Optimizing Busy Time on Parallel Machines[*]

George B. Mertzios [†] Mordechai Shalom [‡] Ariella Voloshin [§]
Prudence W.H. Wong [¶] Shmuel Zaks [§]

Abstract

We consider the following fundamental parallel machines scheduling problem in which the input consists of n jobs to be scheduled on a set of identical machines of bounded capacity g, which is the maximal number of jobs that can be processed simultaneously by a single machine. Each job is associated with a time interval during which it should be processed from start to end (and in one of our extensions it has to be scheduled also in a continuous number of days; this corresponds to a two-dimensional variant of the problem). We consider two versions of the problem. In the scheduling 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 can be scheduled for processing under a budget constraint given in terms of busy time. This is the first study of the maximization version of the problem. The minimization problem is known to be NP-Hard, thus the maximization problem is also NP-Hard. We consider various special cases, identify cases where an optimal solution can be computed in polynomial time, and mainly provide constant factor approximation algorithms for both minimization and maximization problems. Some of our results improve upon the best known results for this job scheduling problem. Our study has applications in energy-aware scheduling, cloud computing, switching cost optimization as well as wavelength assignments in optical networks.

Keywords: Interval scheduling, busy time, resource allocation, approximation algorithms.

1 Introduction

Problem Statement: Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [5, 20]). In particular, much attention was given to *interval scheduling* [19], 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 interval scheduling with *bounded parallelism.* Formally, the input is a set of n jobs $\mathcal{J} = \{J_1, \dots, J_n\}$. Each job, J_j, is associated with an interval $[s_j, c_j]$ during which it should be processed. Also, given is the parallelism parameter, or capacity $g \geq 1$, which is the

[*]This work was supported in part by the Israel Science Foundation grant No. 1249/08, British Council Grant UKTELHAI09, and the EPSRC Grant EP/K022660/1. A preliminary version appeared in the *Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS), Shanghai, China, May 2012.*

[†]School of Engineering and Computing Sciences, Durham University, Durham, UK. Email: george.mertzios@durham.ac.uk

[‡]Tel Hai College, Upper Galilee, 12210, ISRAEL. Email: cmshalom@telhai.ac.il

[§]Department of Computer Science, Technion, Haifa, ISRAEL. Email: [variella,zaks]@cs.technion.ac.il

[¶]Department of Computer Science, University of Liverpool, Liverpool, UK. Email: pwong@liverpool.ac.uk maximal number of jobs that can be processed simultaneously by a single machine. At any given point t in time a machine M_i is said to be *busy* if there is at least one job J_j scheduled to it such that $t \in [s_j, c_j]$, 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*, and denote its length by $busy_i$. In this work we study two optimization problems MinBusy and MaxThroughput. In MinBusy we focus on minimizing the total busy times over all machines, denoted by $\sum_i busy_i$. Note that the number of machines used is part of the output. A solution that minimizes 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 the total machine busy time T and the objective is to maximize the number of scheduled jobs subject to 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.

Applications: Our scheduling problems can be directly interpreted as **energy-aware scheduling** problems in cluster systems. These problems focus on minimizing the energy consumption of a set of machines (see, e.g., [27] and the references therein) which can be measured by the amount of time the machines are switched on and processing, i.e. busy time. It is common that a machine has a capacity on the number of jobs that can be processed at any given time, like in our problems.

Another application of the studied problems comes from **cloud computing** (see, e.g., [23, 26]). 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 proportion to the total amount of computing time of the computing resource. The clients would like to make the most of their money, so they would 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. The simple scenario where each client requires the same amount of computing resources corresponds to our setting where the machines are identical and each job requires the same proportion of their capacity.

Our study is also motivated by problems in **optical network design** (see, e.g., [9, 11, 12]). 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, communications between nodes are realized by *lightpaths*, which are assigned a certain color. As the energy of the signal along a lightpath decreases, *regenerators* are needed and the hardware cost is proportional to the length of the lightpaths. Furthermore, connections can be "groomed" so that a regenerator can be shared by at most g connections, i.e. at any node, at most g connections can have the same color sharing the regenerator there. 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.

Another application concerning optical networks is **wavelength assignment** (see, e.g., [28]). Given a set of connections along the line, each needs to be assigned a wavelength. A fiber can carry at most W wavelengths. Two overlapping connections carried by the same fiber cannot share a wavelength. The wavelength assignment problem is in analogy to our scheduling problem in the sense that the length of fiber used corresponds to the busy time while the amount W of wavelengths that the optical fiber can carry corresponds to the machine capacity.

Related Work: Some of the earlier work on interval scheduling considers the problem of scheduling a valid subset of jobs whose total weight is maximized, i.e., a *maximum weight independent set* (see, e.g., [3] and the comprehensive surveys in [17,18]). 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., [3,6,8,24]). It is also common to consider both minimization and maximization versions of the same scheduling problem (see e.g., [4]), 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: The complexity of MinBusy was studied in [28], showing that the problem is NP-Hard already for $g = 2$. The work [13] considered the problem where jobs are given as intervals on the line with unit demand on capacity. 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, and instances where any two job intervals intersect, i.e., the input forms a clique (see same approximation but different algorithm and analysis in [14]). The work [16] extends the results of [13], considering the case where each job has a different demand on machine capacity.

The minimization problem MinBusy studied in this paper is related to the problems studied in [13,16]. As will be discussed in Section 5, some of our results directly improve upon existing results for these scheduling problems. As for the maximization problem, we are not aware of works that present and study this problem.

Our Contribution: As mentioned above MinBusy is NP-Hard already for $g = 2$. We consider two special cases (clique instances, where all jobs share a common time, and proper instances, where no job interval is properly contained in another one). We also consider a two-dimensional variant that each job is to be processed during given continuous hours over some given continuous days. We show the following results:

- A polynomial-time algorithm for clique instances when $g = 2$ (Lemma 3.1).
- A $\frac{g \cdot H_g}{H_g + g - 1}$-approximation algorithm for clique instances, where H_g is the g-th harmonic number, for any fixed value of g (Lemma 3.2).
- A $(2 - 1/g)$-approximation algorithm for proper instances (Theorem 3.1).
- A polynomial-time algorithm for proper clique instances, based on an interesting combinatorial property of optimal solutions (Theorem 3.2).
- For the scheduling of 2-dimensional intervals, we define $\gamma_k,\ k \in \{1, 2\}$ to be the ratio between the longest and the shortest interval on dimension k. We present a $\min(g, 13.82 \cdot$

$\log \min(\gamma_1, \gamma_2) + O(1)$)-approximation algorithm (Theorem 3.3).

We show that MaxThroughput is NP-Hard whenever MinBusy is NP-Hard. For MaxThroughput we show the following:

- A 4-approximation algorithm for clique instances for any g (Theorem 4.1).
- A polynomial-time algorithm for proper clique instances for any g (Theorem 4.2).

It turns out the algorithms and analysis of the results for MaxThroughput are more involved than those for MinBusy, apparently due to an inherent hardness of the former.

Organization of the paper: In Section 2 we present some preliminaries. MinBusy is studied in Section 3, and MaxThroughput in Section 4. In Section 5 we first summarize our results, then we discuss extensions and further research directions.

2 Notations and Preliminaries

Definitions: 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.

Definition 2.1 *Given a time interval $I = [s_I, c_I]$ with* start time *s_I and* completion time *c_I, the* length *of I is $len(I) = c_I - s_I$. This extends to a set $\mathcal{I}$ of intervals; namely, the* length *of $\mathcal{I}$ is $len(\mathcal{I}) = \sum_{I \in \mathcal{I}} len(I)$.*

Definition 2.2 *For a set $\mathcal{I}$ of intervals we define $SPAN(\mathcal{I}) \overset{def}{=} \cup\mathcal{I}$ and $span(\mathcal{I}) \overset{def}{=} len(SPAN(\mathcal{I}))$ and we refer to both of them as the* span *of a set of intervals, when the intention is clear from the context. Two intervals are said to be* overlapping *if their intersection contains more than one point.*

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 job J is given by a time interval during which it is supposed to be processed. We use jobs and time intervals interchangeably throughout the paper.

As we do not aim at optimizing the number of machines, we assume that the given set $\mathcal{M} = \{M_1, M_2, \ldots\}$ of machines is infinite.

A *(partial) schedule*, is a (partial) function from the set $\mathcal{J}$ of jobs to the set $\mathcal{M}$ of machines. Given a parallelism parameter g, a schedule is *valid* if every machine processes at most g jobs at any given time. In other words every machine has g threads of execution, each of which can process at most one job at any given time. In this definition a job $[s_J, c_J]$ 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 during the interval $[1, 3]$ including time 2. Note that this is consistent with the definition of the term overlapping, and equivalent to saying that the intervals do not contain their completion time, i.e. are half-open intervals.

Given a schedule $s : \mathcal{J} \to \mathcal{M}$, 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. $busy_i^s \overset{def}{=} span(\mathcal{J}_i^s)$.

Given a partial schedule s, we denote the set of jobs scheduled by it as $\mathcal{J}^s \overset{def}{=} \cup_i \mathcal{J}_i^s$. The *cost* $cost^s$ of s is the sum $\sum_i busy_i^s$ of the busy times of all machines. The *throughput* $tput^s$ of s is $\overset{def}{=} |\mathcal{J}^s|$. When there is no ambiguity for the schedule under consideration, we omit the superscripts (e.g. we use $\mathcal{J}_i$ as a shorthand for $\mathcal{J}_i^s$, etc.).

We consider two problem variants: 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 defined as follows:

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

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

Without loss of generality we assume that each machine is busy along a contiguous time interval, i.e. that $SPAN(\mathcal{J}_i^s)$ is an interval. Otherwise we can replace the machine with several machines that satisfy the assumption, changing neither the feasibility nor the total busy time of the schedule.

For MinBusy we assume that the interval graph induced by the jobs is connected; otherwise, the problem can be solved by considering each connected component separately.

