Citation for published item:

Further information on publisher’s website:
http://dx.doi.org/10.1109/IPDPS.2012.31

Publisher’s copyright statement:
© 2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Additional information:

Use policy
The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in DRO
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full DRO policy for further details.
Optimizing Busy Time on Parallel Machines

George B. Mertzios∗, Mordechai Shalom†, Ariella Voloshin‡, Prudence W.H. Wong§ and Shmuel Zaks†

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

Abstract—We consider the following fundamental 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 start time and a completion time; it is supposed to be processed from the start time to the completion time (and in one of our extensions it has to be scheduled also in a continuous number of days; this corresponds to a two-dimensional version 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 are 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 power consumption, cloud computing and optimizing switching cost of optical networks.

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

I. INTRODUCTION

Problem Statement Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [4], [28]). 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 \( \mathcal{J} = \{J_1, \ldots, J_n\} \). 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 \( g \geq 1 \), which is the 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 \( \text{busy}_i \). In this work we study two optimization problems \text{MINBUSY} and \text{MAXTHROUGHPUT}. In \text{MINBUSY} we focus on minimizing the total busy times over all machines, denoted by \( \sum_i \text{busy}_i \). 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 \text{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 power-aware scheduling problems in cluster systems. These problems focus on minimizing the power consumption of a set of machines (see, e.g., [26] 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., [22], [25]). 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., [8], [10], [11]). 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.

**Related Work**  Some of the earlier work on interval scheduling considers the problem of scheduling a feasible subset of jobs whose total weight is maximized, i.e., a maximum weight independent set (see, e.g., [2] 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., [2], [5], [7], [23]). 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 [4]) 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., [4].) 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 \( \text{MinBusy} \) was studied in [27], showing that the problem is NP-Hard already for \( g = 2 \). The work [12] 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, and instances where any two job intervals intersect, i.e., the input forms a clique (see same approximation but different algorithm and analysis in [13]). The work [16] extends the results of [12], considering the case where each job has a different demand on machine capacity.

The minimization problem \( \text{MinBusy} \) studied in this paper is related to the problems studied in [12], [16]. As will be discussed in Section V, 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 \( \text{MinBusy} \) 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). We also consider a two-dimensional version 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 \( g = 2 \).
- A \( \frac{gH_g}{g+H_g} \)-approximation algorithm for clique instances, where \( H_g \) is the \( g \)-th harmonic number, for any fixed value of \( g \).
- A \((2-1/g)\)-approximation algorithm for proper instances.
- A polynomial time algorithm for proper clique instances, based on an interesting combinatorial property of optimal solutions.
- 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 \( \min(g, 6.31 \log \min(\gamma_1, \gamma_2)) \)-approximation algorithm.

We show that \( \text{MaxThroughput} \) is NP-Hard whenever \( \text{MinBusy} \) is NP-Hard. For \( \text{MaxThroughput} \) we show the following:

- A 6-approximation algorithm for clique instances for any \( g \).
- A polynomial time algorithm for proper clique instances.
It turns out the algorithms and analysis of the results for MAX-THROUGHPUT are more involved than those for MINBUSY, apparently due to an inherent hardness of the former.

**Organization of the paper** In Section II we present some preliminaries. MINBUSY is studied in Section III, and MAXTHROUGHPUT in Section IV. In Section V we first summarize our results, then we discuss extensions and further research directions. Some proofs are only sketched in this Extended Abstract.

II. 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 $\text{len}(I) = c_I - s_I$. This extends to a set $\mathcal{I}$ of intervals; namely, the length of $\mathcal{I}$ is $\text{len}(\mathcal{I}) = \sum_{I \in \mathcal{I}} \text{len}(I)$.

**Definition 2.2:** For a set $\mathcal{I}$ of intervals we define $\text{SPAN}(\mathcal{I}) \equiv \bigcup \mathcal{I}$ and $\text{span}(\mathcal{I}) \equiv \text{len}(\text{SPAN}(\mathcal{I}))$ 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 $\text{span}(\mathcal{I}) \leq \text{len}(\mathcal{I})$ 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 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 2 jobs at 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 $s : \mathcal{J} \rightarrow \mathcal{M}$, we denote by $\mathcal{J}_s$ the set of jobs assigned to machine $M_i$ by schedule $s$, i.e. $\mathcal{J}_s \equiv s^{-1}(M_i)$. The cost of machine $M_i$ in this schedule is the length of its busy interval, i.e. $\text{busy}_s \equiv \text{span}(\mathcal{J}_s)$.

Given a partial schedule $s$, we denote the set of jobs scheduled by it as $\mathcal{J}_s \equiv \bigcup \mathcal{J}_s$. Its cost is $\text{cost}_s \equiv \sum \text{busy}_s$, and its throughput is $\text{tput}_s \equiv |\mathcal{J}_s|$. When there is no ambiguity for the schedule under consideration, we omit the superscripts (e.g. we use $\mathcal{J}_i$ for $\mathcal{J}_s$, 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 $s : \mathcal{J} \rightarrow \mathcal{M}$.
- **Objective:** Minimize $\text{cost}_s$.

**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 $s : \mathcal{J} \rightarrow \mathcal{M}$ such that $\text{cost}_s < T$.
- **Objective:** Maximize $\text{tput}_s$.

W.l.o.g. we assume that each machine is busy along a contiguous time interval, because 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 well 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 $J, J' \in \mathcal{J}$ $s_J \leq s_J'$ if and only if $c_J \leq c_J'$. We denote this fact as $J \leq J'$ and w.l.o.g. we assume $J_1 \leq J_2 \leq \ldots \leq J_n$.

