Online Optimization of Busy Time on Parallel Machines*
(Extended Abstract)

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

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. 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.

Keywords: interval scheduling, busy time, resource allocation, online algorithms, cost minimization, throughput maximization

* This work was supported in part by the Israel Science Foundation grant No. 1249/08 and British Council Grant UKTELHAI09

1 Introduction

The problem. Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [5, 10, 29]). 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 of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each job, J_j, is associated with an interval $[r_j, c_j]$ during which it should be processed. Also, given is the parallelism parameter $g \geq 1$, that 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_j scheduled to 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 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 the constraint that the total busy time is at most T.

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.

Online algorithms. We consider the online setting, in which the jobs are given one at a time. The algorithm has to decide on whether to schedule the job (in the MaxThroughput problem) and on which machine to schedule it. The decision cannot be changed later, upon arrival of new jobs. An online algorithm is said to be c-competitive for MinBusy if the total busy time is at most c times that of an optimal solution and c-competitive for MaxThroughput if the number of scheduled jobs is at least $1/c$ times that of an optimal solution; in both cases, an additive constant is allowed. When the additive constant is zero, the online algorithm is said to be strictly c-competitive and c is called the absolute competitive ratio.

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., [32] 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, 31]). 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 f 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, 31]).

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 refers to different flavor of the model than the one presented here (see, e.g., [2, 6, 8, 28]). 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.

The complexity of MinBusy was studied in [33], 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 ("proper" instance), and 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 for which a 6-approximation is proposed for clique instances and a polynomial time algorithm is proposed for "proper" clique instances. These special instances have been considered in [17, 26], though under the off-line setting.

Our contribution. We study deterministic online busy time optimization (both 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. 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 $(1 + \varphi)$ respectively, where $\varphi = (1 + \sqrt{5})/2$ is the Golden Ratio, for one sided clique instances, and extension to the clique instances with a blow up of 2 in the ratio.
- A 5-competitive online algorithm for one sided clique instances.

For the MaxThroughput problem we first show that no online algorithm is better than $(gT/2)$-competitive. We therefore consider special instances, for which we show the following:

- Asymptotic and absolute competitive ratios of at least 2 and at least $2 - \frac{2}{g+1}$, respectively, for feasible one sided clique instances.
- A constant competitive online algorithm with ratio depending on g, but at most 9/2, for feasible one-sided clique instances.

Organization of the paper. In Section 2 we present some preliminaries. We consider online busying time minimization and maximization in Sections 3 and 4, respectively. We then conclude in Section 5 with some open problems and further research directions. Most proofs are sketched or moved to the Appendix in this Extended Abstract.

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 relese time and completion time of the job, respectively. We use jobs and time intervals interchangably 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 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 $len(I) \stackrel{def}{=} c_I - r_I$. 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 $SPAN(\mathcal{I}) \stackrel{def}{=} \cup_{I \in \mathcal{I}} I$ and $span(\mathcal{I}) \stackrel{def}{=} len(SPAN(\mathcal{I}))$ 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 $span(\mathcal{I}) \leq len(\mathcal{I})$ and equality holds if and only if $\mathcal{I}$ is a set of pairwise non-overlapping intervals.

A *(partial) schedule* is a (partial) function from 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. $\mathcal{J}_i^s \stackrel{def}{=} s^{-1}(M_i)$. The cost of machine M_i in this schedule is the length of its busy interval, i.e. $busy_i^s \stackrel{def}{=} span(\mathcal{J}_i^s)$. We further denote the set of jobs scheduled by s as $\mathcal{J}^s \stackrel{def}{=} \cup_i \mathcal{J}_i^s$. The *cost* of schedule s is $cost^s \stackrel{def}{=} \sum_i busy_i^s$, and its *throughput* is $tput^s \stackrel{def}{=} |\mathcal{J}^s|$. 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 $tput^s$.

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 interval, 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 cases 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 a 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]. We denote by s^* an optimal schedule (for MinBusy or MaxThroughput). The cost of s^* is denoted by $cost^*$ and its throughput by $tput^*$. An online algorithm 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, 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, A is said to be *strictly* c-competitive and c is its *absolute* competitive ratio.