Special cases: A set of jobs $\mathcal{J}$ is a *clique set* if there is a time t common to all the jobs in $\mathcal{J}$. It is known that this happens if and only if the corresponding interval graph is a clique. When $\mathcal{J}$ is a clique set we term the corresponding instance ($(\mathcal{J}, g)$ or $(\mathcal{J}, g, T)$) a *clique instance.* A clique instance in which all jobs have the same start time or the same completion time is termed a *one-sided edge 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 $J, J' \in \mathcal{J}$ we have $s_J \leq s_{J'}$ if and only if $c_J \leq c_{J'}$. We denote this fact as $J \leq J'$ and without loss of generality we assume $J_1 \leq J_2 \leq \ldots \leq J_n$. When $\mathcal{J}$ is proper we term the corresponding instance a *proper instance.*

Approximation algorithms: We consider polynomial-time exact and approximation algorithms and analyze their worst case performance. Given an instance of MinBusy or MaxThroughput, we denote by s^* an optimal schedule of this instance. The cost of s^* is denoted by $cost^*$ and its throughput by $tput^*$. An algorithm A for MinBusy is a *ρ-approximation* for $\rho \geq 1$, if for any instance the cost of the schedule returned by it is at most $\rho \cdot cost^*$. For MaxThroughput, A is a ρ-approximation for $\rho \geq 1$ if for any instance the throughput of the schedule returned by it is at least $(1/\rho) \cdot tput^*$.

Basic observations: The next observation gives two immediate lower bounds for the cost of any solution of MinBusy.

Observation 2.1 *For any instance $(\mathcal{J}, g)$ 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 g is the maximum parallelism that can be achieved in any solution. The span bound holds since at any time $t \in \cup\mathcal{J}$ at least one machine is working. The length bound holds because at any time that some machine is busy at least one job is being processed.
By the parallelism bound and length bound we conclude

Proposition 2.1 *Any schedule (algorithm) is a g-approximation for* MinBusy.

Proof: $cost^s \leq len(\mathcal{J}) \leq g \cdot cost^*$. □

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

Proposition 2.2 *There is a polynomial-time reduction from* MinBusy *to* MaxThroughput.

Proof: Given an instance $(\mathcal{J}, g)$ of MinBusy where the values $\{s_J, c_J : J \in J\}$ are rational numbers $p_1/q_1, p_2/q_2, \cdots$, we first multiply them by $\prod q_i$ so that all the values are integers. Note that the size of the input remains polynomial in the original size because every number q_i appears in at most $2|\mathcal{J}| - 1$ numbers of the modified instance. We can now perform a binary search between the lower bound $len(\mathcal{J})/g$ of $cost^*$ and its upper bound $len(\mathcal{J})$. At each iteration we set T to be equal to the guessed value of $cost^*$ and solve the instance $(\mathcal{J}, g, T)$ of MaxThroughput. In the next iteration we guess a smaller value $cost^*$ if and only if $tput^*(\mathcal{J}, g, T) \geq tput(\mathcal{J})$, i.e. all the jobs can be scheduled within the guessed budget. We stop when the difference between the last feasible cost guessed and the last infeasible cost guessed is at most 1. □

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

Proposition 2.3 *If there is a polynomial-time algorithm solving* MinBusy *optimally and a polynomial-time computable family* [1] $X \subseteq 2^{\mathcal{J}}$ *of subsets of* $\mathcal{J}$ *with at least one set* $\mathcal{J}^{s^*} \in X$ *corresponding to some optimal schedule* s^**, then* MaxThroughput *can be solved optimally.*

Proof: Given an instance $(\mathcal{J}, g, T)$ 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. □

For the schedule $\bar{s}$ that assigns every job to a different machine, i.e. $\bar{s}(J_i) = M_i$ we have $cost^{\bar{s}} = len(\mathcal{J})$. For any schedule s we define the saving sav^s of s (in cost, relative to $\bar{s}$) achieved by s as $sav^s \overset{def}{=} len(\mathcal{J}) - cost^s$. As far as optimal schedules are concerned MinBusy can equivalently be reformulated as the problem of maximizing sav^s. However when sub-optimal schedules are considered, the two definitions differ in terms of approximation ratio of a schedule. The following Lemma relates these ratios.

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

Lemma 2.1 *If a schedule is a ρ-approximation to the saving maximization problem for some $\rho \geq 1$, then it is a $(1/\rho + (1 - 1/\rho)\, g)$-approximation to* MinBusy.

Proof: Let $\rho' = 1/\rho$, and let s be a schedule satisfying our assumption, i.e. $sav^s \geq \rho' \cdot sav^*$. We have

$$\begin{aligned} cost^s - cost^* &= sav^* - sav^s \leq (1-\rho')sav^* \\ &= (1-\rho')(len(\mathcal{J}) - cost^*) \\ &= (1-\rho')len(\mathcal{J}) + (\rho' - 1)cost^* \end{aligned}$$

thus

$$\begin{aligned} cost^s &\leq (1-\rho')len(\mathcal{J}) + \rho' \cdot cost^* \\ &\leq (1-\rho')g \cdot cost^* + \rho' \cdot cost^* \\ &= \big(\rho' + (1-\rho')g\big)\, cost^* \end{aligned}$$

where the second inequality follows from the parallelism bound. □

3 Cost Minimization

In this section we study special cases, namely clique instances, proper instances and proper clique instances (in Sections 3.1, 3.2 and 3.3, respectively). We also investigate a generalization of the problem to the two dimensional case, i.e. where the jobs are given with rectangular intervals (Section 3.4).

3.1 Clique Instances

In this section we consider clique instances. We show (in Lemma 3.1) a polynomial-time algorithm for the case $g = 2$ and (in Lemma 3.2) an approximation algorithm with a better ratio for small values of g (improving upon the 2-approximation algorithm of [13] for these cases). We first observe the following for one-sided clique instances.

Observation 3.1 *For one-sided clique instances of* MinBusy *an optimal solution can be obtained by sorting the jobs in non-increasing order of their lengths and grouping them in groups of g in this order (where the last group possibly contains less than g jobs).*

Consider the edge weighted graph $G_m = (\mathcal{J}, E_m)$ where $e_{ij} = \{J_i, J_j\} \in E_m$ if the jobs J_i and J_j overlap. The weight of e_{ij} is the size of the overlap. When the parallelism parameter is $g = 2$, in any valid schedule at most two jobs can share a machine. In other words, a schedule corresponds to a matching in the graph G_m where sav^s is equal the weight of the matching. Therefore the minimization of $cost^s$ is equivalent to the maximization of sav^s, i.e the weight of the matching. As maximum weighted matching is well known to be solvable in polynomial-time, we conclude

Lemma 3.1 *There exists a polynomial-time algorithm for clique instances of* MinBusy *when $g = 2$.*

Using a similar argument as in Lemma 3.1, the MinBusy problem is equivalent to g-dimensional weighted matching, which admits a $\frac{2(g+1)}{3}$-approximation [1]. By Lemma 2.1 this implies a $\frac{2g^2-g+3}{2(g+1)}$-approximation for MinBusy. However using a direct approach we improve this result in the below lemma.

Lemma 3.2 *For any fixed value of g, there is a $\frac{g \cdot H_g}{H_g+g-1}$-approximation algorithm for clique instances of* MinBusy*, where H_g is the g-th harmonic number.*

Proof: As we consider clique instances, a schedule s is valid if and only if for any machine M_i, $|\mathcal{J}_i^s| \leq g$. If we associate with each $\mathcal{Q} \subseteq \mathcal{J}$ such that $|\mathcal{Q}| \leq g$ a weight of $span(\mathcal{Q})$, the problem is equivalent to the problem of finding a minimum weight set cover of $\mathcal{J}$ with subsets of size at most g. As g is fixed we can calculate the weights of all possible subsets and run the well-known H_g-approximation algorithm for set cover.

We can improve the result using the parallelism bound of $PB = len(\mathcal{J})/g$. We modify the costs of the sets, so that they reflect the excess cost from this lower bound. Formally, for each $\mathcal{Q} \subseteq \mathcal{J}$ we assign $weight(\mathcal{Q}) = span(\mathcal{Q}) - len(\mathcal{Q})/g$. The weight corresponding to a schedule s is $weight(s) = \sum_i (span(\mathcal{J}_i^s) - len(\mathcal{J}_i^s)/g) = \sum_i span(\mathcal{J}_i^s) - len(\mathcal{J})/g = cost^s - PB$. For the schedule s returned by the algorithm we have

$$\begin{aligned} weight(s) &\leq H_g \cdot weight(s^*) \\ cost^s - PB &\leq H_g cost^* - H_g \cdot PB \\ cost^s &\leq H_g cost^* - (H_g - 1) \cdot PB \qquad (1) \end{aligned}$$

and by the length bound, we have

$$cost^s \leq len(\mathcal{J}) = g \cdot PB \ . \qquad (2)$$

We choose $0 \leq \rho \leq 1$, satisfying $\rho(H_g - 1) = (1 - \rho)g$, and multiply inequality (1) by ρ, and inequality (2) by $1 - \rho$. Summing up the resulting inequalities we get

$$cost^s \leq \rho \cdot H_g \cdot cost^*$$

Since $\rho = \frac{g}{H_g+g-1}$, the algorithm is a $\frac{g \cdot H_g}{H_g+g-1}$-approximation. $\square$

It can be verified taking the derivative with respect to g, that the function $\frac{g \cdot H_g}{H_g+g-1}$ is monotonically increasing. The value of the function is less than 2 for $g \leq 6$.

Note that the algorithm used in the last lemma is another exact algorithm for $g = 2$, since the set cover problem can be solved optimally when the sizes of the sets are at most 2. However, we have chosen to present both algorithms (lemmata 3.1 and 3.2) as they are based on different techniques and therefore might be extended in different directions to solve different extensions of the problem.

3.2 Proper Instances

In this section we present the algorithm BestCut and show that it is a $(2 - 1/g)$-approximation for proper instances, thus improving upon the 2-approximation algorithm presented in [13]. Recall that we assume without loss of generality that a) the (interval graph corresponding to the) instance is connected, and b) $J_1 \leq J_2 \leq ... \leq J_n$, i.e. the sequence of the jobs' start times is non-decreasing. The algorithm calculates g valid solutions $s^1, \ldots, s^g$ and returns the best one. The solutions are valid because in each solution s^i, every machine j is assigned the set $\mathcal{J}_j^i$ of at most g jobs. Moreover, every machine except possibly the first and last machines is assigned exactly g jobs.