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

**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: $\text{cost}_s \geq \frac{\text{len}(\mathcal{J})}{g}$.
- The span bound: $\text{cost}_s \geq \text{span}(\mathcal{J})$.
- The length bound: $\text{cost}_s \leq \text{len}(\mathcal{J})$.

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 $t \in \mathcal{J}$ 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.
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 \((J, g)\) of MINBUSY we can perform binary search between \(\text{len}(J)/g\) and \(\text{len}(J)\) for the value of \(T\) by solving each time the instance \((J, g, T)\) of MAXTHROUGHPUT to find the smallest value \(T\) such that \(\text{tpu}^*(J, g, T) \geq \text{tpu}^*(J)\), i.e. to find the value \(\text{cost}^*\).

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

**Proposition 2.3:** If there is a polynomial-time computable set \(^1\) \(X \subseteq 2^J\) containing at least one set \(J^*\) for some optimal schedule \(s\), and also MINBUSY can be solved optimally, then MAXTHROUGHPUT can be solved optimally.

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

For any schedule \(s\) we define the saving \(\text{sav}^*(\text{cost, relative to the worst schedule})\) achieved by \(s\) as \(\text{sav}^* \triangleq \text{len}(J) - \text{cost}^*\). As far as optimal schedules are concerned, MINBUSY can equivalently be reformulated as the problem of maximizing \(\text{sav}^*\). 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.

**Lemma 2.1:** If a schedule is a \(\rho\)-approximation to the saving maximization problem for some \(\rho \geq 1\), then it is a \((1/\rho + (1 - 1/\rho) g)^{-1}\)-approximation to MINBUSY.

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

\[
\text{cost}^* - \text{cost}^* = \text{sav}^* - \text{sav}^* \leq (1 - \rho')\text{sav}^*
\]

\[
= (1 - \rho')((\text{len}(J) - \text{cost}^*))
\]

\[
= (1 - \rho')\text{len}(J) + (\rho' - 1)\text{cost}^*
\]

thus

\[
\text{cost}^* \leq (1 - \rho')\text{len}(J) + \rho' \cdot \text{cost}^*
\]

\[
\leq (1 - \rho')g \cdot \text{cost}^* + \rho' \cdot \text{cost}^*
\]

\[
= (\rho' + (1 - \rho')g)\text{cost}^*
\]

where the second inequality follows from the parallelism bound.

### III. Cost Minimization

In this section we study special cases, namely clique instances, proper instances and proper clique instances (in Sections III-A, III-B and III-C, 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 III-D).

---

\(^1\)Also of polynomial size. This assumption holds whenever the set is represented by an explicit list of its elements.
to get
\[ \text{cost}^* \leq \rho \cdot H_g \cdot \text{cost}^* \]
Since \( \rho = \frac{g}{H_g + g - 1} \), the algorithm is a \( \frac{g \cdot H_g}{H_g + g - 1} \)-approximation.

As can be easily verified, the function \( \frac{g \cdot H_g}{H_g + g - 1} \) is monotonically increasing, and it gets values smaller than 2 for \( g \leq 6 \).

Note also that the set cover technique used in the last lemma can be used to derive an exact algorithm for \( g = 2 \); however for this case it seems that the algorithm mentioned in Lemma 3.1, using the matching technique, is better extendable to other contexts.

**B. 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 [12]. Recall that we assume without loss of generality that a) the instance is connected, and b) \( J_1 \leq J_2 \leq \ldots \leq J_n \), i.e. the start time of the jobs is non-decreasing.

**Algorithm 1 BESTCUT(\( \mathcal{J}, g \))**

1. for \( i = 1 \) to \( g \) do
2. \( s_i^0 = \{J_1, \ldots, J_i\} \);
3. \( \text{for } j = 1 \) to \( \lceil(\lvert \mathcal{J} \rvert - 1)/g \rceil \) do
4. \( s_i^j = \{J_{i+g(j-1)+1}, \ldots, J_{i+g \cdot j}\} \);
5. \( \text{end for} \);
6. \( s^i = \{s_i^0, s_i^1, \ldots\} \);
7. \( \text{end for} \);
8. Return as \( s \) one of \( s^1, s^2, \ldots, s^g \) such that \( \text{cost}^* \) is minimized.

**Theorem 3.1:** Algorithm BESTCUT is a \( (2 - 1/g) \)-approximation for proper instances of MINBUSY.

**Proof:** For every \( i \) and \( j \), \( \lvert s_i^j \rvert \leq g \), therefore the schedule \( s \) returned by the algorithm is feasible. We first give an upper bound to \( \text{sav}^* \): by the span bound we have \( \text{cost}^* \geq \text{span}(\mathcal{J}) \).