Basic observations 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:* $cost^s \geq span(\mathcal{J})$,
- *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 1. *For* MinBusy*, any online algorithm is strictly g-competitive.*

The following relationship between MinBusy and MaxThroughput is observed in [26] For the sake of completeness, we give the proof in Appendix B.1.

Proposition 2 ([26]). *There is a polynomial time reduction from the* MinBusy *problem to the* MaxThroughput *problem.*

3 Cost Minimization - MinBusy problem

3.1 General Instances

We consider general instances for MinBusy. We show a lower bound for any online algorithm (Theorem 2) and present a greedy algorithm (Theorem 3).

Lower Bound We first describe a lower bound of 2 on any algorithm Alg. 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 Alg 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 this job 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 paths of length $\ell', \ell' k, \cdots$ 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 paths 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 Alg is no better than 2. By extending this idea we prove the following theorem in Appendix B.2

Theorem 2. *For* MinBusy*, no online algorithm has an absolute competitive ratio better than* g*.*

Upper Bound With Proposition 1 and Theorem 2, the competitive ratio is tight in terms of g. Yet the adversary in Theorem 2 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 classifies jobs according to their lengths and assigns jobs in a First-Fit manner for each class independently.

Algorithm 1 BucketFirstFit

```
1: Classify the jobs into buckets: a job of length in [2^k, 2^{k+1} − 1] belongs to bucket k, for k ≥ 1.
2: Assign machine to jobs in each bucket in a First-Fit manner independently of other buckets.
3: When a job J in bucket k arrives
4: for each machine M already assigned a job from bucket k do
5:     if it is valid to assign J to M then
6:         Assign J to M. return
7:     end if
8: end for
9: Assign J to a new machine.
```

To analyze BucketFirstFit we adapt the proof of Theorem 2.5 in [14], which asserts that if jobs are assigned to machines in descending order of length, a job J assigned M preventing a set of jobs $\mathcal{Q}$ from using machine M would imply $span(\mathcal{Q}) \leq 3 \cdot len(J)$ and using this result, the performance ratio of this procedure can be shown to be at most 4. When the jobs are not in descending order but have a max-min span ratio at most 2 (like a particular bucket in BucketFirstFit), one can then show that $span(\mathcal{Q}) \leq 5 \cdot len(J)$ and the performance ratio becomes 6. We then have Lemma 1 which further implies Theorem 3.

Lemma 1. *Let* $\mathcal{J}^k$ *be the set of jobs of bucket* k *and* s *be a schedule returned by* BucketFirstFit*. Then* $cost^s(\mathcal{J}^k) \leq 6cost^*(\mathcal{J}^k)$*.*

Theorem 3. *For* MinBusy, BucketFirstFit *is* $(6 \log span_{\max})$*-competitive where* $span_{\max}$ *is the maximum span of a job.*

Finally we note that the constant 6 above can be improved to 5 by modifying the algorithm to work with powers of 4 instead of powers of 2. Generally if the algorithm divides the lightpaths into buckets according to powers of some $\alpha > 1$ (like Algorithm 2 below), at each bucket we get a ratio of $2\alpha + 2$ and therefore an overall competitive ratio of $(2\alpha + 2) \log_\alpha span_{\max} = 2\frac{\alpha+1}{\log \alpha} \log span_{\max}$. Choosing $\alpha = 4$ brings the value of the coefficient to the minimum of $2\frac{4+1}{\log 4} = 5$.

3.2 One-Sided Clique Instances

Upper bound We consider one-sided clique instances and present the GreedyBucket algorithm (Algorithm 2). 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.

GreedyBucket depends on a parameter $\alpha > 1$ to be determined in the sequel. A job J is categorized to a bucket according to $len(J)$: the bucket of a job $J \in \mathcal{J}$ is the minimum value of i such that $len(J) \le \alpha^i$. For $i \ge 1$, bucket i consists of jobs J such that $\alpha^{i-1} < len(J) \le \alpha^i$.

Algorithm 2 GreedyBucket(α)

1: Determine the bucket i of the input job J according to the parameter α.
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 GreedyBucket 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 $0 \le r_i < g$. 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 $\ell \le m$. Let L be the span of the longest job in bucket ℓ, thus $\alpha^{\ell-1} < L \le \alpha^\ell$.

