Real-time Scheduling to Minimize Machine Busy Times[*]

Rohit Khandekar[†] Baruch Schieber [‡] Hadas Shachnai[§] Tami Tamir[¶]

Abstract

Consider the following scheduling problem. We are given a set of jobs, each having a release time, a due date, a processing time and demand for machine capacity. Assuming that each machine can process multiple jobs simultaneously, the goal is to schedule all jobs non-preemptively in their release-time-deadline windows, subject to machine capacity constraints, such that the total busy time of the machines is minimized. Our problem naturally arises in power-aware scheduling, optical network design and customer service systems, among others. The problem is APX-hard by a simple reduction from the subset sum problem.

A main result of this paper is a 5-approximation algorithm for general instances. While the algorithm is simple, its analysis involves a non-trivial charging scheme which bounds the total busy-time in terms of *work* and *span* lower bounds on the optimum. This improves and extends the results of Flammini et al. (*Theoretical Computer Science, 2010*). We extend this approximation to the case of *moldable* jobs, where the algorithm also needs to choose, for each job, one of several processing-time vs. demand configurations. Better bounds and exact algorithms are derived for several special cases, including proper interval graphs, intervals forming a clique and laminar families of intervals.

1 Introduction

Traditional research interest in cluster systems has been high performance, such as high throughput, low response time, or load balancing. In this paper we focus on minimizing machine busy times, a recent trend in cluster computing which aims at *reducing power consumption* (see, e.g., [24] and the references therein).

We are given a set of n jobs $\bar{\mathcal{J}} = \{J_1, \dots, J_n\}$ that need to be scheduled on a set of identical machines, each having *capacity* g, for some $g \geq 1$. Each job J has a release time $r(J)$, a due date $d(J)$, a processing time (or, length) $p(J) > 0$ (such that $d(J) \geq r(J) + p(J)$) and a *demand* (or, width) $0 < R(J) \leq g$ for machine capacity; this is the amount of capacity required for processing J on any machine (the demands need not be integral).

[*]A preliminary version of this paper appeared in the Proceedings of the 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Chennai, December 2010.

[†]Knight Capital Group, Jersey City, NJ 07310. E-mail: `rkhandekar@gmail.com`

[‡]IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. E-mail: `sbar@us.ibm.com`

[§]Computer Science Department, Technion, Haifa 32000, Israel. E-mail: `hadas@cs.technion.ac.il`. Work partially supported by funding for DIMACS visitors.

[¶]School of Computer science, The Interdisciplinary Center, Herzliya, Israel. E-mail: `tami@idc.ac.il`

A feasible solution schedules each job J on a machine M *non-preemptively* during a time interval $[t(J), t(J) + p(J))$, such that $t(J) \geq r(J)$ and $t(J) + p(J) \leq d(J)$, and the total demand of jobs running at any given time on each machine is at most g. We say that a machine M is *busy* at time t if there is at least one job J scheduled on M such that $t \in [t(J), t(J) + p(J))$; otherwise, M is *idle* at time t. We call the time period in which a machine M is busy its *busy period* and denote its length by $\texttt{busy}(M)$. The goal is to find a feasible schedule of all jobs on a set of machines such that the total busy time of the machines, given by $\sum_M \texttt{busy}(M)$, is minimized. We consider the offline version of this problem, where the entire input is given in advance.

Note that the number of machines to be used is part of the output (and can take any integral value $m \geq 1$). Indeed, a solution which minimizes the total busy time may not be optimal in the number of machines used. It is NP-hard to approximate the problem within an approximation ratio better than $\frac{3}{2}$ by the following simple reduction from the Partition problem. Given integers $a_1, \ldots, a_n \in \{1, \ldots, g\}$ summing to $2g$, the Partition problem is to determine if it is possible to partition the numbers into two sets, each adding to exactly g. In the reduction, we define for each i, a job J_i with demand a_i, release time 0, processing time 1 and deadline 1. The *yes* instance of Partition results in a busy time of 2, while the *no* instance results in the busy time 3.

1.1 Applications

We list below several natural applications of our problem.

Power-aware scheduling The objective of power-aware scheduling is to minimize the power consumption for running a cluster of machines, while supporting Service Level Agreements (SLAs). SLAs, which define the negotiated agreements between service providers and consumers, include quality of service parameters such as demand for a computing resource and a deadline. The power consumption of a machine is assumed to be proportional to the time the machine is in *on* state. While *on*, a machine can process several tasks simultaneously. The number of these tasks hardly affects the power consumption, but must be below the given machine's capacity. Thus, we get an instance of our problem of minimizing the total busy time of the schedule.

Optical network design Hardware costs in optical networks depend on the usage of switching units such as Optical Add-Drop Multiplexers (or OADMs). Communication is done by *lightpaths*, which are simple paths in the network. A lightpath connecting nodes u and v is using one OADM in each intermediate node. Often, the traffic supported by the network requires transmission rates which are lower than the full wavelength capacity; thus, the network operator has to be able to put together (or, groom) low-capacity demands into the high capacity fibers. Taking g to be the *grooming* factor, for some $g \geq 1$, this can be viewed as grouping the lightpaths with the same wavelength so that at most g of them can share one edge. In terms of OADMs, if g lightpaths of the same wavelength ("color") pass through the same node, they can share a single OADM.

Given a network with a line topology, a grooming factor $g \geq 1$ and a set of lightpaths $\{p_j = (a_j, b_j) | j = 1, ..., n\}$, we need to color the lightpaths such that the number of OADMs is minimized. This yields the following instance of our scheduling problem. The time axis corresponds to the line network. There are n unit demand jobs (corresponding to lightpaths), and the machine capacity is g. For $j = 1, \dots, n$, job J_j needs to be executed in the time interval $[a_j + 1/2, b_j - 1/2]$ (i.e., $p(J_j) = b_j - a_j - 1$, $r(J_j) = a_j + 1/2$ and $d(J_j) = b_j - 1/2$). Grooming up to g lightpaths in the optical network implies that the corresponding jobs are scheduled on the same machine, and vice versa. In other words, different colors in the optical network instance correspond to different machines in the scheduling problem instance, and an OADM at node i corresponds to the interval $[i-1/2, i+1/2]$. Thus, the cost of a coloring of the lightpaths is equal to the cost of the schedule of the corresponding set of jobs. This *grooming problem* has become central in optimizing switching costs for optical networks [12, 11, 10].

Unit commitment given future demand The Unit commitment in power systems involves determining the start-up and shut-down schedule of generation units to meet the required demand. This is one of the major problems in power generation (see, e.g., [26, 2] and the references therein). Under-commitment of units would result in extra cost due to the need to purchase the missing power in the spot market, while overcommitment would result in extra operating cost. In a simplified version of the problem, assume that all generation units have the same capacity and that the blocks of future demands are given in advance. This yields an instance of our real-time scheduling problem, where each generation unit corresponds to a machine, and each block of demand corresponds to a job.

1.2 Related Work

Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [8, 4]). In particular, much attention was given to *interval scheduling* [16], where jobs are given as intervals on the real line, each representing the time interval in 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 time. 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., [1] and the survey in [15]).

There is a wide literature also 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. Also, there are studies of real-time scheduling with demands, where each machine has some capacity; however, to the best of our knowledge, all of this prior art refers to objectives other than minimizing the total busy time of the schedule (see, e.g., [1, 20, 5, 7]). There has been earlier work also on the problem of scheduling the jobs on a set of machines so as to minimize the total cost (see, e.g., [3]), but in these works the cost of scheduling each job is *fixed.* In our problem, the cost of scheduling each of the jobs depends on the other jobs scheduled on the same machine in the corresponding time interval; thus, it may change over time and among different machines. Scheduling *moldable* jobs, where each job can have varying processing times, depending on the amount of resources allotted to this job, has been studied using classic measures, such as minimum makespan, or minimum (weighted) sum of completion times (see, e.g., [23, 17] and a comprehensive survey in [22]).

Our study relates also to *batch scheduling* of conflicting jobs, where the conflicts are given as an interval graph. In the *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, 21].) Our scheduling problem differs from batch scheduling in several aspects. First, as for real-time scheduling, 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. In addition, while job demands imply that each machine can process (at most) 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. Finally, 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. Other work on energy minimization consider utilization of machines with variable capacities, corresponding to their voltage consumption [18], and scheduling of jobs with precedence constraints [13, 27].

The complexity of our scheduling problem was studied in [25]. This paper shows that the problem is NP-hard already for $g = 2$, where the jobs are intervals on the line. The paper [10] considers our scheduling problem where jobs are given as intervals on the line with unit demand. For this version of the problem the paper gives a 4-approximation algorithm for general inputs and better bounds for some subclasses of inputs. In particular, the paper [10] presents a 2-approximation algorithm for instances where no interval is properly contained in another (i.e., the input forms a *proper* interval graph), and a $(2 + \varepsilon)$-approximation for bounded lengths instances, i.e., the length (or, processing time) of any job is bounded by some fixed integer $d \geq 1$.[1] A 2-approximation algorithm was given in [10] for instances where any two intervals intersect, i.e., the input forms a clique (see also [11]). In this paper we improve and extend the results of [10].

1.3 Our Results

Our main result is a 5-approximation algorithm for real-time scheduling of *moldable* jobs. Before summarizing our results, we introduce some notation. Denote by $I(J)$ the interval $[r(J), d(J))$ in which J can be processed.

