The Adwords Problem with Strict Capacity Constraints

Umang Bhaskar[1], Ajil Jalal[2], and Rahul Vaze[3]

1 **School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India**
vaze@tcs.tifr.res.in
2 **Department of Electrical and Computer Engineering, The University of Texas, Austin, USA**
ajiljalal@utexas.edu
3 **School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India**
vaze@tcs.tifr.res.in

Abstract

We study an online assignment problem where the offline servers have capacities, and the objective is to obtain a maximum-weight assignment of requests that arrive online. The weight of edges incident to any server can be at most the server capacity. Our problem is related to the adwords problem, where the assignment to a server is allowed to exceed its capacity. In many applications, however, server capacities are strict and partially-served requests are of no use, motivating the problem we study.

While no deterministic algorithm can be competitive in general for this problem, we give an algorithm with competitive ratio that depends on the ratio of maximum weight of any edge to the capacity of the server it is incident to. If this ratio is 1/2, our algorithm is tight. Further, we give a randomized algorithm that is 6-competitive in expectation for the general problem. Most previous work on the problem and its variants assumes that the edge weights are much smaller than server capacities. Our guarantee, in contrast, does not require any assumptions about job weights. We also give improved lower bounds for both deterministic and randomized algorithms. For the special case of parallel servers, we show that a load-balancing algorithm is tight and near-optimal.

1998 ACM Subject Classification F.1.2 Online Computation

Keywords and phrases Online Algorithms, Adwords, Budgeted Matching, Greedy Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2016.30

1 Introduction

Motivated by the problem of assigning advertising slots to advertisers, the adwords problem is a well known and intensely studied online assignment problem. A set of advertisers or bidders has a fixed budget for buying ad slots. A search engine user enters a search query, based on which an advertisement is to be displayed corresponding to an ad slot. Each advertiser places a bid for the slot, and based on these bids, the slot is assigned to a winning bidder and the bid amount is collected as revenue. The objective is to maximize the revenue for the search engine, given the bids of the advertisers and their budgets. Since the search queries – and hence the bids – are not known in advance, this is an online problem, and the winning advertiser at each step must be chosen without knowing future arrivals.

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).
Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 30; pp. 30:1–30:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

More generally, we can replace the advertisers with budgets by servers with capacities, advertising slots by jobs that require processing, and bids by weighted edges. The objective is then to assign jobs to servers to maximize total server utilization, subject to server capacities. In the adwords application, and in the papers that study the adwords problem, the capacities are assumed to be 'violable' constraints. That is, a bidder's winning bids can exceed its budget. In this case, the revenue from a bidder is the minimum of its budget, and the sum of the winning bids placed by it. This is clearly reasonable, since no bidder is charged more than its budget.

However in many online assignment problems, the capacity may in fact be a 'strict' constraint. As an example, consider the case when advertisers are servers with capacities that dictate how long the server can be operated, slots are jobs that need to be processed, and bids are processing times. A partially-complete job should not contribute towards work completed, and hence server capacities are strict constraints. As another example, server capacities correspond to download limits on a (web- or image-) hosting site, and requests for items hosted on these sites arrive online. A partially-downloaded item is typically of no use, and hence again should not contribute towards quota used from the site.

The violable capacity constraints are particularly useful when the edge weights are small in comparison to the server capacities, since in this case even if the last job assigned to a server exceeds the server capacity, it can be removed with small loss to the objective. In fact most work on the adwords problem focuses on the problem with this restriction (see, e.g., [3, 4, 13]. For deterministic algorithms with violable capacity constraints, if large edge weights are allowed, there is a simple example that shows that the competitive ratio is at least 2, and this is achieved by a greedy algorithm [12].

We study an online maximum weight assignment problem that generalizes these problems, with strict capacity constraints. Our goal is to maximize total utilization of capacitated servers that are available offline. In each time step, a set of jobs arrives, that have to be assigned instantaneously and irrevocably to the servers to maximize capacity utilization, subject to strict capacity constraints on the servers. Further, the set of edges at each time step must constitute a matching, hence each server at each time step can be assigned at most one job. This thus corresponds more closely to the problem of adwords with multiple slots [3]. We consider both small and large edge weights, and obtain both upper and lower bounds for the problem.

1.1 Contributions

We make the following contributions in this paper.

- We propose a simple greedy algorithm that is shown to be 3-competitive, whenever the weight of any job is at most half of the corresponding server capacity. In fact, we prove a more general result that if the weight of any job on a server is at most α times the corresponding server capacity, the greedy algorithm is $\left(1 + \frac{1}{1-\alpha}\right)$-competitive. We show via an example that our analysis of the algorithm is tight. Further, we tighten a lower bound given for online b-matching [8] and show for large edge weights, our algorithm is nearly tight as well.
- When each server has identical capacity 1 and is *parallel*, that is, a job has the same weight on every server, we give a deterministic $1+\epsilon$-competitive algorithm, where ϵ is the maximum job weight. Thus if $\epsilon \to 0$, this algorithm is nearly *optimal*. We also show that no deterministic algorithm obtains a better competitive ratio, even for a single server.
- For the unrestricted edge weights case, we propose a randomized version of the greedy algorithm and show that it is 6-competitive. For our algorithm, we define a job as *heavy*

for a server if its weight is more than half of the server capacity, and *light* otherwise. Our randomization is rather novel, where a server accepts or rejects heavy jobs depending on a coin flip. Typically, the randomization is on the edge side, where an edge is accepted or not depending on the coin flips (e.g., [11]). We also show a lower bound of 2 for any randomized algorithm.