GreedyBucket 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 $(q_i + 1)\alpha^i$, because at most $q_i + 1$ machines are used each with busy time at most α^i. For $i > \ell$ this number is at most $q_i\alpha^i$ (because $r_i = 0$), and for bucket ℓ this number is at most $(q_\ell + 1)L$. Therefore we have:

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

In Appendix B.3 we prove the following proposition.

Proposition 3.

$$cost^* \ge \sum_{i=1}^{m} q_i\alpha^{i-1} + L.$$

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

Theorem 4. *For* MinBusy, GreedyBucket$(1+\varphi)$ *is strictly* $(1+\varphi)$*-competitive for one-sided clique instances, where* $\varphi = \frac{1+\sqrt{5}}{2}$ *is the Golden Ratio.*

Lower bound As of lower bound, we prove the following lemma in Appendix B.4.

Lemma 2. *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* $x^{g-1} = (x+1)/(x-1)$.

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 2 in Appendix A).

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.

Algorithm 3 LeftOrRight$(\mathcal{A})$

1: Two copies of the online algorithm $\mathcal{A}$ are run (each using a different set of machines), $\mathcal{A}_l$ 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**

The correctness of LeftOrRight$(\mathcal{A})$ follows from that of $\mathcal{A}$. Lemma 3 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 $\mathcal{J}$. Detailed proof is given in Appendix B.5.

Lemma 3. *If the competitive ratio of* $\mathcal{A}$ *for one-sided clique instances is* c*, then the competitive ratio of* LeftOrRight$(\mathcal{A})$ *for clique instances is at most* $2 \cdot c$.

Corollary 1. *For* MinBusy *and clique instances,* LeftOrRight(GreedyBucket$(1+\varphi)$) *is* $2(1+\varphi)$*-competitive, where* $\varphi = \frac{1+\sqrt{5}}{2}$ *is the Golden Ratio.*

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., $tput^* = |\mathcal{J}|$. We prove in Appendix B.6 the following proposition asserting that the problem does not admit a small competitive ratio for general instances.

Proposition 4. *No online algorithm for* MaxThroughput *is better than* gT*-competitive even with an additive term* $g - 1$*, while there exists a strictly* gT*-competitive online algorithm.*

Following Proposition 4, from now on we consider only feasible one-sided clique instances. For this case we prove in Appendix B.7 the following proposition asserting that both simple greedy algorithm and algorithm GreedyBucket do not admit a small competitive ratio.

Proposition 5. *For feasible one-sided clique instances of* MaxThroughput (i) *the simple greedy algorithm is $\Omega(R)$-competitive and* (ii) GreedyBucket *is $\Omega(\frac{R}{\log R})$-competitive even when $g = 2$.*

4.2 Lower Bounds for Feasible One-Sided Clique Instances

In this section we show two lower bounds on the competitive ratio of 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 (see Appendix B.8 for the proof).

Lemma 4. *Let* Alg *be a c-competitive online algorithm for feasible one-sided clique instances, with an additive constant b. If $c(b+1) < g$ then*

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

The condition in Lemma 4 holds when $g \gg b$ or $b = 0$, leading to Theorem 5.

Theorem 5. *Consider feasible one-sided clique instances.* (i) *Any online algorithm has a competitive ratio of at least* 2. (ii) *For any fixed value of g, any online algorithm has an absolute competitive ratio of at least $2 - \frac{2}{g+1}$.*

4.3 Online Algorithm for Feasible One-Sided Clique Instances

Note that the negative results in Section 4.2 do not provide any lower bound for the asymptotic competitive ratio for fixed values of g. In this section we propose an online algorithm 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 that will prove helpful in describing the algorithm. We categorize the input jobs into buckets 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 $\mathcal{J}_i \stackrel{def}{=} \{J \in \mathcal{J} \mid bucket(J) = i\}$. In other words we have $\forall J \in \mathcal{J}_i$, $\frac{T}{2^{i+1}} < span(J) \leq \frac{T}{2^i}$. 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 $\mathcal{J}_i$ except its first g jobs in the order of arrival.
- 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) $\sum_i T_i^* \leq cost^*$.

In the pseudo code of BalanceBudget (Algorithm 4), accept(J) stands for scheduling J with the smallest possible machine under use in bucket i such that the schedule continued to be valid after assigning J; and if no such machine exists J is assigned a new machine. reject(J) means that J is not assigned, i.e. $s(J)$ is undefined.