Definition 1.1 *An instance $\bar{\mathcal{J}}$ is said to have* interval jobs *if $d(J) = r(J) + p(J)$ holds for all jobs $J \in \bar{\mathcal{J}}$. An instance $\bar{\mathcal{J}}$ with interval jobs is called*

- proper *if for any two jobs $J, J' \in \bar{\mathcal{J}}$, neither $I(J) \subseteq I(J')$ nor $I(J') \subseteq I(J)$ holds.*
- laminar *if the intervals $I(J)$ for all jobs J form a laminar family, i.e., for any two jobs $J, J' \in \bar{\mathcal{J}}$, we have $I(J) \cap I(J') = \emptyset$ or $I(J) \subset I(J')$, or $I(J') \subset I(J)$.*
- *a* clique *if intervals $I(J)$ for all jobs J form a clique, i.e., for any two jobs $J, J' \in \bar{\mathcal{J}}$, we have $I(J) \cap I(J') \neq \emptyset$.*

We first prove the following result for instances with interval jobs.

Theorem 1.1 *There exists a 5-approximation algorithm for real-time scheduling instances with interval jobs. Furthermore, if the instance is proper, there exists a 2-approximation algorithm.*

We use the above algorithm, as a subroutine, to design our algorithm for the general real-time scheduling problem.

[1]A slight modification of the algorithm yields an improved bound of $1 + \varepsilon$, where $\varepsilon > 0$ is an input parameter.

Theorem 1.2 *There exists a* 5*-approximation algorithm for the real-time scheduling problem.*

Next, we consider an extension to real-time scheduling of *moldable* jobs. In this generalization, a job does not have a fixed processing time and demand; rather, it can be scheduled in one of several possible *configurations*. More precisely, for each job $J \in \bar{\mathcal{J}}$, we have $q \geq 1$ configurations, where configuration i is given by a pair $(p_i(J), R_i(J))$. The problem involves deciding which configuration i_J is to be used in the schedule for each job J. Once a configuration i_J is finalized for a job J, its processing time and demand are given by $p_{i_J}(J)$ and $R_{i_J}(J)$, respectively. We extend our real-time scheduling algorithm to yield the same bound for moldable instances.

Theorem 1.3 *There exists a* 5*-approximation algorithm for real-time scheduling of* moldable *jobs.*

We note that even with small fractional demands, the time complexity of our algorithms is polynomial in n, since at most n jobs can be processed at any given time. Finally, we present improved bounds for some cases of instances with interval jobs.

Theorem 1.4 *Consider an instance $\bar{\mathcal{J}}$ consisting of interval jobs with* unit *demands. There exist* (*i*) *a polynomial time exact algorithm if $\bar{\mathcal{J}}$ is laminar, and* (*ii*) *a PTAS if $\bar{\mathcal{J}}$ is a clique.*

Recent Developments: Following our results in [14], Mertzios at al. [19] considered some subclasses of instances of our problem with interval jobs. They gave a polynomial-time algorithm for proper clique instances and also improved our approximation ratio for proper instances to $2 - 1/g$. Recently, Chang et al. [6] obtained improved results for the unit demand real-time scheduling problem. Similar to our approach in Section 3, they first solve the problem assuming unbounded machine capacity to have a solution that minimizes the projection of the jobs onto the time-axis. Then, the original instance is mapped to one of interval jobs, forcing each job to be processed as in the unbounded capacity solution. This approach is shown in [6] to yield a total busy time that is at most 3 times the optimal.

Organization: The rest of the paper is organized as follows. In Section 1.4, we give some preliminary definitions and observations. Theorems 1.1, 1.2, 1.3, and 1.4 are proved in Sections 2, 3, 4 and 5, respectively.

1.4 Preliminaries

Throughout the paper, we use properties of the interval representation of a given instance.

Definition 1.2 *Given a time interval $I = [s, t)$, the length of I is $\texttt{len}(I) = t - s$. This extends to a set $\mathcal{I}$ of intervals; namely, the length of $\mathcal{I}$ is $\texttt{len}(\mathcal{I}) = \sum_{I \in \mathcal{I}} \texttt{len}(I)$. We define the span of $\mathcal{I}$ as $\texttt{span}(\mathcal{I}) = \texttt{len}(\cup \mathcal{I})$.*

Note that $\texttt{span}(\mathcal{I}) \leq \texttt{len}(\mathcal{I})$ and equality holds if and only if $\mathcal{I}$ is a set of pairwise disjoint intervals.

Given an instance $\bar{\mathcal{J}}$ and machine capacity $g \geq 1$, we denote by $\text{OPT}(\bar{\mathcal{J}})$ the cost of an optimal solution, that is, a feasible schedule in which the total busy time of the machines is minimized. Also, we denote by $\text{OPT}_\infty(\bar{\mathcal{J}})$ the cost of the optimum solution for the instance $\bar{\mathcal{J}}$, assuming that the capacity is $g = \infty$. For any job J, let $w(J) = R(J) \cdot p(J)$ denote the total work required by job

J, then for a set of jobs $\bar{\mathcal{J}}$, $w(\bar{\mathcal{J}}) = \sum_{J \in \bar{\mathcal{J}}} w(J)$ is the total work required by the jobs in $\bar{\mathcal{J}}$. The next observation gives two immediate lower bounds for the cost of any solution.

Observation 1.5 *For any instance $\bar{\mathcal{J}}$ and machine capacity $g \geq 1$, the following bounds hold.*

- *The work bound:* $\text{OPT}(\bar{\mathcal{J}}) \geq \frac{w(\bar{\mathcal{J}})}{g}$.
- *The span bound:* $\text{OPT}(\bar{\mathcal{J}}) \geq \text{OPT}_\infty(\bar{\mathcal{J}})$.

The work bound holds since g is the maximum capacity that can be allocated by a single machine at any time. The span bound holds since the busy-time does not increase by relaxing the capacity constraint.

While analyzing any schedule S that is clear from the context, we number the machines as $M_1, M_2, \ldots$, and denote by $\bar{\mathcal{J}}_i$ the set of jobs assigned to machine M_i under the schedule S. W.l.o.g., the busy period of a machine M_i is contiguous; otherwise, we can divide the busy period to contiguous intervals and assign the jobs of each contiguous interval to a different machine. Obviously, this will not change the total busy time. Therefore, we say that a machine M_i has a *busy interval* which starts at the minimum start time of any job scheduled on M_i and ends at the maximum completion time of any of these jobs. It follows that the cost of M_i is the length of its busy interval, i.e., $\texttt{busy}(M_i) = \texttt{span}(\bar{\mathcal{J}}_i)$ for all $i \geq 1$.

2 Interval Scheduling: Theorem 1.1

2.1 General Instances with Interval Jobs

In this section we present an algorithm for instances with interval jobs (i.e., $d(J) = r(J) + p(J)$ for all $J \in \bar{\mathcal{J}}$), where each job J may have an arbitrary processing time and any demand $1 \leq R(J) \leq g$. Algorithm *First_Fit_with_Demands* ($FF_{\mathcal{D}}$) divides the jobs into two groups, NARROW and WIDE, as given below. It schedules NARROW and WIDE jobs on distinct sets of machines. The WIDE jobs are scheduled arbitrarily, while NARROW jobs are scheduled greedily by considering them one after the other, from longest to shortest. Each job is scheduled on the first machine it can fit. Let $\alpha \in [0, 1]$ be a parameter to be fixed later.

Definition 2.1 *For a subset $\bar{\mathcal{J}}' \subseteq \bar{\mathcal{J}}$ of jobs, let* $\text{NARROW}(\bar{\mathcal{J}}') = \{J \in \bar{\mathcal{J}}' \mid R(J) \leq \alpha \cdot g\}$ *and* $\text{WIDE}(\bar{\mathcal{J}}') = \{J \in \bar{\mathcal{J}}' \mid R(J) > \alpha \cdot g\}$.

```
Algorithm ($FF_{\mathcal{D}}$):
  (i) Schedule jobs in $\textsc{wide}(\bar{\mathcal{J}})$ arbitrarily on some machines. Do not use these machines for
      scheduling any other jobs.
  (ii) Sort the jobs in $\textsc{narrow}(\bar{\mathcal{J}})$ in non-increasing order of length, i.e., $p(J_1) \geq \ldots \geq p(J_{n'})$.
  (iii) For $j = 1, \ldots, n'$ do:
      (a) Let $m$ denote the number of machines used for the jobs $\{J_1, \ldots, J_{j-1}\}$.
      (b) Assign $J_j$ to the first machine that can process it, i.e., find the minimum value of
          $i : 1 \leq i \leq m$ such that, at any time $t \in I(J_j)$, the total capacity allocated on $M_i$ is
          at most $g - R(J_j)$.
      (c) If no such machine exists open $(m+1)$th machine and schedule $J_j$ on it.
```

Theorem 2.1 *If $\alpha = 1/4$ then, for any instance $\bar{\mathcal{J}}$ with interval jobs, we have*

$$FF_{\mathcal{D}}(\bar{\mathcal{J}}) \leq \textsc{opt}_{\infty}(\bar{\mathcal{J}}) + 4 \cdot \frac{w(\bar{\mathcal{J}})}{g} \leq 5 \cdot \textsc{opt}(\bar{\mathcal{J}}).$$

To prove the theorem we bound the costs of the WIDE and NARROW jobs. It is easy to bound the contribution of WIDE jobs to the overall cost. The next result follows directly from the definition of WIDE jobs.