On the other hand \( \text{span}(\mathcal{J}) = \text{len}(\mathcal{J}) - \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert \), where \( I_k \) is the overlap between jobs \( J_k \) and \( J_{k+1} \). Therefore \( \text{cost}^* \geq \text{len}(\mathcal{J}) - \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert \), or equivalently
\[
\text{sav}^* \leq \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert. \tag{1}
\]

We proceed by calculating \( \text{cost}^* \). For every \( 1 \leq i \leq g \), \( \text{cost}^* = \text{span}(s_i^0) + \sum_{j \geq 0} \text{span}(s_i^j) \). Moreover, \( \text{span}(s_i^0) = \text{len}(s_i^0) = \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert \) and \( \text{span}(s_i^j) = \text{len}(s_i^j) = \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert + \sum_{j \geq 0} \lvert I_{i+g \cdot j} \rvert \). Therefore \( \text{cost}^* = \text{len}(\mathcal{J}) - \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert + \sum_{j \geq 0} \lvert I_{i+g \cdot j} \rvert \), or equivalently
\[
\text{sav}^* = \sum_{k=1}^{\lvert \mathcal{J} \rvert - 1} \lvert I_k \rvert - \sum_{j \geq 0} \lvert I_{i+g \cdot j} \rvert. \tag{2}
\]

Combining (1) and (2) we conclude that BESTCUT is a \( \frac{2}{g} \)-approximation for the saving maximization problem. Substituting this for \( \rho \) in Lemma 2.1 we get the \( 2 - \frac{1}{g} \) approximation ratio for MINBUSY.

**C. Proper Clique Instances**

In this section we consider instances that are clique instances and also proper instances. We say that a subset \( Q \subseteq \mathcal{J} \) of jobs is consecutive in \( \mathcal{J} \) (or simply consecutive) if \( Q = \{J_i, J_{i+1}, \ldots, J_j\} \) for some \( 1 \leq i \leq j \leq n \). The following lemma leads to a polynomial-time algorithm for these instances of MINBUSY.

**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:** We first give an informal outline of the proof. Suppose that for any optimal schedule \( s^* \), there is at least one machine \( M_i \), such that 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 \( s^*(J_b) = M_i \neq M_i \). We call such a set of jobs a conflicting triple \( \langle a, b, c \rangle \) of \( s^* \). We consider an optimal schedule \( s^* \), in which the largest possible value of \( b \), such that \( \langle a, b, c \rangle \) is a conflicting triple of \( s^* \), is as small as possible. That is, every optimal schedule \( s^* \) has a conflicting triple \( \langle a', b', c' \rangle \), where \( b' \geq b \). Then, we construct another optimal schedule \( s^{**} \) from \( s^* \) by rescheduling appropriately the subset of jobs that are scheduled to \( M_i \) or \( M_i \) in \( s^* \). We prove that, by this construction, for every conflicting triple \( \langle a'', b'', c'' \rangle \) of the new optimal schedule \( s^{**} \), we have \( b'' < b \). This contradicts the way that the optimal schedule \( s^* \) has been chosen.

We proceed with the formal proof of the lemma. Suppose that for any optimal schedule \( s^* \), there is at least one machine \( M_i \), such that the subset \( \mathcal{J}_i \) is not consecutive. Then there exist three distinct jobs \( J_a \leq J_b \leq J_c \) such that \( s^*(J_a) = s^*(J_c) = M_i \) and \( s^*(J_b) = M_i \neq M_i \) which we call a conflicting triple \( \langle a, b, c \rangle \) of \( s^* \). Given an optimal schedule \( s^* \), a conflicting triple \( \langle a, b, c \rangle \) of \( s^* \) is called rightmost if there exists no conflicting triple \( \langle a', b', c' \rangle \) of \( s^* \), such that \( b' > b \).

Among all optimal schedules, let \( s^* \) be one, in which the value \( b \) of a rightmost conflicting triple \( \langle a, b, c \rangle \) is the smallest possible. That is, for a rightmost conflicting triple \( \langle a', b', c' \rangle \) of an arbitrary optimal schedule \( s' \), we have \( b \geq b' \). Let now \( a, b, c \) be a rightmost conflicting triple of \( w^* \), and let \( s^*(J_a) = s^*(J_c) = M_i \) and \( s^*(J_b) = M_i \). Without loss of generality, we assume \( a \) (resp. \( c \)) is the smallest (resp. largest) index in \( s^* \), for which \( s^*(J_a) = M_i \) (resp. \( s^*(P_c) = M_i \)).
Note that all jobs between \(J_b\) and \(J_c\) are scheduled to \(M_i\) in \(s^*\). Indeed, otherwise there exists an index \(d \in \{b+1, \ldots, c-1\}\) with \(s^*(P_d) = M_{i'}, \neq M_i\), and thus \((a, d, c)\) is a conflicting triple in \(s^*\), for which \(d > b\). This is a contradiction, since \((a, b, c)\) is assumed to be a rightmost conflicting triple of \(s^*\).

We will now construct another optimal schedule \(s^{**}\) that has a rightmost conflicting triple \((a', b', c')\) with \(b' < b\), which is a contradiction to the way that \(s^*\) has been chosen. The schedule \(s^{**}\) will be obtained from \(s^*\) by rescheduling some of the jobs scheduled to \(M_i\) and \(M_{i'}\). That is, for every index \(x\), \(s^*(J_x) \in \{M_i, M_{i'}\}\) if and only if \(s^{**}(J_x) \in \{M_i, M_{i'}\}\). Since the instance is an clique instance, a schedule is feasible if and only if \(|J_x| \leq g\) for each machine \(M_i\). This property will be preserved when obtaining \(s^{**}\) from \(s^*\).

Let \(k\) be the smallest index, for which \(s^*(J_k) = M_{i'}\); clearly \(k \neq a\). We distinguish between the following two cases regarding \(k\) and \(a\).