The analysis of BalanceBudget proceeds as follows. We show that there exists a polynomial-time computable function T_i^* (of the state of the algorithm) satisfying the conditions (a)-(c). We then show that with such T_i^*, BalanceBudget indeed returns a valid schedule with total busy time at most T (Lemma 5). Furthermore, we show that the throughput in every bucket by BalanceBudget is at least a constant fraction of the number of jobs in that bucket. Then in Theorem 6 we can conclude the competitive ratio of BalanceBudget. We start by stating Lemma 5 (see Appendix B.10 for the proof).

Algorithm 4 BalanceBudget

When a job J arrives do:
$i \leftarrow bucket(J)$
if $|\mathcal{J}_i| \leq g$ **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

Lemma 5. BalanceBudget *returns a valid schedule with total busy time at most T provided that there is a polynomial-time computable function to calculate T_i^* satisfying the above mentioned conditions (a)-(c).*

The following claim (for proof see Appendix B.9) shows such T_i^* stated in Lemma 5.

Claim. The function

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

satisfies all the requirements stated for T_i^* and is polynomial time computable.

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 in Appendix B.9 the following claim.

Claim. For every fixed g, there exists a constant $2/9 < c(g) < 1$ such that $\forall i \geq 3, tput_i \geq c(g) \cdot |\mathcal{J}_i|$.

With the above claims and Lemma 5, we prove the following theorem in Appendix B.11.

Theorem 6. *Consider* MaxThroughput *and feasible one-sided clique instances. For every fixed g,* BalanceBudget *is a constant-competitive online algorithm where the constant depends on g and is at most* 9/2.

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 4). 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 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:

- We have shown a constant competitive algorithm for one sided clique instances of the MinBusy problem and extended it to clique instances. Lower bounds are also given. An immediate open question is to close the gaps between the upper and lower bounds.
- The lower bounds and algorithms for the MaxThroughput problem in one-sided clique instances do not have matching counterparts. Specifically is there a constant-competitive algorithm for those instances when g is not fixed? On the other hand is there a lower bound for these instances when g is fixed?
- The lower bounds of the MaxThroughput problem for one-sided clique instances, clearly extend to general instances. 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, 30]).
  - 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, 30]).

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, 34]), 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 extensions can be interpreted clearly within the context of problems in cloud computing (see, e.g., [11, 27, 31]) as presented in Section 1.

- 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].
- 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.
- Another extension is to consider machines that can have different capacities, e.g., there are several types of machines with capacity g_k for machine type k.

Optical network design: 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 can be extended to ring topologies.

The first extension in the context of cloud computing applies also in this context: each lightpath has a capacity (say, a multiple of $1/g$), so that at most g such capacity units can be groomed together on any particular communication line.

References

1. J. Augustine, S. Irani, and C. Swany. Optimal power-down strategies. In *Proceedings of the 45th Annual Symposium on Foundations of Computer Science (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 *Automata, Languages and Programming*, volume 2719, pages 193–193. Springer Berlin / Heidelberg, 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 W. Cook and A. Schulz, editors, *Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 401–414. Springer Berlin / Heidelberg, 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 *Proceedings of 38th International Colloquium on Automata, Languages and Programming (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 *Proceedings of 2010 international conference on Web information systems and mining*, 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 *Prooceedings of IEEE/OSA Conference on Optical Fiber Communications (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. In *Proceedings of the 21st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 154–162, 2009, to appear in IEEE/ACM Transactions on Networking.
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 *14th International European Conference on Parallel and Distributed Computing (EURO-PAR 2008), Canary Island, Spain*, August 2008.
16. J. Kang and S. Ranka. Energy-efficient dynamic scheduling on parallel machines. In *Proceedings of the 15th international conference on High performance computing*, pages 208–219, 2008.
17. R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. In *Intl Conference on Foundations of Software Technology and Theoretical Computer Science (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. Lawler, J. Lenstra, A. Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. *S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin (eds.), Handbooks in Operations Research and Management Science*, 4, 1993.
22. R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms (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 *Proceedings of the 15th International Conference On Principles Of Distributed Systems (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*, 2012. To appear.
27. A. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 351–359, 2010.
28. C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA, pages 879–888, 2000.
29. K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 15–1–15–41. CRC Press, 2004.
30. U. Schwarz. Tightness results for malleable task scheduling algorithms. In *Parallel Processing and Applied Mathematics*, pages 1059–1067. 2008.
31. W. Shi and B. Hong. Resource allocation with a budget constraint for computing independent tasks in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 327–334, 2010.
32. N. Vasić, M. Barisits, V. Salzgeber, and D. Kostic. Making cluster applications energy-aware. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC, pages 37–42, 2009.
33. P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceeding of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 830–831, 2003.
34. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