Observation 2.2 *The cost incurred by jobs in* $\textsc{wide}(\bar{\mathcal{J}})$ *is at most* $\sum_{J \in \textsc{wide}(\bar{\mathcal{J}})} p(J) \leq \frac{w(\textsc{wide}(\bar{\mathcal{J}}))}{\alpha \cdot g}$.

The rest of the section is devoted to bounding the cost of the NARROW jobs. The next observation follows from the fact that the *first-fit* algorithm $FF_{\mathcal{D}}$ assigns a job J to machine M_i with $i \geq 2$ only when it could not have been assigned to machines M_k with $k < i$, due to capacity constraints.

Observation 2.3 *Let J be a job assigned to machine M_i by $FF_{\mathcal{D}}$, for some $i \geq 2$. For any machine $M_k, (k < i)$, there is at least one time $t_{i,k}(J) \in I(J)$ and a set $S_{i,k}(J)$ of jobs assigned to M_k such that, for every $J' \in S_{i,k}(J)$, (a) $t_{i,k}(J) \in I(J')$, and (b) $p(J') \geq p(J)$. In addition, $R(J) + \sum_{J' \in S_{i,k}(J)} R(J') > g$.*

Proof: Assume by contradiction that for any $t \in J$ the total capacity allocated to M_k is at most $g - R(J)$. Since the algorithm assigns jobs to machines incrementally (i.e., it never unassigns jobs), this property was true also when $FF_{\mathcal{D}}$ scheduled job J. Thus, J should have been assigned to M_k by $FF_{\mathcal{D}}$, if it was not already assigned to a machine with smaller index. This is a contradiction to the fact that J is assigned to M_i. Hence, there exists a time $t_{i,k}(J) \in I(J)$, such that the set $S_{i,k}(J)$ of jobs containing $t_{i,k}(J)$, among those assigned to machine M_k, satisfies $R(J) + \sum_{J' \in S_{i,k}(J)} R(J') > g$. Property (b) follows since the jobs are considered by the algorithm in a non-increasing order of their length. ∎

In the subsequent analysis, we assume that each job $J \in \bar{\mathcal{J}}_i$, for $i \geq 2$, fixes a unique time $t_{i,i-1}(J)$ and a unique set of jobs $S_{i,i-1}(J) \subseteq \bar{\mathcal{J}}_{i-1}$. We say that J *blames* jobs in $S_{i,i-1}(J)$.

Lemma 2.4 *For any $1 \leq i \leq m-1$, we have* $\texttt{span}(\bar{\mathcal{J}}_{i+1}) \leq \frac{3 \cdot w(\bar{\mathcal{J}}_i)}{(1-\alpha) \cdot g}$.

Proof: Following Observation 2.3, for a job $J \in \bar{\mathcal{J}}_i$, denote by $b(J)$ the set of jobs in $\bar{\mathcal{J}}_{i+1}$ which blame J, i.e., $b(J) = \{J' \in \bar{\mathcal{J}}_{i+1} \mid J \in S_{i+1,i}(J')\}$. Let J_L (resp. J_R) be the job with earliest start time (resp. latest completion time) in $b(J)$. Since each job in $b(J)$ overlaps J, we have $\mathtt{span}(b(J)) \leq p(J) + p(J_L) + p(J_R) \leq 3 \cdot p(J) = 3 \cdot \frac{w(J)}{R(J)}$. Thus,

$$\sum_{J \in \bar{\mathcal{J}}_i} R(J)\mathtt{span}(b(J)) \leq 3 \cdot w(\bar{\mathcal{J}}_i). \tag{1}$$

Now, we observe that

$$\sum_{J \in \bar{\mathcal{J}}_i} R(J)\mathtt{span}(b(J)) = \int_{t \in \mathtt{span}(\bar{\mathcal{J}}_{i+1})} \sum_{J \in \bar{\mathcal{J}}_i : t \in \mathtt{span}(b(J))} R(J) dt.$$

We bound the right-hand-side as follows. For any $t \in \mathtt{span}(\bar{\mathcal{J}}_{i+1})$, there exists a job $J' \in \bar{\mathcal{J}}_{i+1}$ with $t \in [r(J'), d(J'))$. For all jobs $J \in S_{i+1,i}(J')$, since $J' \in b(J)$, we have $t \in \mathtt{span}(b(J))$. Hence, $\sum_{J \in \bar{\mathcal{J}}_i : t \in \mathtt{span}(b(J))} R(J) \geq \sum_{J \in S_{i+1,i}(J')} R(J)$. Also, $\sum_{J \in S_{i+1,i}(J')} R(J) > g - R(J') \geq (1-\alpha) \cdot g$ follows by Observation 2.3. We thus conclude

$$\sum_{J \in \bar{\mathcal{J}}_i} R(J)\mathtt{span}(b(J)) > \mathtt{span}(\bar{\mathcal{J}}_{i+1}) \cdot (1-\alpha) \cdot g. \tag{2}$$

From (1) and (2) we get the lemma. ■

We show that if $\alpha = 1/4$, the $FF_{\mathcal{D}}$ is a 5-approximation algorithm. The following is an overview of the analysis. The proof combines the span bound and the work bound given in Observation 1.5 with the following analysis. For WIDE jobs we use the work bound. Let m denote the total number of machines used for NARROW jobs, let M_i be the ith such machine in the order considered by $FF_{\mathcal{D}}$, and $\bar{\mathcal{J}}_i$ be the set of NARROW jobs scheduled on M_i. Using Observation 2.3, we relate the cost incurred for the jobs in $\bar{\mathcal{J}}_{i+1}$ to $\mathtt{span}(\bar{\mathcal{J}}_i)$. This relates the overall cost for the jobs in $\bar{\mathcal{J}} \setminus \bar{\mathcal{J}}_1$ to $\text{OPT}(\bar{\mathcal{J}})$, using the work bound. Then we relate the cost incurred for the jobs in $\bar{\mathcal{J}}_1$ to $\text{OPT}(\bar{\mathcal{J}})$, using the span bound.

Proof of Theorem 2.1: The overall cost of the schedule computed by $FF_{\mathcal{D}}$ is the contribution of WIDE jobs and NARROW jobs. Recall that m is the number of machines used for narrow jobs in the current iteration. Then the busy time of M_i, for $1 \leq i \leq m$, is exactly $\mathtt{busy}(M_i) = \mathtt{span}(\bar{\mathcal{J}}_i)$. Now, from Observation 2.2 and Lemma 2.4, we have that the total cost of $FF_{\mathcal{D}}$ is at most

$$\begin{aligned}
\frac{w(\text{WIDE}(\bar{\mathcal{J}}))}{\alpha \cdot g} + \sum_{i=1}^{m} \mathtt{span}(\bar{\mathcal{J}}_i) &\leq \frac{w(\text{WIDE}(\bar{\mathcal{J}}))}{\alpha \cdot g} + \mathtt{span}(\bar{\mathcal{J}}_1) + \sum_{i=1}^{m-1} \frac{3 \cdot w(\bar{\mathcal{J}}_i)}{(1-\alpha) \cdot g} \\
&\leq \frac{w(\text{WIDE}(\bar{\mathcal{J}}))}{\alpha \cdot g} + \text{OPT}_\infty(\bar{\mathcal{J}}) + \frac{3 \cdot w(\text{NARROW}(\bar{\mathcal{J}}))}{(1-\alpha) \cdot g} \\
&\leq \text{OPT}_\infty(\bar{\mathcal{J}}) + \max\left\{\frac{1}{\alpha}, \frac{3}{1-\alpha}\right\} \cdot \frac{w(\bar{\mathcal{J}})}{g} \\
&\leq \text{OPT}_\infty(\bar{\mathcal{J}}) + 4 \cdot \frac{w(\bar{\mathcal{J}})}{g}.
\end{aligned}$$

The second inequality follows from the fact that $\mathtt{span}(\bar{\mathcal{J}}_i) \leq \text{OPT}_\infty(\bar{\mathcal{J}})$, for all $i \geq 1$. The last inequality holds since $\alpha = 1/4$. The proof now follows from Observation 1.5. ■

2.2 Proper Instances with Interval Jobs

In this section we consider instances in which no job interval is contained in another. The intersection graphs for such instances are known as *proper interval graphs*. The simple greedy algorithm consists of two steps. In the first step, the jobs are sorted by their starting times (note that, in a proper interval graph, this is also the order of the jobs by completion times). In the second step the jobs are assigned to machines greedily in a NextFit manner; that is, each job is added to the currently filled machine, unless its addition is invalid, in which case a new machine is opened.

```
Greedy Algorithm for Proper Interval Graphs
  (i) Sort the jobs in non-decreasing order of release times, i.e., $r(J_1) \leq \ldots \leq r(J_n)$.
  (ii) For $j = 1, \ldots, n$ do: Assign $J_j$ to the currently filled machine if this satisfies the capacity
       constraint $g$; otherwise, assign $J_j$ to a new machine and mark it as being currently filled.
```

Theorem 2.5 *Greedy is a 2-approximation algorithm for proper interval graphs.*

Proof: Let D_t be the total demand of jobs active at time t. Also, let M_t^O denote the number of machines active at time t in an optimal schedule, and let M_t^A be the number of machines active at time t in the schedule output by the algorithm.

