Online Assignment with Repeated Matching Constraints

A Project Report

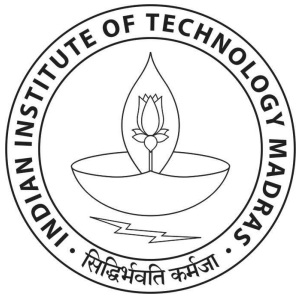
submitted by

AJIL JALAL

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **Online Assignment with Repeated Matching Constraints**, submitted by **Ajil Jalal**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Krishna Jagannathan
Research Guide
Assitant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 16 May 2016

ACKNOWLEDGEMENTS

I would like to thank Prof. Rahul Vaze and Prof. Umang Bhaskar at the Tata Institute of Fundamental Research, Mumbai, under whose guidance this project was initiated. Your hands on mentoring and enthusiasm have eased any worries I may have had about pursuing a PhD. I would also like to thank Prof. Krishna Jagannathan for his valuable time and intellectual inputs over the last year. Conversations with you have always made me think deeper about things that I want- both in academia and life.

To my friends, thank you for being there for me, and I hope that we prevail over the geographic distance that separates us.

Lastly, my parents- I don't think a page is enough to express my gratitude. Your strength has always inspired and will continue to do so.

ABSTRACT

KEYWORDS: Online algorithms; Budgeted matching; Adwords; Mobile offloading; Greedy algorithms

We consider a problem where multiple servers have individual capacity constraints, and at each time slot, a set of jobs arrives, that have potentially different weights to different servers. At each time slot, a one-to-one matching has to be found between jobs and servers, subject to individual capacity constraints, in an online manner. The objective is to maximize the aggregate weight of jobs allotted to servers, summed across time slots and servers, subject to individual server capacity constraints.

For this general problem, we give a randomized online algorithm that is 6-competitive in expectation. Much of previous work on the problem either assumes randomized arrivals, or that the job weights are much smaller than server capacities. Our guarantee, in contrast, holds for worst-case inputs, and does not require any assumptions about job weights. For the special case of identical servers and small-weight jobs, we show that a load-balancing algorithm is optimal. We also consider the case when assignments are temporary — each job arriving has a fixed span in which it consumes resources. This models, e.g., the completion of jobs by the servers. We show that our algorithm can be extended to obtain a 12-competitive algorithm for the case when each node has the same span, and is $O\left(\log \frac{s_{\max}}{s_{\min}}\right)$-competitive for the general case, where $s_{\max}$ and $s_{\min}$ are the maximum and minimum spans respectively.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

We study a basic online resource allocation problem motivated by a number of applications, where the goal is to maximize utilization of capacitated resources that are available ahead of time (offline). In each time step, a set of jobs arrives (online) that have to be matched instantaneously and irrevocably to the servers to maximize capacity utilization subject to individual server capacity constraints. Examples of such applications include online ad auctions with multiple slots, crowdsourcing with multi-agent tasks, and scheduling parallel tasks on multi-processor systems (we describe these applications in detail later). In each of these, a set of capacitated servers is available offline. At each time step, a set of requests arrives that must be irrevocably assigned to the servers, and each request has a different weight (or processing time) depending on its assignment. Crucially, at each time step, the assignment must be a matching: each server can be assigned at most one request, and each request can be assigned to at most a single server. The objective is to maximize the total weight of requests allocated, without exceeding the capacities. We call this general problem the online budgeted repeated matching (OBRM) problem.

As is standard in online problems, we use the competitive ratio — the worst-case ratio over all instances of the optimal value of the objective, to the value obtained by the algorithm — as our metric for performance. In general resource allocation problems, requests have both weights and values, and the goal is either to maximize value subject to constraints on the weight (as in many versions of the secretary problem, (Babaioff *et al.*, 2007)), or to minimize some function of the weight (as in online makespan minimization (Aspnes *et al.*, 1997)). However there are strong lower bounds on the competitive ratio obtainable for these problems with arbitrary inputs. Thus prior work on these general resource allocation problems require assumptions on the input, and either restrict the requests to have small weight, or require the order of request arrivals be randomized.

However, in the applications we study, the value and weight for each request coincide, and the goal is to maximize the weight of requests allocated, or equivalently, utilization of the resources. In this case, we show in this paper that good performance guarantees can be obtained without the earlier assumptions. Instead of randomized arrivals, we construct a randomized algorithm that performs well in expectation on any input. While the coincidence of values and weights for requests allow for competitive algorithms on worst-case inputs, the presence of matching constraints further differentiates the OBRM problem from prior work on online algorithms.

Indeed, in the absence of matching constraints, OBRM reduces to the well-studied online budgeted allocation problem. It is known that a simple greedy algorithm is $1/2$-competitive (Lehmann *et al.*, 2006; Mehta *et al.*, 2007) for the latter problem. This, however, assumes that capacity constraints are violable: in the budgeted allocation problem, the weight of requests allotted to a server can exceed its capacity, and in this case the server's contribution to the objective is its capacity. It is easily seen that without this relaxation, as in our case, where the weight of requests cannot exceed server capacity, no deterministic algorithm is competitive even for a single server.

We briefly describe some applications for the OBRM problem.

1. **ad-auctions with multiple slots:** When a keyword is entered in a search engine, an auction is instantly conducted among advertisers to select an ad to be displayed. Advertisers have limited budgets, and have a value for each keyword which is revealed when the keyword is entered. The goal is to maximize revenue subject to their budgets. In particular, the total value of all keywords assigned to any one advertiser can be more than its budget, but in which case the advertiser is only charged as much as its budget. In ad auctions with multiple slots, there are multiple slots for advertisements for each keyword corresponding to positions on a webpage. Advertisers have different values for each keyword and each slot, and have no value if their advertisements appear in multiple slots for the same keyword. Thus, multiple slots appear in each time step, and feasible allocations correspond to matchings between advertisers and slots. The application is described in more detail by Mehta *et al.* (2007).