Appendices

A Figures

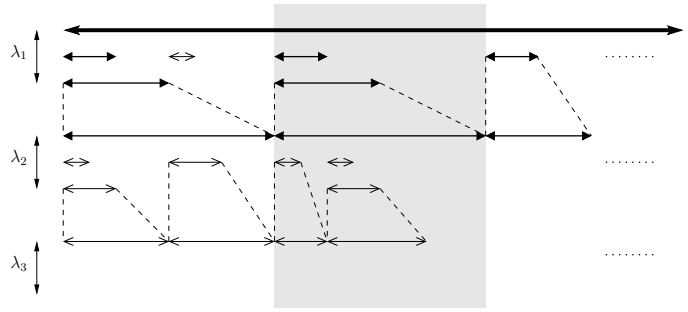

Fig. 1. Lower bound for MinBusy: example for $g = 3$. Jobs with different arrow heads and thicknesses belong to different values of d. Dashed lines show jobs for which the online algorithm uses a new machine and the jobs that are just released before each one of them. Note: Within each such pair the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

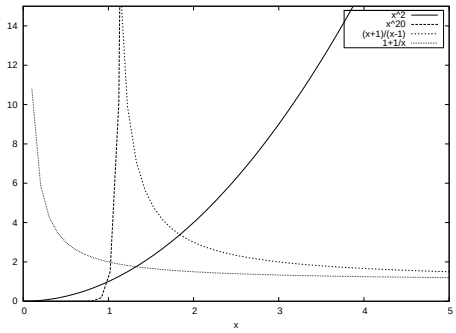

Fig. 2. Lower bound for MinBusy on one-sided clique instances. The line x and the curve $(x+1)/(x-1)$ intersect at $x = 1+\sqrt{2}$ and for this value $1+1/x = \sqrt{2}$. Note that x^g and $(x+1)/(x-1)$ intersect at $x \geq 1$, implying $1 + 1/x \leq 2$.

B Omitted Proofs

B.1 Relationship of MinBusy and MaxThroughput

We observe that MaxThroughput is NP-Hard whenever MinBusy is NP-Hard.

Proposition 2. *There is a polynomial time reduction from the* MinBusy *problem to the* MaxThroughput *problem.*

Proof. Given an instance $(\mathcal{M}, \mathcal{J}, g)$ of MinBusy we can perform binary search between $len(\mathcal{J})/g$ and $len(\mathcal{J})$ for the value of T by solving each time the instance $(\mathcal{M}, \mathcal{J}, g, T)$ of MaxThroughput to find the smallest value T such that $tput^*(\mathcal{M}, \mathcal{J}, g, T) \geq tput(\mathcal{J})$, i.e. to find the value $cost^*$.

We note that the hardness of MaxThroughput stems from the fact that one has to decide which subset of the jobs to schedule.

Proposition 6. *If there is a polynomial-time computable set $X \subseteq 2^{\mathcal{J}}$[4] containing at least one set $\mathcal{J}^s$ for some optimal schedule s, and also* MinBusy *can be solved optimally, then* MaxThroughput *can be solved optimally.*

Proof. Given an instance $(\mathcal{M}, \mathcal{J}, g, T)$ of MaxThroughput, for each set $\mathcal{J}' \in X$ solve the instance $(\mathcal{J}', g)$ of MinBusy. Among all sets $\mathcal{J}'$ with $cost^*(\mathcal{J}') \leq T$ choose one with maximum throughput and return the schedule s returned for this instance. Leave the jobs $\mathcal{J} \setminus \mathcal{J}'$ unscheduled.

