduled after randomization **then**
8: Throw away job s^*, if any.
9: **else**
10: Schedule job s^*, if any.
11: Throw away job l^*.
12: **end if**
13: **else**
14: Schedule job s^*, if any.
15: **end if**
16: **else**
17: Throw away job l^* and job s^*, if any.
18: **end if**
19: **end for**

After randomization, there exist two cases:

- If l^* is scheduled, throw away the short job with highest per-unit value (s^*) (Line 8).
- If l^* is not scheduled, throw away l^* (Line 11) and

[6]A time point t is *available* if there is no job processed at t.

"plug in" the short job with highest per-unit value (i.e., s^*) if any (Line 10).

If there does not exist any long job at time t, $\mathcal{A}_2$ schedules the short job with highest per-unit value (i.e., s^*, if any) directly (Line 14).

Note that for the unavailable time, i.e., some long job is under process, the possible job l^* and s^* at that time (the definitions are the same as above) should also be thrown away (Line 17).

In Figure 3, we use an example to illustrate the operation of algorithm $\mathcal{A}_2$.

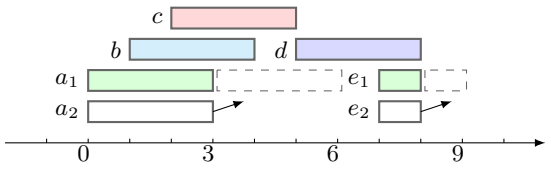

Figure 3: An example of the processing of $\mathcal{A}_2$

EXAMPLE 3. *In this $k = 3$ example, we have two long jobs (a_1 and a_2) which arrive at time 0, and job a_1 has a higher per-unit value than a_2 does. So at the time 0, our algorithm delays a_2 to time 3 and schedules job a_1 with probability $\frac{1}{4}$. If a_1 is finally scheduled at time 0, then job b and job c will be thrown away at time 1 and 2 respectively. Otherwise, the algorithm throws a_1 and schedules job b with probability $\frac{1}{3}$ at time 1. Note that the final probability that job b would be scheduled is $(1 - \frac{1}{4}) \cdot \frac{1}{3} = \frac{1}{4}$.*

In the same way, if job c (or job a_2) had not been thrown away before, then it would be considered to be scheduled with probability $\frac{1}{2}$. The reason is: for job c, there were two jobs (a_1 and b) which might interfere with it; for job a_2 (remember we regard the arrival time of a_2 as time 3), there were also two job (b and c) which might interfere with it. Therefore, we get the desired final probability $\frac{1}{4}$ that job c or job a_2 would be scheduled ($(1 - 2 \cdot \frac{1}{4}) \cdot \frac{1}{2} = \frac{1}{4}$).

As for job d, because there would be only one job (a_2) that might interfere with it, if it had not been thrown away, it would be considered to be scheduled with probability $\frac{1}{3}$, thus, we get the desire final probability $\frac{1}{4}$.

The situation for short jobs (e_1 and e_2) is relatively simple. As e_1 has a higher per-unit value than e_2, e_2 is delayed one time unit afterwards anyway. If job d is scheduled finally, e_1 will be thrown away, otherwise, e_1 will be scheduled.

THEOREM 3. *Algorithm $\mathcal{A}_2$ achieves a competitive ratio of $\frac{1}{k+1}$, for general β.*

PROOF. The proof idea is rooted in the process of this algorithm. Actually, according to the rule of $\mathcal{A}_2$, every long job has the chance of $\frac{1}{k+1}$ to be scheduled at some time t. It is worth mentioning that t is the earliest possible time this job would be scheduled under *opt*.