2. **delayed mobile offloading:** Consider the wireless networking scenario where there are a number of mobile phones at different locations, each of which needs to download a certain amount of data by a deadline. There are two types of access providers (APs) available at various geographical locations: (1) licensed 3G/4G basestations that provide costly connections, and (2) WiFi access points that can be used cheaply by mobiles to download data (e.g., see Lee *et al.* (2013); Deng and Hou (2015)). For each mobile, if the data demand is not met by the deadline

using only the WiFi APs, then the mobile uses the 3G/4G connection to download the rest of the data. The goal is to maximize the sum of the data transferred to all mobiles from only the WiFi APs within their respective deadlines (that may or may not be the same). This problem is equivalent to OBRM, by considering servers as mobile phones and jobs as WiFi APs, where the total data demand of each mobile is the server capacity and the data rate that a WiFi AP can provide to a particular mobile is the weight on the edge connecting the mobile and AP. Depending on relative locations of WiFi APs and mobiles, an AP can provide very different data rate to different mobiles, hence the weights on edges incident on the same job can be very different. Moreover, the weights also change across slots because of mobility of users. To model the different deadlines of each mobile, we consider the case when each server has a finite span of operation or time for which it is active. The problem as before, is to associate each AP to only one mobile in each time slot that maximizes the sum of the data transferred to all mobiles from only the WiFi APs within their deadlines. We note that in practice each WiFi AP can actually serve more than one mobile (server) at each time with time-sharing, which is in contrast to the matching constraint in OBRM, however, for lack of space we don't consider that case here.

1.1 Contributions

We make the following contributions in this paper.

- We propose a simple greedy algorithm for OBRM 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.

- For the unrestricted edge weights case, we propose a randomized version of the greedy algorithm and show that it is 6-competitive when the job weights are arbitrary against an *oblivious* adversary, that decides the input prior to execution of the algorithm. That is, the adversary decides the input before the random bits are generated. 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.

  A simple example (Example 3.1) shows that no deterministic algorithm has bounded competitive ratio when the job weights are arbitrary, and an extension of this gives a lower bound of 2 for OBRM for randomized algorithms.

- 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 $1 + O(\epsilon)$-competitive al-

gorithm, where ϵ is the maximum job weight. Thus if $\epsilon \to 0$, this algorithm is nearly *optimal*.

- 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

Most related to our work is the paper by Buchbinder *et al.* (2007) who give an online algorithm based on primal-dual techniques for the ad allocation problem with multiple slots. Their competitive ratio depends on $R_{\max}$, the ratio of the maximum bid to the minimum budget of any advertiser, and goes to zero as $R_{\max}$ increases. However, for small values of $R_{\max}$, their competitive ratio is $\frac{e}{e-1}$, which is optimal. The ad allocation problem was earlier introduced by Mehta *et al.* (2007) with a similar competitive ratio for the single-slot ad allocation problem (popularly known as the adwords problem).

For stochastic input with known distribution, OBRM with single job arrival at each time has also been studied extensively in literature Aggarwal *et al.* (2011); Devanur and Hayes (2009); Feldman *et al.* (2010); Haeupler *et al.* (2011); Mehta and Panigrahi (2012). Assuming small edge weight, Devanur and Hayes (2009); Feldman *et al.* (2010) achieve near optimal $1+o(\epsilon)$ competitive ratio, while Mehta and Panigrahi (2012) gives a $1/.567$ competitive ratio. The case when estimates are unreliable has been studied in Mahdian *et al.* (2007). From a resource allocation or crowdsourcing job matching perspective, OBRM with single job and stochastic input has been studied in Tan and Srikant (2012); Jaillet and Lu (2012) and Ho and Vaughan (2012). In a minor departure from other work, Tan and Srikant (2012) allowed a little bit of slack in terms of capacity constraint and showed that the derived profit is within a $O(\epsilon)$ of the optimal profit while allowing constraint violations of $O(1/\epsilon)$.

The offline version of our problem is a special case of a separable assignment problem (SAP) (Fleischer *et al.*, 2006). 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 Alaei *et al.* (2013) 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.

There are many related online problems to OBRM, such as maximum weight matching (Korula and Pál, 2009), knapsack (Babaioff *et al.*, 2007) etc., but all of which require the input to be randomized (secretary model) to get non-trivial competitive ratios. For OBRM, we get constant competitive ratio even under the worst case inputs since the value and the weight for each request coincide.

CHAPTER 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. 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 $W(F) := \sum_{e \in F} w(e)$, and $F\{t\} := F \cap E(t)$ as the set of edges incident to jobs in time step t. 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 job is connected to at most one job and one server respectively at each t, and (ii) the total weight of edges incident to each server summed across all time $1, \ldots, T$ is at most its capacity. We will also call a feasible set of edges an *allocation*.

The Online Budgeted Repeated Matching (OBRM) problem is to pick matchings $M(t) \subseteq E(t)$ *irrevocably* at each time step t to maximize $W(\cup_{t \leq T} M(t))$, so that the sum of the weight of edges in $\cup_{t \leq T} M(t)$ incident to server i is at most C_i.

An optimal allocation for an instance of OBRM has maximum weight among all allocations. The *competitive ratio* for an algorithm for the OBRM problem 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. We use $\mu(\mathcal{A})$ to denote the competitive ratio for an algorithm $\mathcal{A}$.

