Online Optimization of Busy Time on Parallel Machines[*]

Mordechai Shalom[1] Ariella Voloshin[2] Prudence W.H. Wong[3]
Fencol C.C. Yung[3] Shmuel Zaks[2]

[1] TelHai College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il
[2] Department of Computer Science, Technion, Haifa, Israel
[variella,zaks]@cs.technion.ac.il
[3] Department of Computer Science, University of Liverpool, Liverpool, UK
pwong@liverpool.ac.uk,ccyung@graduate.hku.hk

January 10, 2014

Abstract

We consider the following online scheduling problem in which the input consists of n jobs to be scheduled on identical machines of bounded capacity g (the maximum number of jobs that can be processed simultaneously on a single machine). Each job is associated with a release time and a completion time between which it is supposed to be processed. When a job is released, the online algorithm has to make decision without changing it afterwards. A machine is said to be busy at a certain time if there is at least one job processing on the machine at that time. We consider two versions of the problem. In the minimization version, the goal is to minimize the total busy time of machines used to schedule all jobs. In the resource allocation maximization version, the goal is to maximize the number of jobs that are scheduled under a budget constraint given in terms of busy time. This is the first study on online algorithms for these problems. We show a rather large lower bound on the competitive ratio for general instances. This motivates us to consider special families of input instances for which we show constant competitive algorithms. Our study has applications in power aware scheduling, cloud computing and optimizing switching cost of optical networks.

1 Introduction

The problem. Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [5, 10]). In particular, much attention was given to *interval scheduling* [21], where jobs are given as intervals on the real line, each representing the time interval during which a job should be processed; each job has to be processed on some machine, and it is commonly assumed that a machine can process a *single* job at any given time.

In this paper we consider online interval scheduling with *bounded parallelism*. Formally, the input is a set $\mathcal{J}$ of n jobs. Each job, $J \in \mathcal{J}$, is associated with an interval $[r_J, c_J]$ during

[*]This work was supported in part by the Israel Science Foundation grant No. 1249/08 and British Council Grant UKTELHAI09. A preliminary version of the paper appeared in Proceedings of the 9th Annual Conference on Theory and Applications of Models of Computation, 2012, pp. 448–460.

which it should be processed. We are also given the parallelism parameter $g \geq 1$, which is the maximum number of jobs that can be processed simultaneously by a single machine. At any given time t a machine M_i is said to be busy if there is at least one job J scheduled on it such that $t \in [r_J, c_J]$, otherwise M_i is said to be idle at time t. We call the time period in which a machine M_i is busy its *busy period.* In this work we study two optimization problems MinBusy and MaxThroughput. In MinBusy we focus on minimizing the total busy time over all machines. Note that a solution minimizing the total busy time may not be optimal in terms of the number of machines used. In Section 5.3 we discuss the relation between MinBusy and minimization of the number of machines. In MaxThroughput, the resource allocation version of the problem, we are given a budget T of total machine busy time and the objective is to maximize the number of scheduled jobs under this constraint.

The input to our scheduling problems can be viewed as an *interval graph*, which is the intersection graph of a set of intervals on the real line. It has one vertex for each interval in the set, and an edge between every pair of vertices corresponding to intersecting intervals. In our setting, each vertex corresponds to a job, and there is an edge between two jobs whose processing times overlap.

Applications. Our scheduling problems can be directly interpreted as **power-aware scheduling** problems in cluster systems. These problems focus on minimizing the power consumption of a set of machines (see, e.g., [31] and references therein) measured by the amount of time the machines are switched on and processing, i.e. the total busy time. It is common that a machine has a bound on the number of jobs that can be processed at any given time.

Another application of the studied problems comes from **cloud computing** (see, e.g., [27,30]). Commercial cloud computing provides computing resources with specified computing units. Clients with computation tasks require certain computing units of computing resources over a period of time. Clients are charged in a way proportional to the total amount of computing time of the computing resource. The clients would like to minimize the charges they have to pay (i.e. minimize the amount of computing time used) or maximize the amount of tasks they can compute with a budget on the charge. This is in analogy to our minimization and maximization problems, respectively.

Our study is also motivated by problems in **optical network design** (see, e.g., [9, 12, 13]). Optical wavelength-division multiplexing (WDM) is the leading technology that enables us to deal with the enormous growth of traffic in communication networks, like the Internet. In an optical network, communication between nodes is realized by *lightpaths*, each of which is assigned a certain color. As the energy of the signal along a lightpath decreases, *regenerators* are needed in order to regenerate the signal, thus the associated hardware cost is proportional to the length of the lightpaths. Furthermore, connections can be "groomed" so that a regenerator placed at some node v and operating at some color λ can be shared by at most g connections colored λ and traversing v. This is known as *traffic grooming.* The regenerator optimization problem on the path topology is in analogy to our scheduling problem in the sense that the regenerator cost measured in terms of length of lightpaths corresponds to the busy time while grooming corresponds to the machine capacity.

In the above three applications, it is natural to consider online version of the problem where jobs arrive at arbitrary time and decisions have to be made straightaway (see e.g., [25, 27, 30]).

Related work. Some of the earlier work on interval scheduling considers the problem of scheduling a feasible subset of jobs with maximum total weight, i.e., a *maximum weight* *independent set* (see, e.g., [2] and surveys in [18, 19]). There is wide literature on *real-time scheduling*, where each job has to be processed on some machine during a time interval between its release time and due date. There are also studies on real-time scheduling, where each machine has some capacity and each job has a demand of a certain machine capacity; however, to the best of our knowledge, all of this prior work (see, e.g., [2, 6, 8, 28]) refers to different flavor of the model than the one presented here. Interval scheduling has been studied in the context of online algorithms and competitive analysis [20, 22]. It is also common to consider both minimization and maximization versions of the same scheduling problem, see e.g., [3] but in that model the machines have unit capacity.

Our study also relates to *batch scheduling* of conflicting jobs, where the conflicts are given as an interval graph. In p-batch scheduling model (see, e.g., Chapter 8 in [5]) a set of jobs can be processed jointly. All the jobs in the batch start simultaneously, and the completion time of a batch is the last completion time of any job in the batch. (For known results on batch scheduling, see, e.g., [5].) Our scheduling problem differs from batch scheduling in several aspects. In our problems, each machine can process g jobs simultaneously, for some $g \geq 1$, the jobs need not be partitioned to batches, i.e., each job can start at different time. Also, while in known batch scheduling problems the set of machines is given, we assume that *any* number of machines can be used for the solution. Finally, while common measures in batch scheduling refer to the maximum completion time of a batch, or a function of the completion times of the jobs, we consider the total busy times of the machines.

Previous work on busy time scheduling. The complexity of MinBusy was studied in [32], which showed that the problem is NP-hard already for $g = 2$. The work [14] considered the problem where jobs are given as intervals on the line with unit demand. For this version of the problem it gives a 4-approximation algorithm for general inputs, and better bounds for some subclasses of inputs. In particular, 2-approximation algorithms were given for instances where no job interval is properly contained in another interval (called "proper" instance), and "clique" instances where any two job intervals intersect, i.e., the input forms a clique (see same approximation but different algorithm and analysis in [15]). The work [17] extends the results of [14], considering the case where each job has a different demand on machine capacity and possibly has some slack time. The work [26] improves upon [14] on some subclasses of inputs and initiates the study of MaxThroughput. A 6-approximation is proposed for clique instances, and a polynomial time algorithm is proposed for proper clique instances, i.e. instances that are both "clique" and "proper". These special instances have been considered in [17, 26]. The study so far has been focused on the offline setting [14, 15, 17, 26].

Our contribution. We extend the study on busy time optimization to the online setting and give deterministic online algorithms for both the minimization and maximization variants. For the MinBusy problem, we first show that

- g is a lower bound for the competitive ratio of any online algorithm.

We therefore consider special instances. We first show that a better bound can be achieved if the lengths of the jobs do not grow exponentially with g. Namely we show

- A strictly $5 \log len_{max}$-competitive online algorithm where len_{max} is the length of the longest job.

One special set of instances we consider is the clique instances where any two job intervals intersect, and the one-sided clique instances where all jobs have the same release time or same completion time. Specifically, we show the following:

- Lower and upper bounds of 2 and $(1 + \varphi) < 2.62$ respectively, where $\varphi = (1 + \sqrt{5})/2$ is the Golden Ratio, for one sided clique instances.
- Extension of the upper bound to the clique instances with a blow up of 2 in the ratio.

For the MaxThroughput problem we first show that no online algorithm is better than $(gT/2)$-competitive. We therefore consider feasible instances for which there exist offline schedules that schedule all jobs. We show the following for feasible one-sided clique instances:

- Asymptotic competitive ratio of at least 2 in general,
- Absolute competitive ratio of at least $2 - \frac{2}{g+1}$ even if g is fixed.
- A constant competitive online algorithm with ratio depending on g, but at most 9/2 in general.

We also extend our results to the context of optimization in optical network with grooming. We show that the results can be applied directly if the underlying network topology is a path and adapt some results to other topologies like ring and tree.

Organization of the paper. In Section 2 we present some preliminaries. We consider online busy time minimization and maximization in Sections 3 and 4, respectively. In Section 5 we discuss the application of our results in the context of optical networks. We then conclude in Section 6 with some open problems and further research directions.

2 Notations and preliminaries

Unless otherwise specified, we use lower case letters for indices and specific times, and upper case letters for jobs, time intervals and machines. Moreover, we use calligraphic characters for sets (of jobs, intervals and machines).

The input consists of a set of machines $\mathcal{M} = \{M_1, M_2, \cdots\}$, an integer g representing the machine parallelism bound, and a set of jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ each of which is associated with an interval $[r_J, c_J]$ during which it is supposed to be processed, where r_J and c_J denote the release time and completion time of the job J, respectively. We use jobs and time intervals interchangeably throughout the paper. We assume that the given set $\mathcal{M}$ of machines is infinite as we do not aim at optimizing the number of machines. We address the issue of number of machines in Section 5. We are to decide a schedule to assign jobs to the machines.