- Lastly, we consider the case when jobs have finite span in addition to their weight, and release the resources consumed at the end of their span. The server capacity is thereafter available for other requests. If all jobs have the same span, then we show that our algorithm is 12-competitive. If they have unequal spans, and the maximum and minimum spans are given to the algorithm, we obtain an $O(\log \frac{s_{\max}}{s_{\min}})$-competitive algorithm, where $s_{\max}$ and $s_{\min}$ are the maximum and minimum spans respectively.

1.2 Related Work

In the adwords problem, a set of bidders with individual budgets is given offline. At each time step, a new request arrives with weighted edges to the bidders. The objective is to assign each request to a bidder, to maximize the total weight of edges selected. If the weight of edges incident to a bidder exceeds its capacity, this capacity is included in the sum, rather than the incident edges. The adwords problem was introduced by Mehta et al. [13], for which the authors give a deterministic algorithm with competitive ratio $e/(e-1)$ when the ratio of each bid to the bidder's budget is small (and budgets are assumed to be a violable constraint). The problem was further studied in a number of other papers (e.g., [3, 4]) that give different algorithms and analyses, but with the same competitive ratio of $e/(e-1)$. In the adwords problem with multiple slots, multiple requests arrive at each time step, and the assignment at each time step must be a matching. This extension is studied by Buchbinder, Jain and Naor [3], and they give an online algorithm based on primal-dual techniques where the competitive ratio is shown to be $(1-1/c)(1-R_{\max})$, for $c = (1+R_{\max})^{1/R_{\max}}$ and $R_{\max}$ is the ratio of the maximum bid to the minimum budget of any advertiser. As $R_{\max} \to 0$, the competitive ratio tends to $\frac{e}{e-1}$, and this is optimal. A special case of the multiple slot setting was earlier studied by Mehta et al. [13] as well.

A variant of the problem where edges have values as well as weights, and the capacities of servers restrict the weight of edges incident, is called the Generalized Assignment Problem. Note that in the adwords problem, the weights and values coincide. This problem is studied by Feldman et al. [5]. With the earlier assumption of small edge weights, and assuming *free disposal*, i.e., earlier assigned items can be discarded later, they give an $e/(e-1)$-competitive algorithm. When the job arrivals are stochastic, rather than adversarial, the $e/(e-1)$ ratio can be improved upon [7]. Without the small bid-to-budget ratio assumption, greedy is known to be 2-competitive, and this is tight [12] for deterministic algorithms.

Our problem is also closely related to the problem of online matching, where each server has capacity 1 and each edge has weight either 1 or 0. Here a randomized $e/(e-1)$-competitive algorithm, called RANKING, was given by Karp, Vazirani, and Vazirani [10], and this was shown to be tight for randomized algorithms. The analysis of this algorithm was simplified and extended in later papers [1, 4]. Further, even if multiple online vertices arrive together, the lower bound of $e/(e-1)$ essentially holds, unless a constant fraction of the vertices arrive together [9]. For the problem of online b-matching, a deterministic algorithm was given with a competitive ratio of $\frac{(1+1/b)^b}{(1+1/b)^b-1}$, which is 2 when $b = 1/2$, and tends to $e/(e-1)$ as b increases.

The offline version of our problem is a special case of a separable assignment problem (SAP) [6]. An SAP is defined by a set of n bins and a set of m items to pack in the bins,

with value v_{ij} for assigning item j to bin i. In addition, there are separable constraints for each bin, describing which subset of items can fit in that bin. The objective is to maximize the total value of items packed in the bins, subject to the bin constraints. The online version of SAP has been studied in [2] with expected competitive ratio $\frac{1}{1-\frac{1}{\sqrt{k}}}$, where similar to prior work two restrictions are made; that the weights and sizes of each item are stochastic and each items' size is less than a fraction $\frac{1}{k}$ of the bin capacity.

2 Problem Definition

We are given a set I of n servers, where server i has capacity C_i. We consider an *online* scenario, in which at each time step $t \in \{1, \ldots, T\}$, a set of jobs $J(t)$ and a set of edges $E(t)$ from servers I to jobs $J(t)$ is revealed[1]. Edges are weighted, and $w(e)$ for $e = (i, j)$ is the weight of job j on server i. In particular, if job j is assigned to server i, it consumes $w(e)$ resources of server i out of the possible C_i. In general, a job may have different weights on different servers, thus for distinct servers i and i', $w(i, j) \neq w(i', j)$. The entire set of jobs is $J = \cup_{t \leq T} J(t)$, and $E = \cup_{t \leq T} E(t)$. For a set of edges F, define $W(F) := \sum_{e \in F} w(e)$. Define $G(t)$ as the bipartite graph $(I \cup J(t), E(t))$. A set of edges F is *feasible* if (i) $F(t)$ is a matching for all $t \leq T$, i.e., each server and each job is connected to at most one job and one server respectively at each t, and (ii) for each server i, the weight of edges in F incident to i is at most C_i (this is the strict capacity constraint). We will also call a feasible set of edges an *allocation*. Our objective is to maximize the weight of the allocation obtained.

An optimal allocation has maximum weight among all allocations. The *competitive ratio* for an algorithm is defined as the maximum over all instances of the ratio of the weight of the optimal allocation, to that obtained by the algorithm. For a *randomized* algorithm, the competitive ratio is obtained by taking the denominator of the previous ratio as the *expected* weight of the allocation obtained by the algorithm. Note that the competitive ratio is always at least 1.

In Section 5, we consider the case where jobs have finite span. Here each edge $e = (i, j)$ has a tuple (w, s) associated with it, where w is the weight and s is the time steps for which job j is active, if assigned to machine i. If job j arrives at time t and is assigned to server i, it consumes w resources from server i in time steps $\tau \in [t, t + s - 1]$. We then say that j is active on server i in this period. Thus a set of edges F is feasible if, for each t, (i) $F(t)$ is a matching, and (ii) the total weight of jobs active on each server i at time t is at most its capacity. Our objective is now to obtain an allocation to maximize the total weight of active jobs, summed over servers as well as time steps.