CHAPTER 3

INFINTE SPAN JOBS

We first consider the OBRM problem when jobs have a weight for each server, and no span; the resources allocated to a job are never released. We begin by illustrating via an example the difficulty in solving the OBRM problem.

Figure 3.1: Illustration for example 3.1 to show non trivial competitive ratio in a fully adversarial setting.

Example 3.1. *In Fig. 3.1, 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 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* $1/\epsilon$*.*

A simple randomized extension to this example, where the input consists of only the job in the first step with probability $1 - \epsilon$*, and both jobs with probability* ϵ*, shows that any randomized algorithm also cannot be better than 2-competitive. 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 Yao (1977).*

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

Definition 3.1. *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.*

3.1.1 GREEDY

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 to form a matching M. The algorithm only adds an edge to the matching if the server connected to it is active.

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**
      $M \leftarrow M \cup e$
end
return *M*

3.1.2 ONLINEGREEDY

We 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

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**
$S \leftarrow S \setminus \{i\}$
end

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 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 3.1. *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 3.1. *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 $e = (i, j) \in M^*(t) \backslash M(t)$, 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 $(i, j) \in M^*(t) \backslash M(t)$, 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.,

$$\sum_{i:i \notin S} \sum_{e \in A_i^*} w(e) < 2 \sum_{i:i \notin S} \sum_{e \in A_i} w(e)\,. \tag{3.1}$$

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 $E_1^S \cup E_2^S$,

$$\sum_{i:i\in S} \sum_{e\in A_i^* \setminus A_i} w(e) = \sum_{e\in E_1^S \cup E_2^S} w(e) \leq 2 \sum_{i:i\in S} \sum_{f\in A_i \setminus A_i^*} w(f) + \sum_{i:i\notin S} \sum_{g\in A_i \setminus A_i^*} w(g)\,. \quad (3.2)$$

Adding (5.2), (5.4),

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

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

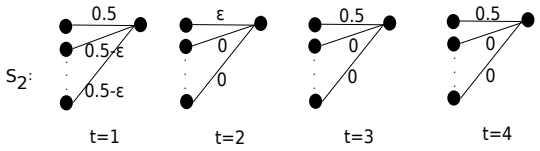


Figure 3.2: Illustration for example 3.2 to show tightness of ONLINEGREEDY

Example 3.2. *This example is used to show the tightness of analysis for* ***Theorem 3.1****. There are n servers with capacity 1. The sequence of jobs is illustrated in Fig. 3.2. At $t = 1$, only the edge to server 1 has weight 0.5, all other edges have weight $(0.5 - \epsilon)$. At*

$t = 2$, only the edge to server 1 has weight ϵ, all other edges have weight 0. At $t = 3, 4$ only the edge to server 1 has weight 0.5, all other edges have weight 0. OnlineGreedy assigns the job at $t = 1, 2$ to server 1, and can't assign any more jobs at $t = 3, 4$, as server 1 is not active during those time slots, and the total weight of the allocation by OnlineGreedy is $0.5 + \epsilon$. The optimal allocation would be to assign the job $(0.5 - \epsilon)$ at $t = 1$ to server 2, and then assign the jobs at time slot $t = 3, 4$ to server 1, so that the optimal weight allocation is $(1.5 - \epsilon)$. Hence OnlineGreedy is a $\frac{1}{3}$-approximation, and this infinite family of instances shows that the analysis of the algorithm is tight.

Remark 3.2. *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 (5.2) 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 3.1. To handle the case of unrestricted job weights, in the next subsection, we present a randomized algorithm RandomOnlineGreedy which is $6-$competitive.*

3.2 Randomized Algorithm for Unrestricted Edge Weights

Now we present a randomized version of OnlineGreedy, 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 3.2. *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.*

Algorithm 3: 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
$S \leftarrow I$
$S_1,\ S_2 \leftarrow \emptyset$
$A_i(0),\ B_i(0) \leftarrow \emptyset \, \forall \, i \in I$
// For each server
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**
$A_i(t) \leftarrow A_i(t-1) \cup \{e\}$
end

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

Proof. Since GREEDY performs a matching at each time step, the degree constraints are always satisfied. We show that the capacity constraints are obeyed as well. Note that $A_i(t) \subseteq B_i(t)$ for all i, t. By construction, if $W(B_i(t)) > \frac{1}{2}C_i$ at any time t, server i is deactivated. Hence every server can accept at most one heavy job. At time t, if a server $i \in S_1$ (i.e., it can accept only heavy jobs) is active, there are no heavy edges in $B_i(t-1)$ and the set $A_i(t-1)$ must be empty. If $\exists e = (i,j) \in M(t)$ which is a heavy edge, it is added to $B_i(t-1)$ and $A_i(t-1)$, and server i is deactivated. As $W(B_i(t-1))$, $W(A_i(t-1))$ increase by at most C_i after adding e to B and A, it may be that $W(B_i(t)) > C_i$ but $W(A_i(t)) \leq C_i$ since $A_i(t)$ was empty before. However, if $\exists e = (i,j) \in M(t)$ which is a light edge, then it is added to $B_i(t-1)$ but not $A_i(t-1)$, and $A_i(t)$ remains empty. Therefore, if $i \in S_1$, $W(A_i(t)) \leq C_i\, \forall\, t$.