To define the objective of the problem, we first define the notion of length and span of intervals. Given a time interval $I = [r_I, c_I]$, the *length* of I is $len(I) \overset{def}{=} c_I - r_I$. The notion extends to a set $\mathcal{I}$ of intervals; namely the *length* of $\mathcal{I}$ is $len(\mathcal{I}) = \sum_{I \in \mathcal{I}} len(I)$. Two intervals are said to be *overlapping* if their intersection contains more than one point. For example, the two intervals [1,2] and [2,3] are considered to be non-overlapping. For a set $\mathcal{I}$ of intervals we define $SPAN(\mathcal{I}) \overset{def}{=} \cup_{I \in \mathcal{I}} I$ and $span(\mathcal{I}) \overset{def}{=} len(SPAN(\mathcal{I}))$. We refer to both of them as the *span* of a set of interval, when the intention is clear from the context. For example, if $\mathcal{I} = \{[1,3],[2,4],[5,6]\}$, then $SPAN(\mathcal{I}) = \{[1,4],[5,6]\}$ $span(\mathcal{I}) = 4$, and $len(\mathcal{I}) = 5$. Note that $span(\mathcal{I}) \le len(\mathcal{I})$ and equality holds if and only if $\mathcal{I}$ is a set of pairwise non-overlapping intervals. A *(partial) schedule* is a function from (a subset of) the set of jobs $\mathcal{J}$ to the set of machines $\mathcal{M}$.

A schedule is said to be *valid* if every machine processes at most g jobs at any given time. In this definition a job $[r_J, c_J]$ is considered as not being processed at time c_J. For instance, a machine processing jobs $[1,2], [2,3], [1,3]$ is considered to be processing two jobs at time 2. Note that this is consistent with the definition of the notion overlapping intervals, and equivalent to saying that the intervals do not contain their completion time, i.e. are half-open intervals.

Given a (partial) schedule $s : \mathcal{J} \mapsto \mathcal{M}$, we denote by $\mathcal{J}_i^s$ the set of jobs assigned to machine M_i by schedule s, i.e. $\mathcal{J}_i^s \overset{def}{=} s^{-1}(M_i)$. The cost of machine M_i in this schedule is the length of its busy interval, i.e. $busy_i^s \overset{def}{=} span(\mathcal{J}_i^s)$. We further denote the set of jobs scheduled by s as $\mathcal{J}^s \overset{def}{=} \cup_i \mathcal{J}_i^s$. The *cost* of schedule s is $cost^s \overset{def}{=} \sum_i busy_i^s$, and its *throughput* is $tput^s \overset{def}{=} |\mathcal{J}^s|$. When there is no ambiguity on the schedule in concern, we omit the superscripts (e.g. we use $\mathcal{J}_i$ for $\mathcal{J}_i^s$, etc.).

We consider two variants of the problem: MinBusy is the problem of minimizing the total cost of scheduling all the jobs, and MaxThroughput is the problem of maximizing the throughput of the schedule subject to a budget given in terms of total busy time. These two problems are formally defined as follows:

MinBusy
Input: $(\mathcal{M}, \mathcal{J}, g)$, where $\mathcal{M}$ is an infinite set of machines, $\mathcal{J}$ is a set of jobs (i.e. time intervals), and g is the parallelism bound.
Output: A valid schedule $s : \mathcal{J} \mapsto \mathcal{M}$.
Objective: Minimize $cost^s$.

MaxThroughput
Input: $(\mathcal{M}, \mathcal{J}, g, T)$ where $\mathcal{M}$ is an infinite set of machines, $\mathcal{J}$ is a set of jobs, g is the parallelism bound, and T is a budget given in terms of total busy time.
Output: A *valid* (partial) schedule $s : \mathcal{J} \mapsto \mathcal{M}$ such that $cost^s \leq T$.
Objective: Maximize $tput^s$.

For both problems we denote by s^* an optimal schedule. The cost of s^* is denoted by $cost^*$ and its throughput by $tput^*$.

Without loss of generality, we assume that each machine is busy over a contiguous time interval. Note that the definition of busy time measures the time that a machine is actually processing some job. If a machine is busy over several contiguous time intervals, then we can replace it with several machines that satisfy the assumption, changing neither the feasibility nor the measure of the schedule. For example, if a machine is busy over $[1,2]$ and $[3,4]$, we can replace this machine with two machines, one busy over $[1,2]$, the other over $[3,4]$ and this does not change the total busy time.

Special instances. A set of jobs $\mathcal{J}$ is a *clique set* if there is a time t common to all the jobs in $\mathcal{J}$. This happens if and only if the corresponding interval graph is a clique. When $\mathcal{J}$ is a clique set we call the corresponding instance ($(\mathcal{M}, \mathcal{J}, g)$ or $(\mathcal{M}, \mathcal{J}, g, T)$) a *clique instance.* A clique instance in which all jobs have the same release time or the same completion time is termed a *one-sided clique instance.*

A set of jobs $\mathcal{J}$ is *proper* if no job in the set properly includes another. Note that in this case for two jobs $J, J' \in \mathcal{J}$, $r_J \leq r_{J'}$ if and only if $c_J \leq c_{J'}$. We denote this fact as $J \leq J'$ and w.l.o.g. we assume the jobs are numbered in such a way that $J_1 \leq J_2 \leq \ldots \leq J_n$.

Online algorithms. When a job is given, an online algorithm has to assign it to a machine or reject it with no future knowledge of jobs to be given. We consider deterministic online algorithms and analyze the performance by competitive analysis [4]. An online algorithm $\mathcal{A}$ for MinBusy is *c-competitive*, for $c \geq 1$, if there exists a constant $b \geq 0$ such that for all input instances, its cost is at most $c \cdot cost^* + b$. For MaxThroughput, $\mathcal{A}$ is c-competitive, for $c \geq 1$, if there exists a constant $b \geq 0$ such that for all input instances, its benefit is at least $(1/c) \cdot tput^* - b$. Note that in both cases, the competitive ratio is ≥ 1. When the additive constant b is zero, $\mathcal{A}$ is said to be *strictly* c-competitive and c is its *absolute* competitive ratio.

Consider MinBusy in which we schedule all jobs in $\mathcal{J}$. The following observation gives two immediate lower bounds for the cost of any schedule of MinBusy.

Observation 1. *For any instance $(\mathcal{M}, \mathcal{J}, g)$ of* MinBusy *and a valid schedule s for it, the following bounds hold:*

- *the* parallelism *bound: $cost^s \geq \frac{len(\mathcal{J})}{g}$,*
- *the* span *bound: $cost^s \geq span(\mathcal{J})$, and*
- *the* length *bound: $cost^s \leq len(\mathcal{J})$.*

The parallelism bound holds since a machine can process at most g jobs at any time. The span bound holds because at any time $t \in SPAN(\mathcal{J})$, at least one machine is busy. The length bound holds because $len(\mathcal{J})$ is the total busy time if each job is allocated a distinct machine.

By the parallelism bound and length bound, the following holds.

Proposition 2. *For* MinBusy, *any online algorithm is strictly g-competitive.*

The following relationship between MinBusy and MaxThroughput is observed in [26].

Proposition 3 ([26])**.** *There is a polynomial time reduction from the* MinBusy *problem to the* MaxThroughput *problem.*

3 Cost Minimization - MinBusy problem

In this section we focus on the MinBusy problem for which we consider general instances in Section 3.1, one-sided clique instances in Section 3.2 and clique instances in Section 3.3.

3.1 General Instances

We consider general instances for MinBusy. We show a lower bound for any online algorithm (Theorem 4) and present a greedy algorithm (Algorithm 2 and Theorem 6).

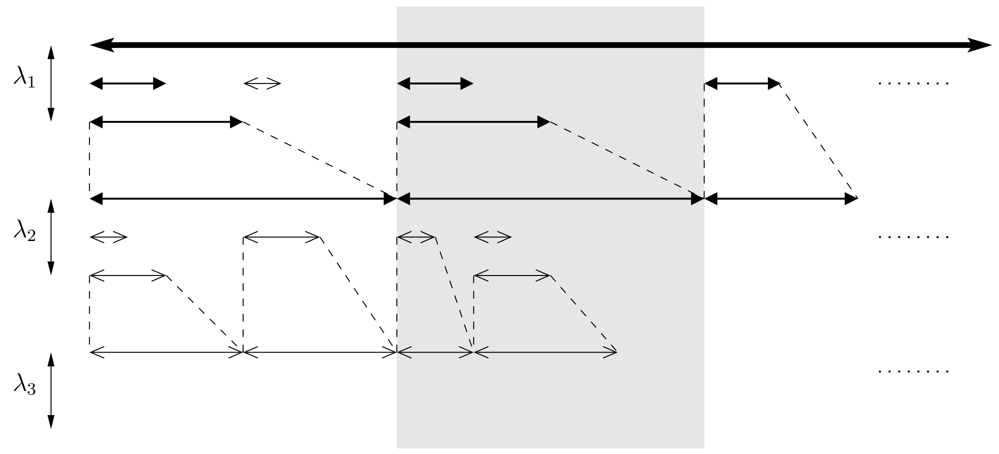

Figure 1: Lower bound for MinBusy: example for $g = 3$. Jobs with different arrow heads and thicknesses belong to different values of d. Dashed lines show jobs for which the online algorithm uses a new machine and the jobs that are just released before each one of them. Note: Within each such pair the shorter job is of length $1/k$ times the longer.

3.1.1 Lower Bound

We first describe a lower bound of 2 on any algorithm $\mathcal{A}$. The adversary releases jobs with release and completion time in the interval $[0, T]$, for some arbitrarily large T. Let k be an integer, $r = 0$ be the release time of the jobs to be released, $t = T$ be the remaining time to be considered, and $\ell = t/k^{g-1}$ be a parameter of the length of jobs to be released.

We first release a job of length t at time r and suppose $\mathcal{A}$ assigns the job to machine M. We then release jobs of lengths $\ell, \ell k, \ell k^2, \cdots$, all at time r, until a machine different from M is used. Suppose that the job assigned to a different machine is of length ℓk^j (note that $j \leq g-1$). Then we set parameters $r' = \ell k^j$, $t' = t - r$ and $\ell' = t'/k^{g-1}$. We then release jobs of length $\ell', \ell' k, \ell k^2, \cdots$ with release time at r', until a machine different from M is used. Repeat until a machine different from M is used for the whole interval $[0, T]$.