We now explain how $\frac{1}{k+1}$ comes out. As for the first arrived long job, the algorithm will schedule it with probability $\frac{1}{k+1}$ clearly. By induction, if we assign $\sigma(t)$ as the probability to schedule for a long job at time t, then in consideration of the interference from possible earlier scheduled jobs, the final probability to schedule this job should be

$$\begin{aligned}&(1 - \frac{1}{k+1} \cdot \sum_{i=1}^{k-1} \mu(L(t-i))) \cdot \sigma(t)\\ =&\frac{1 - \frac{1}{k+1} \cdot \sum_{i=1}^{k-1} \mu(L(t-i))}{k+1-\sum_{i=1}^{k-1} \mu(L(t-i))}\\ =&\frac{1}{k+1}.\end{aligned} \tag{1}$$

As for the short job that arrives at t, the the final probability to schedule is at least

$$1 - \frac{1}{k+1} \cdot \sum_{i=0}^{k-1} \mu(L(t-i)) \geq \frac{1}{k+1}. \tag{2}$$

Since every job has at least $\frac{1}{k+1}$ chance to schedule at some time t which is no later than its earliest possible scheduled time under *opt*. Therefore, the competitive ratio obtained by $\mathcal{A}_2$ is at least $\frac{1}{k+1}$. □

The following table shows the threshold β^* for several typical value of k. That is, given a fixed k, the randomized algorithm $\mathcal{A}_2$ performs better than the deterministic algorithm $\mathcal{A}_1$ when $\beta < \beta^*$.

k	2	3	4	5	10	general k
β^*	1/3	1/2	0.58	0.64	0.77	$(k+1)^{\frac{-1}{k-1}}$

5. BOUNDING THE POWER OF RANDOMIZATION

In this section, we characterize the boundary of our research and give an upper bound on the competitive ratios of any randomized algorithms. Formally, we have the following theorem.

THEOREM 4. *No randomized algorithm can achieve a competitive ratio better than $(1 + \frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$.*

PROOF. Let us take the $k = 3$ case for example, and the proof for general k is essentially the same. We will use an adversary argument and show that under some scenarios, no algorithm can achieve a ratio better than $(1+\frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$.

The adversary will release at most two jobs: a long job l (with length 3) and a short *dominant* job s (with length 1 and much larger per-unit value than l).

At time 0, the long job l is released. For an arbitrary randomized algorithm $\mathcal{A}$, we can use a vector $\langle x_1, x_2, x_3, x_4\rangle$ to represent the probability that $\mathcal{A}$ starts to schedule l at each time point. Formally, x_1, x_2 and x_3 represent the probability that $\mathcal{A}$ intends to schedule the job at time $0, 1$ and 2 respectively, and x_4 is the probability to schedule the job at or later than time 3. Clearly, we have $x_1 + x_2 + x_3 + x_4 = 1$.

Given $\mathcal{A}$, we construct the following possible adversarial behaviors:

(1) No other job is released any more. Then, $\mathcal{A}$ obtains an (expected) reward no greater than $(x_1 + \beta \cdot x_2 + \beta^2 \cdot x_3 + \beta^3 \cdot x_4) \cdot v_l$, while the *opt* can obtain a reward v_l. So the competitive ratio is no greater than $x_1 + \beta \cdot x_2 + \beta^2 \cdot x_3 + \beta^3 \cdot x_4$.

(2) The short dominant job s is released at time 2. Then, $\mathcal{A}$ obtains a competitive ratio of no greater than $\beta \cdot x_1 + \beta^2 \cdot x_2 + x_3 + x_4$.

(3) The short dominant job s is released at time 1. Then, $\mathcal{A}$ obtains a competitive ratio of no greater than $\beta^2 \cdot x_1 + x_2 + x_3 + x_4$.

Suppose the adversary also uses a randomized strategy $\langle y_1, y_2, y_3 \rangle$, i.e., the adversary chooses the i-th adversarial behavior (mentioned above) with probability y_i, $i = 1, 2, 3$, and $y_1 + y_2 + y_3 = 1$. In order to find an upper bound, we come to a min max problem, which is formulated as follows.

$$\min_{y_1,y_2,y_3} \max_{x_1,x_2,x_3,x_4} \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix} \begin{pmatrix} 1 & \beta & \beta^2 & \beta^3 \\ \beta & \beta^2 & 1 & 1 \\ \beta^2 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}.$$

Notice that the coefficients of x_4 is dominated by that of x_3, thus $x_4 = 0$ is a dominant strategy for $\mathcal{A}$. Therefore, the min max problem can be simplified to the following style:

$$\min_{y_1,y_2,y_3} \max_{x_1,x_2,x_3} \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix} \begin{pmatrix} 1 & \beta & \beta^2 \\ \beta & \beta^2 & 1 \\ \beta^2 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