Due to space constraints, all missing proofs appear in the full version.

3 Deterministic Algorithms

We begin by giving a simple example that shows that no deterministic algorithm can be competitive for our problem.

▶ **Example 1.** In Fig. 1, there is a single server with capacity 1. At $t = 1$, a job of weight $\epsilon \ll 1$ arrives. If the algorithm does not accept the job, the input ends; in this case, the optimal value is ϵ while the the algorithm obtains value zero. If the algorithm accepts the

[1] Note that while our algorithms are designed for the setting where multiple jobs arrive at a single time-step, all our lower bounds hold for the case where a single job arrives at each time step.

Figure 1 Illustration for Example 1.

Algorithm 1: GREEDY(G, S)

Input : Weighted bipartite graph G, set of active servers S
Output : Matching M
begin
  $M \leftarrow \emptyset$
  for $e = (i, j) \in G$ *in descending order of weight* **do**
    if *($M \cup e$ is a matching)* ***AND*** $(i \in S)$ **then** $M \leftarrow M \cup e$;
return M

job, the second job with weight 1 arrives. Since the capacity is 1, the algorithm cannot accept this job. In this case, the optimal value is 1 while the algorithm obtains ϵ, and hence any deterministic algorithm has competitive ratio at least $1/\epsilon$.

If we restrict the maximum weight of a job to be $\frac{1}{2}$, then every server can accept at least two jobs, and a deterministic algorithm can give a non-trivial competitive ratio even on adversarial sequences. Under this restriction, we propose an ONLINEGREEDY algorithm that is shown to be 3-competitive next.

In the discussion of the following algorithms, we use $M(t)$ to denote the set of edges selected by the algorithm in time step t, $A(t) := \cup_{\tau \leq t} M(\tau)$, and $M_i(t)$ and $A_i(t)$ to denote the set of edges in $M(t)$ and $A(t)$ incident to server i.

3.1 Deterministic Algorithm for Restricted Edge Weights

We begin with the notion of active servers.

▶ **Definition 2.** Active server: The server i is active at time step $t + 1$ if the sum of the weights of edges assigned to it so far is at most half its capacity, i.e., $W(A_i(t)) \leq \frac{1}{2}C_i$. We will use S to denote the set of active servers.

The deterministic algorithm GREEDY takes as inputs a weighted bipartite graph G, as well as a set S of active servers. GREEDY greedily picks maximum weight edges from the bipartite graph G that are incident to active servers to form a matching M.

3.1.1 OnlineGreedy

We now present a deterministic algorithm ONLINEGREEDY that is 3-competitive for the restricted weights case, where the weight of each edge incident to a server is at most half the server capacity, i.e., $w(i, j) \leq \frac{1}{2}C_i$ for each server i and job j.

ONLINEGREEDY maintains a set of active servers S, along with sets $A_i(t)$ for each server i, where $A_i(t)$ is the set of edges selected that are incident to server i until time t. At each time step t, ONLINEGREEDY calls GREEDY and passes to it as input the weighted bipartite graph $G(t)$ along with the current set of active servers S. For each edge $(i, j) \in M(t)$, where $M(t)$ is the matching returned by GREEDY, edge (i, j) is added to the allocation $A_i(t)$. ONLINEGREEDY then checks if $W(A_i(t)) > \frac{1}{2}C_i$, in which case server i is no longer active and is removed from the set of active servers S for next time slot. If a server i is active at

Algorithm 2: OnlineGreedy

Input : Server capacities $C_1, C_2, ..., C_n$
Weighted bipartite graphs $G(t)$ for $t \leq T$, such that $w(i,j) \leq \frac{1}{2}C_i \, \forall i, j$
Output : Feasible allocation $A(T) = \cup_{t \leq T} M(t)$
begin
$S \leftarrow I$
$A_i(0) \leftarrow \emptyset \; \forall \; i \in I$
for $t \leftarrow 1$ **to** T **do**
$M(t) \leftarrow$Greedy$(G(t), S)$, $A(t) \leftarrow A(t-1) \cup M(t)$
for $(i,j) \in M(t)$ **do**
if $W\left(A_i(t)\right) > \frac{C_i}{2}$ **then** $S \leftarrow S \setminus \{i\}$;

time t, i.e., $W(A_i(t-1)) \leq \frac{1}{2}C_i$, and an edge e is added to $A_i(t-1)$, then $W(A_i(t-1))$ increases by at most $\frac{1}{2}C_i$, and hence $W(A_i(t)) \leq C_i$. Hence, assigning a job to an active server always results in a feasible allocation. Also, since Greedy performs a matching at each time step, the degree constraints (one job/server is assigned to at most one server/job, respectively) are always satisfied. The algorithm continues either until $S = \emptyset$ or $t = T$.

▶ Remark. We note that the restriction on edge weights is only used in proving the feasibility of the allocation obtained, and not in the proof of 3-competitiveness below. In particular, if the edge weights are unrestricted, the allocation obtained may violate the capacity constraints, but will be 3-competitive.

▶ **Theorem 3.** *OnlineGreedy is 3-competitive.*

Proof. For each time step t, let $M(t)$ denote the matching produced by OnlineGreedy, and let $M^*(t)$ denote the corresponding matching given by the optimal offline algorithm. Let $A^*(t) = \cup_{\tau \leq t} M^*(\tau)$, and $A_i^*(t)$ is the set of edges to server i in the optimal allocation until time t. Also, $A_i^* = A_i^*(T)$, $A_i = A_i(T)$, and $A = \cup_{i \in I} A_i$, $A^* = \cup_{i \in I} A_i^*$.