B.2 Proof of Theorem 2

Theorem 2. *No deterministic online algorithm for* MinBusy *has an absolute competitive ratio better than g.*

Proof. We extend the simple adversary described in Section 3.1 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 Alg 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 $2g$ paths would force Alg 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 Alg. Initially, the parameters are $(0, T, 1)$.

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

Algorithm 5 Adversary(r, t, d)

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

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 T. The total length of the remaining jobs is at most $g \sum_{i=1}^{\infty} T/k^i = g \cdot T/(k-1)$. Therefore, $cost^*/cost^s \geq \frac{g}{1+g/(k-1)}$. Picking a large k implies the theorem. □

[4] Also of polynomial size. This assumption holds whenever the set is represented by an explicit list of its elements.

B.3 Proof of Proposition 3

Proposition 3.

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

Proof. Observe that s^* can be obtained by sorting the jobs in decreasing order of lengths, and assigning the same machine to every g consecutive jobs in this order. This can be seen as if s^* considers the jobs in decreasing bucket index and within each bucket in decreasing length. 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 in 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-1} in each machine. In bucket ℓ, s^* spends a busy time of L on the first machine, and at least $\alpha^{\ell-1}$ 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-1}. Note that the last machine is possibly available for bucket $i - 1$. We have:

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

□

B.4 Proof of Lemma 2

Lemma 2. *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 and when the on-line algorithm Alg 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. Then $\frac{cost^s}{cost^*} = (x^{i+1}+x^i)/x^{i+1} = (x+1)/x$.

If Alg 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 Alg 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 Alg is at least $(1 + 1/x)$-competitive. □

B.5 Proof of Lemma 3

Lemma 3. *If the competitive ratio of* $\mathcal{A}$ *for one-sided clique instances is* c*, then the competitive ratio of* LeftOrRight$(\mathcal{A})$ *for clique instances is at most* $2 \cdot c$.

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 $\mathcal{A}_r$.

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 $cost^*(\mathcal{J}'_l) \leq cost^*(\mathcal{J}_l)$. Similarly, $cost^*(\mathcal{J}'_r) \leq cost^*(\mathcal{J}_r)$. Furthermore, $cost^*(\mathcal{J}) \geq cost^*(\mathcal{J}_l) + cost^*(\mathcal{J}_r)$ because these two instances have no time in common (besides t). 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, $cost^{s_l}(\mathcal{J}'_l) \leq c \cdot cost^*(\mathcal{J}'_l)$ and $cost^{s_r}(\mathcal{J}'_r) \leq c \cdot cost^*(\mathcal{J}'_r)$, implying

$$cost^{s_l}(\mathcal{J}'_l) + cost^{s_r}(\mathcal{J}'_r) \leq c \cdot (cost^*(\mathcal{J}_l) + cost^*(\mathcal{J}_r)) \leq c \cdot cost^*(\mathcal{J}).$$

For every job J, LeftOrRight($\mathcal{A}$) selects J_l or J_r (whichever longer) to feed to one of the two copies of algorithm $\mathcal{A}$. Hence, $cost^{s_l}(\mathcal{J}_1) \leq 2 \cdot cost^{s_l}(\mathcal{J}'_l)$ and $cost^{s_r}(\mathcal{J}_2) \leq 2 \cdot cost^{s_r}(\mathcal{J}'_r)$. Then,

$$cost^s(\mathcal{J}) = cost^{s_l}(\mathcal{J}_1) + cost^{s_r}(\mathcal{J}_2) \leq 2(cost^{s_l}(\mathcal{J}'_l) + cost^r(\mathcal{J}'_r)) \leq 2c \cdot cost^*(\mathcal{J}) \ .$$

□

B.6 Proof of Proposition 4

Proposition 4. *No online algorithm for* MaxThroughput *is better than gT-competitive even with an additive term $g-1$, while there exists a strictly gT-competitive online algorithm.*

Proof. To simplify the discussion, we assume that time is divided into integral interval. 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 Alg. The adversary releases (at most) g jobs with interval $[0, T]$ until Alg schedules one job. Alg has to schedule at least one of these g jobs and spends a busy time of T, otherwise, its competitive ratio is unbounded. We then issue gT paths all with length 1, g of which with intervals $[T, T+1]$, g with $[T+1, T+2]$, until g with $[2T-1, 2T]$. Alg cannot schedule any of these jobs while an optimal solution would only schedule these jobs, and the competitive ratio is gT. □