It is interesting that the matrix is a Toeplitz matrix, and we find that the min max problem is not difficult to solve based on the following observation:

OBSERVATION 5. *As the adversary choose his strategy after algorithm $\mathcal{A}$ does, the best strategy for $\mathcal{A}$ is to offer a menu with identical items.*

That is, elements in the following vector must be the same.

$$\begin{pmatrix} 1 & \beta & \beta^2 \\ \beta & \beta^2 & 1 \\ \beta^2 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 + \beta \cdot x_2 + \beta^2 \cdot x_3 \\ \beta \cdot x_1 + \beta^2 \cdot x_2 + x_3 \\ \beta^2 \cdot x_1 + x_2 + x_3 \end{pmatrix}.$$

Together with the constraint $x_1 + x_2 + x_3 = 1$, we can solve x_1, x_2 and x_3, and the upper bound is exactly the result of the min max problem.

For a general k, the proof idea is similar: At time 0, a long job (with length k) is released. Each algorithm $\mathcal{A}$ is associated with a probability vector $\langle x_1, x_2, \ldots, x_k, x_{k+1} \rangle$ (x_{k+1} will be proved to be 0). We then construct k adversarial behaviors, and in fact the adversary can randomly choose such behaviors. Therefore, we come to solve a min max problem:

$$\min_{\vec{y}} \max_{\vec{x}} \ \vec{y}\, T\, \vec{x}', \tag{3}$$

where $\vec{x} = \langle x_1, \ldots, x_k \rangle$, $\vec{y} = \langle y_1, \ldots, y_k \rangle$ and T is the Toeplitz matrix as follows:

$$\begin{pmatrix} 1 & \cdots & \cdots & \beta^{k-1} \\ \beta & \cdots & \beta^{k-1} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \beta^{k-1} & \cdots & 1 & 1 \end{pmatrix}. \tag{4}$$

By the similar analysis, we can construct equations and obtain the upper bound $(1 + \frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$. □

By observing the form of $(1 + \frac{(1-\beta^{k-1})^k}{(1-\beta^k)^{k-1}})^{-1}$, we know the upper bound is increasing with β and decreasing with k. Moreover, the upper bound is always greater than $\frac{1}{2}$ when $k \geq 2$ and $0 < \beta < 1$. There is still a gap between the competitive ratio of $\mathcal{A}_2$ and the upper bound.

6. CONCLUSIONS

In this paper, we studied the online non-preemptive story scheduling problem in web advertising. This new scheduling problem is different from those classic problems studied in the 90's (see [26] for an overview) because of the motivation of online advertising. The key factors which constitute the new problem are (1) discounted time horizon, (2) no deadline, and (3) no preemption. We designed both deterministic and randomized algorithms and proved they all have good performance (in terms of expected reward) through competitive analysis. We also showed upper bounds of competitive ratio for any deterministic and randomized algorithms.

There are multiple aspects to study in the future.

- It remains to close the gap between the upper bound and lower bound for randomized algorithms in our setting. Our hypothesis is that the tight bound is $\frac{\sum_{i=0}^{k-1} \beta^i}{k}$.
- Although the setting of two possible lengths seems to be restrictive, it has already posed great challenge for the competitive analysis of the simplest Greedy algorithm. Our work is the first step towards a general setting, in which job lengths can be any integer in $\{1, 2, \ldots, k\}$. We plan to study this generalization.
- We assumed that a website has no information about future jobs and focused on worst-case analysis in this work. When the website accumulates more and more data as time goes on, it can get some distribution information about future jobs and thus it is interesting to design better algorithms leveraging the distribution information.
- We did not consider advertisers' strategic behaviors, e.g., they may misreport their stories. Taking the strategic behaviors into consideration, this problem can be explored from the perspective of mechanism design.

7. ACKNOWLEDGEMENTS

This work was supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the Natural Science Foundation of China Grant 61033001, 61361136003, 61303077, 61561146398, 61170062, 61222202, 61433014, a Tsinghua Initiative Scientific Research Grant and a China Youth 1000-talent program.

REFERENCES

[1] G. Aggarwal, Y. Cai, A. Mehta, and G. Pierrakos. Biobjective online bipartite matching. In *Web and Internet Economics*, pages 218–231. Springer, 2014.