We say that an edge $e = (i,j) \in M^*(t) \setminus M(t)$, has been *blocked* by a heavier weight edge $f \in M(t)$ if $w(f) \geq w(e)$ and f shares a server vertex (i) or job vertex (j) with e. As f has more weight than e, Greedy would select it first in $M(t)$, and hence e cannot be selected without violating matching constraints. For each edge $(i,j) \in M^*(t) \setminus M(t)$, there are three possible reasons why the edge was not selected by OnlineGreedy:

1. An edge $f = (i,j') \in M(t)$, $j' \neq j$ *blocks* (i,j), i.e. server i was matched to some job j' by Greedy, such that $w(i,j') \geq w(i,j)$.
2. An edge $f = (i',j) \in M(t)$, $i' \neq i$ *blocks* (i,j), i.e. job j was matched to some server i' by Greedy, such that $w(i',j) \geq w(i,j)$.
3. The server i was inactive at time step t, i.e., $i \notin S$.

Let $E_1(t)$, $E_2(t)$ and $E_3(t)$ denote the set of edges in $M^*(t) \setminus M(t)$ that satisfy the first, second and third condition respectively. Clearly, $E_1(t) \cup E_2(t) \cup E_3(t) = M^*(t) \setminus M(t)$. *Note:* No edge can satisfy the first and third condition simultaneously, as a server which is inactive at time t cannot be matched to any job at time t. Therefore, $E_1(t) \cap E_3(t) = \emptyset$. However, in general, $E_1(t) \cap E_2(t) \neq \emptyset$ and $E_2(t) \cap E_3(t) \neq \emptyset$, as edges can satisfy conditions 1 and 2 or 2 and 3.

Let S be the set of active servers at time $T+1$. For all servers $i, i \notin S$, since $W(A_i^*) \leq C_i$ and $W(A_i) > \frac{1}{2}C_i$, the allocation A_i is a $\frac{1}{2}$ approximation to A_i^*, i.e.,

$$\sum_{i: i \notin S} \sum_{e \in A_i^*} w(e) < 2 \sum_{i: i \notin S} \sum_{e \in A_i} w(e) \, . \tag{1}$$

Let $E_1 = \cup_{t=1}^{T} E_1(t), E_2 = \cup_{t=1}^{T} E_2(t), E_3 = \cup_{i=1}^{T} E_3(t)$. Define $E_1^S = \{e = (i,j) \in E_1 \mid i \in S\}, E_2^S = \{e = (i,j) \in E_2 \mid i \in S\}$. Clearly, $E_1^S \cup E_2^S = \cup_{i:i\in S} (A_i^* \setminus A_i)$, as no edge $e = (i,j), i \in S$ can satisfy the third condition.

The edges $e \in E_1^S \cup E_2^S$ were not selected in the greedy allocation as they were blocked by edges of heavier weight from $A \setminus A^*$. The edges in the set $A \setminus A^*$ are of two types:

1. $f = (i,j) \in A_i \setminus A_i^*, i \in S$. As all edges $e = (i',j') \in E_1^S \cup E_2^S$ are such that $i' \in S$, e was blocked either because e and f share a server vertex ($i = i'$) or they share a job vertex ($j = j'$). Thus, for every edge $f = (i,j) \in A_i \setminus A_i^*, i \in S$, there may exist at most two edges $e_1 = (i,j'), e_2 = (i',j)$ that are blocked by f, so that $e_1, e_2 \in E_1^S \cup E_2^S$ and $w(f) \geq w(e_1), w(f) \geq w(e_2)$.
2. $g = (i,j) \in A_i \setminus A_i^*, i \notin S$. As all edges $e = (i',j') \in E_1^S \cup E_2^S$ are such that $i' \in S$, e was blocked only because g and e share the same job vertex ($j = j'$) and g was greedily picked first. Thus, for every edge $g = (i,j) \in A \setminus A^*, i \notin S$, there may exist at most one edge $e_1 = (i',j) \in E_1^S \cup E_2^S$ that is blocked by g and is such that $w(g) \geq w(e_1)$.

As $f = (i,j) \in A_i \setminus A_i^*, i \in S$ can block at most two edges in $E_1^S \cup E_2^S$ and $g = (i,j) \in A_i \setminus A_i^*, i \notin S$ can block at most one edge in $E_1^S \cup E_2^S$,

$$\sum_{i:i\in S} \sum_{e\in A_i^* \setminus A_i} w(e) = \sum_{e\in E_1^S \cup E_2^S} w(e) \leq 2 \sum_{i:i\in S} \sum_{f\in A_i \setminus A_i^*} w(f) + \sum_{i:i\notin S} \sum_{g\in A_i \setminus A_i^*} w(g)\,. \tag{2}$$

Adding (1), (2),

$$\sum_{i:i\notin S} \sum_{e\in A_i^*} w(e) + \sum_{i:i\in S} \sum_{e\in A_i^* \setminus A_i} w(e) \leq 2 \sum_{i:i\notin S} \sum_{e\in A_i} w(e) + 2 \sum_{i:i\in S} \sum_{f\in A_i \setminus A_i^*} w(f) + \sum_{i:i\notin S} \sum_{g\in A_i \setminus A_i^*} w(g).$$

Adding $\sum_{i:i\in S} \sum_{e\in A_i \cap A_i^*} w(e)$ to LHS and RHS,

$$\sum_{i:i\notin S} \sum_{e\in A_i^*} w(e) + \sum_{i:i\in S} \sum_{e\in A_i^*} w(e) \leq \sum_{i:i\in S} \sum_{e\in A_i \cap A_i^*} w(e) + 2 \sum_{i:i\in S} \sum_{f\in A_i \setminus A_i^*} w(f) + 3 \sum_{i:i\notin S} \sum_{g\in A_i} w(g).$$

Simplifying, we get $\sum_{i\in I} \sum_{e\in A_i^*} w(e) \leq 3 \sum_{i\in I} \sum_{e\in A_i} w(e)$, as required. ◀