Claim 2.6 *For any t, we have $D_t > g \left\lfloor \frac{M_t^A - 1}{2} \right\rfloor$.*

Proof: If $M_t^A \leq 2$ then the claim trivially holds. For a given $t > 0$, let $m = M_t^A \geq 3$. The first machine processes at least one job, J, with a positive demand at time t. Similarly, the last machine processes at least one job, J', with a positive demand at time t. Since the graph is a proper interval graph, any job J'' assigned to machines $2, \ldots, m-2$ starts after J and before J', and ends after J and before J', thus, J'' is active at time t. Also, all these machines were active at the time the m-th machine is opened, and thus the total demand of jobs on any two consequent machines starting from the second machine and ending at machine $m-1$ is more than g. If $m-2$ is even, $D_t > g\frac{m-2}{2} = g \left\lfloor \frac{m-1}{2} \right\rfloor$. If $m-2$ is odd, the total demand on the $(m-1)$-th machine and the job causing the opening of the m-th machine is more than g and $D_t > g\frac{m-3}{2} + g = g\frac{m-1}{2}$. ■

Claim 2.7 *For any t, we have $M_t^O \geq M_t^A/2$.*

Proof: Clearly, for any $t \geq 0$, $M_t^O \geq \lceil D_t/g \rceil$. Using Claim 2.6, we get that

$$M_t^O \geq \lceil D_t/g \rceil > \left\lfloor \frac{M_t^A - 1}{2} \right\rfloor .$$

Since M_t^O is integral, we get $M_t^O \geq \lfloor (M_t^A - 1)/2 \rfloor + 1 = \lfloor (M_t^A + 1)/2 \rfloor \geq M_t^A/2$. ■

Therefore, the cost of the output solution is $\int_{t \in \mathtt{span}(\bar{\mathcal{J}})} M_t^A dt \leq \int_{t \in \mathtt{span}(\bar{\mathcal{J}})} 2 \cdot M_t^O dt = 2 \cdot \text{OPT}(\bar{\mathcal{J}})$, as claimed. ■

3 Real-time Scheduling: Theorem 1.2

In this section we show how the results of §2 can extended to scheduling general instances $\bar{\mathcal{J}}$ where each job J can be processed in the time window $[r(J), d(J))$.

Lemma 3.1 *If there exists a β-approximation algorithm for the real-time scheduling with $g = \infty$, there exists an algorithm that computes a feasible solution to the real-time scheduling problem instance, $\bar{\mathcal{J}}$, with cost at most $\beta \cdot \text{OPT}_\infty(\bar{\mathcal{J}}) + 4 \cdot \frac{w(\bar{\mathcal{J}})}{g}$, thus yielding a $(\beta + 4)$-approximation.*

Proof: We first compute a schedule, called S_∞, with busy-time at most $\beta \cdot \text{OPT}_\infty(\bar{\mathcal{J}})$, for the given instance with $g = \infty$. Let $[t_\infty(J), t_\infty(J) + p(J)) \subseteq [r(J), d(J))$ be the interval during which job J is scheduled in S_∞. We next create a new instance $\bar{\mathcal{J}}'$ obtained from $\bar{\mathcal{J}}$ by replacing $r(J)$ and $d(J)$ with $t_\infty(J)$ and $t_\infty(J) + p(J)$, respectively, for each job J. Note that $\text{OPT}_\infty(\bar{\mathcal{J}}') \leq \beta \cdot \text{OPT}_\infty(\bar{\mathcal{J}}) \leq \beta \cdot \text{OPT}(\bar{\mathcal{J}})$. We then run algorithm $FF_{\mathcal{D}}$ on instance $\bar{\mathcal{J}}'$. Theorem 2.1 implies that the resulting solution has busy-time at most $\text{OPT}_\infty(\bar{\mathcal{J}}') + 4 \cdot \frac{w(\bar{\mathcal{J}}')}{g} \leq \beta \cdot \text{OPT}_\infty(\bar{\mathcal{J}}) + 4 \cdot \frac{w(\bar{\mathcal{J}})}{g} \leq (\beta + 4) \cdot \text{OPT}(\bar{\mathcal{J}})$ as claimed. ■

The following theorem with the above lemma implies a 5-approximation algorithm for the real-time scheduling.

Theorem 3.2 *If $g = \infty$, the real-time scheduling problem is polynomially solvable.*

The rest of this section is devoted to proving the above theorem. To describe our *dynamic programming (DP)* based algorithm, we first identify some useful properties of the optimum schedule. Recall that we can assume, w.l.o.g., that the busy period of each machine is a contiguous interval. Next, we show that we can limit the possible start time of the jobs in the optimal schedule and thus in turn limit the possible start and end times of the busy periods.

Lemma 3.3 *W.l.o.g., we may assume that the busy period of any machine in the optimum schedule starts at a time given by $d(J) - p(J)$ for some job J and ends at a time given by either $r(J') + p(J')$, for some job J', or $d(J) - p(J) + p(J')$ for some jobs J and J'. Furthermore, we can assume that the start time of any job J is either its release time $r(J)$ or the start time of the busy period of some machine.*

Proof: We start with an optimum schedule S and modify it without increasing its busy-time so that it satisfies the given properties. Consider a machine M with a busy period starting at time s, and ending at time t. We gradually increase s, if possible, by moving jobs that start at s to the right without violating any constraints (i.e., job due dates). Note that while such a move may increase t, it does not increase the busy-time of machine M. We perform the move till we cannot increase s further. Thus, we have that $s = d(J) - p(J)$ for some job J. Similarly, we decrease t, as much as possible, by moving jobs to the left without violating any constraints, but now also without decreasing s. Then, some job J' cannot be moved to the left either due to its release time $r(J')$ or since it starts at time s. In such cases, either we have $t = r(J') + p(J')$ for this job J' or $t = s + p(J') = d(J) - p(J) + p(J')$ for the jobs J and J'. In addition, we set the start time of each job J scheduled on M to be $\max\{r(J), s\}$ by moving J to the left, if needed. This completes the proof. ■

Motivated by Lemma 3.3, we consider the following definition.

Definition 3.1 *A time t is called* interesting *if $t = r(J)$ or $t = d(J) - p(J)$, for some job J, or $t = r(J) + p(J)$ or $t = d(J) - p(J) + p(J')$, for some jobs J and J'. Let $\mathcal{T}$ denote the set of interesting times.*

Thus, w.l.o.g., we may assume that the busy periods of all machines and placements of all jobs start and end at interesting times. Let the intervals of all the jobs be contained in $[0, T)$. W.l.o.g., we may assume that both 0 and T are interesting times. Note that the number of interesting times is polynomial.

Now we describe our dynamic program. Informally, the algorithm processes the jobs $J \in \bar{\mathcal{J}}$ in the order of non-increasing processing times $p(J)$. It first guesses the placement $[t, t + p(J_1)) \in [r(J_1), d(J_1))$ of job J_1 with the largest processing time. Once this is done, the remainder of the problem splits into two *independent* sub-problems: the "left" problem $[0, t)$ and the "right" problem $[t + p(J_1), T)$. This is so because any job J whose interval $[r(J), d(J))$ has an intersection with $[t, t + p(J_1))$ of size at least $p(J)$ can be scheduled inside the interval $[t, t + p(J_1))$ without any extra cost. The "left" sub-problem then estimates the minimum busy time in the interval $[0, t)$ for scheduling jobs whose placement must intersect $[0, t)$; similarly the "right" sub-problem estimates the minimum busy time in the interval $[t + p(J_1), T)$ for scheduling jobs whose placement must intersect $[t + p(J_1), T)$. More formally,

Definition 3.2 *Let $t_1, t_2 \in \mathcal{T}$ with $t_2 > t_1$ and $\ell = p(J)$ for some job J. Let $\texttt{jobs}(t_1, t_2, \ell)$ denote the set of jobs in $\bar{\mathcal{J}}$ whose processing time is at most ℓ and whose placement must intersect the interval $[t_1, t_2)$, i.e.,*

$$\texttt{jobs}(t_1, t_2, \ell) = \{J \in \bar{\mathcal{J}} \mid p(J) \leq \ell, t_1 - r(J) < p(J), d(J) - t_2 < p(J)\}.$$

Let $\texttt{cost}([t_1, t_2), \ell)$ be the minimum busy-time inside the interval $[t_1, t_2)$ for scheduling jobs in $\texttt{jobs}(t_1, t_2, \ell)$ in a feasible manner.

Note that $\texttt{cost}([t_1, t_2), \ell)$ counts the busy-time only inside the interval $[t_1, t_2)$ assuming that the busy-time outside this interval is already "paid for". For convenience, we define $\texttt{jobs}(t_1, t_2, \ell) = \emptyset$ and $\texttt{cost}([t_1, t_2), \ell) = 0$, whenever $t_2 \leq t_1$.