With this adversary, $cost^s \geq 2T$: T for the very first job, and T for the jobs not assigned to M. One can use the same machine for all jobs except for the shortest ones, and $cost^* \leq T(1 + 1/k^{g-1})$. An arbitrarily large k implies that the competitive ratio of $\mathcal{A}$ is no better than 2. By extending this idea we prove:

Theorem 4. *For* MinBusy*, no online algorithm has an absolute competitive ratio better than g.*

Proof. We extend the simple adversary described above and start with the same parameters k, r, t while ℓ is now defined as t/k^{dg-1} where d is a new parameter initially set to 1. When $\mathcal{A}$ assigns a new machine to a job of length ℓk^j, for some j, we recursively treat ℓk^j as t and $\ell k^j/k^{2g-1}$ as new ℓ. The exponent is $2g - 1$ since at most $2g$ jobs would force $\mathcal{A}$ to use another machine. The adversary is described by three parameters (r, t, d), where $1 \leq d < g$ is the number of machines used by $\mathcal{A}$. Initially, the parameters are $(0, T, 1)$.

The adversary is described in the recursion below (Algorithm 1). See Figure 1 for an example of the input generated by Adversary.

With the above adversary, we have $cost^s \geq gT$, with T for each machine the algorithm uses. One can assign a machine to all the longest jobs in their release sequence for a busy time of

Algorithm 1 ADVERSARY(r, t, d)

```
1: if $d < g$ then
2:     while $t > 0$ do
3:         $\ell = t/k^{dg-1}$
4:         repeat
5:             Release jobs with release time $r$ and length $\ell, \ell k, \ell k^2, \cdots$
6:         until a new machine is used by the algorithm for a job of length $\ell k^j$
7:         ADVERSARY$(r, \ell k^j, d+1)$
8:         $t = t - \ell k^j$, $r = r + \ell k^j$
9:     end while
10: end if
```

T (these jobs are the jobs that are scheduled on new machines by algorithm $\mathcal{A}$). The total length of the remaining jobs is at most $g \sum_{i=1}^{\infty} T/k^i = g \cdot T/(k-1)$. By the length bound, we have $cost^* \leq T + g \cdot T/(k-1)$ even if the remaining jobs are scheduled individually on different machines. Therefore, $cost^s/cost^* \geq \frac{g}{1+g/(k-1)}$. Picking a large k implies the theorem. □

3.1.2 Upper Bound

With Proposition 2 and Theorem 4, the competitive ratio is tight in terms of g. Yet the adversary in Theorem 4 makes use of jobs of many different lengths. In particular, the adversary needs to generate jobs of length k^{g^2}, requiring $k^{g^2} \leq T$, i.e. $g \leq \sqrt{\log_k T}$. When this is not the case, we have a better result: Algorithm BUCKETFIRSTFIT achieves a competitive ratio depending on the span of the longest job. BUCKETFIRSTFIT takes a parameter $\alpha > 1$ and classifies jobs according to their lengths. A job J is classified in a bucket according to $len(J)$: a job with $\alpha^i \leq len(J) < \alpha^{i+1}$ belongs to bucket i, for $i \geq 0$. Jobs in a bucket are assigned to machines in a First-Fit manner independently of other buckets.

Algorithm 2 BUCKETFIRSTFIT(α)

```
1: When a job $J$ arrives, determine the bucket $i$ of $J$ according to the parameter $\alpha$.
2: for each machine $M$ already assigned a job from bucket $i$ do
3:     if it is valid to assign $J$ to $M$ then
4:         Assign $J$ to $M$
5:         return
6:     end if
7: end for
8: Assign $J$ to a new machine.
```

To analyze BUCKETFIRSTFIT we adapt the proof of Theorem 2.1 in [14]. The offline algorithm in [14] basically sorts the jobs in non-increasing order of length and then assigns jobs in this order to machines in a First-Fit manner. The analysis [14] proves the following properties:

1. Consider the set of jobs $\mathcal{J}_j$ assigned to machine M_j and a job $J \in \mathcal{J}_j$. J may prevent a set of jobs $\mathcal{Q}$ from using machine M_j since they overlap with J. It can be shown that

$$len(J) \geq span(\mathcal{Q})/3. \tag{1}$$

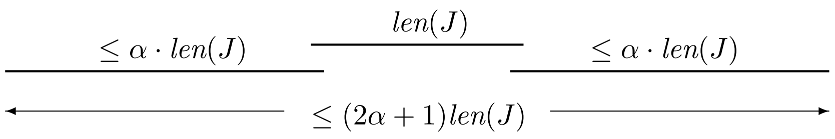

Figure 2: Analysis of BucketFirstFit: Job J (top middle) overlaps with some other jobs in the same bucket. The span of those overlapping jobs is at most $(2\alpha + 1)len(J)$.

Indeed, as jobs are considered in descending order of length, any job in $\mathcal{Q}$ has length at most $len(J)$. The span of any set of jobs overlapping with J is at most $len(J)$ to the left of r_J and at most $len(J)$ to the right of c_J, and hence $span(\mathcal{Q}) \leq 3 \cdot len(J)$.

2. Using Inequality (1), one can then show that for any $j \geq 1$,

$$len(\mathcal{J}_j) \geq \frac{g}{3} span(\mathcal{J}_{j+1}) \tag{2}$$

where the denominator 3 follows from the same denominator in Inequality (1).

3. Summing up Inequality (2) for all j and considering that $span(\mathcal{J}_1)$ is no more then the optimum, it is then shown that the cost of the First Fit algorithm is at most 4 times that of an optimal solution. Note that the coefficient 4 is obtained by adding 1 to the denominator 3 in Inequality (1).

A similar analysis can be applied to BucketFirstFit on the jobs in the same bucket. In the same bucket, unlike in [14], jobs are considered in arbitrary order. However the max-min length ratio of the jobs' length in one bucket is at most α. Then the factor in (1) becomes $2\alpha + 1$ because the span of jobs overlapping with J must be in $[r_J - \alpha\, len(J), c_J + \alpha\, len(J)]$ and that $c_J = r_J + len(J)$ (see Figure 2). By Property 3, the performance ratio for each bucket is therefore $2\alpha + 2$. The result is summarized in Lemma 5.

Lemma 5. *Let $\mathcal{J}^{(i)}$ be the set of jobs in bucket i and let s be a schedule returned by* BucketFirstFit. *Then $cost^s(\mathcal{J}^{(i)}) \leq (2\alpha + 2) cost^*(\mathcal{J}^{(i)})$.*

By Lemma 5 and the bound on the total number of buckets, we can prove the following theorem.

Theorem 6. *For* MinBusy*, by choosing $\alpha = 4$,* BucketFirstFit(4) *is strictly* $(5 \log len_{\max})$*-competitive where $len_{\max}$ is the maximum length of a job.*

Proof. Let $B = \lceil \log_\alpha len_{\max} \rceil$. Note that $B + 1$ is an upper bound on the total number of buckets. Then, $cost^s(\mathcal{J}) = \sum_{0 \leq i \leq B} cost^s(\mathcal{J}^{(i)})$. By Lemma 5, we have

$$cost^s(\mathcal{J}) \leq (2\alpha + 2) B \max_{0 \leq i \leq B} cost^*(\mathcal{J}^{(i)}) \;\; = \frac{(2\alpha + 2) \lceil \log len_{\max} \rceil}{\log \alpha} \max_{0 \leq i \leq B} cost^*(\mathcal{J}^{(i)}).$$

On the other hand, an optimal schedule for all jobs in $\mathcal{J}$ clearly defines a schedule for the jobs in any bucket. Therefore, we have $cost^*(\mathcal{J}) \geq \max_{0 \leq i \leq B} cost^*(\mathcal{J}^{(i)})$. By choosing $\alpha = 4$, $\frac{(2\alpha+2)}{\log \alpha} = 5$ and hence $cost^s(\mathcal{J}) \leq 5 \log len_{\max} cost^*(\mathcal{J})$ and the theorem follows. $\square$

By Proposition 2 and Theorem 6, we have the following corollary.

Corollary 7. *For* MinBusy*, there is a strictly* $\min\{g, 5 \log len_{\max}\}$*-competitive online algorithm where $len_{\max}$ is the maximum length of a job.*

3.2 One-Sided Clique Instances

In this and the next section we consider special instances. We first consider one-sided clique instances in which all jobs have the same release time or the same completion time. We then consider in the next section clique instances in which there is a time t common to all jobs.

3.2.1 Upper bound

For one-sided clique instances at most g jobs can be assigned to one machine because all jobs share a common time. Algorithm BucketFirstFit in Section 3.1.2 can be simplified. For each bucket, instead of checking machines one by one, we only need to keep one "current" machine that has fewer than g machines assigned to it. We give the details in Algorithm 3. We show that it is strictly $(1+\varphi)$-competitive where $\varphi = (1+\sqrt{5})/2$ is the Golden Ratio. Without loss of generality, we assume that all jobs have the same release time.

BucketGreedy depends on a parameter $\alpha > 1$ to be determined in the sequel. In the same way as in the previous section, a job J is categorized to a bucket according to $len(J)$: for $i \geq 0$, bucket i consists of jobs J such that $\alpha^i \leq len(J) < \alpha^{i+1}$.

Algorithm 3 BucketGreedy(α)

```
1: When a job $J$ arrives, determine the bucket $i$ of $J$ according to the parameter $\alpha$.
2: If bucket $i$ has no current machine, then use a new machine and make it the current machine
   of bucket $i$.
3: If there are already $g$ jobs assigned to the current machine of bucket $i$, then use a new
   machine and make it the current machine of bucket $i$.
4: $s(J) \leftarrow$ the current machine of bucket $i$.
```

The correctness of BucketGreedy stems from the validation in Step 3. We proceed with its competitive analysis. Let $q_i \cdot g + r_i$ be the number of jobs in bucket i, where $0 \leq r_i < g$. Let m be the non-empty bucket with biggest index and let ℓ be the biggest index such that $r_\ell > 0$. Clearly, bucket ℓ is non-empty (otherwise $r_\ell = 0$), thus $\ell \leq m$. Let L be the maximum length of a job in bucket ℓ, thus $\alpha^\ell \leq L < \alpha^{\ell+1}$.