▶ Remark. In the more general case, where edge weights are restricted to be at most α (≤ 1) times the corresponding server capacities, i.e., if $w(i,j) \leq \alpha C_i \, \forall \, i,j$, the following modification of OnlineGreedy makes it $\left(1 + \frac{1}{1-\alpha}\right)$-competitive. Instead of removing a server i from the set of active servers S when $W(A_i(t)) > \frac{1}{2} C_i$, if we remove it when $W(A_i(t)) > (1-\alpha) C_i$, then (1) can be changed to $\sum_{i:i\notin S} \sum_{e\in A_i^*} w(e) < \left(\frac{1}{1-\alpha}\right) \sum_{i:i\notin S} \sum_{e\in A_i} w(e)$. The rest of the proof follows directly to give a $\left(1 + \frac{1}{1-\alpha}\right)$-competitive algorithm. Clearly, as $\alpha \to 1$, the competitive ratio tends to 0, and OnlineGreedy will fail, as expected from Example 1. To handle the case of unrestricted job weights, in the next subsection, we present a randomized algorithm RandomOnlineGreedy which is 6−competitive.

▶ **Example 4.** This example is used to show the tightness of analysis for **Theorem 3**. There are 2 servers with capacity 1. We assume for simplicity that $1/\alpha$ is integral, but the example can be modified to remove this constraint. The sequence of jobs is illustrated in Fig. 2. There

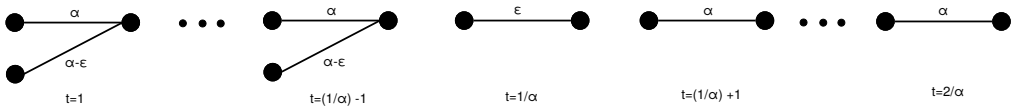

Figure 2 Illustration for Example 4.

are $(1/\alpha) - 1$ jobs that have weight α to the first server, and weight $\alpha - \epsilon$ to the second server. OnlineGreedy assigns all of these to the first server, as well as the next job, and thus the first server now has remaining capacity $\alpha - \epsilon$. The online algorithm then cannot assign any of the remaining jobs, and obtains weight $1 - \alpha + \epsilon$. The optimal offline assigns the first $(1/\alpha) - 1$ jobs to the second machine, ignores the job of weight of ϵ, and assigns the remaining $1/\alpha$ jobs to the first machine, obtaining a total weight of $1 + (\alpha - \epsilon)\left(\frac{1}{\alpha} - 1\right)$. As ϵ tends to zero, this gives a lower bound of $1 + \frac{1}{1-\alpha}$.

3.2 A lower bound for deterministic algorithms

The lower bound example by Kalyanasundaram and Pruhs [8] for online b-matching holds for our problem as well, and shows that if the maximum ratio of edge-weight to server capacity is α, then no deterministic algorithm can obtains competitive ratio better than $(1 + 1/b)^b/(1 + 1/b)^b - 1$, where $b := \lceil 1/\alpha \rceil$. As α goes to zero, this ratio tends to $e/e - 1$.

We can use the strict capacity constraint and strengthen this lower bound slightly, to obtain a lower bound of $\frac{(1+1/b)^{b-1}}{(1+1/b)^{b-1}-1}$. For $\alpha = 1/2$ and $1/3$, this evaluates to 3 and 2.28 respectively, while our algorithm is 3- and 2.5-competitive respectively in these cases. Thus for $\alpha = 1/2$, the competitive ratio we obtain is tight. Since the construction and proof are similar to the earlier example in [8], we give a sketch of the proof here.

▶ **Theorem 5.** *No deterministic algorithm obtains competitive ratio better than* $\frac{(1+1/b)^{b-1}}{(1+1/b)^{b-1}-1}$ *for the online max-weight assignment problem with strict capacity constraints.*

Proof Sketch. Our example closely follows the lower bound for b-matching [8], deviating only at the very beginning. Informally, the earlier example starts with $(b+1)^b$ servers of capacity 1, and jobs of weight $1/b$ to a subset of machines. The example sends jobs in $b+1$ groups, and ensures that jobs in group i can only be assigned by the online algorithm to servers in S_i, where S_1 is the set of all $(b+1)^b$ servers. Group R_i, $i \leq b$, consists of $b^i(1+b)^{b-i}$ jobs, and group R_{b+1} consists of b^{b+1} jobs. Further, it ensures that the last group R_{b+1} of jobs cannot be assigned to any server by the algorithm, while the offline optimal assigns all jobs. This gives the earlier lower bound of $(1 + 1/b)^b/(1 + 1/b)^b - 1$. We modify the example by ensuring that the last *two* groups cannot be assigned by any online algorithm, giving us the improved lower bound.

If $b = 1$, then the lower bound from Example 1 can be used for the theorem. Otherwise, $b \geq 2$. To ensure that jobs from the penultimate group b cannot be assigned by the online algorithm, we will start off with $(b+1)^{b+1}$ servers, and send $(b+1)^{b+1}$ jobs of weight ϵ, each of which can only be assigned to a distinct machine. After these jobs are sent, if at most $(b+1)^b$ jobs are assigned to their unique machine, then we stop. In this case, the optimal offline algorithm assigns all the jobs, and the online algorithm has competitive ratio $(b+1)$, which is at least the bound in the theorem for $b \geq 2$. If at least $(b+1)^b$ machines have a job assigned to them, we select $(b+1)^b$ of these machines, let these machines be S_1, and run the earlier lower bound example. In the example, the jobs in group R_b only have edges