Theorem 3.1 *Algorithm* BestCut *is a $(2-1/g)$-approximation for proper instances of* MinBusy.

Algorithm 1 BestCut$(\mathcal{J}, g)$

```
1: for $i = 1$ to $g$ do
2:     $\mathcal{J}_0^i = \{J_1, \ldots, J_i\}$;        ▷ $\mathcal{J}_j^i$ is the set of jobs to be assigned to machine $j$ in schedule $i$.
3:     for $j = 1$ to $\lfloor (|\mathcal{J}| - 1)/g \rfloor$ do
4:         $\mathcal{J}_j^i = \left\{ J_{i+g(j-1)+1}, \ldots, J_{i+g \cdot j} \right\}$
5:     end for
6:     $s^i = \left\{ \mathcal{J}_0^i, \mathcal{J}_1^i, \ldots \right\}$
7: end for
8: $s \leftarrow \text{argmin} \left\{ cost^{s^i} \mid s^1, s^2, \ldots, s^g \right\}$
9: return $s$
```

Proof: For every i and j, $\left| \mathcal{J}_j^i \right| \leq g$, therefore the schedule s returned by the algorithm is valid. We first give an upper bound to sav^*: by the span bound we have $cost^* \geq span(\mathcal{J})$. On the other hand $span(\mathcal{J}) = len(\mathcal{J}) - \sum_{k=1}^{|\mathcal{J}|-1} |I_k|$, where I_k is the overlap $J_k \cap J_{k+1}$ of the jobs J_k and J_{k+1}. Therefore $cost^* \geq len(\mathcal{J}) - \sum_{k=1}^{|\mathcal{J}|-1} |I_k|$, or equivalently

$$sav^* \leq \sum_{k=1}^{|\mathcal{J}|-1} |I_k|. \tag{3}$$

We proceed by calculating $cost^s$. For every $1 \leq i \leq g$, $cost^{s^i} = span(\mathcal{J}_0^i) + \sum_{j>0} span(\mathcal{J}_j^i)$. Moreover $span(\mathcal{J}_0^i) = len(\mathcal{J}_0^i) - \sum_{k=1}^{i-1} |I_k|$ and $span(\mathcal{J}_j^i) = len(\mathcal{J}_j^i) - \sum_{k=i+g(j-1)+1}^{i+g \cdot j-1} |I_k|$. Therefore $cost^{s^i} = len(\mathcal{J}) - \sum_{k=1}^{|\mathcal{J}|-1} |I_k| + \sum_{j \geq 0} |I_{i+g \cdot j}|$, or equivalently $sav^{s^i} = \sum_{k=1}^{|\mathcal{J}|-1} |I_k| - \sum_{j \geq 0} |I_{i+g \cdot j}|$.

Then

$$\begin{aligned}
\sum_{i=1}^{g} sav^{s^i} &= g \sum_{k=1}^{|\mathcal{J}|-1} |I_k| - \sum_{i=1}^{g} \sum_{j \geq 0} |I_{i+g \cdot j}| \\
&= (g-1) \sum_{k=1}^{|\mathcal{J}|-1} |I_k| \\
sav^s &= \max_{1 \leq i \leq g} sav^{s^i} \geq \frac{g-1}{g} \sum_{k=1}^{|\mathcal{J}|-1} |I_k| \qquad (4)
\end{aligned}$$

Combining (3) and (4) we conclude that BestCut is a $\frac{g}{g-1}$-approximation for the saving maximization problem. Substituting this for ρ in Lemma 2.1 we get the $2 - \frac{1}{g}$ approximation ratio for MinBusy.

□

3.3 Proper Clique Instances

In this section we consider instances that are both clique instances and proper instances. We start with a simple observation and some definitions regarding proper instances.

Property 3.1 *Consider any two jobs J_i and $J_{i'}$ in a proper instance. If $i \leq i'$, then we have $s_i \leq s_{i'}$ and $c_i \leq c_{i'}$.*

We say that a subset $\mathcal{Q} \subseteq \mathcal{J}$ of jobs is *consecutive* in $\mathcal{J}$ (or simply consecutive) if $\mathcal{Q} = \{J_i, J_{i+1}, \cdots, J_j\}$ for some $1 \leq i \leq j \leq n$. We show in Lemma 3.3 that there is an optimal schedule such that the jobs allocated to every machine are consecutive. To prove this lemma, we introduce the notion of conflicting triple. Consider a schedule s such that there is at least one machine M_i for which the subset $\mathcal{J}_i$ is not consecutive. Then there exists at least one triple of distinct jobs $J_a \leq J_b \leq J_c$ such that $s(J_a) = s(J_c) = M_i$ and $s(J_b) \neq M_i$. We call such a triple of jobs a *conflicting triple* $\langle a, b, c\rangle$ of s. A conflicting triple $\langle a, b, c\rangle$ is said to be *maximal* if J_a (resp. J_c) is the leftmost (resp. rightmost) job scheduled on M_i in s. Furthermore, a conflicting triple $\langle a, b, c\rangle$ is said to be a *rightmost* conflicting triple if there exists no conflicting triple $\langle a', b', c'\rangle$ of s such that $b' > b$. Note that a rightmost conflicting triple is not necessarily unique. We now state a property of rightmost conflicting triples.

Property 3.2 *Consider a schedule s and a rightmost conflicting triple $\langle a, b, c\rangle$ of s.* (i) *All jobs from $J_{b+1}, \cdots, J_c$ are all scheduled on the same machine.* (ii) *The job J_b is the rightmost job scheduled on machine $s(J_b)$.*

Proof: (i) If there is a job J_d with $b < d < c$ that is scheduled on a machine different from $s(J_c)$, then $\langle a, d, c\rangle$ is a conflicting triple of s with $d > b$. This contradicts the fact that $\langle a, b, c\rangle$ is a rightmost conflicting triple of s.

(ii) If there is a job J_e with $e > b$ that is scheduled on the same machine $s(J_b)$, then either $\langle a, e, c\rangle$ is a conflicting triple (if $b < e < c$) or $\langle b, c, e\rangle$ is a conflicting triple (if $e > c$). In either case, it contradicts the fact that $\langle a, b, c\rangle$ is a rightmost conflicting triple of s. □

We now give a framework of the proof of Lemma 3.3. Suppose on the contrary that in every optimal schedule there exists at least one conflicting triple. We consider an optimal schedule s^* in which the value b of a rightmost conflicting triple $\langle a, b, c\rangle$ is the smallest. That is, every optimal schedule s' has a rightmost conflicting triple $\langle a', b', c'\rangle$ with $b' \geq b$. We say that this optimal schedule s^* has a *smallest rightmost conflicting triple*. The proof works by constructing from s^* another optimal schedule s^{**} and showing that for every conflicting triple $\langle a^{**}, b^{**}, c^{**}\rangle$ of s^{**}, we have $b^{**} < b$. This contradicts the definition of s^* being an optimal schedule having a smallest rightmost conflicting triple.

Lemma 3.3 *Given a proper clique instance of* MinBusy*, there is an optimal schedule such that for every machine M_i, the subset $\mathcal{J}_i$ is consecutive in $\mathcal{J}$.*

Proof: Assume, by contradiction, that the lemma does not hold, i.e. that every optimal schedule has a conflicting triple. Let s^* be an optimal schedule with a smallest rightmost conflicting triple $\langle a, b, c\rangle$. Let also $s^*(J_a) = s^*(J_c) = M_i$ and $s^*(J_b) = M_{i'} \neq M_i$. Without loss of generality, we assume that $\langle a, b, c\rangle$ is a maximal conflicting triple.

We now construct another optimal schedule s^{**} that has a rightmost conflicting triple $\langle a', b', c'\rangle$ with $b' < b$, which is a contradiction to the definition of s^*. The schedule s^{**} is obtained from s^* by rescheduling some jobs scheduled on M_i and $M_{i'}$. Specifically, for every index x, if $s^*(J_x) \in \{M_i, M_{i'}\}$ then $s^{**}(J_x) \in \{M_i, M_{i'}\}$, and otherwise $s^{**}(J_x) = s^*(J_x)$. Since the instance is a clique instance, a schedule s is valid if and only if $|\mathcal{J}_i^s| \leq g$ for each machine M_i. This property will be preserved when obtaining s^{**} from s^*.

Construction, validity and optimality of s^{**}: Let k be the smallest index, for which $s^*(J_k) = M_{i'}$; clearly $k \neq a$, and also $k \leq b$ by Property 3.2. We distinguish between the following two cases regarding k and a.

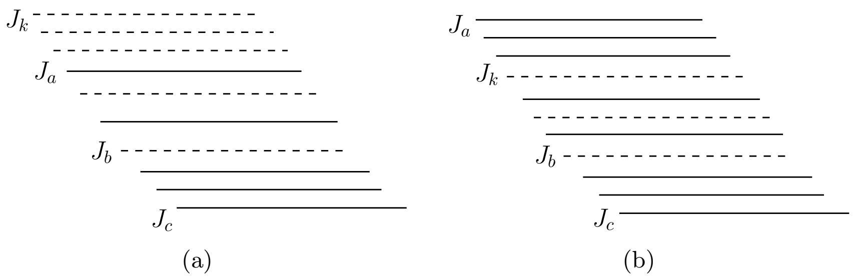

Figure 1: The subset of jobs scheduled on $M_i, M_{i'}$ (represented by solid and dashed lines, respectively) in an optimal schedule s^* of a proper clique instance, where (a) $k < a$ and (b) $a < k$.

Case 1: $k < a$ (see Figure 1(a)). We construct the schedule s^{**} by exchanging the machine assignments of J_a and J_b, i.e., $s^{**}(J_a) = M_{i'}$, $s^{**}(J_b) = M_i$, and $s^{**}(J_x) = s^*(J_x)$ for every $x \notin \{a, b\}$. The schedule s^{**} remains a valid schedule. Since J_a is the leftmost job scheduled on M_i and J_b is the rightmost job scheduled on $M_{i'}$ in s^* (Property 3.2(ii)), it follows that $span(\mathcal{J}_i^{s^{**}}) \leq span(\mathcal{J}_i^{s^*})$ and $span(\mathcal{J}_{i'}^{s^{**}}) \leq span(\mathcal{J}_{i'}^{s^*})$, i.e. s^{**} is optimal.