[2] S. Alaei, E. Arcaute, S. Khuller, W. Ma, A. Malekian, and J. Tomlin. Online allocation of display advertisements subject to advanced sales contracts. In *Proceedings of the third international workshop on data mining and audience intelligence for advertising*, pages 69–77. ACM, 2009.

[3] S. Alaei, M. Hajiaghayi, and V. Liaghat. Online prophet-inequality matching with applications to ad allocation. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 18–35. ACM, 2012.

[4] S. Albers and A. Passen. New online algorithms for story scheduling in web advertising. In *Automata, Languages, and Programming*, pages 446–458. Springer, 2013.

[5] G. Christodoulou, E. Koutsoupias, and A. Vidali. A lower bound for scheduling mechanisms. *Algorithmica*, 55(4):729–740, 2009.

[6] P. W. H. Coopers. *IAB internet advertising revenue report*. Coopers, Price Water House, 2014.

[7] A. Dasgupta, A. Ghosh, H. Nazerzadeh, and P. Raghavan. Online story scheduling in web advertising. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1275–1284. Society for Industrial and Applied Mathematics, 2009.

[8] J. Ding, T. Ebenlendr, J. Sgall, and G. Zhang. Online scheduling of equal-length jobs on parallel machines. In *Proceedings of the 15th annual European conference on Algorithms*, pages 427–438. Springer-Verlag, 2007.

[9] J. Ding and G. Zhang. Online scheduling with hard deadlines on parallel machines. In *Algorithmic Aspects in Information and Management*, pages 32–42. Springer, 2006.

[10] T. Ebenlendr and J. Sgall. A lower bound for scheduling of unit jobs with immediate decision on parallel machines. In *Approximation and Online Algorithms*, pages 43–52. Springer-Verlag, 2009.

[11] J. Feldman, N. Korula, V. Mirrokni, S. Muthukrishnan, and M. Pál. Online ad assignment with free disposal. In *Internet and network economics*, pages 374–385. Springer, 2009.

[12] E. J. Friedman and D. C. Parkes. Pricing wifi at starbucks: issues in online mechanism design. In *Proceedings of the 4th ACM conference on Electronic commerce*, pages 240–241. ACM, 2003.

[13] A. Ghosh and A. Sayedi. Expressive auctions for externalities in online advertising. In *Proceedings of the 19th international conference on World wide web*, pages 371–380. ACM, 2010.

[14] G. Goel, M. Hajiaghayi, and M. R. Khani. Randomized revenue monotone mechanisms for online advertising. In *Web and Internet Economics*, pages 324–337. Springer, 2014.

[15] S. A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34(2):370–389, 2000.

[16] M. H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. *Journal of Scheduling*, 6(2):183–211, 2003.

[17] M. Hajiaghayi, R. Kleinberg, M. Mahdian, and D. C. Parkes. Online auctions with re-usable goods. In *Proceedings of the 6th ACM conference on Electronic commerce*, pages 165–174. ACM, 2005.

[18] A. W. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[19] R. Lavi and N. Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 233–241. ACM, 2000.

[20] R. Lipton. Online interval scheduling. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete Algorithms*, pages 302–311. Society for Industrial and Applied Mathematics, 1994.

[21] A. Mäcker, M. Malatyali, F. Meyer auf der Heide, and S. Riechers. Non-preemptive scheduling on machines with setup times. In *Proceedings of the 13th International Symposium on Algorithms and Data Structures (WADS)*, LNCS. Springer, 2015.

[22] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.

[23] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 129–140. ACM, 1999.

[24] N.Y.Times. What is surround sessions?, 2002.

[25] D. C. Parkes. Online mechanisms. *Algorithmic Game Theory, ed. N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, Cambridge University Press*, pages 411–439, 2007.

[26] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

[27] R. Porter. Mechanism design for online real-time scheduling. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 61–70. ACM, 2004.

[28] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, pages 15–1, 2004.

[29] H. Zhang, B. Li, H. Jiang, F. Liu, A. V. Vasilakos, and J. Liu. A framework for truthful online auctions in cloud computing with heterogeneous user demands. In *Proceedings of INFOCOM*, pages 1510–1518. IEEE, 2013.