BucketGreedy uses distinct machines for every bucket, thus $cost^s$ is the sum of the busy time of machines in each bucket. The sum of busy time of machines in bucket i is at most $(q_i+1)\alpha^{i+1}$, because at most $q_i + 1$ machines are used each with busy time at most α^{i+1}. For $i > \ell$ this number is at most $q_i\alpha^{i+1}$ (because $r_i = 0$), and for bucket ℓ this number is at most $(q_\ell + 1)L$. Therefore we have:

$$\begin{aligned} cost^s &\leq \sum_{i=0}^{\ell-1}(q_i+1)\alpha^{i+1} + (q_\ell+1)L + \sum_{i=\ell+1}^{m} q_i\alpha^{i+1} \leq \sum_{i=0}^{m} q_i\alpha^{i+1} + \sum_{i=0}^{\ell-1}\alpha^{i+1} + L \\ &< \sum_{i=0}^{m} q_i\alpha^{i+1} + \frac{\alpha^{\ell+1}}{\alpha-1} + L \leq \sum_{i=0}^{m} q_i\alpha^{i+1} + \frac{\alpha^{\ell+1}L}{(\alpha-1)\alpha^\ell} + L \\ &= \alpha\sum_{i=0}^{m} q_i\alpha^{i} + \frac{2\alpha-1}{\alpha-1}L \ . \end{aligned} \tag{3}$$

We then bound the cost of the optimal schedule in the following proposition.

Proposition 8.

$$cost^* \geq \sum_{i=0}^{m} q_i \alpha^i + L \ .$$

Proof. Observe that an optimal schedule s^* can be obtained by sorting the jobs in non-increasing order of lengths, and assigning the same machine to every g consecutive jobs in this order. Note that such a solution might assign jobs of different buckets to one machine. This order is equivalent to the non-increasing lexicographic order in $(bucket(J), len(J))$ where $bucket(J)$ is the index of the bucket of job J. We now consider the buckets in decreasing order. The number of jobs in bucket i for $i > \ell$ is a multiple of g, thus in s^* no machine used for jobs of bucket i is available for bucket $i-1$. The number of machines used is therefore q_i, and the busy time is at least α^i in each machine. In bucket ℓ, s^* spends a busy time of L on the first machine, and at least α^ℓ for each of the subsequent q_ℓ machines. Note that the last machine is available for bucket $\ell - 1$. In bucket i with $i < \ell$, s^* may use the last machine of bucket $i+1$ and needs at least q_i more machines, each with a busy time of at least α^i. Note that the last machine is possibly available for bucket $i-1$. We have:

$$cost^* \geq \sum_{i=\ell+1}^{m} q_i \alpha^i + L + q_\ell \alpha^\ell + \sum_{i=0}^{\ell} q_i \alpha^i = \sum_{i=0}^{m} q_i \alpha^i + L \ .$$

□

For $\alpha = \frac{3+\sqrt{5}}{2}$ we have $\frac{2\alpha-1}{\alpha-1} = \alpha$. Then Inequality (3) can be written as $cost^s < \alpha \sum_{i=0}^{m} q_i \alpha^i + \alpha L \leq \alpha \cdot cost^*$ implying

Theorem 9. *For* MinBusy, BucketGreedy$(1+\varphi)$ *is strictly* $(1+\varphi)$*-competitive for one-sided clique instances, where* $\varphi = \frac{1+\sqrt{5}}{2}$ *is the Golden Ratio.*

3.2.2 Lower bound

Lemma 10. *For* MinBusy, *no on-line algorithm has an absolute competitive ratio better than* $(1+1/x)$ *for one-sided clique instances, where x is the root of the equation* $x^{g-1} = (x+1)/(x-1)$.

Proof. The adversary starts by releasing jobs one by one, of lengths 1, x, x^2, $\cdots$, x^{g-1}, with the same release time. If the on-line algorithm $\mathcal{A}$ uses a second machine for a job of length x^{i+1} for some $0 \leq i < g-1$, the adversary stops releasing the remaining jobs. In this case $\frac{cost^s}{cost^*} = (x^{i+1} + x^i)/x^{i+1} = 1 + 1/x$.

If $\mathcal{A}$ assigns the same machine for all these g jobs, the adversary releases one more job of length x^{g-1} at the same arrival time, forcing $\mathcal{A}$ to use a different machine. In this case, $cost^s = 2x^{g-1}$ while s^* assigns one machine to the job of length 1 and another machine for the rest, resulting in $cost^* = x^{g-1} + 1$. Then, $\frac{cost^s}{cost^*} = 2x^{g-1}/(x^{g-1}+1)$.

If we set x to be the root of the equation $x^{g-1} = (x+1)/(x-1)$, then $2x^{g-1}/(x^{g-1}+1) = 1 + 1/x$, implying that $\mathcal{A}$ is at least $(1+1/x)$-competitive. □

For $g = 2$, this implies a lower bound of $\sqrt{2}$ and for larger values of g, we have an increasing lower bound approaching 2 (see Figure 3).

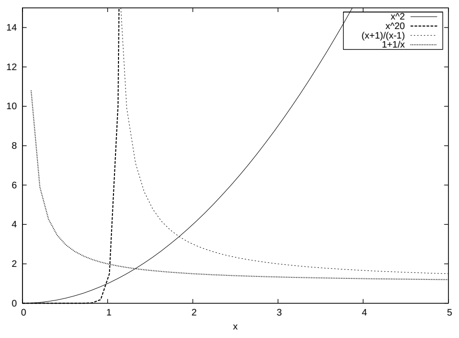

Figure 3: Lower bound for MinBusy on one-sided clique instances. The line x and the curve $(x+1)/(x-1)$ intersect at $x = 1+\sqrt{2}$ and for this value $1+1/x = \sqrt{2}$. Note that x^g and $(x+1)/(x-1)$ intersect at $x \geq 1$, implying $1 + 1/x \leq 2$.

3.3 Clique Instances

We present the online algorithm LeftOrRight($\mathcal{A}$) for clique instances, which assumes the knowledge of the time t common to all jobs, and takes as parameter an online algorithm $\mathcal{A}$ for one-sided clique instances. Roughly speaking LeftOrRight runs two copies of $\mathcal{A}$: $\mathcal{A}_L$ for the jobs with longer left side (from r_J to t) and $\mathcal{A}_R$ for jobs with longer right side (from t to c_J). We give the details in Algorithm 4.

The correctness of LeftOrRight($\mathcal{A}$) follows from that of $\mathcal{A}$. Lemma 11 asserts that its competitive ratio is at most twice that of $\mathcal{A}$. Roughly speaking, the lemma stems from the fact that the sum of the optimal cost for the sets of jobs fed to $\mathcal{A}_L$ and $\mathcal{A}_R$ is at most that the optimal cost for $\mathcal{J}$.

Algorithm 4 LeftOrRight($\mathcal{A}, t$)

```
1: Two copies of the online algorithm $\mathcal{A}$ are run, $\mathcal{A}_L$ for some jobs on the left of the common
   time $t$ and $\mathcal{A}_R$ for some jobs on the right.
2: When a job $J$ with interval $[r_J, c_J]$ arrives, compute the lengths to the left and right to the
   common time $t$, i.e., the two quantities $t - r_J$ and $c_J - t$.
3: if $t - r_J \geq c_J - t$ then
4:     Create an input job $J_L$ with interval $[r_J, t]$
5:     Feed $J_L$ to $\mathcal{A}_L$.
6:     Assign $J$ to the machine that $\mathcal{A}_L$ assigns $J_L$
7: else
8:     Create an input job $J_R$ with interval $[t, c_J]$
9:     Feed $J_R$ to $\mathcal{A}_R$.
10:    Assign $J$ to the machine that $\mathcal{A}_R$ assigns $J_R$.
11: end if
```

Lemma 11. *If $\mathcal{A}$ is c-competitive for one-sided clique instances of* MinBusy*, then* LeftOrRight($\mathcal{A}, t$) *is $2 \cdot c$-competitive for clique instances of* MinBusy *where t is any common point to all the jobs.*

Proof. Consider a given set $\mathcal{J}$ of jobs. For each job $J \in \mathcal{J}$ with interval $[r_J, c_J]$, let J_L be the job with interval $[r_J, t]$ and J_R be the job with interval $[t, c_J]$. Let $\mathcal{J}_L = \{J_L \mid J \in \mathcal{J}\}$

and $\mathcal{J}_R = \{J_R \mid J \in \mathcal{J}\}$. Let $\mathcal{J}'_L \subseteq \mathcal{J}_L$ be the set of jobs that are fed to $\mathcal{A}_L$, and $\mathcal{J}_1$ be the corresponding subset of $\mathcal{J}$. Similarly, we define $\mathcal{J}'_R$ and $\mathcal{J}_2$ for $\mathcal{A}_R$. Note that $\mathcal{J}'_R$ and $\mathcal{J}'_L$ are one-sided clique instances by the way they are built by the algorithm, and by the fact that $t \in J$ for every $J \in \mathcal{J}$.

As $\mathcal{J}'_L \subseteq \mathcal{J}_L$, any schedule for $\mathcal{J}_L$ defines a schedule for $\mathcal{J}'_L$ of same or smaller busy time, implying that $\mathit{cost}^*(\mathcal{J}'_L) \leq \mathit{cost}^*(\mathcal{J}_L)$. Similarly, $\mathit{cost}^*(\mathcal{J}'_R) \leq \mathit{cost}^*(\mathcal{J}_R)$. Furthermore, since $\mathcal{J}_L$ and $\mathcal{J}_R$ have no time in common except t, a schedule for $\mathcal{J}$ defines two schedules, one for $\mathcal{J}_L$ and one for $\mathcal{J}_R$. Therefore, we have

$$\mathit{cost}^*(\mathcal{J}) \geq \mathit{cost}^*(\mathcal{J}_L) + \mathit{cost}^*(\mathcal{J}_R). \tag{4}$$

Let s_L and s_R be the schedules returned by $\mathcal{A}_L$ and $\mathcal{A}_R$ on $\mathcal{J}'_L$ and $\mathcal{J}'_R$, respectively. Since $\mathcal{A}$ is c-competitive for one-sided clique instances, $\mathit{cost}^{s_L}(\mathcal{J}'_L) \leq c \cdot \mathit{cost}^*(\mathcal{J}'_L)$ and $\mathit{cost}^{s_R}(\mathcal{J}'_R) \leq c \cdot \mathit{cost}^*(\mathcal{J}'_R)$, implying

$$\mathit{cost}^{s_L}(\mathcal{J}'_L) + \mathit{cost}^{s_R}(\mathcal{J}'_R) \leq c \cdot (\mathit{cost}^*(\mathcal{J}_L) + \mathit{cost}^*(\mathcal{J}_R)) \leq c \cdot \mathit{cost}^*(\mathcal{J}).$$

For every job J, LeftOrRight$(\mathcal{A}, t)$ selects J_L or J_R (whichever longer) to feed to one of the two copies of algorithm $\mathcal{A}$. Hence, $\mathit{cost}^s(\mathcal{J}_1) \leq 2 \cdot \mathit{cost}^{s_L}(\mathcal{J}'_L)$ and $\mathit{cost}^s(\mathcal{J}_2) \leq 2 \cdot \mathit{cost}^{s_R}(\mathcal{J}'_R)$. Then,

$$\mathit{cost}^s(\mathcal{J}) = \mathit{cost}^s(\mathcal{J}_1) + \mathit{cost}^s(\mathcal{J}_2) \leq 2(\mathit{cost}^{s_L}(\mathcal{J}'_L) + \mathit{cost}^{s_R}(\mathcal{J}'_R)) \leq 2c \cdot \mathit{cost}^*(\mathcal{J}) \ . \tag{5}$$

□

By Theorem 9 and Lemma 11, the following corollary follows.