On the other hand, if the server $i \in S_2$ is active at time t, then $W(A_i(t-1)), W(B_i(t-1)) \leq \frac{1}{2}C_i$. If $\exists e = (i,j) \in M(t)$ which is a heavy edge, then e is added to $B_i(t-1)$ and i is deactivated. However, e is not added to $A_i(t-1)$ and $W(A_i(t)) \leq \frac{1}{2}C_i$ as no edge has been added to $A_i(t-1)$ at time t. If $\exists e = (i,j) \in M(t)$ which is a light edge, then e is added to $B_i(t-1)$ and $A_i(t-1)$. With the addition of a light edge, $W(B_i(t-1)), W(A_i(t-1))$ increase by at most $\frac{1}{2}C_i$, and as $W(A_i(t-1)) \leq \frac{1}{2}C_i,\ W(A_i(t)) \leq C_i$. Therefore, if $i \in S_2$, $W(A_i(t)) \leq C_i\, \forall\, t$. □

Example 3.3. *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.*

Example 3.4. *This example illustrates how* RANDOMONLINEGREEDY *performs well on Example 3.1. If the job weights to server i are C at $t = 2$, then the optimal matching decision would be to not make any allocations to server i at $t = 1$, an event which occurs in* RANDOMONLINEGREEDY *with probability 0.5 (i.e., if $i \in S_1$). Similarly, if the job weights to server i are 0 at $t = 2$, then the optimal matching decision would be to allocate a job of weight ϵ, an event which occurs in* RANDOMONLINEGREEDY *with probability 0.5 (i.e., if $i \in S_2$). Thus, for the sequence in Example 3.1, with probability 0.5,* RANDOMONLINEGREEDY *finds the optimum allocation for a server.*

Theorem 3.3. *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 $\mathbb{E}[W(A)] = \mathbb{E}[W(\cup_{i=1}^{n} A_i(T))]$ be the expected value of the feasible allocation $A(T)$ made by RANDOMONLINEGREEDY(denoted as $\mathcal{A}$), then from **Lemma 3.4** and **Lemma 3.5**, $\mu(\mathcal{A}) = \max\left(\frac{W(A^*(T))}{\mathbb{E}[W(A(T))]}\right) = 6.$ □

Lemma 3.4. $\frac{W(A^*(T))}{W(B(T))} \leq 3.$

Proof. As the arguments for (5.2), (5.4) hold for the sets $B_i(t) \,\forall\, i$, the proof for **Lemma 3.4** follows similar to the proof for **Theorem 3.1**. A full proof is provided in the Appendix. □

Lemma 3.5. $\frac{W(B(T))}{\mathbb{E}[W(A(T))]} = 2.$

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

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

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

3.3 Deterministic Algorithm for Small Job Weights and Parallel Servers

Servers are *parallel* if $C_i = C_{i'}$ and $e_{ij} = e_{i'j}$ 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 $\dfrac{1}{1-2\epsilon/C}$-competitive.

Algorithm 4: 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
end

Lemma 3.6. *After any time step t, the remaining capacity of any pair of machines i, i' differs by at most ϵ with the PARALLELLOADBALANCE algorithm.*

Theorem 3.7. *Algorithm PARALLELLOADBALANCE is $\dfrac{1}{(1-2\epsilon/C)}$-competitive.*

CHAPTER 4

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 4.1. *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 sequence $S_1 = \{(\epsilon, T), \underbrace{(0,1), \ldots, (0,1)}_{T-1}\}$, while $S_2 = \{(\epsilon, T), \underbrace{\left(\frac{C}{2}, T-1\right), \left(\frac{C}{2}/0, 1\right), \ldots, \left(\frac{C}{2}/0, 1\right)}_{T-1}\}$, 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.*

4.1 Uniform Span

We first look at the case where all jobs have the same span s. Algorithms UNIFORMGREEDY and RANDOMUNIFORMGREEDY are similar to 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 algorithms and the analyses are formally presented in the Appendix. 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 4.1. *UniformGreedy is 6-competitive where all job requests to a server are at most half the capacity of the corresponding server.*

Theorem 4.2. *RandomUniformGreedy is 12-competitive.*

Proof. The proof follows similar to Theorem 3.3, with Theorem 4.1 replacing Theorem 3.1.

□

4.2 Non Uniform Span

In this section we present RandomNonUniformGreedy, a $O\left(\log\left(\frac{s_{max}}{s_{min}}\right)\right)-$competitive algorithm for the case where all jobs do not have the same span.

Algorithm 5: RandomNonUniformGreedy

Input : Server capacities $C_1, C_2, ..., C_n$
Weighted bipartite graph $G(t)$ for $t \leq T$, such that $w(i,j) \leq C_i \, \forall \, i,j$
Minimum span s_{min}; Maximum span s_{max}
Output: Random feasible allocation $A = \cup_{i \in I} A_i$
begin
$r = \left\lceil \log_2\left(\frac{s_{max}+1}{s_{min}}\right) \right\rceil$
$k \sim \mathcal{U}\left(\{0, 1, 2, \cdots, r-1\}\right)$
$s_1 = 2^k \cdot s_{min}$
$s_2 = 2^{k+1} \cdot s_{min}$
Run RandomUniformGreedy, only accepting jobs of span s, such that $s_1 \leq s < s_2$
end

Theorem 4.3. *RandomNonUniformGreedy is $O\left(\log\left(\frac{s_{max}}{s_{min}}\right)\right)$ competitive.*

4.3 Finite Span Servers

We now consider the case in which servers have finite span, but jobs have infinite span. At time t', the adversary announces that a server i will be *alive* for s_i time steps, i.e., for t such that $t' \leq t < t' + s_i$, jobs can be scheduled on server i. We argue that our previous algorithm RANDOMONLINEGREEDY is 6-competitive in such a setting.

Theorem 4.4. *RANDOMONLINEGREEDY is 6-competitive for the case of finite span servers.*

Proof. Assume that the adversary has the sequence of job requests and server spans s_i fixed for all servers i(this does not restrict the adversary, as the competitive ratio is taken as the worst case over all possible sequences), and call this sequence S. Using S, construct the sequence S' as follows:

1. If the number of distinct servers in S is n, then S' will have n servers at each time step.
2. If a server i is announced at time t_i and has span s_i, then the job weights to server i at time t is non-zero if and only if $t_i \leq t < t_i + s_i$, i.e., if a server is not alive at time t, then all job weights presented to it are strictly 0.
3. If server i is alive at time t, then the job weights to server i at time t in the sequences S' and S are the same.