Case 2: $a < k \leq b$ Figure 1(b)). Consider the set $\mathcal{J}_i^{s^*} \cup \mathcal{J}_{i'}^{s^*}$. Since $a < k$, the leftmost (indexed from a to $k-1$) and the rightmost jobs (indexed from $b+1$ to c) in $\mathcal{J}_i^{s^*} \cup \mathcal{J}_{i'}^{s^*}$ are scheduled on M_i in s^*. For the sake of the following discussion in which only these jobs are relevant we re-index jobs in $\mathcal{J}_i^{s^*} \cup \mathcal{J}_{i'}^{s^*}$ such that the indices are consecutive and $a = 0$. We have $c + 1 = \left|\mathcal{J}_i^{s^*} \cup \mathcal{J}_{i'}^{s^*}\right| \leq 2g$. We have also $c+1 \leq b+g$ because all the $c-b$ jobs between J_{b+1} and J_c and one job J_a are assigned to one machine, namely M_i. Let $x = \min\{g, b\}$. s^{**} assigns all the jobs from J_0 to J_{x-1} to M_i, and all the subsequent jobs to $M_{i'}$. The number of jobs assigned to M_i is $x = \min\{g, b\} \leq g$. The number of jobs assigned to $M_{i'}$ is $c + 1 - x = c + 1 - \min\{g, b\}$. If $g < b$ then $x = c + 1 - g \leq g$, otherwise $x = c + 1 - b \leq g$. Therefore s^{**} is valid.

In s^*, the total cost of M_i and $M_{i'}$ is $span(\{J_0, J_c\}) + span(\{J_k, J_b\}) = c_c - s_0 + c_b - s_k$. In s^{**}, the cost of M_i and $M_{i'}$ is $span(\{J_a, J_x\}) + span(\{J_{x+1}, J_c\}) = c_x - s_0 + c_c - s_{x+1}$. In order to show that s^{**} is optimal, it suffices to show that $c_x - s_{x+1} \leq c_b - s_k$. Indeed, as $x \leq b$, by Property 3.1 we have $c_x \leq c_b$. Furthermore as $k \leq g$ and $k \leq b$ we have $k \leq x < x + 1$, and by Property 3.1, $s_k \leq s_{x+1}$.

Rightmost conflicting triple of s^{**}: To complete the proof, it remains to show that any rightmost conflicting triple $\langle a^{**}, b^{**}, c^{**}\rangle$ of the constructed optimal schedule s^{**} satisfies $b^{**} < b$. Note that, as jobs assigned to other machines will be relevant, we revert to the original indices in the rest of the discussion.

If the machines assigned to $J_{b^{**}}$ and $J_{c^{**}}$ in s^{**} are in $\{M_i, M_{i'}\}$, then s^{**} is constructed in Case 1 where $k < a$ because in Case 2 where $a < k$, there is no conflicting triple in s^{**} with machines in $\{M_i, M_{i'}\}$. In the construction in Case 1, we have $b^{**} < b$.

Otherwise, one of the jobs $J_{b^{**}}, J_{c^{**}}$ is scheduled on $M_{i''} \notin \{M_i, M_{i'}\}$ by s^{**}. Then s^{**} may be constructed in any of the two cases $a < k$ or $k < a$. Suppose on the contrary that $b^{**} \geq b$. We further consider two cases.

Case A: $b^{**} > c$. I.e., $c^{**} > b^{**} > c > b$. Since c and b are the largest indices for which $s^*(J_c) = M_i$ and $s^*(J_b) = M_{i'}$, respectively (the former due to $\langle a, b, c\rangle$ being a maximal conflicting triple while the latter due to Property 3.2(ii)), it follows that $J_{a^{**}}, J_{b^{**}}, J_{c^{**}}$ are all scheduled on machines different from $M_i, M_{i'}$ in both s^* and s^{**}. Therefore $\langle a^{**}, b^{**}, c^{**}\rangle$ is also a conflicting triple in s^*, where $b^{**} > b$, contradicting that $\langle a, b, c\rangle$ is a rightmost conflicting triple of s^*.

Case B: $b \leq b^{**} \leq c$. By Property 3.2(i), all jobs from J_{b+1} to J_c are scheduled on M_i in s^*, and thus in particular $s^*(J_{b^{**}}) \in \{M_i, M_{i'}\}$. Furthermore, by the construction of s^{**} in both Cases 1 and 2, it follows that $s^{**}(J_{b^{**}}) = M_{i'}$. Therefore $s^{**}(J_{c^{**}}) = M_{i''}$ by assumption, where $i'' \notin \{i, i'\}$, and thus also $s^*(J_{c^{**}}) = M_{i''}$, implying that $c^{**} > c$ (again due to Property 3.2(i)). Therefore, $\langle a^{**}, c, c^{**} \rangle$ is a conflicting triple of s^*, where $c > b$, contradicting that $\langle a, b, c \rangle$ is a rightmost conflicting triple of s^*.

Summarizing Cases A and B, we have $b^{**} < b$. This means that all conflicting triples of the optimal schedule s^{**} have $b^{**} < b$, contradicting the definition of s^*. In conclusion, there exists an optimal schedule s^*, such that for every machine M_i, the subset $\mathcal{J}_i$ is consecutive. □

Using Lemma 3.3 we design a polynomial-time dynamic programming algorithm to find an optimal schedule. Let us consider a proper clique instance consisting of n jobs $J_1 \leq J_2 \leq \ldots \leq J_n$. We denote by $cost^*(i)$ the cost of an optimal schedule of the sub-instance consisting of the leftmost i jobs, and by $cost^*(i,j)$ the minimum cost of those schedules of the same sub-instance that assign the same machine to the last (exactly) j jobs. Clearly $j \leq \min(g,i)$, and $cost^*(i) = \min_{1 \leq j \leq \min(i,g)} cost^*(i,j)$. Let I_k be the overlap between jobs J_k and J_{k+1}.

Algorithm 2 FINDBESTCONSECUTIVE

```
1: $cost^*(1) \leftarrow cost^*(1,1) \leftarrow span(J_1)$
2: **for** $i = 2$ to $n$ **do**
3:     $cost^*(i,1) \leftarrow |J_i| + cost^*(i-1)$
4:     **for** $j = 2$ to $\min(g,i)$ **do**
5:         $cost^*(i,j) \leftarrow cost^*(i-1,j-1) + |J_i| - |I_{i-1}|$
6:     **end for**
7: **end for**
```

Note that the assignments in lines 3 and 5 are correct by Lemma 3.3.

As for the time complexity, if $n < g$ all the jobs are scheduled on the same machine. Otherwise we run the dynamic programming algorithm FINDBESTCONSECUTIVE. For each job J_i, for $i = 1, \ldots, n$, we have to compute $cost^*(i,j)$ for $i = 1, \ldots, g$. Thus, the total running time of the algorithm is $O(n \cdot g)$, implying the following theorem.

Theorem 3.2 *Given a proper clique instance of* MINBUSY*,* FINDBESTCONSECUTIVE *computes an optimal schedule in polynomial time.*

3.4 Rectangular Intervals

In this section we consider a generalization of the problem to two dimensions. In graph theoretic terms we are considering rectangle graphs (i.e., intersection graphs of rectangles), instead of interval graphs. The problem can be relevant in contexts where rectangle graphs are relevant. For instance, we can consider periodic jobs that are run in a specific time interval every day, between two given dates. Another application is the case of a path topology optical networks (see also discussion in Section 5) in which clients may have communication requests between two points of the network for a specific time interval. In this generalization all the times, in particular the start and completion times are pairs of real numbers and jobs are given with rectangular intervals.

Definition 3.1 *Given a rectangular interval $I = [s_I, c_I]$, where $s_I = (s_{I,1}, s_{I,2}), c = (c_{I,1}, c_{I,2}) \in \mathbb{R} \times \mathbb{R}$ and $s_k < c_k$, for $k \in \{1,2\}$ we define $\pi_k(I)$ as the projection $[s_{I,k}, c_{I,k}]$ of I in dimension k. We further define $len_k(I) \overset{def}{=} len(\pi_k(I))$ and $len(I) \overset{def}{=} len_1(I) \cdot len_2(I)$.*

Definition 3.2 *For a set $\mathcal{I}$ of rectangular intervals we define $SPAN(\mathcal{I}) \overset{def}{=} \cup\mathcal{I}$ and $span(\mathcal{I})$ is defined as the area of $SPAN(\mathcal{I})$.*

Given the above definition of *len* and *span*, all the definitions given for the one dimensional case in Section 2 extend to this case. Moreover it is easy to verify that the span, length and parallelism bounds hold for this case too.

We also define $\gamma_k = \frac{\max_{J \in \mathcal{J}} len_k(J)}{\min_{J \in \mathcal{J}} len_k(J)}$ for $k \in \{1,2\}$. In the sequel we assume without loss of generality $\gamma_1 \leq \gamma_2$ and present an $O(\log \gamma_1)$-approximation algorithm.

Algorithm FirstFit (Algorithm 3) is similar to algorithm FirstFit presented in [13] for the one dimensional case. In this work we adapt the algorithm and its proof to the case of two dimensions. FirstFit sorts the jobs according to their len_2 values and considers them in descending order. Every job is assigned to the first thread of execution among the threads of the first machine that is feasible for it.

Algorithm 3 FirstFit($\mathcal{J}, g$)

```
1: Sort the jobs in non-increasing order of their lengths in the second dimension, i.e., $len_2(J_1) \geq len_2(J_2) \geq \ldots \geq len_2(J_n)$.
2: for $j = 1$ to $n$ do                        ▷ Consider the jobs according to the above order
3:     for $i = 1$ to $\infty$ do                              ▷ Consider the machines
4:         for $\tau = 1$ to $g$ do                      ▷ Consider the threads of execution
5:             if $J_j$ does not intersect any job assigned to thread $\tau$ of $M_i$ then
6:                 Assign $J_j$ to thread $\tau$ of $M_i$
7:                 return
8:             end if
9:         end for
10:    end for
11: end for
```

The following observation is a direct consequence of the behavior of FirstFit.