B.7 Proof of Proposition 5

Proposition 5. *For* MaxThroughput *and feasible one sided clique instances,* (i) *the simple greedy algorithm is $\Omega(T)$-competitive and* (ii) GreedyBucket *is $\Omega(\frac{T}{\log T})$-competitive 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 GreedyBucket 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, GreedyBucket 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 $\Omega(\frac{T}{\log T})$. □

B.8 Proof of Lemma 4

Lemma 4. *For feasible one-sided clique instances, let* ALG *be a c-competitive online algorithm with an additive constant b. If* $c(b+1) < g$ *then*

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

Proof. For any prefix of the input ALG returns a schedule s such that $tput^s \geq tput^*/c - b$. We describe an adaptive adversary for ALG. 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 $T/2-1$. ALG 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. ALG uses two machines, one with busy time $T/2+1$ and the other $T/2-1$ and $cost^s = T$. Then ALG 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. ALG uses only one machine. The adversary then releases $g - c(b+1)$ more jobs each of length $T/2+1$. ALG 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. □

B.9 Proofs of Claims in Section 4.3

We give the proofs for the two claims in Section 4.3.

Claim. The function

$$T_i^* \stackrel{def}{=} \left\lceil \frac{\max(|\mathcal{J}_i| - (g-1), 0)}{g} \right\rceil \frac{T}{2^{i+1}} \tag{3}$$

satisfies all the requirements stated for T_i^* and is polynomial time computable.

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

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 $\max(|\mathcal{J}_i| - (g-1), 0)$ jobs are assigned machines not used in previous buckets. This corresponds to $\left\lceil \frac{\max(|\mathcal{J}_i|-(g-1),0)}{g} \right\rceil$ machines, and each machine has a busy time of at least $\frac{T}{2^{i+1}}$. Therefore, $cost_i^* \geq \sum_i \left\lceil \frac{\max(|\mathcal{J}_i|-(g-1),0)}{g} \right\rceil \frac{T}{2^{i+1}} = \sum_i T_i^*$. □

Claim. For every fixed g, there exists a constant $2/9 < c(g) < 1$ such that $\forall i \geq 3, tput_i \geq c(g) \cdot |\mathcal{J}_i|$.

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

Otherwise $|\mathcal{J}_i| > g/c$ 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. $tput_i < c \cdot |\mathcal{J}_i|$. 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 $tput_i - 1 < c \cdot |\mathcal{J}_i| - 1 < c(|\mathcal{J}_i| - 1)$, 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:

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

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 (2) we conclude

$$\frac{tput_i + 1}{g} \geq \left\lceil \frac{tput_i + 1}{g} \right\rceil - 1 \geq \frac{3}{8} \left(\left\lceil \frac{|\mathcal{J}_i| + 1}{g} \right\rceil - 1 \right) \geq \frac{3}{8} \frac{\mathcal{J}_i - g + 1}{g}$$
$$tput_i \geq \frac{3}{8} \left(|\mathcal{J}_i| - g - \frac{5}{3} \right)$$

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

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

If c is chosen such that $\frac{3}{8}(1 - (1 + \frac{5}{3g})c) \geq c$ then $tput_i > c \cdot |\mathcal{J}_i|$, 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 $g = 2$. □

B.10 Proof of Lemma 5

Lemma 5. 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.*

Proof. Clearly 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 4. 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 in $\mathcal{J}_i$ BalanceBudget uses one machine with busy time at most $T/2^i$ except for $i < 3$ where none of the first g jobs are 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 $cost^s \leq \sum_{i \geq 3} \frac{T}{2^i} + \frac{3}{4} \sum T_i^* < \frac{T}{4} + \frac{3}{4} T = T$. The last inequality is due to $\sum T_i^* \leq cost^* \leq T$. □

B.11 Proof of Theorem 6

Theorem 6. *Consider* MaxThroughput *and feasible one-sided clique instances. For every fixed g,* BalanceBudget *is a constant-competitive online algorithm where the constant depends on g and is at most $9/2$.*

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 clearly does not change the value of T_i, 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 the claim in Appendix B.9 implies that $\forall i \geq 0, tput'_i \geq c(g) \cdot |\mathcal{J}_i|$. Summing over all buckets we get

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

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