Corollary 12. *LeftOrRight(BucketGreedy$(1+\varphi), t)$ is $2(1+\varphi)$-competitive for clique instances of* MinBusy *where $\varphi = \frac{1+\sqrt{5}}{2}$ is the Golden Ratio and t is any point common to all the jobs.*

We note that if t is not known a priori, we can use a modified algorithm that divides jobs into buckets according to their length $c_J - r_J$. As $\frac{c_J - r_j}{2} \leq \max\{t - r_J, c_J - t\} \leq c_J - r_j$ the index of the bucket decided by the modified algorithm differs from the index decided by the original algorithm by at most 1, thus loosing a factor of at most 2 in the performance with respect to the original algorithm.

4 Throughput Maximization - MaxThroughput problem

4.1 Basic Results

In this section we consider the MaxThroughput problem $(\mathcal{M}, \mathcal{J}, g, T)$. An input instance is said to be *feasible* if there is a schedule that assigns all the jobs with total machine busy time at most T, i.e., $\mathit{tput}^* = |\mathcal{J}|$. The following proposition asserts that the problem does not admit a small competitive ratio for general instances.

Proposition 13. *No online algorithm for* MaxThroughput *is better than strictly gT-competitive, while there exists a strictly gT-competitive online algorithm.*

Proof. To simplify the discussion, we assume that time is divided into unit-length intervals. A simple algorithm that schedules every possible job using any machine is strictly gT-competitive because it schedules at least one job and at most gT jobs can be scheduled.

We then show a matching lower bound. Consider any online algorithm $\mathcal{A}$. The adversary releases a job with interval $[0, T]$. $\mathcal{A}$ has to schedule this job, and spends a busy time of T, otherwise, its competitive ratio is unbounded. The adversary then issues gT jobs, all with length 1, g of which with intervals $[T, T+1]$, g with $[T+1, T+2]$, $\cdots$, g with $[2T-1, 2T]$. $\mathcal{A}$ cannot schedule any of these jobs while an optimal solution would only schedule these jobs, and the competitive ratio is gT. □

Following Proposition 13, from now on we consider only feasible instances. Furthermore, we focus on feasible one-sided clique instances. Proposition 14 asserts that both the simple greedy algorithm and algorithm BucketGreedy do not admit a small competitive ratio. This suggests that some form of admission control seems necessary and we will give such an algorithm in the sequel.

Proposition 14. *For feasible one-sided clique instances of* MaxThroughput (i) *the simple greedy algorithm is* $\Omega(T)$*-competitive, and* (ii) BucketGreedy *is* $\Omega(\frac{T}{\log T})$*-competitive even when* $g = 2$*.*

Proof. (i) The simple greedy algorithm simply assigns the first machine that the new job can be assigned, i.e., the resulting schedule is still valid. The following adversary shows that the algorithm is $\Omega(T)$-competitive. Let $g = T/2$. The adversary releases g groups of jobs each with g jobs and all with the same arrival time (one-sided clique instance). For each group, the first job has length $T-g$ and the next $g-1$ jobs has length 1. Note that this instance is feasible since one can schedule all the long jobs in one machine with busy time $T-g$ and the other $g(g-1)$ jobs in $g-1$ machines with total busy time $g-1$. So the total busy time of all machines is $T-1$. The simple greedy algorithm schedules jobs in the order of groups and would schedule the same group in the same machine, Then only two groups would be scheduled since $T-g = T/2$, resulting in a total busy time of T. Therefore, the competitive ratio is at least $g^2/2g = T/4$.

(ii) The following adversary shows that BucketGreedy is $\Omega(\frac{T}{\log T})$-competitive when $g = 2$. Release $(\log T - 1)$ jobs of length 2, 2^2, $\cdots$, $T/2$ followed by $2T/3$ jobs with length 1. This is a feasible instance since one can schedule the first $(\log T - 1)$ jobs in $(\log T - 1)/2$ machines with total busy time $2T/3$ and the remaining $2T/3$ short jobs on $T/3$ machines with total busy time $T/3$. On the other hand, BucketGreedy only schedules the first $(\log T - 1)$ jobs each on a different machine and two short jobs on one machine, with a total busy time of T. The competitive ratio is therefore $\Omega(\frac{T}{\log T})$. □

4.2 Lower Bounds for Feasible One-Sided Clique Instances

In this section we show two lower bounds on the competitive ratio of any online algorithms for feasible one-sided clique instances, one for the absolute competitive ratio for any fixed value of g and the other for the general case

Lemma 15. *Let* $\mathcal{A}$ *be a* c*-competitive online algorithm for feasible one-sided clique instances of* MaxThroughput*, with an additive constant* b*. If* $c(b+1) < g$ *then*

$$c \geq 2 - \frac{4b+2}{g+2b+1} \ .$$

Proof. By the definition of competitiveness, for any prefix of the input $\mathcal{A}$ returns a schedule s such that $tput^s \geq tput^*/c - b$. We describe an adaptive adversary for $\mathcal{A}$. Let T be a large even integer. Release $c(b+1)$ jobs each of length $T/2+1$, followed by $g - c(b+1)$ jobs each of length $T/2-1$. $\mathcal{A}$ has to schedule at least one of the first $c(b+1)$ jobs, otherwise after the first $c(b+1)$ jobs, $tput^s = 0 < 1 = c(b+1)/c - b$. We consider two cases.

Case 1. $\mathcal{A}$ uses two machines, one with busy time $T/2+1$ and the other $T/2-1$ and $cost^s = T$. Then $\mathcal{A}$ cannot use a third machine, i.e. $tput^s \leq 2g$ regardless of the rest of the input. The adversary further releases $g(T/2-1)$ jobs with length 1. One can schedule all the jobs by assigning the first g jobs to one machine with busy time $T/2+1$ and the rest on $T/2-1$ machines with a total busy time of $T/2-1$. Therefore $tput^* = g + g(T/2-1) = gT/2$, and the competitive ratio is $tput^*/(tput^s + b) \geq \frac{T}{4+2b/g}$. For any g and b by choosing an arbitrarily large even T, c is unbounded.

Case 2. $\mathcal{A}$ uses only one machine. The adversary then releases $g - c(b+1)$ more jobs each of length $T/2+1$. $\mathcal{A}$ cannot use a second machine for these jobs. Therefore $tput^s \leq g$. One can schedule all jobs by assigning the same machine to the first $c(b+1)$ and the last $g - c(b+1)$ jobs using a busy time of $T/2+1$ and leaving a busy time of $T/2-1$ for the rest. The competitive ratio is $c \geq tput^*/(tput^s + b) \geq (2g - c(b+1))/(g+b)$. Then we have $c(g+2b+1) \geq 2g$ and the result follows. ☐

The condition in Lemma 15 holds when $g \gg b$ or $b = 0$, leading to two lower bounds

Theorem 16. *For feasible one-sided clique instances of* MaxThroughput, *any online algorithm has* (i) *a competitive ratio of at least* 2, (ii) *an absolute competitive ratio of at least* $2 - \frac{2}{g+1}$ *even if g is fixed.*

Proof. Assume by contradiction that there is an algorithm with a competitive ratio $c = 2 - \epsilon$ for some $\epsilon \in (0, 1]$.

(i) Consider instances with $g = 1 + \lceil \max\left\{c(b+1), 2(2b+1)(\frac{2}{\epsilon} - 1)\right\} \rceil$. Then $g > c(b+1)$, thus satisfies the condition of Lemma 15. Therefore $c \geq 2 - \frac{4b+2}{g+2b+1} = 2 - \frac{\epsilon}{2} > 2 - \epsilon = c$, a contradiction.

(ii) As we consider absolute competitive ratio, we have $b = 0$. Therefore, $g \geq 2 > c = c(b+1)$, i.e. the condition of Lemma 15 is satisfied. We conclude that $c \geq 2 - \frac{4b+2}{g+2b+1} = 2 - \frac{2}{g+1}$ ☐

4.3 Online Algorithm for Feasible One-Sided Clique Instances

In this section we propose an online algorithm called BalanceBudget that achieves a constant asymptotic competitive ratio for every fixed g. Since the given instance is feasible, we have $tput^* = |\mathcal{J}|$.

We start by defining a few terms and notations for the algorithm. We categorize the input jobs into buckets[1] according to their lengths. Namely, given a job $J \in \mathcal{J}$ we define $bucket(J)$ as the smallest non-negative integer i such that $\frac{T}{2^{i+1}} < len(J)$ and $\mathcal{J}^{(i)} \overset{def}{=} \{J \in \mathcal{J} \mid bucket(J) = i\}$. In other words we have $\forall J \in \mathcal{J}^{(i)}$, $\frac{T}{2^{i+1}} < len(J) \leq \frac{T}{2^i}$. We also define the following two dynamic variables (i.e., their values depend on the state of the algorithm).

- T_i: The total busy time incurred by the algorithm to schedule the jobs of $\mathcal{J}^{(i)}$ except its first g jobs in the order of arrival.

[1]Note that the definition of bucket is slightly different from that defined for MinBusy.

- T_i^*: A set of $\lceil \log T \rceil$ variables (one for each bucket) satisfying, (a) $T_i^* \geq 0$, (b) T_i^* is non-decreasing, and (c) $\sum_i T_i^* \leq \mathit{cost}^*$.

In the pseudo code of BalanceBudget (Algorithm 5), accept(J) stands for scheduling J to the machine with smallest index that is under use in bucket i and the schedule continues to be valid after assigning J. If no such machine exists J is assigned a new machine. In this case we also set $\mathcal{J}^{(i)} \leftarrow \mathcal{J}^{(i)} \cup \{J\}$, where i is the bucket J belongs to. reject(J) means that J is not assigned, i.e. $s(J)$ is undefined, and $\mathcal{J}^{(i)}$ remains the same.