Observation 3.2 *Let J be a job assigned to machine M_{i+1} by* FirstFit*, for some $i \geq 1$. Then for every thread τ of M_i there is a job $b_\tau(J) \in \mathcal{J}_i$ such that $b_\tau(J) \cap J \neq \emptyset$ and $len_2(b_\tau(J)) \geq len_2(J)$.*

The following key lemma is based on the above observation.

Lemma 3.4 *For any $i \geq 1$,*

$$span(\mathcal{J}_{i+1}) \leq \frac{6\gamma_1 + 3}{g} len(\mathcal{J}_i).$$

Proof: Whenever a job $J \in \mathcal{J}_{i+1}$ intersects with more than one job of $\mathcal{J}_i$ assigned to the same thread τ of M_i we fix one of them (arbitrarily) to be $b_\tau(J)$. Let $\mathcal{J}_{i,\tau}$ be the set of jobs assigned by FirstFit to thread τ of M_i. Clearly $b_\tau : \mathcal{J}_{i+1} \rightarrow \mathcal{J}_{i,\tau}$ is a function. We define the set valued inverse function, in the usual way, as $b_\tau^{-1} : \mathcal{J}_{i,\tau} \rightarrow 2^{\mathcal{J}_{i+1}}$ such that $b_\tau^{-1}(J') = \{J \in \mathcal{J}_{i+1} \mid b_\tau(J) = J'\}$. We claim that for every $J' \in \mathcal{J}_{i,\tau}$

$$span(b_\tau^{-1}(J')) \leq (6\gamma_1 + 3) \cdot len(J'). \qquad (5)$$

Indeed, by Observation 3.2, every job $J \in b_\tau^{-1}(J')$ intersects with J' and $len_2(J) \leq len_2(J')$. Moreover, $len_1(J) \leq \gamma_1 \cdot len_1(J')$ by definition of γ_1. We conclude that J lies entirely within a

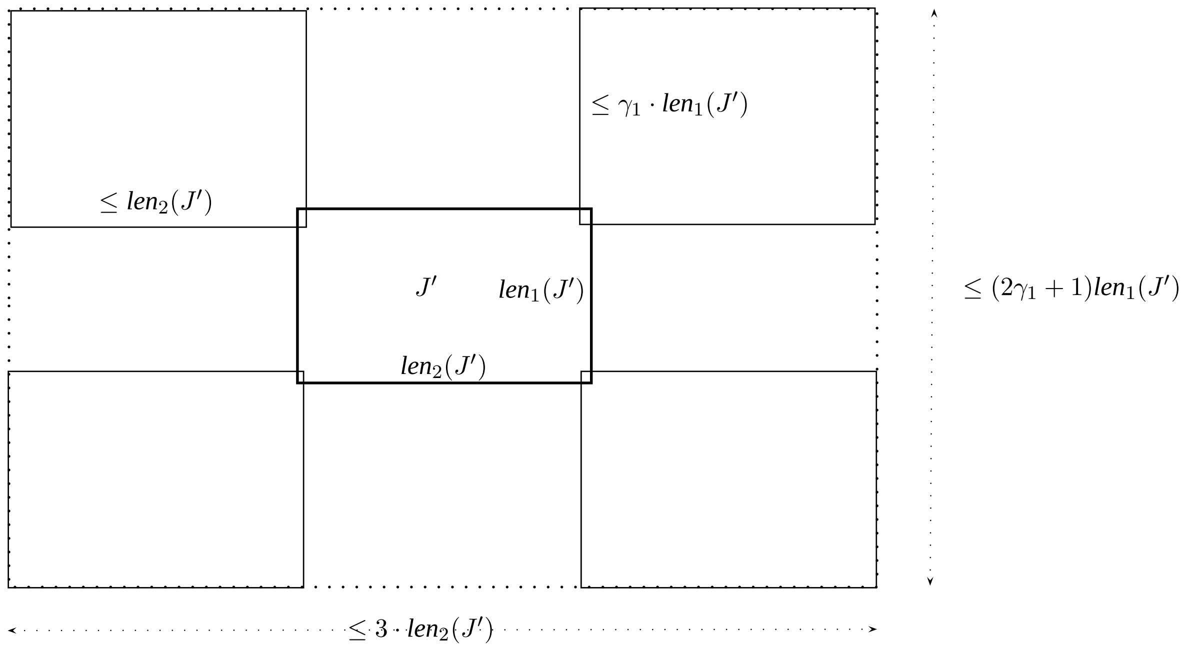

Figure 2: A bounding rectangle of $SPAN(b_\tau^{-1}(J'))$

bounding rectangle of lengths $(2\gamma_1+1) \cdot len_1(J')$ and $3 \cdot len_2(J')$ in the two dimensions. Therefore $SPAN(b\tau^{-1}(J'))$ lies entirely within the same rectangle whose area is $(2\gamma_1+1) \cdot len_1(J') \cdot 3 \cdot len_2(J') = (6\gamma_1+3) \cdot len(J')$ (see Figure 2).

As b_τ is a function, we have $\mathcal{J}_{i+1} = \cup_{J' \in \mathcal{J}_{i,\tau}} b_\tau^{-1}(J')$. Therefore $SPAN(\mathcal{J}_{i+1}) = \cup_{J' \in \mathcal{J}_{i,\tau}} SPAN(b_\tau^{-1}(J'))$. Now, we use the union bound and sum up inequality (5) for all the jobs $J' \in \mathcal{J}_{i,\tau}$ to get

$$span(\mathcal{J}_{i+1}) \leq \sum_{J' \in \mathcal{J}_{i,\tau}} span(b_\tau^{-1}(J')) \leq (6\gamma_1+3) \sum_{J' \in \mathcal{J}_{i,\tau}} len(J') = (6\gamma_1+3) \cdot len(\mathcal{J}_{i,\tau}).$$

Summing for all g possible values of τ, we get

$$g \cdot span(\mathcal{J}_{i+1}) \leq (6\gamma_1+3) \cdot \sum_{\tau=1}^{g} len(\mathcal{J}_{i,\tau}) = (6\gamma_1+3) \cdot len(\mathcal{J}_i).$$

□

Lemma 3.5 *The approximation ratio of* FirstFit *is between* $6\gamma_1+3$ *and* $6\gamma_1+4$*.*

Proof: **Upper bound:** By definition, all the jobs in $\mathcal{J}_{i+1}$ are assigned to one machine, i.e. M_{i+1}. For such a set the cost of the assignment is exactly its span, i.e. FirstFit$(\mathcal{J}_{i+1}) = busy_{i+1} =$

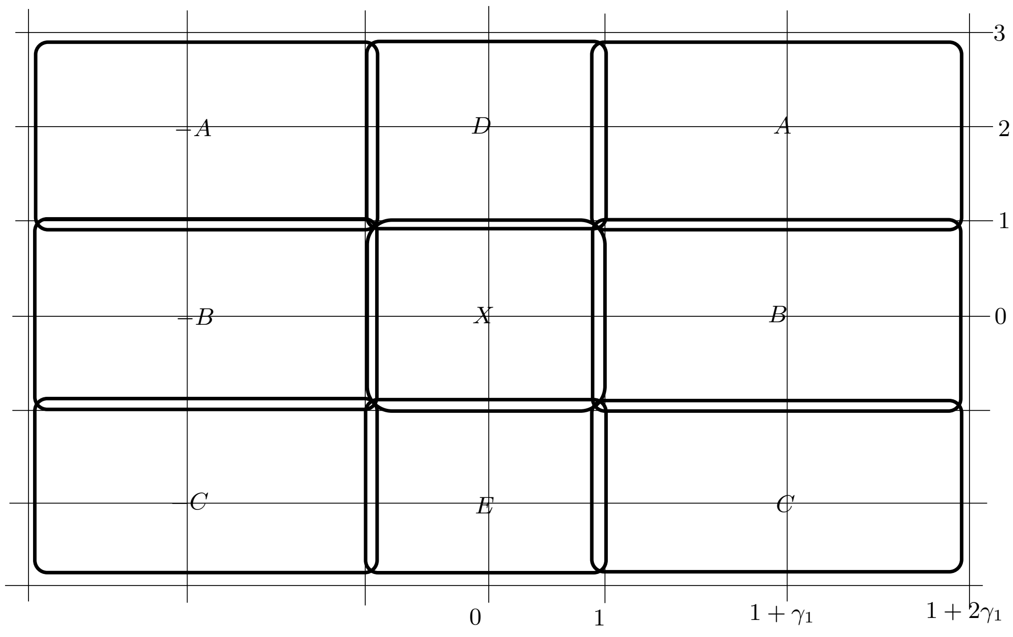

Figure 3: The rectangles in the instance used in the lower bound proof.

$span(\mathcal{J}_{i+1})$. Let $m \geq 1$ be the number of machines used by FirstFit. Then

$$\begin{aligned}
\textsc{FirstFit}(\mathcal{J}) &= \sum_{i=1}^{m} \textsc{FirstFit}(\mathcal{J}_i) = span(\mathcal{J}_1) + \sum_{i=2}^{m} span(\mathcal{J}_i) \\
&= span(\mathcal{J}_1) + \sum_{i=1}^{m-1} span(\mathcal{J}_{i+1}) \leq span(\mathcal{J}) + \frac{6\gamma_1+3}{g} \sum_{i=1}^{m-1} len(\mathcal{J}_i) \\
&< span(\mathcal{J}) + (6\gamma_1+3)\frac{len(\mathcal{J})}{g} \\
&\leq (6\gamma_1+4) \cdot cost^*(\mathcal{J})
\end{aligned}$$

where the last inequality uses both the span and parallelism bounds.

