Minimizing Busy Time in Multiple Machine Real-time Scheduling

Rohit Khandekar[1], Baruch Schieber[1], Hadas Shachnai[2], and Tami Tamir[3]

1 **IBM T.J. Watson Research Center**
`{rohitk,sbar}@us.ibm.com`
2 **Computer Science Department, Technion**
`hadas@cs.technion.ac.il`
3 **School of Computer Science, The Interdisciplinary Center**
`tami@idc.ac.il`

Abstract

We consider the following fundamental scheduling problem. The input consists of n jobs to be scheduled on a set of machines of bounded capacities. Each job is associated with a release time, a due date, a processing time and demand for machine capacity. The goal is to schedule all of the 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 has important applications in power-aware scheduling, optical network design and unit commitment in power systems. Scheduling to minimize busy times is APX-hard already in the special case where all jobs have the same (unit) processing times and can be scheduled in a fixed time interval.

Our main result is a 5-approximation algorithm for general instances. We extend this result to obtain an algorithm with the same approximation ratio for the problem of scheduling *moldable* jobs, that requires also to determine, 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.

Keywords and phrases real-time scheduling, busy time, preemption, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.169

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 fundamental problem in cluster computing, which aims at reducing power consumption (see, e.g., [22] and the references therein).

Given is a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ that need to be scheduled on a set of identical machines, each of which 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* $1 \leq R(J) \leq g$ for machine capacity; this is the amount of capacity required for processing J on any machine.

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

© Rohit Khandekar, Baruch Schieber, Hadas Shachnai and Tami Tamir;
licensed under Creative Commons License NC-ND

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. Also, it is NP-hard to approximate our problem within ratio better than $\frac{3}{2}$, already in the special case where all jobs have the same (unit) processing times and can be scheduled in a fixed time interval, by a simple reduction from the subset sum problem.[1]

1.1 Applications

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. Communication in an optical network is achieved by *lightpaths*, which are simple paths in the network. Hardware cost for operating such a network is proportional to the number of switching units such as Optical Add-Drop Multiplexers (or OADMs) installed at the nodes in the network. A lightpath j with transmission rate $R(j)$ uses that capacity in an OADM at each internal node on the path. Assuming that the OADMs have transmission capacity g, one would like to "groom" multiple lightpaths together so that their aggregate transmission rate is at most g. It is easy to see that this grooming problem [11, 10, 9] for minimizing switching costs in optical networks with *path-topologies* can be reduced to our real-time scheduling problem.

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., [24, 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 [7, 4]). In particular, much attention was given to *interval scheduling* [14], where jobs are given

[1] Given the integers $a_1, \ldots, a_n \in \{1, \ldots, g\}$ summing to $2g$, the subset sum problem (SSP) is to determine if there is a subset of numbers adding to exactly g. In the reduction, we create for each i, a job J_i with demand a_i, release time 0, processing time 1 and deadline 1. The *yes* instance of SSP results in the busy time of 2 while the *no* instance results in the busy time of 3.

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 [13]).

There is 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. 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, 18, 5, 6]). 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., [21, 16] and a comprehensive survey in [20]). Scheduling moldable jobs differs from *malleable* jobs in which the amount of resources allotted to jobs may change over their execution [15].

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, 19].) Our scheduling problem differs from batch scheduling in several aspects. While 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. 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. Other work on energy minimization consider utilization of machines with variable capacities, corresponding to their voltage consumption [17], and scheduling of jobs with precedence constraints [12, 25].

The complexity of our scheduling problem was studied in [23]. This paper shows that the problem is NP-hard already for $g = 2$, where the jobs are intervals on the line. Flammini et al. [9] consider our scheduling problem where jobs are given as intervals on the line with unit demand. For this version, they give a 4-approximation algorithm for general inputs and better bounds for some subclasses of inputs. In particular, the paper 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.[2] A 2-approximation algorithm was given in [9] for instances where any two intervals intersect, i.e., the input forms a clique (see also in [10]). In this paper we improve and extend the results of [9].