- \(k < a\). This case is illustrated in Figure 1(a). In this case, we construct the schedule \(s^{**}\) by exchanging the machine assignments of \(J_a\) and \(J_b\). That is, \(s^*(J_a) = M_{i'}\), \(s^*(J_b) = M_i\), and \(s^*(J_x) = s^*(J_x)\) for every \(x \neq \{a, b\}\). Then, since \(J_a\) is the leftmost job scheduled to \(M_i\) and \(J_b\) is the rightmost job scheduled to \(M_{i'}\) in \(s^*\), it follows that \(\text{span}(J_a^{**}) \leq \text{span}(J_b^{**})\) and \(\text{span}(J_b^{**}) \leq \text{span}(J_a^{**})\), i.e. \(s^{**}\) is optimal.

- \(a < k\). This case is illustrated in Figure 1(b). Note that the set of jobs scheduled to either \(M_i\) or \(M_{i'}\) in \(s^*\) contains \(t \leq 2g\) elements, since \(|J_a^{**}| \leq g\) and \(|J_b^{**}| \leq g\). Furthermore, since \(a < k\), the leftmost and the rightmost jobs of this set are scheduled to \(M_i\) in \(s^*\). Suppose that in this set of jobs, there are \(t_1 \geq 1\) jobs to the left of \(J_k\) and \(t_2 \geq 1\) jobs to the right of \(J_b\). That is, the leftmost \(t_1\) jobs and the rightmost \(t_2\) jobs of this set are scheduled to \(M_i\) in \(s^*\), and thus \(t_1 + t_2 \leq t \leq g\). Let \(t_0 = \min\{g, t - t_2 - 1\}\).

We construct \(s^{**}\) by scheduling the leftmost \(t_0 \leq g\) jobs to \(M_i\), and the remaining \(t - t_0 \leq g\) jobs to machine \(M_{i'}\), and thus that \(s^{**}\) is feasible. Let \(J_x\) be the rightmost job scheduled to \(M_i\) and \(J_y\) be the leftmost job scheduled to \(M_{i'}\) in \(s^{**}\). Note that, if \(k = b\), then \(y = k = b\). Otherwise, if \(k \neq b\), then \(k \leq y < b\). In the schedule \(s^*\), the cost of the jobs scheduled to \(M_i\) and \(M_{i'}\) is \(\text{span}(\{J_x, J_y\}) + \text{span}(\{J_y, J_b\})\). Similarly, in \(s^{**}\), the cost of the jobs scheduled to \(M_i\) and \(M_{i'}\) is \(\text{span}(\{J_x, J_y\}) + \text{span}(\{J_y, J_b\})\). Note that, since we assumed an clique instance, the completion time of any job is greater than or equal to the start time of any job. Therefore, in both cases where \(k = b\) and \(k \neq b\), it is easy to verify that \(\text{span}(\{J_x, J_y\}) + \text{span}(\{J_y, J_b\}) \leq \text{span}(\{J_x, J_y\}) + \text{span}(\{J_y, J_b\})\), and thus \(s^{**}\) is optimal.

It remains to show now that, in both cases where \(k < a\) and \(a < k\), the constructed optimal schedule \(s^{**}\) has a rightmost conflicting triple \((a^*, b^*, c^*)\) with \(b^* < b\). Let \(\langle a^*, b^*, c^*\rangle\) be a maximal conflicting triple of \(s^{**}\). If the machines assigned to \(J_{a^*}\) and \(J_{c^*}\) in \(s^{**}\) are in \(\{M_i, M_{i'}\}\), then \(k < a\) (indeed, according to the above construction of \(s^{**}\), in the case where \(a < k\), there is no conflicting triple in \(s^{**}\) with machines in \(\{M_i, M_{i'}\}\)). In this case, \(b^* < b\) by the construction of \(s^{**}\).

Consider now the case where at least one of the jobs \(J_{b^*}, J_{c^*}\) is scheduled to \(M_{i'} \notin \{M_i, M_{i'}\}\) by \(s^*\) (in both cases \(k < a\) and \(a < k\)). Suppose that \(b^* \geq b\). Let first \(b^* > c\), i.e. \(b^* > c > b\). Then, since \(c\) is assumed to be the largest index in \(s^*\), for which \(s^*(J_c) = M_i\), it follows that all jobs \(J_{a^*}, J_{b^*}, J_{c^*}\) are scheduled to machines different than \(M_i, M_{i'}\) in both \(s^*\) and \(s^{**}\). Therefore \(\langle a^*, b^*, c^*\rangle\) is also a conflicting triple in \(s^*\), where \(b^* > b\), which is a contradiction, since \((a, b, c)\) is assumed to be a rightmost conflicting triple of \(s^*\). Let now \(b^* < b\). Then, as we proved above, all jobs between \(J_b\) and \(J_c\) are scheduled to \(M_i\) in \(s^*\), and thus in particular \(s^*(J_{b^*}) \in \{M_i, M_{i'}\}\). Furthermore, by the above construction of the new optimal schedule \(s^{**}\) (in both cases where \(k < a\) and \(a < k\)), it follows that \(s^{**}(J_{b^*}) = M_{i'}\). Therefore \(s^{**}(J_{c^*}) = M_{i'}\) by assumption, where \(i^* \notin \{i, i'\}\), and thus also \(s^{**}(J_{c^*}) = M_{i'}\).

Furthermore, since all jobs between \(J_b\) and \(J_c\) are scheduled to \(M_i\) in \(s^*\), it follows that \(c^* > c\). Therefore, \(\langle a^*, c^*, c^*\rangle\) is a conflicting triple of \(s^*\), where \(c > b\), which is a contradiction, since \((a, b, c)\) is a rightmost conflicting triple of \(s^*\). Therefore, again \(b^* < b\).