As the job weights are the same in S' and S, the optimal offline algorithm will produce the same allocation for both sequences. When job requests are presented from sequences S and S' at time t, RANDOMONLINEGREEDY will make the same allocation at each time step. From theorem 3.3, as RANDOMONLINEGREEDY is 6-competitive on the sequence S', it is also 6-competitive on the sequence S. □

CHAPTER 5

MODIFIED OBJECTIVE FUNCTION

In this chapter we extend our results to the mobile offloading problem by making a small modification in our objective function. For each server, the we count the minimum of its capacity and the cumulative weight of jobs assigned to it, i.e., $\min\{C_i, W(A_i(T))\}$, where C_i is the capacity of server i and $A_i(T)$ denotes the jobs allocated to server i by our algorithm. Thus the new objective is:

$$\max \sum_i \min\{C_i, W(A_i(T))\} \tag{5.1}$$

The previous formulation places a hard constraint that if the remnant capacity of a server is C_i', then it cannot serve jobs that have weight greater than the remnant capacity. However, in the mobile offloading problem, the remnant capacity denotes the remaining data that a mobile user requires. Thus, if the remaining data rate requirement is C_i' for mobile user i, then it is acceptable for an AP with data rate greater than C_i' to be matched to user i, and the hard constraint $W(A_i(T)) \leq C_i$ is unnecessary. The case of finite span servers help model the limited life time of APs, and along with the new formulation given by 5.1, we find that OBRM with finite span servers models the mobile offloading problem nicely.

The original formulation of AdWords by Mehta *et al.* (2007) uses the same formulation as in 5.1.

All our previous results(except those associated with finite span jobs- the new objective function does not work in this scenario) hold for the new ojective function. We present the proof of Theorem 3.1 as an example. The proofs for the other algorithms follow along the same lines.

Theorem 5.1. *OnlineGreedy is 3-competitive for the formulation given by (5.1)*

Remark 5.1. *In prior work, under the restriction that any job weight is at most half the corresponding capacity, the best competitive ratio known is* $\frac{2\sqrt{3}}{\sqrt{3}-1}$ *Buchbinder* et al. *(2007).*

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 $A^*(t) = \cup_{\tau \le t} M^*(\tau)$, 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 $e = (i, j) \in M^*(t) \backslash M(t)$, has been *blocked* by a heavier weight edge $f \in M(t)$ if $w(f) \ge w(e)$ 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 $(i, j) \in M^*(t) \backslash M(t)$, 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 $w(i, j') \ge w(i, j)$.
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 $w(i', j) \ge w(i, j)$.
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 $W(A_i) > \frac{1}{2} C_i$ and $\min\left(W(A_i^*), C_i\right) \le C_i$, the allocation A_i is a $\frac{1}{2}$ approximation to A_i^*, i.e.,

$$\sum_{i : i \notin S} \min\left(C_i, \sum_{e \in A_i^*} w(e)\right) < 2 \sum_{i : i \notin S} \sum_{e \in A_i} w(e)\,. \tag{5.2}$$

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 $E_1^S \cup E_2^S$,

$$\sum_{i:i\in S} \min\left(\sum_{e\in A_i^*\setminus A_i} w(e), C_i\right) \leq \sum_{i:i\in S}\sum_{e\in A_i^*\setminus A_i} w(e) \leq 2\sum_{i:i\in S}\sum_{f\in A_i\setminus A_i^*} w(f) + \sum_{i:i\notin S}\sum_{g\in A_i\setminus A_i^*} w(g). \tag{5.3}$$

Note that on the RHS, no $\min$ is required, since all job weights are less than half of the corresponding server capacity, and the server is deactivated as soon as more than half of its capacity is exhausted, hence the contribution from all jobs associated to any server can be counted in full without involving the $\min$.

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

$$\begin{aligned}
\sum_{i:i\in S} \min\left(\sum_{e\in A_i^*} w(e), C_i\right) &\leq \sum_{i:i\in S} \min\left(\sum_{e\in A_i^*\setminus A_i} w(e), C_i\right) + \sum_{i:i\in S}\sum_{e\in A_i\cap A_i^*} w(e)\,, \\
&\leq \sum_{i:i\in S}\sum_{e\in A_i\cap A_i^*} w(e) + 2\sum_{i:i\in S}\sum_{f\in A_i\setminus A_i^*} w(f) + \sum_{i:i\notin S}\sum_{g\in A_i\setminus A_i^*} w(g)\,, \\
&\leq 2\sum_{i:i\in S}\sum_{f\in A_i} w(f) + \sum_{i:i\notin S}\sum_{g\in A_i} w(g)\,.
\end{aligned} \tag{5.4}$$

Adding (5.2), (5.4),

$$\sum_{i\in I} \min\left(\sum_{e\in A_i^*} w(e), C_i\right) \leq 2\sum_{i:i\notin S}\sum_{e\in A_i} w(e) + 2\sum_{i:i\in S}\sum_{f\in A_i} w(f) + \sum_{i:i\notin S}\sum_{g\in A_i} w(g)\,.$$

Simplifying, $\sum_{i\in I} \min\left(\sum_{e\in A_i^*} w(e), C_i\right) \leq 3\sum_{i\in I}\sum_{e\in A_i} w(e)$, as required. □

REFERENCES

1. **Aggarwal, G.**, **G. Goel**, **C. Karande**, and **A. 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*. 2011.