Algorithm 5 BalanceBudget

```
When a job $J$ arrives do:
$i \leftarrow bucket(J)$
if $\left|\mathcal{J}^{(i)}\right| \leq g$ then
    if $i \geq 3$ then accept($J$)                                          ▷ (+)
    else reject($J$)
    end if
else
    if $T_i \leq \frac{3}{4}T_i^*$ would hold after accepting $J$ then accept($J$)        ▷ (*)
    else reject($J$)
    end if
end if
```

The analysis of BalanceBudget proceeds as follows. In Lemma 17 we show that, provided a polynomial-time computable function T_i^* (of the state of the algorithm) that satisfies conditions (a)-(c), BalanceBudget returns a valid schedule with total busy time at most T. We then show in Lemma 18 the existence of such a function T_i^*. Furthermore, we show that the throughput in every bucket of BalanceBudget is at least a constant fraction of the number of jobs in that bucket. Then. in Theorem 20 we can conclude the competitive ratio of BalanceBudget. We start by stating Lemma 17 that BalanceBudget returns a valid schedule with nice properties.

Lemma 17. BalanceBudget *returns a valid schedule with total busy time at most T, provided that there is a polynomial-time computable function T_i^* satisfying the above mentioned conditions (a)-(c).*

Proof. Clearly, the semantics of accept guarantees that the schedule is valid, because an existing machine is used only if it can accommodate the new job. It remains to show that the total busy time is at most T.

Initially $T_i = 0 \leq \frac{3}{4}T_i^*$. Observe that T_i increases only after an accept(J) marked with a (*) in Algorithm 5 since T_i does not account for the first g jobs accepted. However in this case the algorithm ensures that $T_i \leq \frac{3}{4}T_i^*$ holds, thus we conclude that for all i, $T_i \leq \frac{3}{4}T_i^*$ at all times during the execution of BalanceBudget.

For the first g jobs of $\mathcal{J}^{(i)}$ BalanceBudget uses one machine with busy time at most $T/2^i$ except for $i < 3$ where none of the first g jobs is accepted and thus the busy time incurred for these jobs is zero. For the other jobs in $\mathcal{J}^{(i)}$ BalanceBudget incurs a busy time of $T_i \leq \frac{3}{4}T_i^*$. Therefore the total busy time of BalanceBudget is at most $\mathit{cost}^s \leq \sum_{i\geq 3} \frac{T}{2^i} + \sum T_i \leq \sum_{i\geq 3} \frac{T}{2^i} + \frac{3}{4}\sum T_i^* < \frac{T}{4} + \frac{3}{4}T = T$. The last inequality is due to $\sum T_i^* \leq \mathit{cost}^* \leq T$. □

The following lemma presents a function T_i^* that satisfies the conditions (a)-(b) required by Lemma 17.

Lemma 18. *The function*

$$T_i^* \overset{def}{=} \left\lceil \frac{\max(\left|\mathcal{J}^{(i)}\right| - (g-1), 0)}{g} \right\rceil \frac{T}{2^{i+1}} \tag{6}$$

is polynomial time computable, and satisfies the requirements (a)-(c) stated for T_i^*.

Proof. T_i^* is clearly polynomial time computable, non-negative, and also non-decreasing because $\left|\mathcal{J}^{(i)}\right|$ is non-decreasing. It remains to show that $\sum_i T_i^* \leq cost^*$.

It is easy to show (for details see Lemma 3.3 in [26]) that an optimal solution can be obtained by sorting the jobs of $\mathcal{J}$ in decreasing order of length, dividing into consecutive sets of size g with the last set possibly containing less than g jobs and then scheduling each set using a different machine. Note that in this ordering the jobs appear in non-decreasing order of their buckets. The jobs assigned to the same machine may span multiple consecutive buckets. We charge the busy time $cost^*$ of an optimal solution to every bucket, so that the busy time of every machine is charged to the first bucket containing a job of this machine. Consider the optimal solution. In each bucket, at most $g-1$ jobs can share a machine with jobs of previous buckets, therefore at least $\max(\left|\mathcal{J}^{(i)}\right| - (g-1), 0)$ jobs are assigned machines not used in previous buckets. This corresponds to $\left\lceil \frac{\max(\left|\mathcal{J}^{(i)}\right| - (g-1), 0)}{g} \right\rceil$ machines, and each machine has a busy time of at least $\frac{T}{2^{i+1}}$. Therefore, $cost_i^* \geq \sum_i \left\lceil \frac{\max(\left|\mathcal{J}^{(i)}\right| - (g-1), 0)}{g} \right\rceil \frac{T}{2^{i+1}} = \sum_i T_i^*$. □

To analyze the competitiveness of BalanceBudget, let $tput_i$ be the throughput in bucket i, i.e. the number of jobs in $\mathcal{J}^{(i)}$ scheduled by BalanceBudget. In each bucket i BalanceBudget assigns $tput_i - g$ jobs to $\lceil (tput_i - g)/g \rceil$ machines each with busy time at most $T/2^i$, therefore:

$$T_i \leq \left\lceil \frac{tput_i - g}{g} \right\rceil \frac{T}{2^i} = \left(\left\lceil \frac{tput_i}{g} \right\rceil - 1 \right) \frac{T}{2^i}.$$

Using the above inequality we prove the following lemma.

Lemma 19. *For every fixed g, there exists a constant $c(g)$ with $2/9 < c(g) < 1$ such that $\forall i \geq 3, tput_i \geq c(g) \cdot \left|\mathcal{J}^{(i)}\right|$.*

Proof. Let $c < 1$ be a constant to be fixed later. Note that for $i \geq 3$ the first g jobs are always scheduled by BalanceBudget. If $\left|\mathcal{J}^{(i)}\right| \leq g$ then $tput_i = \left|\mathcal{J}^{(i)}\right| \geq c \cdot \left|\mathcal{J}^{(i)}\right|$, otherwise if $g < \left|\mathcal{J}^{(i)}\right| \leq g/c$ then $tput_i \geq g \geq c \cdot \left|\mathcal{J}^{(i)}\right|$. In both cases the claim is correct.

Otherwise $\left|\mathcal{J}^{(i)}\right| > g/c$ and we assume, by way of contradiction, that the claim does not hold. Consider the first step during the algorithm that the claim does not hold, i.e. $tput_i < c \cdot \left|\mathcal{J}^{(i)}\right|$. Suppose that the decision of the algorithm at this step is accept. Then the number of jobs scheduled by the algorithm up to (but not including) this step is $tput_i - 1 < c \cdot \left|\mathcal{J}^{(i)}\right| - 1 < c(\left|\mathcal{J}^{(i)}\right| - 1)$, i.e. the claim did not hold one step before, a contradiction. Therefore the decision of the algorithm has to be reject. In this case the value of $tput_i$ does not change from the last step. We denote by $\widetilde{T}_i$ the value that T_i would take if the decision would be ACCEPT. We have:

$$\begin{aligned}\widetilde{T}_i &> \frac{3}{4}T_i^* \\ \widetilde{T}_i &\leq \left(\left\lceil \frac{tput_i + 1}{g} \right\rceil - 1\right)\frac{T}{2^i}\end{aligned}$$

The first inequality is due to the fact that this step was REJECT. The second inequality is because the number of scheduled jobs would be $tput_i + 1$, the first g jobs do not affect T_i, and the length of each job is at most $T/2^i$. By combining the last two inequalities with Equation (6) we conclude

$$\begin{aligned}\frac{tput_i + 1}{g} &\geq \left\lceil \frac{tput_i + 1}{g} \right\rceil - 1 \geq \frac{3}{8}\left(\left\lceil \frac{\left|\mathcal{J}^{(i)}\right| + 1}{g} \right\rceil - 1\right) \geq \frac{3}{8}\frac{\left|\mathcal{J}^{(i)}\right| - g + 1}{g} \\ \therefore \quad tput_i &\geq \frac{3}{8}\left(\left|\mathcal{J}^{(i)}\right| - g - \frac{5}{3}\right)\end{aligned}$$

On the other hand as $\left|\mathcal{J}^{(i)}\right| > g/c$ we have $\left|\mathcal{J}^{(i)}\right| - g - 5/3 > \left|\mathcal{J}^{(i)}\right|(1 - (1 + 5/3g)c)$, thus

$$tput_i > \frac{3}{8}\left|\mathcal{J}^{(i)}\right|\left(1 - \left(1 + \frac{5}{3g}\right)c\right)$$

If c is chosen such that $\frac{3}{8}(1 - (1 + \frac{5}{3g})c) \geq c$ then $tput_i > c \cdot \left|\mathcal{J}^{(i)}\right|$, a contradiction. It is easy to verify that this holds for any $c \leq \frac{3g}{11g+5}$. The expression on the right hand side is monotonically increasing and attains the minimum of $2/9$ for $g = 2$. □

With Lemmas 17-19, we prove the following theorem.

Theorem 20. *For any g,* BALANCEBUDGET *is a $c(g)$-competitive online algorithm for feasible one-sided clique instances of* MAXTHROUGHPUT*, where $c(g) \leq 9/2$.*

Proof. Assume that BALANCEBUDGET is given an extra budget of busy time to schedule the first g jobs of the first 3 buckets. This modification does not change the value of T_i since T_i ignores the first g jobs, therefore it does not affect the execution of the algorithm besides the fact that now it schedules the first g jobs in every bucket. Let $tput'_i$ be the throughput of the modified algorithm for bucket i. For every $i \geq 3$ we have $tput'_i = tput_i$ and for $i < 3$ we have $tput'_i \leq tput_i + g$. For the modified algorithm Lemma 19 implies that $\forall i \geq 0, tput'_i \geq c(g) \cdot \left|\mathcal{J}^{(i)}\right|$. Summing over all buckets we get

$$\sum_{i \geq 0} tput'_i \geq c(g) \sum_{i \geq 0} \left|\mathcal{J}^{(i)}\right| = c(g) \cdot |\mathcal{J}| = c(g) \cdot tput^*.$$

On the other hand $\sum_{i \geq 0} tput'_i \leq \sum_{i \geq 0} tput_i + 3g$, therefore $tput^s = \sum_{i \geq 0} tput_i \geq c(g) \cdot tput^* - 3g$. □

Note that BALANCEBUDGET can be modified so that it gets some integer parameter $\beta \geq 2$ to indicate how many buckets from which we do not accept the first g paths (marked (+) in Algorithm 5). In the above presentation we assumed, for simplicity that $\beta = 3$. In general the competitive ratio is a decreasing function of β, but the additive constant of $\beta \cdot g$ increases with β.