Summarizing, we constructed an optimal schedule \(s^{**}\), having a rightmost conflicting triple \((a^*, b^*, c^*)\) with \(b^* < b\). This is a contradiction to our assumption on \(s^*\). Therefore, there exists an optimal schedule \(s^*\), such that for every machine \(M_i\), the subset \(J_i\) is consecutive. This completes the proof of the lemma.

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 \(\text{cost}^*(i)\) the cost of an optimal schedule of the sub-instance consisting of the leftmost \(i\) jobs, and by \(\text{cost}^*(i, j)\) 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 \(\text{cost}^*(i) = \min(\text{cost}^*(i, 0), \text{cost}^*(i, 1), \ldots, \text{cost}^*(i, g-1))\).
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) ← cost^*(1, 1) ← span(J_1)
2: for i = 2 to n do
3: cost^*(i, 1) ← |J_i| + cost^*(i - 1)
4: for j = 2 to \min(g, i) do
5: cost^*(i, j) ← 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 to 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.

D. Rectangular Intervals

In this section we consider a generalization of the problem to 2 dimensions. In graph theoretic terms we are now dealing with rectangle graphs 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. 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 \( I = [s, c] \), where \( s = (s_1, s_2), c = (c_1, c_2) \in \mathbb{R} \times \mathbb{R} \) and \( s < c \), we define \( len_k(I) \) as the length of the interval.

Definition 3.2: For a set \( \mathcal{I} \) of rectangular intervals we define \( SPAN(\mathcal{I}) = \bigcup \mathcal{I} \) 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 II extend to this case. Moreover the bounds mentioned therein hold for this case too.

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

The following algorithm FirstFit is actually presented in [12] for the one dimensional case. We present it here for completeness and also adapt it to our case.

The following lemma generalizes Theorem 2.5 in [12] that proves that Algorithm FirstFit is a 4-approximation.

Lemma 3.4: Algorithm FirstFit is a \( (3\gamma_1 + 1) \)-approximation for MinBusy on rectangular intervals.

Proof: By Lemma 2.3 of [12] we have \( len_2(J_i) \geq \frac{g}{4} span_2(J_{i+1}) \). Let \( \ell = \min_{J \in \mathcal{J}} len_1(J). \) Then for every job \( J \in \mathcal{J}, \ell \leq len_1(J) \leq \ell \cdot \gamma_1, \) thus \( \ell \cdot len_2(J) \leq len(J) \leq \ell \cdot \gamma_1 \cdot len_2(J). \) Therefore for any subset \( \mathcal{J}' \subseteq \mathcal{J} \) of jobs \( \ell \cdot len_2(J') \leq len(J') \leq \ell \cdot \gamma_1 \cdot len_2(J') \) and \( span(J') \leq \ell \cdot \gamma_1 \cdot span_2(J') \). 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. Thus,

\[
\text{FirstFit}(\mathcal{J}_{i+1}) = \text{bus}_{i+1} = \text{span}(\mathcal{J}_{i+1}) \leq \ell \cdot \gamma_1 \cdot \text{span}_2(\mathcal{J}_{i+1}) \leq \ell \cdot \gamma_1 \cdot \frac{3}{2} len_2(J_i) \leq \gamma_1 \cdot \frac{3}{2} len(J_i).
\]

Let \( m \geq 1 \) be the number of machines used by FirstFit. Then

\[
\sum_{i=2}^{m} \text{FirstFit}(\mathcal{J}_i) = \sum_{i=1}^{m-1} \text{FirstFit}(\mathcal{J}_{i+1}) \leq \gamma_1 \cdot \frac{3}{2} \sum_{i=1}^{m-1} \text{len}(\mathcal{J}_i) \leq 3 \gamma_1 \cdot \text{cost}^*(\mathcal{J}) (3)
\]

where the last inequality follows from the parallelism bound.

Now, using the span bound, we have that

\[\text{FirstFit}(\mathcal{J}_i) = \text{bus}_1 = \text{span}(\mathcal{J}_i) \leq \text{span}(\mathcal{J}) \leq \text{cost}^*(\mathcal{J}).\]

Combining with (3) we get

\[\text{FirstFit}(\mathcal{J}) \leq (3 \gamma_1 + 1) \cdot \text{cost}^*(\mathcal{J}).\]

Algorithm 3 FirstFit(\( J, g \))
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: Consider the jobs in the above order: assign the next job, \( J_j \), the first machine that is available for it, i.e., find the minimum value of \( i \geq 1 \) such that, at any time \( t \in J_j \), \( M_i \) is processing at most \( g - 1 \) jobs. If no such machine exists, then use a new machine for \( J_j \).

Algorithm 4 BucketFirstFit(\( G, J, g, \beta \))
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 \( \lceil \log_\beta \gamma_1 \rceil \) do
4: Let \( \mathcal{J}^{(b)} = \{ J \in \mathcal{J} : \ell \cdot \beta^{b-1} \leq len_1(J) \leq \ell \cdot \beta^b \} \)
5: Schedule the jobs in \( \mathcal{J}^{(b)} \) to a set of unused machines using algorithm FirstFit.
6: end for
\[ \text{cost}^*(J) \geq \text{cost}^*(J(b)) \]. Therefore

\[ \text{cost}^*(J) = \sum_{b=1}^{\log_\beta \gamma_1} \text{cost}^*(J(b)) \]

\[ \leq \sum_{b=1}^{\log_\beta \gamma_1} (3\beta + 1)\text{cost}^*(J(b)) \]

\[ \leq \sum_{b=1}^{\log_\beta \gamma_1} (3\beta + 1)\text{cost}^*(J) \]

\[ = \left( \frac{3\beta + 1}{\log \beta} \log \gamma_1 + O(\beta) \right) \cdot \text{cost}^*(J). \]

Substituting \( \beta = 3 \) and recalling our assumption \( \gamma_1 \leq \gamma_2 \) we get

**Theorem 3.3:** BUCKETFIRSTFIT \((G, J, g, 3)\) constitutes a \( \min(g, 6.31 \log \min(\gamma_1, \gamma_2) + O(1)) \)-approximation algorithm for MINBUSY on rectangular intervals.