2. **Alaei, S.**, **M. Hajiaghayi**, and **V. Liaghat**, The online stochastic generalized assignment problem. *In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2013, 11–25.

3. **Aspnes, J.**, **Y. Azar**, **A. Fiat**, **S. A. Plotkin**, and **O. Waarts** (1997). On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, **44**(3), 486–504.

4. **Babaioff, M.**, **N. Immorlica**, and **R. Kleinberg**, Matroids, secretary problems, and online mechanisms. *In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

5. **Buchbinder, N.**, **K. Jain**, and **J. S. Naor**, Online primal-dual algorithms for maximizing ad-auctions revenue. *In Algorithms–ESA 2007*. Springer, 2007, 253–264.

6. **Deng, H.** and **I. H. Hou**, Online scheduling for delayed mobile offloading. *In 2015 IEEE Conference on Computer Communications (INFOCOM)*. 2015. ISSN 0743-166X.

7. **Devanur, N. R.** and **T. P. Hayes**, The adwords problem: online keyword matching with budgeted bidders under random permutations. *In Proceedings of the 10th ACM conference on Electronic commerce*. ACM, 2009.

8. **Feldman, J.**, **M. Henzinger**, **N. Korula**, **V. S. Mirrokni**, and **C. Stein**, Online stochastic packing applied to display ad allocation. *In Algorithms–ESA 2010*. Springer, 2010, 182–194.

9. **Fleischer, L.**, **M. X. Goemans**, **V. S. Mirrokni**, and **M. Sviridenko**, Tight approximation algorithms for maximum general assignment problems. *In Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. Society for Industrial and Applied Mathematics, 2006.

10. **Haeupler, B.**, **V. S. Mirrokni**, and **M. Zadimoghaddam**, Online stochastic weighted matching: Improved approximation algorithms. *In Internet and Network Economics*. Springer, 2011, 170–181.

11. **Ho, C.-J.** and **J. W. Vaughan**, Online task assignment in crowdsourcing markets. *In AAAI*, volume 12. 2012.

12. **Jaillet, P.** and **X. Lu** (2012). Near-optimal online algorithms for dynamic resource allocation problems. *arXiv preprint arXiv:1208.2596*.

13. **Korula, N.** and **M. Pál**, Algorithms for secretary problems on graphs and hypergraphs. *In Automata, Languages and Programming*. Springer, 2009, 508–520.

14. **Lee, K.**, **J. Lee**, **Y. Yi**, **I. Rhee**, and **S. Chong** (2013). Mobile data offloading: How much can wifi deliver? *IEEE/ACM Transactions on Networking (TON)*, **21**(2), 536–550.

15. **Lehmann, B.**, **D. J. Lehmann**, and **N. Nisan** (2006). Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, **55**(2), 270–296.

16. **Mahdian, M.**, **H. Nazerzadeh**, and **A. Saberi**, Allocating online advertisement space with unreliable estimates. *In Proceedings of the 8th ACM conference on Electronic commerce*. ACM, 2007.

17. **Mehta, A.** and **D. Panigrahi**, Online matching with stochastic rewards. *In Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*. IEEE, 2012.

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

19. **Tan, B.** and **R. Srikant** (2012). Online advertisement, optimization and stochastic networks. *Automatic Control, IEEE Transactions on*, **57**(11), 2854–2868.

20. **Yao, A. C.-C.**, Probabilistic computations: Toward a unified measure of complexity. *In Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE, 1977.

APPENDIX

5.1 UNIFORMGREEDY

We use the set $A\{t\}$ to denote the set of jobs that consume resources at time t, i.e., if an edge e was allocated at time t_1, and $t_1 \leq t < t_1 + s$, then $e \in A\{t\}$.

Algorithm 6: UNIFORMGREEDY

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**
$S \leftarrow S \setminus \{i\}$
for $i \notin S$ **do**
for $(i,j) \in A_i\{t\}$ **do**
if j *is released at* $t+1$ **then**
$S \leftarrow S \cup \{i\}$
end

In the same way we modified ONLINEGREEDY to get UNIFORMGREEDY, we can modify RANDOMONLINEGREEDY to get RANDOMUNIFORMGREEDY.

5.2 Proof of Lemma 3.4

Proof. For each time step t, let $M(t)$ denote the matching produced by RANDOMONLINEGREEDY, and let $M^*(t)$ denote the corresponding matching given by the optimal offline algo-

rithm. Let $A^*(t) = \cup_{\tau \le t} M^*(\tau)$, and $A_i^*(t)$ is the set of edges to server i in the optimal allocation until time t. Also, $A_i^* = A_i^*(T)$ and $B_i = B_i(T)$.

For each edge $(i,j) \in M^*(t) \setminus M(t)$, there are three possible reasons why the edge was not selected by RANDOMONLINEGREEDY:

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 $w(i,j') \ge w(i,j)$.
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 $w(i',j) \ge w(i,j)$.
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.

For all servers $i, i \notin S$, since $W(A_i^*) \le C_i$ and $W(B_i) > \frac{1}{2} C_i$, the allocation B_i is a $\frac{1}{2}$ approximation to A_i^*, i.e.,

$$\sum_{i: i \notin S} \sum_{e \in A_i^*} w(e) < 2 \sum_{i: i \notin S} \sum_{e \in B_i} w(e)\,. \tag{5.5}$$

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, $E_1^S \cup E_2^S = \cup_{i: i \in S} (A_i^* \setminus B_i)$, 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 $B \setminus A^*$. The edges in the set $B \setminus A^*$ are of two types:

1. $f = (i,j) \in B_i \setminus A_i^*, i \in S$. 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 $f = (i,j) \in B_i \setminus A_i^*, i \in S$, there may exist at most two edges $e_1 = (i,j'), e_2 = (i',j)$ such that $e_1, e_2 \in E_1^S \cup E_2^S$ and $w(f) \ge w(e_1), w(f) \ge w(e_2)$.