Algorithm 3: ParallelLoadBalance

Input : Capacities C of servers
Jobs $J(t)$ at each time step $t \in \{1, \ldots, T\}$, with weight $w(j)$ for $j \in J(t)$.
Output : Feasible server allocations A_i, $i \in \{1, 2, ..., n\}$
begin
$A_i \leftarrow \emptyset \ \forall i \in \{1, \ldots, n\}$ initially.
for $t \leftarrow 1$ **to** T **do**
for $j \in J(t)$, in decreasing order of weight **do**
Let i be the machine with highest remaining capacity $C - W(A_i)$ that is not assigned a job in current time step.
if $W(A_i \cup \{j\}) \leq C$ **then** $A_i \leftarrow A_i \cup \{j\}$;
else return;

to servers in S_b, and have weight $1/b$ to these servers. However, with our initial step, when the jobs in group R_b arrive, each server in S_b has remaining capacity at most $(1/b) - \epsilon$, and hence cannot serve any more jobs. Hence, jobs in both groups R_b and R_{b+1} cannot now be scheduled by any online algorithm, while the optimal offline algorithm ignores the initial jobs of weight ϵ, and successfully assigns all the remaining jobs. Thus, the optimal offline algorithm obtains total weight $(b+1)^b$, while any online algorithm obtains total weight at most $(b+1)^b - b^{b-1} - b^b$. ◀

3.3 Parallel Servers

Servers are *parallel* if $C_i = C_{i'}$ and $w(i, j) = w(i', j)$ for all jobs j and all servers i, i'. That is, the servers are identical, and each job consumes the same quantity of resources on each server. Thus instead of edge weights we now refer to the weight of each job. If servers are parallel, each with capacity C, and each job has weight at most ϵ, then we show a simple deterministic load-balancing algorithm that is $\frac{1}{1-\epsilon/C}$-competitive.

▶ **Lemma 6.** *After any time step t, the remaining capacity of any pair of machines i, i' differs by at most ϵ with the* ParallelLoadBalance *algorithm.*

Proof. The proof is by induction. Suppose the lemma is true at the end of time step $t-1$, and $A_i(t-1)$, $A_{i'}(t-1)$ are the set of jobs assigned by the algorithm to machines i, i' until time step $t-1$. Assume without loss of generality that $W(A_i(t-1)) \leq W(A_{i'}(t-1))$. Then by the inductive hypothesis, $W(A_i(t-1)) \geq W(A_{i'}(t-1)) - \epsilon$. Further if j, j' are the jobs assigned to i, i' respectively in time step t, then by the algorithm $\epsilon \geq w(j) \geq w(j')$. It follows that $|W(A_i(t)) - W(A_{i'}(t))| \leq \epsilon$. ◀

▶ **Theorem 7.** *Algorithm* ParallelLoadBalance *is $\frac{1}{(1-\epsilon/C)}$-competitive. Further, no deterministic algorithm can perform better.*

Proof. If the **else** condition in ParallelLoadBalance is never encountered, then at every time step the n jobs of largest weight are assigned, and hence the assignment obtained is optimal. Suppose that for some time step t, job j, and server i, the **else** condition is encountered. Then $W(A_i(t-1)) + w(j) > C$, and in fact every server has remaining capacity at most ϵ. This is obviously true of server i, since $w(j) \leq \epsilon$. To see this for the other servers, consider any server i' with $W(A_{i'}(t-1)) < W(A_i(t-1))$. Then in

Algorithm 4: RandomOnlineGreedy

Input : Server capacities $C_1, C_2, ..., C_n$
Weighted bipartite graph $G(t)$ for $t \leq T$, such that $w(i,j) \leq C_i \, \forall \, i,j$
Output : Random feasible allocation $A = \cup_{i \in I} A_i$
begin
$S \leftarrow I$
S_1, S_2, $A_i(0)$, $B_i(0) \leftarrow \emptyset \; \forall \; i \in I$
for $k \leftarrow 1$ **to** n **do**
$v_k \sim Bernoulli(\frac{1}{2})$
if $v_k = 1$ **then** $S_1 \leftarrow S_1 \cup \{k\}$ // accept only heavy jobs ;
else $S_2 \leftarrow S_2 \cup \{k\}$ // accept only light jobs ;
for $t \leftarrow 1$ **to** T **do**
$M(t) \leftarrow$Greedy$(G(t), S)$
for $e = (i,j) \in M(t)$ **do**
$B_i(t) \leftarrow B_i(t-1) \cup \{e\}$
if $W\left(B_i(t)\right) > \frac{C_i}{2}$ **then** $S \leftarrow S \setminus \{i\}$
;
if $\left(i \in S_1$ ***AND*** $w(i,j) > \frac{C_i}{2}\right)$ ***OR*** $\left(i \in S_2$ ***AND*** $w(i,j) \leq \frac{C_i}{2}\right)$ **then**
$A_i(t) \leftarrow A_i(t-1) \cup \{e\}$

time step t, server i' is assigned a job j' with weight at least $w(j)$. Further, by Lemma 6, $W(A_{i'}(t-1)) \geq W(A_i(t-1)) - \epsilon$, and hence

$$W(A_{i'}(t)) = W(A_{i'}(t-1)) + w(j') \geq W(A_{i'}(t-1)) + w(j) \geq W(A_i(t-1)) + w(j) - \epsilon > C - \epsilon \, .$$

Thus in this case, on any machine the remaining capacity is at most ϵ. The proof of the upper bound immediately follows.

For the lower bound, consider a single server with capacity 1. The adversary behaves as follows. At any time, if the total weight of requests sent is at least 1, the adversary stops. The adversary first sends requests of size at most ϵ to the online algorithm, until the server has exactly ϵ remaining capacity. The adversary now sends requests of size $\delta << \epsilon$ until the online algorithm assigns exactly one. The remaining capacity on the server is then $\epsilon - \delta$. It then sends a single request of size ϵ. The optimal offline algorithm ignores the request of size δ and obtains weight 1, while the online algorithm has weight at most $1 - \epsilon + \delta$, giving the lower bound for small δ. ◀

4 A Randomized Algorithm for Unrestricted Edge Weights

Now we present a randomized algorithm, called RandomOnlineGreedy, that is 6-competitive for the general case of unrestricted edge weights.