Lemma 3.4 *If $\texttt{jobs}(t_1, t_2, \ell) = \emptyset$ then $\texttt{cost}([t_1, t_2), \ell) = 0$. Otherwise, let $J \in \texttt{jobs}(t_1, t_2, \ell)$ be a job with the longest processing time among the jobs in $\texttt{jobs}(t_1, t_2, \ell)$. Then,*

$$\begin{aligned}\texttt{cost}([t_1, t_2), \ell) \quad &= \min_{t \in [r(J), d(J) - p(J)) \cap \mathcal{T}} \Big(\min\{p(J), t + p(J) - t_1, t_2 - t, t_2 - t_1\} \\ &\quad + \texttt{cost}([t_1, t), p(J)) \quad + \texttt{cost}([t + p(J), t_2), p(J)) \Big). \end{aligned} \tag{3}$$

Proof: The proof is by induction on the cardinality of the set $\texttt{jobs}(t_1, t_2, \ell)$. The base case of cardinality 0, i.e., $\texttt{jobs}(t_1, t_2, \ell) = \emptyset$, is trivial since in this case $\texttt{cost}([t_1, t_2), \ell) = 0$. Assume that $|\texttt{jobs}(t_1, t_2, \ell)| > 0$ and that the claim is true for all sets of lower cardinality. We first show that $\texttt{cost}([t_1, t_2), \ell)$ is at most the expression (3), by constructing a feasible schedule of jobs in $\texttt{jobs}(t_1, t_2, \ell)$ with total busy-time in $[t_1, t_2)$ equal to (3). Given a job J, consider the time t that achieves the minimum in (3). Schedule J in $[t, t + p(J))$. This job contributes $\min\{p(J), t + p(J) - t_1, t_2 - t, t_2 - t_1\}$ to the busy-time inside $[t_1, t_2)$, depending on whether both $t, t + p_J(J) \in [t_1, t_2)$,

$t \notin [t_1, t_2)$ and $t + p_J(J) \in [t_1, t_2)$, $t \in [t_1, t_2)$ and $t + p_J(J) \notin [t_1, t_2)$, or both $t, t + p_J(J) \notin [t_1, t_2)$. Since $p(J) \geq p(J')$ for any other job $J' \in \texttt{jobs}(t_1, t_2, \ell)$, we get that $\texttt{jobs}(t_1, t, p(J)) \cap \texttt{jobs}(t + p(J), t_2, p(J)) = \emptyset$. We use the solution of cost $\texttt{cost}([t_1, t), p(J))$ (resp., $\texttt{cost}([t + p(J), t_2), p(J)))$ to schedule jobs in $\texttt{jobs}(t_1, t, p(J))$ (resp., $\texttt{jobs}(t + p(J), t_2, p(J)))$ in the interval $[t_1, t)$ (resp., $[t + p(J), t_2)$). The jobs in $\texttt{jobs}(t_1, t_2, \ell) \setminus \{\{J\} \cup \texttt{jobs}(t_1, t, p(J)) \cup \texttt{jobs}(t + p(J), t_2, p(J))\}$ can be scheduled inside the interval $[t, t + p(J))$ without paying extra cost. Thus the overall cost of the solution is the expression (3). From the definition of $\texttt{cost}([t_1, t_2), \ell)$, we get that $\texttt{cost}([t_1, t_2), \ell)$ is at most the expression (3), as claimed.

Now we show that $\texttt{cost}([t_1, t_2), \ell)$ is at least the expression (3). Consider the schedule S of cost $\texttt{cost}([t_1, t_2), \ell)$, and let $J \in \texttt{jobs}(t_1, t_2, \ell)$ be a job with the largest processing time. Let t be the start time of J in S. Then S pays the cost $\min\{p(J), t + p(J) - t_1, t_2 - t, t_2 - t_1\}$ inside $[t_1, t_2)$, depending on whether both $t, t+p(J) \in [t_1, t_2)$, $t \notin [t_1, t_2)$, or $t+p(J) \notin [t_1, t_2)$. W.l.o.g., we may assume that S schedules the jobs in $\texttt{jobs}(t_1, t_2, \ell) \setminus (\{J\} \cup \texttt{jobs}(t_1, t, p(J)) \cup \texttt{jobs}(t + p(J), t_2, p(J)))$ inside the interval $[t, t + p(J))$. The cost of S for scheduling jobs $\texttt{jobs}(t_1, t, p(J))$ (respectively, $\texttt{jobs}(t + p(J), t_2, p(J)))$ in the interval $[t_1, t)$ (respectively, $[t + p(J), t_2)$) is, by definition, at least $\texttt{cost}([t_1, t), p(J))$ (respectively, $\texttt{cost}([t + p(J), t_2), p(J)))$. Thus $\texttt{cost}([t_1, t_2), \ell)$ is at least the expression (3), as claimed.

This completes the proof. ■

Note that the number of interesting times and the number of distinct processing lengths are polynomial. Thus, the quantities $\texttt{cost}([t_1, t_2), \ell)$ for $t_1, t_2 \in \mathcal{T}$ and $\ell = p(J)$ for some $J \in \bar{\mathcal{J}}$ and their corresponding schedules can be computed, using the relation in Lemma 3.4, in polynomial time. We finally output the schedule corresponding to $\texttt{cost}([0, T), \max_{J \in \bar{\mathcal{J}}} p(J))$. By definition, this gives the optimum solution.

4 Real-time Scheduling for Moldable Jobs: Theorem 1.3

A job J in an instance $\bar{\mathcal{J}}$ of the real-time scheduling problem with moldable jobs is described by a release time $r(J)$, a due date $d(J)$, and a set of configurations $\{(p_i(J), R_i(J))\}_{i=1,\ldots,q}$. We assume, w.l.o.g., that $p_i(J) \leq d(J) - r(J)$ for all $1 \leq i \leq q$. The goal is to pick a configuration $1 \leq i_J \leq q$ for each job J and schedule these jobs on machines with a capacity g such that the total busy-time is minimized while satisfying the capacity constraints. Given configurations $\vec{i} = \{i_J\}_{J \in \bar{\mathcal{J}}}$, let $\bar{\mathcal{J}}(\vec{i})$ denote the instance of real-time scheduling problem derived from $\bar{\mathcal{J}}$ by fixing configuration i_J for each job J. Let $\text{OPT}(\bar{\mathcal{J}})$ denote the cost of the optimum schedule, and let $\vec{i}^* = \{i^*_J\}_{J \in \bar{\mathcal{J}}}$ denote the configurations used in this schedule. From Observation 1.5, we have

$$\text{OPT}_\infty(\bar{\mathcal{J}}(\vec{i}^*)) + 4 \cdot \frac{w(\bar{\mathcal{J}}(\vec{i}^*))}{g} \leq 5 \cdot \text{OPT}(\bar{\mathcal{J}}). \tag{4}$$

In this section, we prove the following main lemma.

Lemma 4.1 *Given an instance $\bar{\mathcal{J}}$ of the real-time scheduling with moldable jobs, we can find in polynomial time configurations $\vec{i} = \{i_J\}_{J \in \bar{\mathcal{J}}}$, such that $\text{OPT}_\infty(\bar{\mathcal{J}}(\vec{i})) + 4 \cdot \frac{w(\bar{\mathcal{J}}(\vec{i}))}{g}$ is minimized.*

Proof: Motivated by Lemma 3.3 and Definition 3.1, we define the set of interesting times as follows.

Definition 4.1 *A time t is called* interesting *if $t = r(J)$ or $t = d(J) - p_i(J)$, for some job J and configuration i, or $t = r(J) + p_i(J)$ or $t = d(J) - p_i(J) + p_{i'}(J')$, for some jobs J and J' and their respective configurations i and i'. Let $\mathcal{T}$ denote the set of interesting times.*

Note that the size of $\mathcal{T}$ is polynomial and we can assume, w.l.o.g., that the busy periods of all machines and placements of all jobs start and end at interesting times. Let the intervals of all the jobs be contained in $[0, T)$. W.l.o.g. we can assume that both 0 and T are interesting times. For a job $J \in \bar{\mathcal{J}}$ and a configuration $1 \le i_J \le q$, let $w_{i_J}(J) = p_{i_J}(J) \cdot R_{i_J}(J)$.

Definition 4.2 *Let $t_1, t_2 \in \mathcal{T}$ with $t_2 > t_1$ and $\ell \ge 0$. Let*

$$\texttt{jobs}(t_1, t_2, \ell) = \{J \in \bar{\mathcal{J}} \mid r(J) > t_1 - \ell, d(J) < t_2 + \ell\}.$$

For a choice of configurations $\vec{i} = \{i_J\}_{J \in \texttt{jobs}(t_1,t_2,\ell)}$, let $\texttt{cost}([t_1, t_2), \ell, \vec{i})$ denote the minimum busy-time inside interval $[t_1, t_2)$ for scheduling jobs in $\texttt{jobs}(t_1, t_2, \ell)(\vec{i})$ in a feasible manner. Let $\texttt{ub}([t_1, t_2), \ell)$ be the minimum value of

$$\texttt{cost}([t_1, t_2), \ell, \vec{i}) + 4 \cdot \frac{w(\texttt{jobs}(t_1, t_2, \ell)(\vec{i}))}{g},$$

where the minimum is taken over all configurations $\vec{i}$ that satisfy $p_{i_J}(J) \le \ell$, for all jobs $J \in \texttt{jobs}(t_1, t_2, \ell)$.

As before, $\texttt{cost}([t_1, t_2), \ell, \vec{i})$ counts the busy-time only inside the interval $[t_1, t_2)$, assuming that the busy-time outside this interval is already "paid for". We define the $\min_A = \infty$, if $A = \emptyset$.