5 Applications to Optical Networks and Extensions

In this section we extend techniques and results on MinBusy to optical networks. Furthermore we discuss the performance of some of our algorithms with respect to the number of machined used. We first give some background on a coloring problem in optical networks with traffic grooming. In the context of Wavelength Division Multiplexing (WDM) Optical networks, one is given a network and a set of paths on it and each such path (called a *lightpath*) has to be assigned a color. In such a coloring a set of paths that is assigned the same color has to satisfy the following two conditions:

- The load condition: The number of such paths that use the same edge is at most g, where g is an integer called the *grooming factor*.
- The no-split condition: The union of the paths cannot contain split nodes, i.e. it is a graph of maximum degree 2.

Clearly, when the graph is a path, the second condition is satisfied by any coloring. It is easy to see that the first condition is similar to the validity condition for schedules. Indeed, when the graph is a path, a lightpath (resp. color, coloring) is analogous to a job (resp. machine, schedule) and the satisfaction of load condition by a coloring is analogous to the validity of a schedule.

Every set of lightpaths that is assigned some color, uses a separate set of devices called *regenerators*. The number of regenerators used by this set is proportional to the number of internal vertices used by these paths. In our terminology this is exactly the span of this set. Therefore the problem of minimizing the number of regenerators in such a network is equivalent to the problem of minimizing total machine busy time, and the results on MinBusy can be applied directly to optical networks when the topology is a path.

In the sequel we extend the definitions of the problems MinBusy and MaxThroughput to general topologies, while keeping the above analogy in mind. We then extend some of the results in previous sections to the ring and tree topologies. Note that in a general topology, a lightpath (a job) can be any path in the input graph and the measurement is still the span of the set of jobs assigned to a machine. We give the detailed definitions as follows.

Problem definition. We provide new definitions of the terms defined in Section 2 wherever they differ from the original ones. The instances of our problems contain a graph $G = (V(G), E(G))$, and a set of jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ where each job is a simple path on G. Given a job J, $len(J)$ denotes its length on G. For a set $\mathcal{Q}$ of jobs, $SPAN(\mathcal{Q})$ is the graph induced by the union of the paths in $\mathcal{Q}$, and $span(\mathcal{Q}) \overset{def}{=} |E(SPAN(\mathcal{Q}))|$.

A schedule s of $\mathcal{J}$ is *valid* if it satisfies the following two conditions:

- The *load* condition: For any edge $e \in E(G)$ and machine M_i, the number of jobs processed by M_i and using e is at most g.
- The *no-split* condition: The set of jobs assigned to any machine M_i satisfies $SPAN(\mathcal{J}_i^s)$ is a graph with maximum degree at most 2.

5.1 General Instances in Ring Topology

Algorithm BucketFirstFit (in Section 3.1.2) can be shown to have the same competitive ratio, i.e. $\min(g, 5 \log len_{max})$ in ring topology. It can be verified that all the arguments given therein hold for the ring topology too. Yet, we present here a slightly weaker but more general result.

Lemma 21. *For* MinBusy*, if there is a c-competitive algorithm for one-sided clique instances in path topology, then there is a* $4c$*-competitive algorithm for clique instances in ring topology.*

Proof. The proof goes in the same lines of Lemma 11. It can be verified that all the arguments therein hold for the ring topology, except Inequality (4) that should be replaced by

$$cost^*(\mathcal{J}) \geq \max\{cost^*(\mathcal{J}_L), cost^*(\mathcal{J}_R)\}$$

doubing the factor of the right hand side of Inequality (5). □

Corollary 22. *There is a* $(4(1+\varphi))$*-competitive algorithm for clique instances of* MinBusy *in ring topologies, where* $\varphi = \frac{1+\sqrt{5}}{2}$ *is the Golden Ratio.*

Lemma 23. *For* MinBusy*, if there is a* c_p*-competitive algorithm for general instances in path topology, and there is a* c_c*-competitive algorithm for clique instances in ring topology then there is a* $(c_p + c_c)$*-competitive algorithm for ring topology.*

Proof. Let $\mathcal{AC}$ be a c_c-competitive online algorithm for clique instances in ring topology with an additive constant b_c, and let $\mathcal{AP}$ be a c_p-competitive online algorithm for path topology with an additive constant b_p. Consider an online algorithm that given an instance $(G, \mathcal{J}, g)$ chooses some arbitrary edge e of the ring G, schedules the jobs $\mathcal{J}_e$ containing e using algorithm $\mathcal{AC}$, and schedules the rest of the jobs using algorithm $\mathcal{AP}$ with a different set of colors.

An optimal solution for an instance costs at least as much as an optimal solution for any of its sub-instances, therefore $cost^*(\mathcal{J}) \geq \max\{cost^*(\mathcal{J}_e), cost^*(\mathcal{J} \setminus \mathcal{J}_e)\}$. Let s_C and s_P be the schedules returned by $\mathcal{AC}$ and $\mathcal{AP}$ respectively. Then

$$\begin{aligned}
cost^s(\mathcal{J}) &= cost^{s_C}(\mathcal{J}_e) + cost^{s_P}(\mathcal{J} \setminus \mathcal{J}_e) \\
&\leq c_c \cdot cost^*(\mathcal{J}_e) + b_c + c_p \cdot cost^*(\mathcal{J} \setminus \mathcal{J}_e) + b_p \\
&\leq (c_c + c_p) \cdot \max\{cost^*(\mathcal{J}_e), cost^*(\mathcal{J} \setminus \mathcal{J}_e)\} + b_c + b_p \\
&\leq (c_c + c_p) \cdot cost^*(\mathcal{J}) + (b_c + b_p) \ .
\end{aligned}$$

Therefore, the algorithm is $(c_c + c_p)$-competitive. □

Combining Corollaries 7 and 22 and Lemma 23, we have the following corollary.

Corollary 24. *There is a* $(\min\{g, 5\log len_{max}\} + O(1))$*-competitive algorithm for* MinBusy *in ring topology.*

5.2 One-Sided Clique Instances in Tree Topology

In this section we consider MinBusy on tree topology. We focus on one-sided clique instances in which all the jobs in $\mathcal{J}$ share a common edge e incident to a leaf of G. We present algorithm NoSplitBucket (Algorithm 6), that partitions the jobs in $\mathcal{J}$ into disjoint subsets $\mathcal{S}_1, \mathcal{S}_2, \cdots$ such that for every j, the jobs in $\mathcal{S}_j$ satisfy the no-split condition, in other words, each $\mathcal{S}_j$ is like an input instance on a path topology and if $g = \infty$, all the jobs in $\mathcal{S}_j$ can be assigned to

Algorithm 6 NoSplitBucket(α)

```
1: When a job $J$ arrives:
2: Find the first no-split subset $\mathcal{S}_j$ such that $\mathcal{S}_j \cup \{J\}$ satisfies the no-split condition. If there
   is no such subset, start a new one.
3: Determine the bucket $i$ for job $J$ according to the parameter $\alpha$.
4: If bucket $i$ of $\mathcal{S}_j$ has no current machine, then choose an unused machine arbitrarily and
   make it the current machine of bucket $i$ of $\mathcal{S}_j$.
5: If there are already $g$ jobs assigned to the current machine of bucket $i$ of $\mathcal{S}_j$, then choose an
   unused machine arbitrarily and make it the current machine of bucket $i$.
6: $s(J) \leftarrow$ the current machine of bucket $i$ of $\mathcal{S}_j$.
```

the same machine. Within each $\mathcal{S}_j$, the algorithm employs the same classification of jobs into buckets as the algorithm BucketGreedy in Section 3.2.1.

The correctness of the algorithm follows immediately from Steps 2 and 5. Step 2 guarantees that the no-split condition is never violated, and Step 5 guarantees that the load condition is never violated. We now analyze the performance of the algorithm. First, we rearrange the no-split subsets by non-increasing order of the length of the longest job in each subset. Let L_j be the length of the longest job in $\mathcal{S}_j$ and b_j be the bucket that this longest job falls into. Then, $L_1 \geq L_2 \geq \cdots$, and $\alpha^{b_j} \leq L_j < \alpha^{b_j+1}$. For $i, j \geq 0$, let the number of paths in bucket i of $\mathcal{S}_j$ be $q_{i,j} \cdot g + r_{i,j}$, where $0 \leq r_{i,j} < g$. Then we have

$$\begin{aligned}
cost^s(\mathcal{J}) &= \sum_j cost^s(\mathcal{S}_j) \\
&\leq \sum_j \left(\sum_{i=0}^{b_j-1} (q_{i,j}+1)\alpha^{i+1} + (q_{b_j,j}+1)L_j \right) \\
&\leq \sum_j \sum_{i=0}^{b_j} q_{i,j}\alpha^{i+1} + \sum_j \left(\sum_{i=0}^{\infty} L_j/\alpha^i + L_j \right) \\
&\leq \sum_{i,j} q_{i,j}\alpha^{i+1} + \sum_j L_j \frac{2\alpha-1}{\alpha-1} \quad .
\end{aligned}$$

As NoSplitBucket finds a no-split subset greedily, for any $j_1 \neq j_2$, the longest jobs in $\mathcal{S}_{j_1}$ and $\mathcal{S}_{j_2}$ cannot be assigned to the same machine in any schedule, in particular in an optimal schedule s^*. Together with the grooming bound, we have two bounds on the cost of s^*: $cost^* \geq \sum_j L_j$ and $cost^* \geq \sum_{i,j} q_{i,j}\alpha^i$. Hence, $\frac{cost^s}{cost^*} \leq \alpha + \frac{2\alpha-1}{\alpha-1}$. The ratio attains the minimum value of 5 when $\alpha = 2$. We conclude that NoSplitBucket(2) is 5-competitive.