### IV. 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.

#### A. Clique Instances

For one-sided clique instances we note that if a schedule \( s \) with \( \text{cost}^* \leq T \) schedules \( \text{put}^* \) jobs, then there is a schedule \( s' \) with \( \text{cost}^* \leq \text{cost}^* \leq T \) that schedules the shortest \( \text{put}^* \) jobs. In particular there is an optimal schedule that schedules the shortest \( j \) jobs for some \( 0 \leq j \leq |J| \). By Propositions 2.3 and 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. Let \( t \) be the time that is common to all the jobs. For a job \( J = [s, c] \) we term the sub-interval \([s, t]\) (resp. \([t, c]\)) 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 they are equal 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 \( J^{(L)} \) (resp. \( J^{(R)} \)) the subset of left-heavy (resp. right-heavy) jobs of \( J \). For \( X \in \{ L, R \} \), a subset containing the \( j \) jobs of \( J^{(X,j)} \) with shortest head lengths is termed a prefix of size \( j \) and denoted by \( J^{(X,j)} \).

The reduced cost of a schedule \( s \) of \( J \) is the cost of \( s \) where each job is replaced by its head, and we denote it by \( \text{cost}^*(J) \). In other words, in the reduced cost model the tails of the jobs do not consume machine time. Clearly \( \text{cost}^*(J) \leq \text{cost}^*(J) \). Moreover \( \text{cost}^*(J) \leq 2\cdot \text{cost}^*(J) \) because for each machine \( M_i, \text{span}(F_i) \) is at most twice the longest head of \( J \).

A schedule minimizing \( \text{cost}^*(J) \) is termed reduced-optimal, and the corresponding reduced cost is denoted by \( \text{cost}^* \). Note that for \( J^{(L)}, J^{(R)} \) and their subsets the calculation of \( \text{cost}^* \) is equivalent to solving a one-sided clique instance under the normal cost model and this can be done in polynomial time by Proposition 3.1.

We proceed with the performance analysis of Algorithm ALG1 below that chooses the maximal number \( j + k \) of jobs with shortest heads in \( J^{(L)} \) and \( J^{(R)} \) \((j \text{ jobs of } J^{(L,j)} \text{ and } k \text{ jobs of } J^{(R,k)} \text{, with total machine busy time at most } T/2) \). Then it schedules the jobs in each of these sets in a reduced-optimal manner. Consider an optimal schedule \( s' \) and the set \( J^* \) of jobs scheduled by it. We split this set into sets of left- and right-heavy jobs, namely \( Q^L = J^* \cap J^{(L)} \) and \( Q^R = J^* \cap J^{(R)} \). Then \( \text{put}^* = |J^*| = |Q^L| + |Q^R| \) and also \( T \geq \text{cost}^*(J^*) \geq \text{cost}^*(Q^L) + \text{cost}^*(Q^R) \).

The last equality holds because in the reduced cost model each one of the sets \( Q^L \) and \( Q^R \) incurs cost either before or after time \( t \), but not both. Let \( |Q^L| = 2 \cdot g \cdot q_L + r_L \) where \( q_L \) and \( r_L \) are the quotient and remainder in the division of \( |Q^L| \) by \( 2g \), i.e. \( q_L \geq 0, 0 \leq r_L < 2g \) and define \( q_R \) and \( r_R \) in a similar way.

**Claim 4.1:** The approximation ratio of ALG1 is at most \( 2 + \frac{4}{q_L + q_R} \).

**Proof:** Let \( Q^L = \{ J_0, J_1, \ldots \} \) 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 longest \( g \) jobs on one machine, the next \( g \) jobs on another machine and so on. Therefore \( \text{cost}^*(Q^L) \geq \sum_{i=0}^{2g-1} \text{len}(J_{i,g}) \). As the sequence is non-increasing the odd indexed elements of the sequence sum up to at most half of the sum, namely \( \sum_{j=0}^{2g-1} \text{len}(J_{2i+1,g}) \leq \text{cost}^*(Q^L)/2 \).

This sum is the reduced cost of the schedule \( s \) that schedules the \( g \) jobs \( J_{2i+1,g}, J_{2i+1,g+1}, \ldots, J_{2i+1,g+g-1} \) on machine \( i \) for \( 0 \leq i < q_L \). Thus \( \text{cost}^* \leq \text{cost}^*(Q^L)/2 \) and \( \text{put}^* = g \cdot q_L \). Now we observe that the reduced cost of a schedule \( s' \) that schedules the \( g \cdot q_L \) jobs with shortest heads in \( Q^L \) is at most \( \text{cost}^* \) because jobs are only replaced with jobs with shorter heads. The same holds for the schedule \( s'' \) that schedules \( J^{(L,g-q_L)} \) by a similar argument. Therefore \( \text{cost}^*(J^{(L,g-q_L)}) \leq \text{cost}^* \leq \text{cost}^*(Q^L)/2 \), or \( \text{cost}^*(J^{(L,g-q_L)}) \leq \text{cost}^*(Q^L) \). We schedule jobs from \( Q^R \) in a similar manner and get \( \text{cost}^*(J^{(R,g-q_R)}) \leq \text{cost}^*(Q^L) \). Therefore \( s'' \) is within budget, i.e. the total machine busy time it uses is at most \( T \). Observe that ALG1 considers this prefix pair in one of its iterations, thus it will return a schedule with throughput no less than the throughput of \( s'' \), i.e. \( (q_L + q_R)g \). Recalling that \( s'' \) schedules \( |Q^L| + |Q^R| = 2g(q_L + q_R)g + r_L + r_R g \) jobs.