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

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

- *proper* if for any two jobs $J, J' \in \mathcal{J}$, 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 $J, J' \in \mathcal{J}$, we have $I(J) \cap I(J') = \emptyset$ or $I(J) \subseteq I(J')$, or $I(J') \subseteq I(J)$.
- a *clique* if intervals $I(J)$ for all jobs J form a clique, i.e., for any two jobs $J, J' \in \mathcal{J}$, we have $I(J) \cap I(J') \neq \emptyset$.

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

▶ **Theorem 2.** *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.

▶ **Theorem 3.** *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 $J \in \mathcal{J}$, 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 assume that q is polynomially bounded in the input size.

▶ **Theorem 4.** *There exists a* 5*-approximation algorithm for real-time scheduling of* moldable *jobs.*

Finally, we present improved bounds for some cases of instances with interval jobs.

▶ **Theorem 5.** *Consider an instance $\mathcal{J}$ consisting of interval jobs with* unit *demands. There exist* (*i*) *a polynomial time exact algorithm if $\mathcal{J}$ is laminar, and* (*ii*) *a PTAS if $\mathcal{J}$ is a clique.*

1.4 Preliminaries

▶ **Definition 6.** 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 $\mathcal{J}$ and machine capacity $g \geq 1$, we denote by $\text{OPT}(\mathcal{J})$ 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 $\text{OPT}_\infty(\mathcal{J})$ the cost of the optimum solution for the instance $\mathcal{J}$, 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 $\mathcal{J}$, $w(\mathcal{J}) = \sum_{J \in \mathcal{J}} w(J)$ is the total work required by the jobs in $\mathcal{J}$. The next observation gives two immediate lower bounds for the cost of any solution.

▶ **Observation 7.** *For any instance $\mathcal{J}$ and machine capacity $g \geq 1$, the following bounds hold.*

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

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 $\mathcal{J}_i$ 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., $\texttt{busy}(M_i) = \texttt{span}(\mathcal{J}_i)$ for all $i \geq 1$.[3]

2 Interval Scheduling: Theorem 2

2.1 General Instances with Interval Jobs

In this section we present an algorithm for instances with interval jobs, where each job $J \in \mathcal{J}$ may have an arbitrary processing time and any demand $1 \leq R(J) \leq g$. Algorithm *First_Fit_with_Demands* ($FF_\mathcal{D}$), shown in the frame below, 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 8.** For a subset $\mathcal{J}' \subseteq \mathcal{J}$ of jobs, let $\text{NARROW}(\mathcal{J}') = \{J \in \mathcal{J}' \mid R(J) \leq \alpha \cdot g\}$ and $\text{WIDE}(\mathcal{J}') = \{J \in \mathcal{J}' \mid R(J) > \alpha \cdot g\}$.

```
Algorithm ($FF_\mathcal{D}$):
1. Schedule jobs in WIDE($\mathcal{J}$) arbitrarily on some machines. Do not use these machines
   for scheduling any other jobs.
2. Sort the jobs in NARROW($\mathcal{J}$) by non-increasing lengths, i.e., $p(J_1) \geq \ldots \geq p(J_{n'})$.
3. For $j = 1, \ldots, n'$ do:
   a. Let $m$ denote the number of machines used for 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.
```

Let $FF_\mathcal{D}(\mathcal{J})$ denote the total busy time of the schedule computed by $FF_\mathcal{D}$ on job $\mathcal{J}$.

▶ **Theorem 9.** *If $\alpha = 1/4$, for any instance $\mathcal{J}$ with interval jobs, we have*

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

[3] By $\texttt{span}(\mathcal{J}_i)$ we refer to the span of the set of intervals representing the jobs, as scheduled on M_i.

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

▶ **Lemma 10.** *The cost incurred by jobs in* WIDE$(\mathcal{J})$ *is at most* $\sum_{J \in \text{WIDE}(\mathcal{J})} p(J) \leq \frac{w(\text{WIDE}(\mathcal{J}))}{\alpha \cdot g}$.