Note that while $w(i,j)$ can be unbounded, any edge such that $w(i,j) > C_i$ will be ignored as it can never be allocated to server i.

▶ **Definition 8.** An edge $e = (i,j)$ that satisfies $\frac{C_i}{2} < w(i,j) \leq C_i$ is called a *heavy edge* and the corresponding job is called a *heavy job* for that server. In other words, the weight of a heavy edge (i,j) connected to a server i is at least half the server's initial capacity. An edge that is not heavy is called *light*, and the corresponding job is called *light* for that server.

At the start of the algorithm RandomOnlineGreedy, an unbiased coin is flipped for each server i. If heads, then server i is added to set S_1, else it is added to set S_2. If server

$i \in S_1$, it can only accept jobs corresponding to heavy edges, while if $i \in S_2$, it can only accept jobs corresponding to light edges.

Similar to OnlineGreedy, RandomOnlineGreedy maintains a set of active servers S, along with sets $A(t)$ and $B(t)$. At each time step t, the weighted bipartite graph G_t and set of active servers S are passed as input to Greedy, which returns a matching $M(t)$. The set $B(t) := \cup_{\tau \leq t} M(\tau)$ and $B_i(t)$ represents the set of edges in $B(t)$ connected to server i. The set $A_i(t)$ is conditioned on the coin toss for server i. If $i \in S_1$, $A_i(t)$ only contains the heavy edges in $B_i(t)$. Otherwise, if $i \in S_2$, $A_i(t)$ only contains the light edges in $B_i(t)$.

At time t, if RandomOnlineGreedy adds an edge $e = (i, j)$ to B, the algorithm checks the weight $W(B_i(t))$ to see if it should be active for the next time step. If $W(B_i(t)) > \frac{1}{2}C_i$, then server i is removed from S. The reason for maintaining two sets B and A is that it is possible for $B_i(T)$ to be infeasible for some server i. However, $A_i(T)$ is a feasible allocation $\forall\, i$, and $\mathbb{E}\left[W(A_i(T))\right] = \frac{1}{2}W(B_i(T))$. The algorithm continues until either $S = \emptyset$ or $t = T$.

▶ **Lemma 9.** *The allocation $A_i(T)$ is feasible for each machine $i \in I$.*

▶ **Example 10.** This example illustrates how $B_i(T)$ may be an infeasible allocation, while $A_i(T)$ is feasible. Consider a single server with capacity C. At each time step, one job is presented, and $T = 2$. At $t = 1$, a job of weight $\frac{C}{2} - \epsilon$ is presented, while at time $t = 2$, a job of weight C is presented. RandomOnlineGreedy will put both jobs into $B(2)$. If the coin showed heads, $A(2)$ will contain the second edge. If the coin showed tails, $A(2)$ will contain the first edge at time $t = 1$, i.e., $A(2) = \{\frac{1}{2}C - \epsilon\}$ or $A(2) = \{C\}$, and both allocations occur with probability $\frac{1}{2}$. However, $W(B(2)) = \left(\frac{3}{2}C - \epsilon\right)$, which is an infeasible allocation.

▶ **Lemma 11.** $\dfrac{W(A^*(T))}{W(B(T))} \leq 3.$

Proof. As the arguments for (1), (2) hold for the sets $B_i(t)\,\forall\, i$, the proof for **Lemma 11** follows similar to the proof for **Theorem 3**. ◀

▶ **Lemma 12.** $\dfrac{W(B(T))}{\mathbb{E}\left[W(A(T))\right]} = 2.$

Proof. The set $B_i(t)$ can be partitioned into two mutually exclusive subsets $X_i(t)$ and $Y_i(t)$, such that $X_i(t)$ only contains heavy edges, while $Y_i(t)$ only contains light edges. Note that $|X_i(t)| \leq 1$. Let $v_i = 1 (= 0)$ if server i accepts only heavy (light) jobs. As $A_i(t)$ is a feasible allocation $\forall\, t$ and $A_i(t) = X_i(t), t \leq T$ if $v_i = 1$, and $A_i(t) = Y_i(t), t \leq T$ if $v_i = 0$, $X_i(t), Y_i(t), t \leq T$ are both feasible allocations. Therefore $B_i(t) = X_i(t) \cup Y_i(t)$, $X_i(t) \cap Y_i(t) = \emptyset\, \forall\, t$, and $W(B_i(t)) = W(X_i(t)) + W(Y_i(t))$. Hence

$$\begin{aligned}\mathbb{E}\left[W(A_i(T))\right] &= \mathbb{P}\left[v_i = 1\right] W(A_i(T) \mid v_i = 1) + \mathbb{P}\left[v_i = 0\right] W(A_i(T) \mid v_i = 0),\\ &= \frac{1}{2}\left(W(X_i(T)) + W(Y_i(T))\right).\end{aligned}$$

Therefore, $\mathbb{E}\left[W(A_i(T))\right] = \frac{1}{2}W(B_i(T))$. Summing over all servers i, $\sum_{i=1}^{n} \mathbb{E}\left[W(A_i(T))\right] = \frac{1}{2}\sum_{i=1}^{n} W(B_i(T))$, and $\dfrac{\mathbb{E}\left[W(A(T))\right]}{W(B(T))} = \dfrac{1}{2}$. ◀