**Algorithm 5 ALG1**

1. Among all the \( O \left( \frac{|J^L|}{|J^R|} \right) \)-possible prefix pairs \( J^{(L,j)}, J^{(R,k)} \)
2. Choose a pair with \( \text{cost}^*(J^{(L,j)}) + \text{cost}^*(J^{(R,k)}) \leq T/2 \)
3. Schedule the jobs of \( J^{(L,j)} \) in a reduced-optimal manner.
4. Schedule the jobs of \( J^{(R,k)} \) in a reduced-optimal manner.
the approximation ratio of ALG1 is
\[
\frac{2(q_L + q_R)g + r_L + r_R}{(q_L + q_R)g} = 2 + \frac{r_L + r_R}{(q_L + q_R)g} < 2 + \frac{4}{q_L + q_R}.
\]

If \( q_L = q_R = 0 \) then \( ptu^* \leq r_L + r_R \leq 4g - 2 \). By contraposition and using the above claim we conclude

**Lemma 4.1:** If \( ptu^* > 4g - 2 \) then ALG1 is a 6-approximation algorithm for clique instances of MAXTHROUGHPUT.

It remains to find a good approximation for the case \( ptu^* \leq 4g - 2 \). 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 \( Q \subseteq J \) of jobs if all the jobs in \( Q \) are contained in it. The coverage of \( I \) is the maximal subset \( Q \subseteq J \) that \( I \) covers. The coverage of some given interval \( I \) can be computed in time \( O(|J|) \) by considering each job and testing whether it is covered by \( I \). Now we observe that in a clique instance, for any subset \( Q \subseteq J \) of jobs, \( \text{SPAN}(Q) \) is determined by at most two jobs, each one determining one endpoint. In other words there exist two (not necessarily distinct) jobs \( J_i, J_j \in Q \) such that \( \text{SPAN}(\{J_i, J_j\}) = \text{SPAN}(Q) \). We conclude that the number of all possible distinct intervals \( \text{SPAN}(Q) \) is at most \( |J|^2 \).

**Algorithm 6 ALG2**

```plaintext
1: Try each possible pair \( J_i, J_j \) of jobs with \( \text{span}(\{J_i, J_j\}) \leq T \).
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 \( \text{SPAN}(\{J_i, J_j\}) \) to the same machine.
5: else
6: choose arbitrarily \( g \) jobs from the coverage of \( \text{SPAN}(\{J_i, J_j\}) \).
7: assign them to the same machine.
8: end if
```

**Lemma 4.2:** If \( ptu^* \leq 4g - 2 \) then ALG2 is a 4-approximation algorithm for clique instances of MAXTHROUGHPUT.

**Proof:** Consider the span \( \text{SPAN}(J^*) \) of all jobs scheduled by some optimal schedule \( s^* \). Then \( T \geq \text{cost}(s^*) \geq \text{cost}(J^*) \). ALG2 will consider this span in one of the iterations therefore the value of \( m \) will be at least equal to \( ptu^* \). If \( ptu^* \geq g \) then \( m \geq g \) and the algorithm will schedule \( g \) jobs. In this case the approximation ratio is at most \((4g - 2)/g < 4 \). If \( ptu^* < g \) then \( ptu^* \leq \min(m, g) \) 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 schedule we conclude by Lemmas 4.1 and 4.2:

**Theorem 4.1:** There is a 6-approximation algorithm for clique instances of MAXTHROUGHPUT.

**Algorithm 6 ALG2**

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

**Lemma 4.3:** For proper clique instances of MAXTHROUGHPUT there is an optimal (partial) schedule such that \( J_i \) is consecutive in \( J \) 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 \( J^* \subseteq J \) be the set of jobs scheduled by \( s^* \). For each machine \( M_i, J^*_i \) is consecutive in \( J^* \) by Lemma 3.3. It remains to prove that \( J^*_i \) is consecutive in \( J \). Assume, by way of contradiction that for some machine \( M_i, J_i \) is consecutive in \( J^* \) but not consecutive in \( J \). Then \( J^*_i = \{J_{i_1}, J_{i_2}, \ldots, J_{i_k}\} \), 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 feasible schedule and since the instance is a proper instance, \( J_x \) is entirely within the span of \( J^*_i \), thus the cost can only decrease. This process can be repeated until \( J^*_i \) is consecutive in \( J \).