Lemma 4.2 *If $\texttt{jobs}(t_1, t_2, \ell) = \emptyset$ we have $\texttt{ub}([t_1, t_2), \ell) = 0$. If $t_2 \le t_1$ then*

$$\texttt{ub}([t_1, t_2), \ell) = \sum_{J \in \texttt{jobs}(t_1,t_2,\ell)} \min_{i_J : p_{i_J}(J) \le \ell} 4 \cdot \frac{w_{i_J}(J)}{g}.$$

Otherwise, let $I_L = [t_1, \min\{t, t_2\})$ and $I_R = [\max\{t + p_{i_J}(J), t_1\}, t_2)$, then we have

$$\begin{aligned} \texttt{ub}([t_1, t_2), \ell) \;=\; & \min_{J \in \texttt{jobs}(t_1,t_2,\ell)} \; \min_{i_J : p_{i_J}(J) \le \ell} \; \min_{t \in [r(J), d(J) - p_{i_J}(J)) \cap \mathcal{T}} \\ & \Bigg(\min \Big\{ p_{i_J}(J), \max\{0, t + p_{i_J}(J) - t_1\}, \max\{0, t_2 - t\} \Big\} \\ & + \sum_{\substack{J' \in \texttt{jobs}(t_1,t_2,\ell) \setminus \\ (\texttt{jobs}(I_L, p_{i_J}(J)) \cup \texttt{jobs}(I_R, p_{i_J}(J)))}} \; \min_{i_{J'} : p_{i_{J'}}(J') \le p_{i_J}(J)} 4 \cdot \frac{w_{i_{J'}}(J')}{g} \\ & + \texttt{ub}(I_L, p_{i_J}(J)) \\ & + \texttt{ub}(I_R, p_{i_J}(J)) \Bigg). \end{aligned} \tag{5}$$

Note that the number of interesting times and the number of distinct processing lengths are polynomial. Thus, the quantities $\texttt{ub}([t_1, t_2), \ell)$ for $t_1, t_2 \in \mathcal{T}$ and $\ell = p_i(J)$, for some $J \in \bar{\mathcal{J}}, 1 \le i \le q$ and their corresponding job-configurations and schedules can be computed, using the relation in

Lemma 4.2, in polynomial time. The algorithm finally outputs the job-configurations corresponding to $\texttt{ub}([0,T), \max_{J\in\bar{\mathcal{J}},1\le i\le q} p_i(J))$. By definition, this proves Lemma 4.1. ■

Proof of Lemma 4.2: By definition, it follows that if $\texttt{jobs}(t_1,t_2,\ell)=\emptyset$, we have $\texttt{ub}([t_1,t_2),\ell)=0$. If $t_2\le t_1$, the jobs in $\texttt{jobs}(t_1,t_2,\ell)$ can be scheduled in the interval $[t_1-\ell, t_2+\ell)$ without paying any additional busy-time. Therefore, $\texttt{cost}([t_1,t_2),\ell,\vec{i})=0$, for all $\vec{i}$, and

$$\texttt{ub}([t_1,t_2),\ell) = \sum_{J\in\texttt{jobs}(t_1,t_2,\ell)} \min_{i_J : p_{i_J}(J)\le\ell} 4\cdot\frac{w_{i_J}(J)}{g}$$

follows from the definition.

The rest of the proof is by induction on the cardinality of $\texttt{jobs}(t_1,t_2,\ell)$ similar to the proof of Lemma 3.4.

We first show that $\texttt{ub}([t_1,t_2),\ell)$ is at most the expression (5), by constructing a feasible schedule of jobs in $\texttt{jobs}(t_1,t_2,\ell)$ with total busy-time in $[t_1,t_2)$ equal to (5). Consider the job J, its configuration i_J, and time t that achieve the minimum in (5). Schedule J, fixing its configuration i_J, in $[t, t+p_{i_J}(J))$. This job contributes $\min\{p_{i_J}(J), \max\{0, t+p_{i_J}(J)-t_1\}, \max\{0, t_2-t\}\}$ to the busy-time inside $[t_1,t_2)$ depending on whether both $t, t+p_{i_J}(J)\in[t_1,t_2)$, $t\notin[t_1,t_2)$ and $t+p_{i_J}(J)\in[t_1,t_2)$, $t\in[t_1,t_2)$ and $t+p_{i_J}(J)\notin[t_1,t_2)$, or both $t, t+p_{i_J}(J)\notin[t_1,t_2)$. All jobs $J'\in\texttt{jobs}(t_1,t_2,\ell)\setminus(\texttt{jobs}(I_L,p_{i_J}(J))\cup\texttt{jobs}(I_R,p_{i_J}(J)))$ can either be scheduled inside $[t,t+p_{i_J}(J))$ or outside $[t_1,t_2)$, provided we chose configuration $i_{J'}$ satisfying $p_{i_{J'}}(J')\le p_{i_J}(J)$ for each such job J'. Thus, these jobs do not contribute anything additional to the busy-time inside $[t_1,t_2)$. However such jobs J' contribute $\min_{i_{J'}:p_{i_{J'}}(J')\le p_{i_J}(J)} 4\cdot\frac{w_{i_{J'}}(J')}{g}$ to $\texttt{ub}([t_1,t_2),\ell)$. We next use the solution of cost $\texttt{ub}(I_L,p_{i_J}(J))$ (respectively, $\texttt{ub}(I_R,p_{i_J}(J)]))$ to schedule jobs in $\texttt{jobs}(I_L,p_{i_J}(J))$ (respectively, $\texttt{jobs}(I_R,p_{i_J}(J)))$ in the interval I_L (respectively, I_R). Thus the overall cost of the solution is the expression (5). From the definition of $\texttt{ub}([t_1,t_2),\ell)$, we get that $\texttt{ub}([t_1,t_2),\ell)$ is at most the expression (5), as claimed.

Now, we show that $\texttt{ub}([t_1,t_2),\ell)$ is at least the expression (5). Consider the schedule S and configurations $\{i_J\}_{J\in\texttt{jobs}(t_1,t_2,\ell)}$ corresponding to $\texttt{ub}([t_1,t_2),\ell)$, and let $J\in\texttt{jobs}(t_1,t_2,\ell)$ and its configuration i_J be the job with the largest processing time $p_{i_J}(J)$. Let t be the start time of J in S. Thus, the cost upper bound pays $\min\{p_{i_J}(J), \max\{0,t+p_{i_J}(J)-t_1\}, \max\{0,t_2-t\}\}$ inside $[t_1,t_2)$. W.l.o.g., we can assume that S schedules the jobs in

$$\texttt{jobs}(t_1,t_2,\ell)\setminus(\texttt{jobs}(I_L,p_{i_J}(J))\cup\texttt{jobs}(I_R,p_{i_J}(J)))$$

inside the interval $[t,t+p_{i_J}(J))$, or outside the interval $[t_1,t_2)$. These jobs J', w.l.o.g., contribute $\min_{i_{J'}:p_{i_{J'}}(J')\le p_{i_J}(J)} 4\cdot\frac{w_{i_{J'}}(J')}{g}$ to the cost upper bound. The cost upper bound for scheduling jobs $\texttt{jobs}(I_L,p_{i_J}(J))$ (resp., $\texttt{jobs}(I_R,p_{i_J}(J)))$ in the interval I_L (resp., I_R) is, by definition, at least $\texttt{ub}(I_L,p_{i_J}(J))$ (resp., $\texttt{ub}(I_R,p_{i_J}(J)))$. Thus, $\texttt{ub}([t_1,t_2),\ell)$ is at least the expression (5), as claimed.

This completes the proof. ■

Now, recall that Lemma 3.1 and Theorem 3.2 together imply that given an instance $\bar{\mathcal{J}}(\vec{i})$ of the real-time scheduling problem, we can compute in polynomial time a feasible schedule with busy-time at most $\text{OPT}_\infty(\bar{\mathcal{J}}(\vec{i})) + 4 \cdot \frac{w(\bar{\mathcal{J}}(\vec{i}))}{g}$. Thus, equation (4), Lemma 4.1, Lemma 3.1, and Theorem 3.2 together imply that we can find a schedule with cost at most

$$\text{OPT}_\infty(\bar{\mathcal{J}}(\vec{i})) + 4 \cdot \frac{w(\bar{\mathcal{J}}(\vec{i}))}{g} \;\leq\; \text{OPT}_\infty(\bar{\mathcal{J}}(\vec{i^*})) + 4 \cdot \frac{w(\bar{\mathcal{J}}(\vec{i^*}))}{g} \;\leq\; 5 \cdot \text{OPT}(\bar{\mathcal{J}}),$$

thus yielding a 5-approximation.

5 Interval Scheduling with Unit Demands: Theorem 1.4

In this section, we consider instances with interval jobs, where all jobs have unit demands, i.e., $p(J) = d(J) - r(J)$ and $R(J) = 1$.

5.1 Laminar Instances

We show a polynomial time exact algorithm for the case in which job intervals $I(J)$, for all jobs J, form a laminar family, i.e., for any two jobs $J, J' \in \bar{\mathcal{J}}$, it holds that $I(J) \cap I(J') = \emptyset$ or $I(J) \subset I(J')$, or $I(J') \subset I(J)$.

Since the job intervals are laminar, the jobs can be represented by a forest F of rooted trees, where each vertex in a tree $T \in F$ corresponds to a job, and a vertex $v(J)$ is an ancestor of a vertex $v(J')$ if and only if $I(J') \subset I(J)$. Let the level of a vertex be defined as follows. Any root of a tree in the forest is at level 1. For all other vertices v, the level of v is 1 plus the level of its parent. Consider an algorithm which assigns jobs in level ℓ to machine $M_{\lceil \ell/g \rceil}$.