▶ **Theorem 13.** *RandomOnlineGreedy is $6-$competitive.*

Proof. Let $W(A^*(T)) = W(\cup_{i=1}^{n} A_i^*(T))$ be the value of the allocation made by the optimal offline algorithm, and $W(B(T)) = W(\cup_{i=1}^{n} B_i(T))$ be the value of the infeasible allocation $B(T)$. Moreover, let $\mathbb{E}\left[W(A)\right] = \mathbb{E}\left[W(\cup_{i=1}^{n} A_i(T))\right]$ be the expected value of the feasible allocation $A(T)$ made by RandomOnlineGreedy(denoted as $\mathcal{A}$), then from **Lemma 11** and

Lemma 12, the competitive ratio of RandomOnlineGreedy $= \max\left(\frac{W(A^*(T))}{\mathbb{E}\left[W(A(T))\right]}\right) \leq 6$. ◀

▶ **Example 14.** Example 1 can be extended to show that a lower bound of 2 on the competitive ratio of any randomized algorithm. Consider the randomized adversary that sends only the job in the first step (with weight ϵ) with probability $1-\epsilon$, and both jobs (with weights ϵ and 1) with probability ϵ. Then any deterministic algorithm for this distribution gets value at most ϵ while the optimal expected value is $2\epsilon - \epsilon^2$. The lower bound on randomized algorithms follows by an application of Yao's lemma [14].

5 Finite Span Jobs

We now generalise our model by assuming that the jobs do not consume server resources for infinite time, i.e, along with the job weight, the adversary also announces the span over which the job remains in the server. If a job is presented at time t' and has span s, then it consumes resources for t such that $t' \leq t < t' + s$. Once an allocated job expires, the capacity corresponding to the weight of that job is made available to the server for future job requests.

▶ **Example 15.** For each job let (w, s) be the tuple representing the weight and span, respectively. Let there be a single server with capacity C. Let the input sequences be $S_1 = \{(\epsilon, T), \underbrace{(0,1), \ldots, (0,1)}_{T-1}\}$, and $S_2 = \{(\epsilon, T), \underbrace{\left(\frac{C}{2}, T-1\right), \left(\frac{C}{2}/0, 1\right), \ldots, \left(\frac{C}{2}/0, 1\right)}_{T-1}\}$, where $\frac{C}{2}/0$ means either the weight is $\frac{C}{2}$ or 0 depending on earlier matchings. If at time $t = 1$, the job is not matched to the server, the competitive ratio on S_1 is ∞. Otherwise, the adversary presents the sequence S_2, where if job at time $t = 2$ is not matched then the weights of jobs for all further time instants are 0, and the competitive ratio is $0.5C/\epsilon$. If the server does accept the job at $t = 2$, then all future jobs have weight $\frac{C}{2}$ and span 1. The server cannot accept any jobs for $t \geq 3$ due to lack of capacity, and the competitive ratio is $\frac{0.5C(T-2)}{0.5C+\epsilon} \approx T - 2$. This shows that as $T \to \infty$, the competitive ratio can be made arbitrarily bad for all deterministic algorithms, even when edge weights are restricted to be at most half the server capacity.

5.1 Uniform Span

We first look at the case where all jobs have the same span s. For this case, we propose algorithms UniformGreedy and RandomUniformGreedy, which are similar to our previous algorithms OnlineGreedy (if each job weight is at most half the server capacity) and RandomOnlineGreedy(for general weights), with the following modification. If the algorithm assigns job i to server j at time t, the resources used are released at time $t + s$. The analysis is similar to OnlineGreedy and RandomOnlineGreedy. However, a more intricate argument is required since we can no longer argue about the jobs allocated at each time step. Instead, our analysis considers a window of size s, and obtains bounds on the weight of all jobs that are active within this window.

▶ **Theorem 16.** *UniformGreedy is 6-competitive where all job requests to a server are at most half the capacity of the corresponding server.*

▶ **Theorem 17.** *RandomUniformGreedy is 12-competitive.*

The proof follows similar to Theorem 13, with Theorem 16 replacing Theorem 3.

5.2 Non Uniform Span

RandomNonUniformGreedy is a $\mathcal{O}\left(\log\left(\frac{s_{max}}{s_{min}}\right)\right)$ −competitive algorithm for the case when jobs may not have the same span. The algorithm works by dividing the jobs into $\log\left(\frac{s_{max}}{s_{min}}\right)$ logarithmically spaced levels based on their span and accepting jobs that belong to only one level. This level is chosen uniformly at random before execution of the algorithm.

▶ **Theorem 18.** *RandomNonUniformGreedy is* $\mathcal{O}\left(\log\left(\frac{s_{max}}{s_{min}}\right)\right)$*–competitive.*

Acknowledgments. We are grateful to Anupam Gupta for helpful discussions on the problem, as well as to the anonymous referees of a previous version of this paper for their constructive comments.

References

1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1253–1264, 2011.

2 Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. The online stochastic generalized assignment problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 11–25. Springer, 2013.

3 Niv Buchbinder, Kamal Jain, and Joseph Seffi Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms–ESA 2007*, pages 253–264. Springer, 2007.

4 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107, 2013.

5 Jon Feldman, Nitish Korula, Vahab S. Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Internet and Network Economics, 5th International Workshop, WINE 2009, Rome, Italy, December 14-18, 2009. Proceedings*, pages 374–385, 2009.

6 Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 611–620. Society for Industrial and Applied Mathematics, 2006.

7 Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *Internet and Network Economics – 7th International Workshop, WINE 2011, Singapore, December 11-14, 2011. Proceedings*, pages 170–181, 2011.

8 Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000. doi:10.1016/S0304-3975(99)00140-1.

9 Ming-Yang Kao and Stephen R. Tate. Online matching with blocked input. *Inf. Process. Lett.*, 38(3):113–116, 1991.

10 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358, 1990.

11 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Automata, Languages and Programming*, pages 508–520. Springer, 2009.

12 Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

13 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.

14 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 222–227. IEEE, 1977.