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 11.** *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, $R(J) + \sum_{J' \in s_{i,k}(J)} R(J') > g$.*

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

▶ **Lemma 12.** *For any $1 \leq i \leq m-1$, we have $g \cdot \texttt{span}(\mathcal{J}_{i+1}) \leq \frac{3 \cdot w(\mathcal{J}_i)}{1-\alpha}$.*

Proof. Following Observation 11, for a job $J \in \mathcal{J}_i$, denote by $b(J)$ the set of jobs in $\mathcal{J}_{i+1}$ which blame J, i.e., $b(J) = \{J' \in \mathcal{J}_{i+1} \mid J \in s_{i+1,i}(J')\}$. Let J_L (resp. J_R) be the job with earliest start time (resp. latest completion time) in $b(J)$. Since each job in $b(J)$ intersects J, we have $\texttt{span}(b(J)) \leq p(J) + p(J_L) + p(J_R) \leq 3 \cdot p(J) = 3 \cdot \frac{w(J)}{R(J)}$. Thus,

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

Now, we observe that

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

We bound the right-hand-side as follows. For any $t \in \texttt{span}(\mathcal{J}_{i+1})$, there exists a job $J' \in \mathcal{J}_{i+1}$ with $t \in [r(J'), d(J'))$. For all jobs $J \in s_{i+1,i}(J')$, since $J' \in b(J)$, we have $t \in \texttt{span}(b(J))$. Hence, $\sum_{J \in \mathcal{J}_i : t \in \texttt{span}(b(J))} R(J) \geq \sum_{J \in s_{i+1,i}(J')} R(J)$. By Observation 11 $\sum_{J \in s_{i+1,i}(J')} R(J) > g - R(J') \geq (1-\alpha) \cdot g$. We thus conclude

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

From (1) and (2) we get the lemma. ◀

Proof of Theorem 9: The overall cost of the schedule computed by $FF_{\mathcal{D}}$ is the contribution of WIDE jobs and NARROW jobs. Note that the busy time of M_i, for $1 \leq i \leq m$ is exactly $\texttt{busy}(M_i) = \texttt{span}(\mathcal{J}_i)$. Now from Lemmas 10 and 12, we have that the total cost of $FF_{\mathcal{D}}$ is at most

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

The second inequality follows since $\mathtt{span}(\mathcal{J}_1) \leq \textsc{opt}_\infty(\mathcal{J})$. The last inequality holds since $\alpha = 1/4$. The proof now follows from Observation 7.