Theorem 5.1 *The algorithm yields an optimal solution for laminar instances.*

Proof: Clearly, the algorithm outputs a feasible solution, since at most g jobs are scheduled on any machine at any time. Let M_t (resp., N_t) be the number of active machines (resp., jobs) at time t. Then $M_t = \lceil N_t/g \rceil$. This proves the claim, by exploiting the work bound in Observation 1.5. ■

5.2 Instances that Form a Clique

In the following we show that if all jobs have unit demands, and the corresponding graph is a clique, then the problem can be approximated within factor $1 + \varepsilon$, for any $\varepsilon > 0$. Recall that for general instances of job intervals with unit demands the problem is NP-hard already for $g = 2$ [25]. We show that for inputs that form a clique the problem with $g = 2$ is solvable in polynomial time. Finally, we show that the maximum revenue problem is solvable for any $g \geq 1$. In this variant, each job is associated with a profit, the busy intervals of the machines are given, and the goal is to find a feasible schedule of maximum profit.

5.2.1 A PTAS for Cliques

Since the instance $\bar{\mathcal{J}}$ forms a clique, there is a time t_0 such that $t_0 \in I(J)$ for all $J \in \mathcal{J}$. Thus, the busy period of each machine is a contiguous interval. The PTAS consists of two main phases. First, it extends the interval lengths, then it finds (using dynamic programming) an optimal schedule of the resulting instance on $m = \lceil n/g \rceil$ machines, where n is the number of jobs. We give the pseudocode of the scheme below.

Theorem 5.2 *For any $\varepsilon \in (0, 1]$, the scheme with $c = \lceil 1/\varepsilon \rceil$ is a PTAS for any clique.*

Approximation Scheme for a Clique:

1. Let $c > 1$ be a constant.
2. Let t_0 be such that $t_0 \in I(J)$ for all $J \in \bar{\mathcal{J}}$. Let $\texttt{left}(J) = t_0 - r(J)$, $\texttt{right}(J) = d(J) - t_0$. Also, let $\texttt{sh}(J) = \min\{\texttt{left}(J), \texttt{right}(J)\}$ and $\texttt{lo}(J) = \max\{\texttt{left}(J), \texttt{right}(J)\}$ be the length of the short (resp. long) segment of J w.r.t. t_0. If $\texttt{sh}(J)/\texttt{lo}(J) \in ((k-1)/c, k/c]$ for some $1 \le k \le c$, stretch the short segment to round the ratio to k/c.
3. Partition the jobs into $2c - 1$ classes. For $\ell \in \{1, \ldots, c\}$, the ℓ-th class consists of all jobs for which $\texttt{sh}(J)/\texttt{lo}(J) = \ell/c$ and $\texttt{left}(J) \ge \texttt{right}(J)$. For $\ell \in \{c+1, \ldots, 2c-1\}$, the ℓ-th class consists of all jobs for which $\texttt{sh}(J)/\texttt{lo}(J) = (\ell - c)/c$ and $\texttt{left}(J) < \texttt{right}(J)$. Let n_ℓ be the number of jobs in class ℓ, $1 \le \ell \le 2c - 1$.
4. For $i \ge 1$, let $C_i(n'_1, \ldots, n'_{2c-1})$ be the minimum cost of scheduling the longest n'_ℓ jobs of class ℓ, for all ℓ, on i machines. Let $m = \lceil n/g \rceil$. Use dynamic programming to find a schedule achieving $C_m(n_1, \ldots, n_{2c-1})$.

We prove Theorem 5.2 using the following results.

Observation 5.3 *Given a clique, for any $c \ge 1$ stretching the short segment of any interval in step 2 of the scheme may increase the busy time of the schedule at most by a factor of $1 + 1/c$.*

Proof: Given a schedule of the input jobs, consider the subset of jobs assigned to M_i, for some $i \ge 1$. Clearly, there exist two jobs, J_ℓ and J_r on M_i, such that $\texttt{busy}(M_i) = \texttt{left}(J_\ell) + \texttt{right}(J_r)$. Stretching the short segments in step 2 may extend some left segment of an interval, causing it to start earlier, extending $\texttt{left}(J_\ell)$ by at most a factor of $1 + 1/c$. Similarly, a stretch of a short right segment may cause it to exceed the length $\texttt{right}(J_r)$ by at most factor $1 + 1/c$. ■

The DP in step 4 schedules the jobs in $m = \lceil n/g \rceil$ machines. This is justified by the following lemma.

Lemma 5.4 *If the instance is a clique then any optimal schedule uses exactly $m = \lceil n/g \rceil$ machines.*

Proof: Clearly, at least m machines are active at time t_0. Assume that $m' > \lceil n/g \rceil$ machines are used in some schedule. We show that the number of machines can be reduced while decreasing the total busy time. We say that a machine is *full* if it processes g jobs. Assume that there are k non-full machines, $M_1, M_2, ..., M_k$. Let the busy intervals of these non-full machines be $[s_1, e_1], [s_2, e_2], \ldots, [s_k, e_k]$ such that $s_1 \le \ldots \le s_k$. W.l.o.g., these intervals form a proper interval graph, that is, $e_1 \le \ldots \le e_k$, or else, if for some i, j, $[s_i, e_i]$ is contained in $[s_j, e_j]$ then it is possible to move jobs from M_i to M_j, until either j is full or i is empty.

Since $m' > \lceil n/g \rceil$, it holds that $m' \geq (n+g)/g$ and $n \leq g(m'-1)$. The full machines hold exactly $(m'-k)g$ jobs. Thus, $n-(m'-k)g \leq g(k-1)$ jobs are scheduled on the k non-full machines. Given that $s_1 \leq \ldots \leq s_k$ and $e_1 \leq \ldots \leq e_k$ and that all k machines together hold at most $(k-1)g$ jobs, we show how to reassign the jobs scheduled on the non-full machines, on at most $k-1$ machines while decreasing the total busy time.

First, move the rightmost jobs (those with the largest $d(J)$) from M_2 to M_1, until either M_1 holds g jobs or M_2 is empty. The busy time of M_1 is increased by $e_2 - e_1$. If M_2 becomes empty we are done (the busy time of M_2 is 0 and we have one less active machine as needed). Next, if M_2 still has jobs on it, move jobs from M_3 to M_2. By the same argument, the busy time of M_2 is increased by $e_3 - e_2$. Continue 'filling machines' until some machine becomes empty. Since the total number of jobs on non-full machines is at most $(k-1)g$, this must happen in M_k at the latest. Let M_j be the machine that becomes empty. The total increase in the busy time of all involved machines is at most $(e_2-e_1)+(e_3-e_2)+(e_4-e_3)+\cdots+(e_j-e_{j-1}) = e_j - e_1$. On the other hand, machine M_j is not active anymore, so its busy time of $e_j - s_j > e_j - e_1$ is saved. It holds that $e_1 > s_j$ since the instance is a clique. We conclude that the total busy time saved is larger than the total added busy time. ■

As stated in step 4, the DP only considers schedules in which the jobs from each class are assigned to machines from longest to shortest. That is, when scheduling $\hat{n}_\ell$ jobs from class ℓ on machine i, the DP selects the $\hat{n}_\ell$ longest unscheduled jobs in this class. Note that since the jobs of a class share the same $\texttt{sh}(J)/\texttt{lo}(J)$ ratio and the same long side, then the intervals from each class, when sorted according to length, are nested in each other. The following Lemma assures that there exists an optimal schedule of the extended intervals in which the jobs are assigned to machines according to the nesting order.

Lemma 5.5 *(Nesting property) Given a clique of intervals, there exists an optimal schedule in which the machines can be ordered such that for any class $1 \leq \ell \leq 2c-1$ and $1 \leq i < m$, every job of class ℓ assigned to machine i is longer than every job of class ℓ assigned to any machine i', for $i < i' \leq m$.*

Proof: We show that if the nesting property does not hold in some schedule, then it is possible to achieve it without increasing the total cost of the schedule. Note that the nesting property is implied by the following two properties:

$P1$: The jobs of class ℓ assigned to one machine form a contiguous subset in the nested order of class ℓ.

$P2$: For all $1 \leq i < m$, the set of jobs of class ℓ assigned to machine i precedes the set of jobs of class ℓ assigned to any machine i', for $i < i' \leq m$, in the nested order of class ℓ.

Consider a given schedule. First, we show how to convert the schedule into one in which property 1 holds: assume that for some ℓ, machine i is assigned the k-th and the $(k+\delta)$-th job from class ℓ (for some $\delta > 1$) but not the $(k+1)$-st job. It is possible to exchange the assignments of the $(k+1)$-st and the $(k+\delta)$-th jobs without increasing the schedule cost (recall that the $(k+\delta)$-th job is contained in the $(k+1)$-st, which is contained in the k-th job). By performing such inner-class exchanges for each class according to the nesting order, we get a schedule in which a contiguous subset of jobs from each class is scheduled on each machine.

Next we show that it is possible to convert a schedule fulfilling $P1$ into a one fulfilling $P2$. We say that the pair of machines M' and M'' have *inversion* if there exist two classes, j and k, such that M' contains a set of short jobs from j, A'_j, and a set of long jobs from k, A'_k, and M'' contains a set of long jobs from j, A''_j, and a set of short jobs from k, A''_k. By 'short' and 'long' we refer to the containment relation that we defined on the jobs in each class, i.e., all the jobs in A'_j are contained in the shortest job in A''_j (and similarly for A''_k and A'_k).