2. $g = (i, j) \in B_i \setminus A_i^*, i \notin S$. 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 $g = (i, j) \in B \setminus A^*, i \notin S$, there may exist at most one edge $e_1 = (i', j) \in E_1^S \cup E_2^S$ such that $w(g) \geq w(e_1)$.

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

$$\sum_{i:i\in S} \sum_{e\in A_i^*\setminus B_i} w(e) \quad = \quad \sum_{e\in E_1^S\cup E_2^S} w(e) \leq 2 \sum_{i:i\in S} \sum_{f\in B_i\setminus A_i^*} w(f) + \sum_{i:i\notin S} \sum_{g\in B_i\setminus A_i^*} w(g) \tag{5.6}$$

Adding (5.5), (5.6),

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

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

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

Simplifying,

$$\sum_{i\in I} \sum_{e\in A_i^*} w(e) \leq 3 \sum_{i\in I} \sum_{e\in B_i} w(e). \tag{5.7}$$

□

5.3 Proof of Lemma 3.6.

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 $|W(A_i(t)) - W(A_{i'}(t))| \leq \epsilon$. □

5.4 Proof of Theorem 3.7

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 machine i, the **else** condition is encountered. Thus $W(A_i \cup \{j\}) > C$, and since each job has weight at most ϵ, $W(A_i) \geq C - \epsilon$. By Lemma 3.6, for any machine i', $W(A_{i'}) \geq C - 2\epsilon$. The proof immediately follows. □

5.5 Proof of Theorem 4.1

Proof. Without loss of generality, assume that all edges have an even span s. If an allocated edge e was presented at time t_1, then it remains in the server for t' such that $t_1 \leq t' < t_1 + s$. We use the notation $B(t)$ to denote the set of edges which consume resources over the time interval $\left[t - \frac{s}{2}, t + \frac{s}{2}\right]$. Stated formally, if an edge e allocated by our algorithm was presented at time t_1, then $e \in B(t)$ if $[t_1, t_1 + s) \cap \left[t - \frac{s}{2}, t + \frac{s}{2}\right] \neq \emptyset$. $B(t)$ can be thought of as the set of all edges that consume resources over a time window of width s centered at time t. $B_i(t)$ is used to denote the set of edges in $B(t)$ that have been allocated to server i, and the weight of the set $B(t)$ is defined as $W(B(t)) := \sum_{e \in B(t)} w(e)$. Similarly, let $B^*(t)$ and $B^*_i(t)$ denote the analogous set of edges in the optimal allocation. The sets $M(t), M^*(t), A(t), A^*(t)$ are used as defined in Theorem 3.1.

For each edge $(i, j) \in B^*(t) \setminus B(t)$, there are three possible reasons why the edge was not selected by UNIFORMGREEDY:

1. An edge $(i', j) \in B(t)$, $i' \neq i$ was selected instead, i.e. job j was matched to some server i' by GREEDY, such that $w(i', j) \geq w(i, j)$, as the edges were chosen in decreasing order of weight.

2. An edge $(i, j') \in B(t)$, $j' \neq j$ was selected instead, i.e. server i was matched to some job j' by GREEDY, such that $w(i, j') \geq w(i, j)$, as the edges were chosen in decreasing order of weight.
3. The server i was inactive when the edge was presented.

Let $E_1(t)$, $E_2(t)$ and $E_3(t)$ denote the set of edges in $B^*(t) \setminus B(t)$ that satisfy the first, second and third condition respectively. Clearly, $E_1(t) \cup E_2(t) \cup E_3(t) = B^*(t) \setminus B(t)$ and hence we get,

$$W(B^*(t) \setminus B(t)) \leq \sum_{k=1}^{3} W(E_k(t)) \,. \tag{5.8}$$

For each $e_1 \in E_1(t)$ $\exists$ some edge $f(e_1) \in B(t) \setminus B^*(t)$ such that $w\left(f(e_1)\right) \geq w(e_1)$ and $f(e_1)$, e_1 share a common vertex. Similarly, for each $e_2 \in E_2(t)$ $\exists$ $f(e_2) \in B(t) \setminus B^*(t)$ such that $w\left(f(e_2)\right) \geq w(e_2)$ and $f(e_2), e_2$ share a common vertex.

$$\sum_{e_1 \in E_1(t)} w(e_1) + \sum_{e_2 \in E_2(t)} w(e_2) \leq \sum_{e_1 \in E_1(t)} w(f(e_1)) + \sum_{e_2 \in E_2(t)} w(f(e_2)). \tag{5.9}$$

Each $f \in F = \{f(e_1) \mid e_1 \in E_1(t)\} \cup \{f(e_2) \mid e_2 \in E_2(t)\}$ appears at most twice in the RHS of (5.9), and $F \subseteq B(t) \setminus B^*(t)$,

$$\sum_{e_1 \in E_1(t)} w(f(e_1)) + \sum_{e_2 \in E_2(t)} w(f(e_2)) \leq 2 \sum_{f \in F} w(f).$$