Lower bound: We will show that for any $\epsilon > 0$, $\textsc{FirstFit}(\mathcal{J})/cost^*(\mathcal{J}) > 6\gamma_1 + 3 - \epsilon$. Let ϵ', $0 < \epsilon' < 1$, be a real number that depends on ϵ and γ_1 whose value will be determined later. For ease of discussion, we allow times to be negative. Given a rectangle $A = [(s_1, s_2), (c_1, c_2)]$, we denote by $-A$ the rectangle $[(-c_1, s_2), (-s_1, c_2)]$. We also denote for $s_1, s_2 \geq 0$ by $\pm(s_1, s_2)$ the rectangle $[(-s_1, -s_2), (s_1, s_2)]$ centered at the origin. We define the following rectangles (see Figure 3):

$$\begin{array}{rcl} A & = & [(1-\epsilon', 1-\epsilon'), (1+2\gamma_1-\epsilon', 3-\epsilon')] \\ B & = & [(1-\epsilon', -1), (1+2\gamma_1-\epsilon', 1)] \\ C & = & [(1-\epsilon', -3+\epsilon'), (1+2\gamma_1-\epsilon', -1+\epsilon')] \\ D & = & [(-1, 1-\epsilon'), (1, 3-\epsilon')] \\ E & = & [(-1, -3+\epsilon'), (1, -1+\epsilon')] \\ X & = & \pm(1,1) \\ Y & = & \pm(1+2\gamma_1-\epsilon', 3-\epsilon') \end{array} \tag{6}$$

The following facts are easy to verify:

- $len_1(A) = len_1(B) = len_1(C) = 2\gamma_1$, $len_1(D) = len_1(E) = len_1(X) = 2$, $len_1(Y) = 2(1+2\gamma_1-\epsilon')$,
- $len_2(A) = len_2(B) = len_2(C) = len_2(D) = len_2(E) = len_2(X) = 2$, $len_2(Y) = 2(3-\epsilon')$,
- $A, C, -A, -C$ are pairwise disjoint (because $\epsilon' < 1$),
- $D \cap E = \emptyset$, $B \cap -B = \emptyset$ (because $\epsilon' < 1$),
- X intersects with each one of $A, B, C, D, E, -A, -B$ and $-C$,
- A, B, D are pairwise intersecting, C, B, E are pairwise intersecting,
- $Y = X \cup A \cup B \cup C \cup D \cup E \cup -A \cup -B \cup -C$.

Consider the input consisting of $g\cdot(g-3)$ copies of X and g copies of $A, B, C, D, E, -A, -B, -C$. We first note that $\frac{\max_{J\in\mathcal{J}} len_1(J)}{\min_{J\in\mathcal{J}} len_1(J)} = \frac{2\gamma_1}{2} = \gamma_1$ as required.

A possible solution is to assign all the copies of $g(g-3)$ rectangles X to $g-3$ machines, and to assign all the g copies of each rectangle type to one machine. This solution is feasible because each machine is processing g jobs. In such a solution the busy time of each one of the $g-3$ machines is $span(X)$ and the busy time of the other machines are $span(A), span(B), \ldots$. Therefore

$$cost^*(\mathcal{J}) \leq (g-3)span(X)+2(span(A)+span(B)+span(C))+span(D)+span(E) = 4(g-3)+24\gamma_1+8.$$

On the other hand, FirstFit sorts the jobs by their len_2. As they are all equal, breaking ties arbitrarily it might consider them in the following order: $g-3$ copies of X, one copy of each one of A, C, $-A$, $-C$, B, $-B$, D, E, $g-3$ copies of X, one copy of each one of A, C, $-A$, $-C$, B, $-B$, D, E, and so on [2]. The jobs X are assigned to the first $g-3$ threads of execution of the first machine. As all the other jobs intersect X these threads cannot process any other jobs. The jobs A, C, $-A$, $-C$ are assigned to the $(g-2)$th thread of execution of the first machine. As all the other jobs intersect one of these rectangles this thread cannot process any other jobs. The jobs B and $-B$ are assigned to thread number $g-1$ of the first machine. As these jobs intersect with all the rest, this thread too is not usable by other jobs. The jobs D, E are assigned to the last thread

[2]For any variant of FirstFit that does not break ties arbitrarily the instance can be modified by altering the len_2 value of each rectangle slightly so that the total cost is not affected and the stated order is enforced.

of the first machine. Similarly, this thread is not usable by any other job. The algorithm continues in the same way and fills g machines in the same way. Therefore

$$\textsc{FirstFit}(\mathcal{J}) = \sum_{i=1}^{g} span(\mathcal{J}_i) = g \cdot span(Y) = 4g(1+2\gamma_1-\epsilon')(3-\epsilon').$$

We conclude that the approximation ratio of FirstFit is at least

$$\frac{4g(1+2\gamma_1-\epsilon')(3-\epsilon')}{4(g-3)+24\gamma_1+8} = \frac{g(1+2\gamma_1-\epsilon')(3-\epsilon')}{g+6\gamma_1-1} = \frac{(1+2\gamma_1-\epsilon')(3-\epsilon')}{1+\frac{6\gamma_1-1}{g}}.$$

For any $\gamma_1 \geq 1$ one can choose g sufficiently big to make the denominator arbitrarily close to 1, and ϵ' sufficiently small to make the nominator close to $6\gamma_1+3$. Therefore, the approximation ratio can be made arbitrarily close to $6\gamma_1+3$.

□

Algorithm 4 BucketFirstFit($\mathcal{J}, g, \beta$)

```
1: Let $\ell = \min_{J\in\mathcal{J}} len_1(J)$.
2: Let $\gamma_1 = (\max_{J\in\mathcal{J}} len_1(J))/\ell$.
3: for $b = 1$ to $\max(\lceil\log_\beta \gamma_1\rceil, 1)$ do
4:     Let $\mathcal{J}^{(b)} = \{J \in \mathcal{J} \mid \ell\cdot\beta^{b-1} \leq len_1(J) \leq \ell\cdot\beta^b\}$.
5:     Schedule the jobs in $\mathcal{J}^{(b)}$ to a set of unused machines using algorithm FirstFit.
6: end for
```

Consider algorithm BucketFirstFit that gets an additional parameter $\beta \geq 1$ and invokes FirstFit as a subroutine, such that in each invocation the sub-instance $\mathcal{J}^{(b)}$ satisfies $\gamma_1 \leq \beta$. FirstFit is a $(6\beta+4)$-approximation on each sub-instance, i.e. $cost^s(\mathcal{J}^{(b)}) \leq (6\beta+4)\cdot cost^*(\mathcal{J}^{(b)}) \leq (6\beta+4)\cdot cost^*(\mathcal{J})$. Therefore

$$\begin{aligned} cost^s(\mathcal{J}) &= \sum_{b=1}^{\max(\lceil\log_\beta\gamma_1\rceil,1)} cost^s(\mathcal{J}^{(b)}) \\ &\leq \sum_{b=1}^{\max(\lceil\log_\beta\gamma_1\rceil,1)} (6\beta+4)cost^*(\mathcal{J}) \\ &= \max(\lceil\log_\beta\gamma_1\rceil,1)\cdot(6\beta+4)\cdot cost^*(\mathcal{J}) \\ &\leq (\lceil\log_\beta\gamma_1\rceil+1)\cdot(6\beta+4)\cdot cost^*(\mathcal{J}) \\ &\leq (\log_\beta\gamma_1+2)\cdot(6\beta+4)\cdot cost^*(\mathcal{J}) \\ &= \left(\frac{6\beta+4}{\log\beta}\cdot\log\gamma_1 + O(\beta)\right)\cdot cost^*(\mathcal{J}) \end{aligned}$$

where the logarithms are base 2 unless written otherwise. Substituting $\beta = 3.3$ and recalling our assumption $\gamma_1 \leq \gamma_2$ we get

Theorem 3.3 BucketFirstFit($\mathcal{J}, g, 3.3$) *constitutes a* $\min(g, 13.82\cdot\log\min(\gamma_1,\gamma_2)+O(1))$-*approximation algorithm for* MinBusy *on rectangular intervals.*

4 Throughput Maximization

Although MaxThroughput is at least as hard as MinBusy, it turns out that in some cases we can achieve similar results as for MinBusy.

4.1 Clique Instances

For one-sided clique instances we note that if a schedule s with $cost^s \leq T$ schedules $tput^s$ jobs, then there is a schedule s' with $cost^{s'} \leq cost^s \leq T$ that schedules the shortest $tput^s$ jobs. In particular there is an optimal schedule that schedules the shortest j jobs for some $0 \leq j \leq |\mathcal{J}|$. By Proposition 2.3 and Observation 3.1 we conclude

Proposition 4.1 *One-sided clique instances of* MaxThroughput *can be solved optimally in polynomial time.*

We now present a constant approximation algorithm for clique instances of MaxThroughput. We start with some terminology. We first fix an arbitrary time t that is common to all the jobs. For a job $J = [s_J, c_J]$ we term the sub-interval $[s_J, t]$ (resp. $[t, c_J]$) as the *left part* (resp. *right part*) of J. The longer (resp. shorter) among these parts is termed the *head* (resp. *tail*) of J, and whenever these parts have the same length the left part is the head. A job J is *left-heavy* (resp. *right-heavy*) if its left (resp. right) part is the head.

We denote by $\mathcal{J}^{(L)}$ (resp. $\mathcal{J}^{(R)}$) the subset of left-heavy (resp. right-heavy) jobs of $\mathcal{J}$. For $X \in \{L, R\}$, a subset containing j jobs of $\mathcal{J}^{(X)}$ with shortest heads is termed a *prefix* of size j and denoted by $\mathcal{J}^{(X,j)}$.

The reduced cost of a schedule s of $\mathcal{J}$ is the cost of s where each job is replaced by its head, and we denote it by $\overline{cost}^s(\mathcal{J})$. In other words, in the reduced cost model the tails of the jobs do not consume machine time. Clearly $\overline{cost}^s(\mathcal{J}) \leq cost^s(\mathcal{J})$. Moreover $cost^s(\mathcal{J}) \leq 2 \cdot \overline{cost}^s(\mathcal{J})$ because for each machine M_i, $span(\mathcal{J}_i)$ is at most twice the longest head of $\mathcal{J}_i$, i.e. at most twice the busy time of the machine in the reduced cost model. A schedule minimizing $\overline{cost}^s(\mathcal{J})$ is termed *reduced-optimal*, and the corresponding reduced cost is denoted by $\overline{cost}^*$. Note that for $\mathcal{J}^{(L)}$, $\mathcal{J}^{(R)}$ and their subsets the calculation of $\overline{cost}^*$ is equivalent to solving a one-sided clique instance under the normal cost model and this can be done in polynomial time by Proposition 4.1.