Let $h = \min(|A'_j|, |A''_k|)$, namely, h is the smaller between the sets of short jobs. W.l.o.g., suppose that the smaller is A'_j. We move the jobs in A'_j to M'' and move the h longest jobs in A''_k to M'. This does not affect the total busy time, since we add to each machine a set of jobs that are contained in some job scheduled on this machine, which belongs to the long job sets A''_j and A'_k, respectively. In the resulting schedule, all of the above jobs of class j are scheduled on M', and we have decreased the number of inversions. If, as a result of the new assignment, the jobs of classes j, k assigned to M' or M'', do not form a contiguous subset in the nested order of their class, i.e., $P1$ is violated, then apply inner-class exchanges to close the gap and to keep the relative order of jobs from each class across the machines. These shifts guarantee that $P1$ holds and also, that no new inversions are created when one inversion is removed.

We continue decreasing the number of inversions for pairs of machines and for pairs of classes, until no inversions exist. At this point, it is possible to order the machines in a way that fulfills the nesting property. ■

To complete the proof of Theorem 5.2 we describe the dynamic programming that is based on Observation 5.3, and lemmas 5.4 and 5.5. For simplicity we modify the number of jobs so that it is a multiple of g, this is done without impacting the approximation ratio, by adding $mg - n$ dummy jobs of length $\varepsilon \to 0$ into the c-th class (around t_0). After this addition all machines are full.

The dynamic programming proceeds as follows: For $1 \leq \ell \leq 2c - 1$, for any $n' \leq n_\ell$, and $h \leq n_\ell - n'$, let $J_\ell(n')$ be the n' longest job in class ℓ, and let $J_\ell(n', h) = J_\ell(n' + h) \setminus J_\ell(n')$ be the set of h jobs at rank $n' + 1, \ldots, n' + h$ in class ℓ. In particular, if $h = 0$ then $J_\ell(n', h) = \emptyset$. For any $(n'_1, \ldots, n'_{2c-1})$ such that $n'_\ell \leq n_\ell$ for $\ell = 1, \ldots, 2c-1$, and any $(h_1, \ldots, h_{2c-1})$ such that $\sum_\ell h_\ell = g$ and $0 \leq h_\ell \leq n_\ell - n'_\ell$, let $f(n'_1, \ldots, n'_{2c-1}; h_1, \ldots, h_{2c-1}) = \mathtt{span}(J_1(n'_1, h_1) \cup \cdots \cup J_\ell(n'_{2c-1}, h_{2c-1}))$. Note that $f(n'_1, \ldots, n'_{2c-1}; h_1, \ldots, h_{2c-1})$ is the busy time of the machine that is assigned h_ℓ jobs from class ℓ starting after the n'_ℓ longest job, for $\ell = 1, \ldots, 2c - 1$. Recall that $C_i(n'_1, \ldots, n'_{2c-1})$ denotes the minimum cost of scheduling the longest n'_ℓ jobs of each class ℓ on i machines. Since all machines are full, we need to compute $C_i(n'_1, \ldots, n'_{2c-1})$, for $1 \leq i \leq m$, only for $(n'_1, \ldots, n'_{2c-1})$ such that $\sum_\ell n'_\ell = i \cdot g$. Obviously, for $(h_1, \ldots, h_{2c-1})$ such that $\sum_\ell h_\ell = g$, $C_1(h_1, \ldots, h_{2c-1}) = f(0, \ldots, 0; h_1, \ldots, h_{2c-1})$. For $1 < i \leq m$, and $(n'_1, \ldots, n'_{2c-1})$ such that $\sum_\ell n'_\ell = i \cdot g$, let

$$\begin{aligned} C_i(n'_1, \ldots, n'_{2c-1}) &= \min_{(h_1,\ldots,h_{2c-1}) s.t \sum_\ell h_\ell = g} \{C_{i-1}(n'_1 - h_1, \ldots, n'_{2c-1} - h_{2c-1}) \\ &\qquad + f(n'_1 - h_1, \ldots, n'_{2c-1} - h_{2c-1}; h_1, \ldots, h_{2c-1})\}. \end{aligned}$$

Since c is a constant, the number of entries in the dynamic programming table (i.e., the table containing the partial solution values used for computing $C_i(n'_1, \ldots, n'_{2c-1})$) is polynomial in n and g, and the time required to calculate each entry is $n^{O(c)}$. The optimal schedule of the extended instance for m machines is given by $C_m(n_1, \ldots, n_{2c-1})$.

5.2.2 Polynomially Solvable Instances

The case $g = 2$: It is possible to solve the problem optimally for cliques by reducing it to minimum-weight perfect matching in a complete graph. Given $\bar{\mathcal{J}}$, let t_0 be the intersection point for all intervals, i.e., $t_0 \in [r(J), d(J))$ for all $J \in \mathcal{J}$. If n is odd add a dummy job of length $\varepsilon \to 0$ around t_0. Construct a complete graph in which each job corresponds to a vertex and for every pair i, j, the edge (J_i, J_j) has weight $\mathtt{span}(J_i \cup J_j)$. Use Edmond's algorithm [9] to find a minimum-weight perfect matching in the graph. It is easy to verify that an optimal schedule corresponds to a partition of the jobs into $n/2$ pairs. The busy time for each pair is the weight of the corresponding edge in the complete graph.

The Max Revenue Problem: Consider the following variant of our problem on cliques. As before, we have interval jobs of unit demands. Each job J is associated with a processing time, as well as a profit $w(J)$, that is gained if J is completed. Also, the busy intervals of the machines are given. The goal is to find a feasible schedule of a maximal-profit subset of jobs. It is possible to solve this max revenue problem, for any $g \geq 1$, by reducing it to the following min-cost max-flow problem. Given the set $\mathcal{I}$ of busy intervals, and the set of jobs $\bar{\mathcal{J}}$, construct a network flow in which $V = \{s, t\} \cup \mathcal{J} \cup \mathcal{I}$. There is an edge (s, J_j) of cost 0 and capacity 1 connecting the source to every job J_j, an edge (J_j, I_k) of cost $-w(J_j)$ and capacity 1 connecting job j and interval k if J_j is contained in I_k (that is, job j can be processed by machine k), and an edge (I_k, t) of cost 0 and capacity g connecting every busy interval with the target. It is easy to verify that any feasible flow in this network corresponds to a feasible schedule on the given set of busy intervals. In particular, the min-cost max-flow corresponds to an optimal schedule.

Acknowledgment: We thank Dor Arad for helpful discussions on an earlier version of the paper. We are also grateful to the two anonymous referees for valuable comments and suggestions.

References

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

[2] L. Belede, A. Jain, and R. Reddy Gaddam. Unit commitment with nature and biologically inspired computing. In *World Congress on Nature and Biologically Inspired Computing (NABIC)*, pages 824–829, 2009.

[3] S. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. *ACM Transactions on Algorithms*, 3(1), 2007.

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

[5] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. An improved approximation algorithms for resource allocation. *ACM Transactions on Algorithms*, 7(4):1–7, 2011.

[6] J. Chang, S. Khuller, and K. Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. In *26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2014.

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

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

[9] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[10] 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. *Theoretical Computer Science*, 411(40-42):3553–3562, 2010.

[11] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem with respect to ADMs and OADMs. In *14th Euro-Par*, 2008.

[12] O. Gerstel, R. Ramaswami, and G. Sasaki. Cost effective traffic grooming in wdm rings. In *INFOCOM*, 1998.

[13] J. Kang and S. Ranka. Energy-efficient dynamic scheduling on parallel machines. In *High Performance Computing (HiPC)*, pages 208–219, 2008.

[14] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 169–180, 2010.

[15] 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.

[16] E. Lawler, J.K. Lenstra, A.H.G. Rinnooy 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.

[17] W. T. Ludwig. *Algorithms for Scheduling Malleable and Nonmalleable Parallel Tasks.* PhD thesis, Dept. of Computer Science, Univ. of Wisconsin - Madison, 1995.

[18] A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy/power. *IEEE Trans. VLSI Syst. 11(2)*, pages 501–507, 2003.

[19] G. B. Mertzios, M. Shalom, A. Voloshin, P. W. H. Wong, and S. Zaks. Optimizing busy time on parallel machines. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 238–248, 2012.

[20] C.A. Phillips, R.N. Uma, and J. Wein. Off-line admission control for general scheduling problems. *J. of Scheduling*, 3:365–381, 2000.

[21] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, 2008.

[22] U.M. Schwarz. Tightness results for malleable task scheduling algorithms. In *Parallel Processing and Applied Mathematics*, 2009.

[23] J. Turek, J.L. Wolf, and P. S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *4th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1992.

[24] N. Vasić, M. Barisits, V. Salzgeber, and D. Kostić. Making cluster applications energy-aware. In *1st workshop on Automated Control for Datacenters and Clouds (ACDC)*, 2009.

[25] P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 830–831, 2003.

[26] A.J. Wood and B. Wollenberg. *Power Generation Operation and Control.* Wiley, 2nd edition, 1996.

[27] Y. Zhang, X. Hu, and D.Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference (DAC)*, pages 183–188, 2002.