By Lemma 4.3, each machine processes a set of consecutive jobs in \( J \), 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 \( \text{cost}(i, j, u, t) \) as the minimum cost of scheduling the instance consisting of the first \( i \) jobs of \( J \) 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 \( \text{cost}(n, j, u, t) \leq T \) and an optimal schedule is one that has the minimum value of \( t \). In other words, the maximum throughput is

\[
\max \{t \mid \text{cost}(n, j, u, t) \leq T \}.
\]

\( \text{cost}(i, j, u, t) \) can be calculated recursively as follows. For any \( 1 \leq i \leq n, 1 \leq j \leq \min(i, g), 0 \leq u \leq i - j, \) and \( u \leq t \leq i - j, \)

\[
\text{cost}(i, j, u, t) = \begin{cases} 
\text{cost}(i - 1, j, u - 1, t - 1) & \text{if } u > 0, \\
\text{cost}(i - 1, j - 1, u, t) + |P_t| - |I_{i-1}| & \text{if } u = 0 \text{ and } j > 1, \\
\min_{u'} \text{cost}(i - 1, j', u', t) + |P_t| & \text{if } u = 0 \text{ and } j = 1.
\end{cases}
\]

(4)
In the last case, the ranges of \( j' \) and \( u' \) are: 1 ≤ \( j' \) ≤ \( \min(g, i - 1 - t) \) and 1 ≤ \( u' \) ≤ \( \min(i - 1 - j', u) \).

When \( u > 0 \), the schedule with \( \text{cost}(i, j, u, t) \) 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 \( \text{cost}(i - 1, j, u - 1, t - 1) \). 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 \( j - 1 \) 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 \( \text{cost}(i - 1, j - 1, u, t) + |J_i| - |I_{i-1}| \). 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 4. The 4-dimensional table can be filled by the dynamic programming algorithm MOSTTHROUGHPUTCONSECUTIVE.

**Algorithm 7 MOSTTHROUGHPUTCONSECUTIVE**

\[
\begin{align*}
1: & \text{cost}(1, 1, 0, 0) \leftarrow |T| \\
2: & \text{for } i = 2 \text{ to } n \text{ do} \\
3: & \hspace{1em} \text{for } j = 1 \text{ to } \min(i, g) \text{ do} \\
4: & \hspace{2em} \text{for } u = 0 \text{ to } i - j \text{ do} \\
5: & \hspace{3em} \text{if } u > 0 \text{ then} \\
6: & \hspace{4em} \text{cost}(i, j, u, t) = \text{cost}(i - 1, j - 1, u - 1, t - 1) \\
7: & \hspace{1em} \text{else} \\
8: & \hspace{2em} \text{if } j \geq 1 \text{ then} \\
9: & \hspace{3em} \text{cost}(i, j, u, t) = \text{cost}(i - 1, j - 1, u - 1, t - 1) + |P_i| - |P_{i-1}| \\
10: & \hspace{1em} \text{else} \\
11: & \hspace{2em} \text{cost}(i, j, u, t) = \min_{1 \leq j' \leq \min(g, i - 1 - t)} \min_{1 \leq u' \leq \min(i - 1 - j', u)} \text{cost}(i - 1, j', u', t) + |P_i| \\
12: & \text{end if} \\
13: & \text{end if} \\
14: & \text{end for} \\
15: & \text{end for} \\
16: & \text{end for} \\
17: & \text{end for}
\end{align*}
\]

The 4-dimensional table contains \( n^3 g \) entries. Among these entries, 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 \( O(|J|^3 \cdot g) \).

V. SUMMARY, EXTENSIONS AND FUTURE WORK

In this paper we revisit the problem MINBUSY and initiate the study of the problem MAXTHROUGHPUT in optimization of busy times. The full list of results is given in Section I. In particular, three of our results on MINBUSY directly improve upon existing results as follows.

- A polynomial-time algorithm for clique instances when \( g = 2 \).

- \( \frac{g - H_g}{H_g} \)-approximation algorithm for clique instances, where \( H_g \) is the \( g \)-th harmonic number, for small values of \( g \).

- A \((2 - 1/g)\)-approximation algorithm for proper instances.

The following open problems are of interest:

- **MINBUSY:** The exact complexity of MINBUSY for clique instances and for proper instances is still open. In this work approximation algorithms were presented.

- **MAXTHROUGHPUT:** We have shown that this problem is NP-Hard whenever MINBUSY is NP-Hard. We note that the problem is NP-Hard in the weak sense even for proper instances, by a simple reduction from the KNAPSACK problem (\( T \) being the size of the knapsack). The question of whether it is strictly harder than MINBUSY generally or in a special case is open.

As we have mentioned, 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 we mentioned in Section I, 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], [21], [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 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], [6]. 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., [9], [22], [25]) as presented in Section I.

- We measure the throughput as the number of jobs satisfied. In general, we can consider jobs having an associated benefit and maximize the total benefit obtained from the scheduled jobs.

- 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 required is at most \( g \) [16].

- 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 \( k \).
• In this work the jobs are supposed to be processed during the whole period from start time $s_j$ 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 $p_j$ time units during the interval $[s_j, c_j]$ (see e.g. [24]).
  - 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., [20], [24]).

Optical network design: As detailed in Section I, 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 Proposition 3.1 for one-sided clique instances can be extended to tree topologies.
- Theorem 3.3 can be extended to ring and tree topologies.

The first two above-mentioned extensions in the context of cloud computing are of importance also in the context of optical networks; namely, the case where each job has a benefit associated with it, and the case where each job has its own capacity demand. The first case corresponds to the extension of the regenerator placement problem to the case where there is a benefit associated with each lightpath to be established, and in the maximization problem the objective is to maximize this benefit. The second case corresponds to the extension of the regenerator placement problem to the case where each lightpath has a capacity (say, a multiple of $1/g$), so that up to a total of $g$ such capacity units can be groomed together on any particular communication line, and the objective is to find optimal solution for the MINBUSY and MAXTHROUGHPUT problems.

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

REFERENCES