We first present Algorithm Alg1 that achieves a constant approximation ratio in most cases. Alg1 chooses the maximal number $j + k$ of jobs with shortest heads in $\mathcal{J}^{(L)}$ and $\mathcal{J}^{(R)}$ (j jobs of $\mathcal{J}^{(L,j)}$ and k jobs of $\mathcal{J}^{(R,k)}$), with total reduced machine busy time at most $T/2$. Alg1 then schedules the jobs in each one of these sets in a reduced-optimal manner. The algorithm is clearly correct because the cost of the solution is at most twice its reduced cost, i.e. at most T.

Algorithm 5 Alg1

```
1: Among all the $O\left(\left|\mathcal{J}^{(L)}\right| \cdot \left|\mathcal{J}^{(R)}\right|\right)$ possible prefix pairs $\mathcal{J}^{(L,j)}, \mathcal{J}^{(R,k)}$
2:     Choose a pair with $\overline{cost}^*(\mathcal{J}^{(L,j)}) + \overline{cost}^* \mathcal{J}^{(R,k)} \leq T/2$ maximizing $j + k$.
3: Schedule the jobs of $\mathcal{J}^{(L,j)}$ in a reduced-optimal manner.
4: Schedule the jobs of $\mathcal{J}^{(R,k)}$ in a reduced-optimal manner.
```

We now proceed with the performance analysis of Alg1.

Lemma 4.1 *If $tput^* > 4g$ then* Alg1 *is a 4-approximation algorithm for clique instances of* MaxThroughput.

Proof: Consider an optimal schedule s^* and the set $\mathcal{J}^*$ of jobs scheduled by it. We partition this set into two sets of left-heavy and right-heavy jobs, namely $\mathcal{Q}^{(L)} = \mathcal{J}^* \cap \mathcal{J}^{(L)}$ and $\mathcal{Q}^{(R)} = \mathcal{J}^* \cap \mathcal{J}^{(R)}$. Then $tput^* = |\mathcal{J}^*| = \left|\mathcal{Q}^{(L)}\right| + \left|\mathcal{Q}^{(R)}\right|$ and also $T \geq cost^*(\mathcal{J}^*) \geq \overline{cost}^*(\mathcal{J}^*) = \overline{cost}^*(\mathcal{Q}^{(L)}) + \overline{cost}^*(\mathcal{Q}^{(R)})$. The last equality holds because in the reduced cost model each one of the sets $\mathcal{Q}^{(L)}$ and $\mathcal{Q}^{(R)}$ incurs cost either before or after time t, but not both. For $X \in \{L, R\}$, let $q_X = \lfloor \left|\mathcal{Q}^{(X)}\right| / g \rfloor$ and $r_X = \left|\mathcal{Q}^{(X)}\right| \mod g$. Let also $q = q_L + q_R$. Then

$$tput^* = \left|\mathcal{Q}^{(L)}\right| + \left|\mathcal{Q}^{(R)}\right| = g \cdot q + r_L + r_R.$$

By the assumption of our lemma $tput^* > 4g$. Therefore, $q \geq 3$. Let $\mathcal{Q}^{(X)} = \{J_0, J_1, \ldots\}$ where the jobs are indexed in non-increasing order of their head lengths. In a one-sided clique instance the minimum cost is obtained by scheduling the first g jobs on one machine, the next g jobs on another machine and so on. Therefore $\overline{cost}^*(\mathcal{Q}^{(X)})$ is the sum of the head lengths of the first elements of each set, i.e. $len(J_0) + len(J_g) + len(J_{2g}) + \cdots$. As the sequence is non-increasing, the elements in odd indices of the sequence sum up to at most half of the sum, namely $len(J_g) + len(J_{3g}) + \cdots \leq \overline{cost}^*(\mathcal{Q}^{(X)})/2$. This sum is the reduced cost of the schedule s_X that schedules the jobs of every second group of $\mathcal{Q}^{(X)}$ and leaves the rest unscheduled. Let $k_X = tput^{s_X}$. We observe that the reduced cost of a schedule s'_X that schedules k_X jobs of $\mathcal{J}^{(X)}$ with shortest heads (i.e. the prefix $\mathcal{J}^{(X,k_X)}$) is at most $\overline{cost}^{s_X}$ because jobs are only replaced with jobs with shorter heads. Therefore

$$\overline{cost}^{s'_X}(\mathcal{J}^{(L,k_X)}) \leq \overline{cost}^{s_X} \leq \overline{cost}^*(\mathcal{Q}^{(X)})/2.$$

Consider the schedule s' that is obtained by the union of the schedules s'_L and s'_R. We have $\overline{cost}^{s'} = \overline{cost}^{s'_L}(\mathcal{J}^{(L,k_L)}) + \overline{cost}^{s'_R}(\mathcal{J}^{(R,k_R)}) \leq \overline{cost}^*/2 \leq T/2$. Observe that ALG1 considers the prefix pair $\mathcal{J}^{(L,k_L)}, \mathcal{J}^{(R,k_R)}$ in one of its iterations, thus it will return a schedule with throughput no less than the throughput of s', i.e. $k_L + k_R$.

We are now ready to finalize the proof using the above facts. Let $p_X = q_X \mod 2$. s_X schedules the jobs of every second group. The first $\lfloor q_X/2 \rfloor$ groups contain g jobs each, and whenever q_X is odd there is a last group with r_X jobs. Therefore

$$k_X = g \cdot \lfloor q_X/2 \rfloor + p_X \cdot r_X = g \cdot \frac{q_X}{2} + p_X \cdot r_X - p_X \frac{g}{2}.$$

The schedule returned by the algorithm contains at least

$$k_L + k_R = g \cdot \frac{q}{2} + p_L \cdot r_L + p_R \cdot r_R - (p_L + p_R)\frac{g}{2}$$

jobs. Then the approximation ratio ρ of the algorithm is at most

$$\rho \leq \frac{g \cdot q + r_L + r_R}{g \cdot \frac{q}{2} + p_L \cdot r_L + p_R \cdot r_R - (p_L + p_R)\frac{g}{2}}.$$

We consider three cases depending on p_R and p_L. Recall that $q \geq 3$.

- $p_L = p_R = 0$:
$$\rho \leq \frac{g \cdot q + r_L + r_R}{g \cdot \frac{q}{2}} < \frac{g \cdot (q+2)}{g \cdot \frac{q}{2}} = 2 + \frac{4}{q} < 4.$$

- $p_L = 1, p_R = 0$ ($p_L = 0, p_R = 1$ is symmetric):
$$\rho \leq \frac{g \cdot q + r_L + r_R}{g \cdot \frac{q}{2} + r_L - \frac{g}{2}} \leq \frac{g \cdot q + r_R}{g \cdot \frac{q}{2} - \frac{g}{2}} < \frac{g \cdot (q+1)}{(q-1)\frac{g}{2}} = 2 + \frac{4}{q-1} \leq 4.$$

- $p_L = p_R = 1$: In this case q_L and q_R are both odd, i.e. q is even. Therefore $q \geq 4$. Then

$$\rho \leq \frac{g \cdot q + r_L + r_R}{g \cdot \frac{q}{2} + r_L + r_R - g} \leq \frac{g \cdot q}{g \cdot \frac{q}{2} - g} = 2 + \frac{4}{q-2} \leq 4.$$

□

It remains to find a good approximation for the case *tput*$^* \leq 4g$. In this case to find a schedule that schedules g jobs on one machine would be a 4-approximation. We say that an interval I *covers* a subset $\mathcal{Q} \subseteq \mathcal{J}$ of jobs if $SPAN(Q) \subseteq I$ are contained in it. The *coverage* of I is the subset $\mathcal{Q}$ of $\mathcal{J}$ of jobs contained in I. The coverage of some given interval I can clearly be computed in linear time. Now we observe that in a clique instance, for any subset $\mathcal{Q} \subseteq \mathcal{J}$ of jobs, $SPAN(\mathcal{Q})$ is determined by at most two jobs, each one determining one endpoint. In other words there exist two (not necessarily distinct) jobs $J_l, J_r \in \mathcal{Q}$ such that $SPAN(\{J_l, J_r\}) = SPAN(\mathcal{Q})$. We conclude that the number of all possible distinct intervals $SPAN(\mathcal{Q})$ is at most $|\mathcal{J}|^2$. ALG2 below is a polynomial-time algorithm by the preceding discussion.

Algorithm 6 ALG2

```
1: Try each possible pair $J_i, J_j$ of jobs with $span(\{J_i, J_j\}) \leq T$.
2: choose the pair whose span covers the maximum number $m$ of jobs.
3: if $m \leq g$ then
4:     assign all the jobs in the coverage of $SPAN(\{J_i, J_j\})$ to the same machine.
5: else
6:     choose arbitrarily $g$ jobs from the coverage of $SPAN(\{J_i, J_j\})$.
7:     assign them to the same machine.
8: end if
```

Lemma 4.2 *If tput*$^* \leq 4g$ *then* ALG2 *is a 4-approximation algorithm for clique instances of* MAX-THROUGHPUT.

Proof: Consider the span $SPAN(\mathcal{J}^*)$ of all jobs scheduled by some optimal schedule s^*. Then $T \geq cost^* \geq span(\mathcal{J}^*)$. ALG2 will consider this span in one of the iterations therefore the value of m will be at least *tput**. If *tput*$^* \geq g$ then $m \geq g$ and the algorithm will schedule g jobs. In this case the approximation ratio is at most $4g/g = 4$. If *tput*$^* < g$ then *tput*$^* \leq \min(m, g)$ and the algorithm schedules $\min(m, g)$ jobs, therefore optimal.

□

By considering the combined algorithm that runs ALG1 and ALG2 and returns the best of the two schedules, we conclude by Lemmata 4.1 and 4.2:

Theorem 4.1 *There is a* 4*-approximation algorithm for clique instances of* MAXTHROUGHPUT.

Algorithms ALG1 and ALG2 can be implemented more efficiently by sorting the jobs and calculating the costs of every prefix. The cost of every prefix can be calculated based on the cost of the previous prefix. Finally for each left prefix, the corresponding best feasible right prefix can be found at logarithmic time using binary search. As we are interested mainly in approximation ratios, we have chosen the above description for ease of exposition.