Since $F \subseteq B(t) \setminus B^*(t)$,

$$2 \sum_{f \in F} w(f) \leq \; 2 \sum_{g \in B(t) \setminus B^*(t)} w(g).$$

From (5.9),

$$W(E_1(t)) + W(E_2(t)) \leq \; 2W(B(t) \setminus B^*(t)). \tag{5.10}$$

Now consider the weight of the set $E_3(t)$. Let S_t denote the set of servers that are active in the allocation by UNIFORMGREEDY for all time steps $t' \in [t - \frac{s}{2}, t + \frac{s}{2}]$. As each edge has span s, and the window is of width s, it implies that for each server i, there are

at most 2 edges $e_{i1}, e_{i2} \in E_3(t)$ whose spans never overlap. Since over the span of a job, the cumulative weight of edges allocated to server i must be less than C_i, we have,

$$\sum_{e_3 \in E_3(t)} w(e_3) \leq \sum_{i:i \notin S_t} 2C_i. \tag{5.11}$$

As all the servers $s_i \in \{s_1, s_2, ..., s_n\} \setminus S_t$ are inactive at some $t' \in [t - \frac{s}{2}, t + \frac{s}{2}]$, $W(B_i(t)) > \frac{1}{2}C_i$, and hence,

$$2W(B(t)) \geq \sum_{i:i \notin S_t} 2W(B_i(t)) \geq \sum_{i:i \notin S_t} C_i. \tag{5.12}$$

From (5.11) and (5.12),

$$4W(B(t)) \geq \sum_{e_3 \in E_3(t)} w(e_3). \tag{5.13}$$

From (5.8), (5.10) and (5.13),

$$W(B^*(t) \setminus B(t)) \leq 2W(B(t) \setminus B^*(t)) + 4W(B(t)).$$

Adding $W\left(B^*(t) \cap B(t)\right)$ to the LHS and $2W\left(B^*(t) \cap B(t)\right)$ to the RHS,

$$W(B^*(t)) \leq 6W(B(t)). \tag{5.14}$$

We now show how the sets $A(T)$ and $A^*(T)$ can be windowed using $B(t)$ and $B^*(t)$. Consider the sets $B(t_k), t_k = 1 + k \cdot s, k \geq 0$. As each edge $e \in A(t)$ is of span s, and sets $B(t_k)$ are windows of width s centered around t_k, e belongs to exactly 2 sets $B(t_k), B(t_{k+1})$. If e was presented at time t_1, then $e \in B(t_k) \cap B(t_{k+1})$, if $t_1 \leq 1 + s \cdot (k + \frac{1}{2}) \leq t_1 + s \leq 1 + s \cdot ((k + 1) + \frac{1}{2})$.

Summing (5.14) over k, we get,

$$\begin{aligned}
\sum_k W(B^*(t_k)) &\leq \sum_k 6W(B(t_k)), \qquad (5.15)\\
\sum_{e\in B^*(t_k)} w(e) &\leq \sum_{e\in B(t_k)} 6w(e),\\
2\sum_{e\in A^*(T)} w(e) &\leq 2\sum_{e\in A(T)} 6w(e),\\
2\cdot W(A^*(T)) &\leq 2\cdot 6W(A(T)),
\end{aligned}$$

as required. □

5.6 Proof of Theorem 4.3

Proof. Let $r = \lceil \log_2 \left(\frac{s_{max}+1}{s_{min}}\right) \rceil$, and let

$$D(k) = \{(i,j) \mid (i,j) \in A^*(T), 2^k \cdot s_{min} \leq span(j) < 2^{k+1} \cdot s_{min}\},\ 0 \leq k \leq r-1,$$

where A^* is the optimal allocation.

The allocation A returned by our algorithm has minimum span $s_1 = 2^k \cdot s_{min}$ and maximum span $s_2 = 2^{k+1} \cdot s_{min}$, where $k \sim \mathcal{U}\{0, 1, 2, \cdots, r-1\}$. Define $B(t)$, $B^*(t) \subseteq D(k)$, $E_1(t)$, $E_2(t)$, $E_3(t)$ as in the proof of Theorem 4.1. In this case, however, $B(t)$ and $B^*(t)$ are windows of width s_2 centered around t.

The proof follows along similar lines as that of Theorem 4.1. To start with, assume that the weights are restricted to be half the capacity, as in UNIFORMGREEDY. Then equation (5.10) holds for RANDOMNONUNIFORMGREEDY. As s_2 is the width of the window, and s_1 is the minimum span of all jobs in the allocation, with $s_2 = 2s_1$, it means that there are at most 3 jobs $e_{i1}, e_{i2}, e_{i3} \in E_3(t)$ such that their spans do not overlap. Thus (5.11) should be modified to

$$\sum_{e_3\in E_3(t)} w(e_3) \leq \sum_{i:i\notin S_t} 3C_i,$$

and hence we get $W(B^*(t)) \leq 2W(B(t)) + 6W(B(t)) = 8W(B(t))$.

As each window $B(t_j)$ is centered at $t_j = 1 + s_2 \cdot j$ and has width s_2, an edge

$e \in B(t)$ can be present in at least one set $B(t_j)$ or at most 2 sets $B(t_j), B(t_{j+1})$. Proceeding similar to (5.15), we get

$$1 \cdot W(D(k)) \leq \sum_j W(B^*(t_j)) \leq \sum_j 8W(B(t_j)) \leq 2 \cdot 8W(A(T)). \tag{5.16}$$

Using randomisation as in Theorem 3.3, we can extend this to $W(D(k)) \leq 32\mathbb{E}[W(A(T) \mid k)]$ for the case of unrestricted edge weights.

Summing over k, $\sum_{k=0}^{r-1} W(D(k)) \leq 32 \sum_{k=0}^{r-1} \mathbb{E}[W(A(T) \mid k)]$, which implies $\sum_{k=0}^{r-1} W(D(k)) \leq 32r \cdot \sum_{k=0}^{r-1} \frac{1}{r}\mathbb{E}[W(A(T) \mid k)]$, and finally $W(A^*(T)) \leq 32r \cdot \mathbb{E}_k\left[\mathbb{E}[W(A(T))]\right]$.

□