On the other hand, when $\alpha = 2$, an instance with g jobs satisfying the no-split condition and having lengths $2, 2^2, \cdots, 2^g$ forces NoSplitBucket(2) to assign every job to a different machine while s^* can assign all of them to the same machine. In this case, $\frac{cost^s}{cost^*} \geq 2 - \frac{1}{2^g}$. We thus conclude

Theorem 25. *Algorithm* NoSplitBucket(2) *is* 5*-competitive and not better than* 2*-competitive, for one-sided clique instances of* MinBusy *on a tree topology.*

5.3 Minimization of Number of Machines

We note that the (offline) problem of minimizing the number of machines can be solved in polynomial time. Let ω be the clique number of the interval graph, i.e. the maximum number of jobs that have to be processed at some given time t. Clearly any schedule has to use $\lceil \omega/g \rceil$ machines. This can be achieved by coloring the intervals with ω colors and assigning g color classes to every machine. Therefore unless P = NP, there are instances of MinBusy having solutions with a minimum number of machines that are not optimal. On the other hand there are optimal solutions of MinBusy that do not use the smallest number of machines. Indeed, consider the instance consisting of g identical jobs $[0,1]$, $g(g-1)$ identical jobs $[0,2]$ and g identical jobs $[1,2g+1]$. These jobs can be scheduled using g machines each of which is assigned one $[0,1]$ job, one $[1,2g+1]$ job and $g-1$ $[0,2]$ jobs. However, the following schedule s^* that assigns g identical jobs to every machine uses $g+1$ machines and it is optimal for MinBusy with a total busy time $cost(s^*) = 1 + 2g + 2(g-1) = 4g-1$. In order to see that s^* is optimal, first note that every optimal solution must assign the $[1,2g+1]$ jobs to one machine, because otherwise its cost would be at least $4g$. As the other jobs constitute a one sided clique instance the optimum is to assign the $[0,1]$ jobs to one machine and $[0,2]$ jobs to $g-1$ machines.

In this section we provide a short analysis showing that our algorithms have good competitive ratio also for the problem of minimizing the number of machines. We first analyze the performance of BucketFirstFit(α). The following lemma analyzes its behaviour in one bucket.

Lemma 26. *Let $\mathcal{J}^{(i)}$ be the set of jobs in bucket i and s be a schedule returned by* BucketFirstFit *using m_i machines for the jobs $\mathcal{J}^{(i)}$. Let s^* be a solution using m_i^* machines for the same jobs. Then $m_i \leq \lceil \alpha + 1 \rceil m_i^*$.*

Proof. For simplicity we assume that α is integral, and that without loss of generality the jobs of $\mathcal{J}^{(i)}$ are assigned to the first m_i machines. Consider a job $J \in \mathcal{J}_{m_i}$, i.e. assigned to machine m_i. Let $\mathcal{J}_J$ be the jobs of $\mathcal{J}^{(i)}$ intersecting J. By the behavior of the algorithm, every machine $m_{i'}$ with $i' < i$ is assigned at least g jobs of $\mathcal{J}^{(i)}$ intersecting J. Therefore $|\mathcal{J}_J| \geq g(m_i - 1) + 1$. Let $t_1 = r_J < t_2 < \cdots < t_{\alpha+1} = c_J$ be the times that divide J into α equal length intervals, i.e. of length $len(J)/\alpha$ each. Consider a job $J' \in \mathcal{J}_J$. As J, J' are in the same bucket we have $len(J') \geq len(J)/\alpha$. Therefore J' intersects at least one of the times t_k. Then at least one of these times contains at least $\frac{g(m_i-1)+1}{\alpha+1}$ jobs. We conclude that any solution needs at least $\lceil \frac{g(m_i-1)+1}{g(\alpha+1)} \rceil \geq \frac{m_i}{\alpha+1}$ machines to schedule these jobs. □

By the above lemma and the bound on the total number of buckets, we can prove the following theorem.

Theorem 27. BucketFirstFit(α) *is $(\lceil \alpha + 1 \rceil \log len_{\max} / \log \alpha)$-competitive in terms of the number of machines where $len_{\max}$ is the maximum length of a job.*

As for algorithm BucketGreedy, it is easy to see that it uses the minimum number of machines in each bucket, concluding

Theorem 28. BucketGreedy(α) *is $(\log len_{\max} / \log \alpha)$-competitive for one-sided clique instances, in terms of the number of machines where $len_{\max}$ is the maximum length of a job.*

6 Summary and Future Work

In this work we have studied online busy time optimization problems. We have shown some rather large lower bounds for general instances, and this motivated us to consider special families of instances for which we have shown better online algorithms. This is the first work that deals with online algorithms for this setting, and, as such, it calls for a variety of open problems, as detailed below. Some open problems are closely related to those studied in this work, including:

- For the MinBusy problem, an immediate open question is to close the gaps between the upper and lower bounds for one-sided clique instances and clique instances.
- For the MaxThroughput problem in one-side clique instances, is there a constant-competitive algorithm when g is not fixed? On the other hand is there a lower bound for these instances when g is fixed?
- For the MaxThroughput problem are there better lower bounds for the general instances or is there a constant-competitive algorithm?

More general open problems naturally arise, including:

- Consider jobs having associated benefit and maximize the total benefit of scheduled jobs.
- In this work the jobs are supposed to be processed during the whole period from start time r_J to completion time c_J. We can consider jobs of other characteristics.
  - One may consider jobs that also have a processing time p_J and have to be processed for p_J consecutive time units during the interval $[r_J, c_J]$ (see e.g. [17, 29]).
  - One may also consider *malleable* jobs which can be assigned several machines and the actual processing time depends on the number of machines allocated (see e.g., [23, 29]).
- We assume that each job requires the same amount of capacity ($1/g$) of a machine. An extension is to allow a job requiring different amount of capacity and a machine can process jobs as long as the sum of capacity requirements of its jobs is at most g [17].

As we have mentioned in Section 1, our work is closely related to power-aware scheduling, cloud computing and optical network design. Our problems can be extended to cover more general problems in these three applications.

Power-aware scheduling: As said, machine busy time reflects how long the processor is switched on and how much energy is used. Energy saving can also be achieved via other mechanisms.

- Modern processors support Dynamic Voltage Scaling (DVS) (see, e.g., [16, 24, 33]), where the processor speed can be scaled up or down. The scheduler may speed up the processor to shorten the busy time, resulting in shorter time of processing but higher power usage per time unit. It is interesting to derive algorithms that can make a wise tradeoff.

- We assume that we can use as many machines as we like without any overhead. In reality, switching on a machine from a sleep state requires some energy and it may save energy to leave a machine to idle if jobs will be scheduled on it again soon [1, 7]. To take this advantage, different optimization criteria have to be considered.

Cloud computing: The following extension can be interpreted clearly within the context of problems in cloud computing (see, e.g., [11, 27, 30]) as presented in Section 1. We assume that the machines have identical computing power. An extension is to have different machines types and to allow a job to require a specified list of machines type.

Optical network design: As shown in Section 5 the busy time scheduling problems have a direct application in placement of regenerators in optical network design: In MinBusy we are given a set of paths and a grooming factor g and the objective is to find a valid coloring for all paths with minimum number of regenerators. In MaxThroughput we are also given a budget T and the objective is to find a valid coloring with at most T regenerators maximizing the number of satisfied paths. Some of our results for MinBusy can be extended to other topologies: the result for one-sided clique instances can be extended to tree topologies while the result for clique instances and general instances can be adapted to ring topologies.

References

[1] J. Augustine, S. Irani, and C. Swany. Optimal power-down strategies. In *FOCS*, pages 530–539, 2004.

[2] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, pages 1–23, 2000.

[3] R. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. In *ICALP*, pages 193–193, 2003.

[4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] P. Brucker. *Scheduling Algorithms, 5th ed.* Springer, 2007.

[6] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *IPCO*, pages 401–414, 2006.

[7] S.-H. Chan, T. W. Lam, L.-K. Lee, C.-M. Liu, and H.-F. Ting. Sleep management on multiple machines for energy and flow time. In *ICALP*, pages 219–231, 2011.

[8] B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34:501–507, 2002.

[9] S. Chen, I. Ljubic, and S. Raghavan. The regenerator location problem. *Networks*, 55(3):205–220, May 2010.

[10] J. Y.-T. L. (ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRS Press, 2004.

[11] Y. Fang, F. Wang, and J. Ge. A task scheduling algorithm based on load balancing in cloud computing. In *WISM*, pages 271–277, 2010.

[12] R. Fedrizzi, G. M. Galimberti, O. Gerstel, G. Martinelli, E. Salvadori, C. V. Saradhi, A. Tanzi, and A. Zanardi. A framework for regenerator site selection based on multiple paths. In *OFC*, pages 1–3, 2010.

[13] M. Flammini, A. Marchetti-Spaccamela, G. Monaco, L. Moscardelli, and S. Zaks. On the complexity of the regenerator placement problem in optical networks. *IEEE/ACM Transactions on Networking*, 19(2):498 –511, April 2011.

[14] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *Theor. Comput. Sci.*, 411(40-42):3553–3562, 2010.

[15] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem with respect to ADMs and OADMs. In *EURO-PAR*, August 2008.

[16] J. Kang and S. Ranka. Energy-efficient dynamic scheduling on parallel machines. In *HIPC*, pages 208–219, 2008.

[17] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. In *FSTTCS*, pages 169–180, 2010.

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

[19] M. Y. Kovalyov, C. T. Ng, and T. C. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.

[20] S. O. Krumke, C. Thielen, and S. Westphal. Interval scheduling on related machines. *Computers and Operations Research*, 38(12):1836–1844, 2011.

[21] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science*, 4:445 – 522, 1993.

[22] R. J. Lipton and A. Tomkins. Online interval scheduling. In *SODA*, pages 302–311, 1994.

[23] W. T. Ludwig. *Algorithms for scheduling malleable and nonmalleable parallel tasks.* PhD thesis, 1995.

[24] A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy/power. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(2):270 –276, 2003.

[25] G. B. Mertzios, M. Shalom, , P. W. H. Wong, and S. Zaks. Online regenerator placement. In *OPODIS*, pages 4–17, 2011.

[26] G. B. Mertzios, M. Shalom, A. Voloshin, , P. W. H. Wong, and S. Zaks. Optimizing busy time on parallel machines. In *IPDPS*, pages 238–248, 2012.

[27] A. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *CloudCom*, pages 351–359, 2010.

[28] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.

[29] U. Schwarz. Tightness results for malleable task scheduling algorithms. In *Parallel Processing and Applied Mathematics*, pages 1059–1067. 2008.

[30] W. Shi and B. Hong. Resource allocation with a budget constraint for computing independent tasks in the cloud. In *CloudCom*, pages 327–334, 2010.

[31] N. Vasić, M. Barisits, V. Salzgeber, and D. Kostic. Making cluster applications energy-aware. In *ACDC*, pages 37–42, 2009.

[32] P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *SODA*, pages 830–831, 2003.

[33] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.