4.2 Proper Clique Instances

In this section we give a polynomial-time dynamic programming algorithm for proper clique instances of MaxThroughput. We first show, in Lemma 4.3, a structural property of an optimal solution which is an extension of the consecutiveness property proven in Lemma 3.3. Recall that without loss of generality $J_1 \leq J_2 \leq \cdots \leq J_n$, and that a subset $\mathcal{Q} \subseteq \mathcal{J}$ is said to be consecutive in $\mathcal{J}$ if $\mathcal{Q} = \{J_i, J_{i+1}, \ldots, J_j\}$ for some $i \leq j$.

Lemma 4.3 *For proper clique instances of* MaxThroughput *there is an optimal (partial) schedule such that $\mathcal{J}_i$ is consecutive in $\mathcal{J}$ for every machine M_i.*

Note that the statement of Lemma 4.3 is the same as Lemma 3.3 except that an optimal schedule may be partial, and therefore some jobs may be left unscheduled.
Proof: Let s^* be a schedule that schedules maximum number of jobs and has minimum cost among such schedules. Let $\mathcal{J}^* \subseteq \mathcal{J}$ be the set of jobs scheduled by s^*. For each machine M_i, $\mathcal{J}_i^*$ is consecutive in $\mathcal{J}^*$ by Lemma 3.3. It remains to prove that $\mathcal{J}_i^*$ is consecutive in $\mathcal{J}$. Assume, by way of contradiction that for some machine M_i, $\mathcal{J}_i^*$ is consecutive in $\mathcal{J}^*$ but not consecutive in $\mathcal{J}$. Then $\mathcal{J}_i^* = \{J_{i_1}, J_{i_2}, \cdots, J_{i_k}\}$, where $i_1 < i_2 < \cdots < i_k$ and there is at least one non-scheduled job J_x such that $i_1 < x < i_k$. Then we can schedule J_x on M_i and unschedule J_{i_1}. Since the instance is a clique instance this still gives a valid schedule and since the instance is a proper instance, J_x is entirely within the span of $\mathcal{J}_i^*$, thus the cost can only decrease. This process can be repeated until $\mathcal{J}_i^*$ is consecutive in $\mathcal{J}$.

□

By Lemma 4.3, each machine processes a set of consecutive jobs in $\mathcal{J}$, and a (possibly empty) set of consecutive unscheduled jobs are between the jobs of two consecutive machines M_i and M_{i+1}. With this observation, we formulate a dynamic program to find the minimum cost of scheduling a subset of jobs. We define the cost function $\mathit{cost}(i,j,u,t)$ as the minimum cost of scheduling the instance consisting of the first i jobs of $\mathcal{J}$ such that the last machine processes exactly j jobs, the last (exactly) u jobs are not scheduled and a total of t (out of the i jobs) are not scheduled. A valid schedule is a scheduling such that $\mathit{cost}(n,j,u,t) \leq T$ and an optimal schedule is one that has the minimum value of t. In other words, the maximum throughput is

$$n - \min\{t \mid \mathit{cost}(n,j,u,t) \leq T\} \ .$$

$\mathit{cost}(i,j,u,t)$ can be calculated based on previously calculated values $\mathit{cost}(i-1,?,?,?)$ as follows. For any $1 \leq i \leq n$, $1 \leq j \leq \min(i,g)$, $0 \leq u \leq i-j$, and $u \leq t \leq i-j$,

$$\mathit{cost}(i,j,u,t) = \begin{cases} \mathit{cost}(i-1,j,u-1,t-1) \\ \qquad \text{if } u > 0, \\ \mathit{cost}(i-1,j-1,u,t) + |P_i| - |I_{i-1}| \\ \qquad \text{if } u = 0 \text{ and } j > 1, \\ \min_{j',u'} \mathit{cost}(i-1,j',u',t) + |P_i| \\ \qquad \text{if } u = 0 \text{ and } j = 1. \end{cases} \tag{7}$$

In the last case, the ranges of j' and u' are: $1 \leq j' \leq \min(g, i-1-t)$ and $1 \leq u' \leq \min(i-1-j', u)$.

When $u > 0$, the schedule with $\mathit{cost}(i,j,u,t)$ is supposed to have the least u jobs non-scheduled and thus the schedule of the first $i-1$ jobs would have the least $u-1$ jobs non-scheduled, i.e., with $\mathit{cost}(i-1,j,u-1,t-1)$. When $u = 0$, it means the last j jobs are supposed to be processed on the same machine. If $j > 1$, it means the schedule for the first $i-1$ jobs should have the last $j-1$

Algorithm 7 MostThroughputConsecutive

```
1: $cost(1,1,0,0) \leftarrow |T|$
2: for $i = 2$ to $n$ do
3:     for $j = 1$ to $\min(i, g)$ do
4:         for $u = 0$ to $i - j$ do
5:             for $t = u$ to $i - j$ do
6:                 if $u > 0$ then
7:                     $cost(i,j,u,t) = cost(i-1,j,u-1,t-1)$
8:                 else
9:                     if $j > 1$ then
10:                        $cost(i,j,u,t) = cost(i-1,j-1,u,t) + |P_i| - |I_{i-1}|$
11:                    else
12:                        $cost(i,j,u,t) = \min_{\substack{1 \le j' \le \min(g,i-1-t) \\ 1 \le u' \le \min(i-1-j',u)}} cost(i-1,j',u',t) + |P_i|$
13:                    end if
14:                end if
15:            end for
16:        end for
17:    end for
18: end for
```

jobs assigned to the same machine and J_i is scheduled on the same machine as these $j-1$ jobs, then the cost would become $cost(i-1,j-1,u,t) + |J_i| - |I_{i-1}|$. Otherwise if $j = 1$, it means that J_i is scheduled to a new machine and the schedule for the first $i-1$ machines can have any valid value of j' and u' and so the cost can be computed as shown in recurrence 7. The 4-dimensional table can be filled by the dynamic programming algorithm MostThroughputConsecutive. It contains $n^3 g$ entries, among which the computation of at most n^2 entries requires $O(gn)$ time and the rest requires $O(1)$ time. We conclude

Theorem 4.2 *There exists a polynomial-time algorithm for proper clique instances of* MaxThroughput *that computes an optimal schedule in time* $O(|\mathcal{J}|^3 \cdot g)$.

5 Summary, Extensions and Future Work

In this paper we revisited the problem MinBusy and initiated the study of the problem MaxThroughput of busy time optimization. The full list of results is given in Section 1. In particular we presented three algorithms for MinBusy improving upon existing ones. Specifically, we presented a) a polynomial-time optimal algorithm for clique instances when $g = 2$, b) A $\frac{g \cdot H_g}{H_g + g - 1}$-approximation algorithm for clique instances, and c) A $(2 - 1/g)$-approximation algorithm for proper instances. The second algorithm led to an improvement when g attains small values.

The following open problems are of interest:

- Complexity: The exact complexity of MinBusy and MaxThroughput for clique instances and for proper instances is still open. In this work approximation algorithms were presented.
- MinBusy and MaxThroughput: We have shown that MaxThroughput is NP-Hard whenever MinBusy is NP-Hard. The question of whether MaxThroughput is strictly harder than MinBusy generally or in a special case is open.

- Approximation algorithms: Another open question is to improve the approximation ratio for MinBusy and derive approximation ratio for MaxThroughput, both for the general input instances.

- MaxThroughput: In this paper the throughput is measured by the number of jobs. A natural question is whether we can extend the results to weighted throughput.

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

Energy-aware scheduling: As we mentioned in Section 1, 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., [15, 22, 29]), which means the processor speed can be scaled up or down. In the context of busy time scheduling, the scheduler may speed up the processor to shorten the busy time, resulting in shorter time of processing but higher energy 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 [2,7]. 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., [10, 23, 26]) as presented in Section 1.

- An extension is to allow jobs requiring different amount of capacities and a machine can process jobs as long as the sum of capacity required is at most g [16].

- Other extensions are to have different machine types of different computing power, different capacities, and/or allow migration of jobs with possibly a penalty.

- In this work the jobs are supposed to be processed during the whole period from start time s_j to completion time c_j. One may consider also
  - jobs that also have processing time $p_j \geq c_j - s_j$ and have to be processed for p_j consecutive time units during the interval $[s_j, c_j]$ (see e.g. [25]),
  - *malleable* jobs which can be assigned several machines and the actual processing time depends on the number of machines allocated (see e.g., [21, 25]).

Optical network design: As detailed in Section 1, the scheduling problems studied in this paper have a direct application to problems in placement of regenerators in optical network design. Our work is related to two regenerator optimization problems with traffic grooming for network with a line topology. 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 total 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 that maximizes the number of satisfied paths. Some of our results can be extended to other topologies. In particular:

- The algorithm in Observation 3.1 for one-sided clique instances maintains a current machine with at most g jobs, a new job is added to this machine if it has less than g jobs, otherwise a new machine with only the new job is created and designated as the current machine. This extends to tree topologies as follows: We maintain multiple current sets, and process the paths in non-increasing order. The *opening job* of a set $\mathcal{J}_i$ is the first (i.e. longest) job it processes. A current set $\mathcal{J}_i$ is possible for a new job J if J is contained in the opening path of $\mathcal{J}_i$ and also $|\mathcal{J}_i| < g$. Each path is added to the possible set with the biggest number of paths.

- Theorem 3.3 is valid in ring topologies. In this case jobs represent communication requests in a ring optical network that arrive with start and end times. More specifically, a customer needs an optical communication line between two given notes during a time period. It can be verified that Lemma 3.4 holds for ring topologies. The conclusion of Theorem 3.3 from this lemma is topology-independent.

Other extensions include the following:

- In the regenerator placement problem the extension of variable capacity requirement applies, i.e. input paths require different amount of bandwidth.

- The scheduling problem we considered, in the context of regenerator placement, corresponds to a requirement that a regenerator needs to be placed in every node along a path. This requirement can be generalized to the case where a regenerator is only needed within every d hops, for some constant d.

- Another natural extension is to consider other topologies.

Acknowledgement: We thank Dariusz R. Kowalski and Mariusz A. Rokicki for helpful discussions.

References

[1] A compendium of NP optimization problems. http://www.nada.kth.se/~viggo/wwwcompendium/node143.html#5968.

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

[3] 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*, 48(5):1069–1090, 2001.

[4] 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.

[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] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] J. Y.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

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

[22] 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.

[23] 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.

[24] 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 '00, pages 879–888, 2000.

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

[26] 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.

[27] 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 '09, pages 37–42, 2009.

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

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