2.2 Proper Instances with Interval Jobs

We now 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:
1. Sort the jobs in non-decreasing order of release times, i.e., $r(J_1) \leq \ldots \leq r(J_n)$.
2. 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
   current filled.
```

▶ **Theorem 13.** *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. The proofs of the following lemmas are omitted due to lack of space.

▶ **Lemma 14.** *For any t, we have $D_t > g \left\lfloor \frac{M_t^A - 1}{2} \right\rfloor$.*

▶ **Lemma 15.** *For any t, we have $M_t^O \geq M_t^A/2$.*

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

3 Real-time Scheduling: Theorem 3

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

▶ **Lemma 16.** *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, $\mathcal{J}$, with cost at most $\beta \cdot \textsc{opt}_\infty(\mathcal{J}) + 4 \cdot \frac{w(\mathcal{J})}{g}$, thus yielding a $(\beta + 4)$-approximation.*

Proof. We first compute a schedule, called S_∞, with busy-time at most $\beta \cdot \textsc{opt}_\infty(\mathcal{J})$, 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 $\mathcal{J}'$ obtained from $\mathcal{J}$ by replacing $r(J)$ and $d(J)$ with $t_\infty(J)$ and $t_\infty(J) + p(J)$, respectively, for each job J. Note that $\textsc{opt}_\infty(\mathcal{J}') \leq \beta \cdot \textsc{opt}_\infty(\mathcal{J}) \leq \beta \cdot \textsc{opt}(\mathcal{J})$. We then run algorithm $FF_{\mathcal{D}}$ on instance $\mathcal{J}'$. Theorem 9 implies that the resulting solution has busy-time at most $\textsc{opt}_\infty(\mathcal{J}') + 4 \cdot \frac{w(\mathcal{J}')}{g} \leq \beta \cdot \textsc{opt}_\infty(\mathcal{J}) + 4 \cdot \frac{w(\mathcal{J})}{g} \leq (\beta + 4) \cdot \textsc{opt}(\mathcal{J})$ as claimed. ◀

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

▶ **Theorem 17.** *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 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.

▶ **Lemma 18.** *W.l.o.g., we can 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.*

Motivated by Lemma 18, we consider the following definition.

▶ **Definition 19.** A time t is called *interesting* if $t = r(J)$ or $d(J) - p(J)$ for some job J, or $t = r(J) + p(J)$ or $d(J) - p(J) + p(J')$ 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 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 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 20.** 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 $\mathcal{J}$ whose processing time is at most ℓ and whose placement must intersect the interval $[t_1, t_2)$, i.e.,

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

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 21.** *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,*

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

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 $J \in \mathcal{J}$ and their corresponding schedules can be computed, using the relation in Lemma 21, in polynomial time. We finally output the schedule corresponding to $\texttt{cost}[0, T, \max_{J \in \mathcal{J}} p(J)]$. By definition, this gives the optimum solution.

4 Real-time Scheduling for Moldable Jobs: Theorem 4

A job J in an instance $\mathcal{J}$ 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 $p_i(J) \le d(J) - r(J)$ for all $1 \le i \le q$. The goal is to pick a configuration $1 \le i_J \le q$ 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 $\vec{i} = \{i_J\}_{J \in \mathcal{J}}$, let $\mathcal{J}(\vec{i})$ denote the instance of real-time scheduling problem derived from $\mathcal{J}$ by fixing configuration i_J for each job J. Let $\text{OPT}(\mathcal{J})$ denote the cost of the optimum solution, and let $\vec{i}^* = \{i^*_J\}_{J \in \mathcal{J}}$ denote the configurations used in the optimum schedule. From Observation 7, we have

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

In this section, we prove the following main lemma.

▶ **Lemma 22.** *Given an instance $\mathcal{J}$ of the real-time scheduling with moldable jobs, we can find in polynomial time configurations $\vec{i} = \{i_J\}_{J \in \mathcal{J}}$, such that $\text{OPT}_\infty(\mathcal{J}(\vec{i})) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}))}{g}$ is minimized.*

Proof. Motivated by Lemma 18 and Definition 19, we define the set of interesting times as follows.

▶ **Definition 23.** A time t is called *interesting* if $t = r(J)$ or $d(J) - p_i(J)$ for some job J and configuration i, or $t = r(J) + p_i(J)$ or $d(J) - p_i(J) + p_{i'}(J')$ 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 $J \in \mathcal{J}$ and a configuration $1 \le i_J \le q$, let $w_{i_J}(J) = p_{i_J}(J) \cdot R_{i_J}(J)$.

▶ **Definition 24.** Let $t_1, t_2 \in \mathcal{T}$ with $t_2 > t_1$ and $\ell = p_i(J)$ for some job J. Let

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

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".

▶ **Lemma 25.** *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, we have

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

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 $J \in \mathcal{J}, 1 \le i \le q$ and their corresponding job-configurations and schedules can be computed, using the relation in Lemma 25, in polynomial time. The algorithm finally outputs the job-configurations corresponding to $\texttt{ub}[0, T, \max_{J \in \mathcal{J}, 1 \le i \le q} p_i(J)]$. By definition, this proves Lemma 22. ◀

Now, recall that Lemma 16 and Theorem 17 together imply that given an instance $\mathcal{J}(\vec{i})$ of the real-time scheduling problem, we can compute in polynomial time a feasible schedule with busy-time at most $\text{OPT}_\infty(\mathcal{J}(\vec{i})) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}))}{g}$. Thus, equation (4), Lemma 22, Lemma 16, and Theorem 17 together imply that we can find a schedule with cost at most

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

thus yielding a 5-approximation.

5 Interval Scheduling with Unit Demands: Theorem 5

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 in case the job intervals $I(J)$, for all jobs J, form a laminar family, i.e., for any two jobs $J, J' \in \mathcal{J}$, 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 26.** *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. ◀

5.2 A PTAS and more for Cliques

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 $g = 2$ [23]. We show that for inputs that form cliques, the problem with $g = 2$ is solvable in polynomial time.

Since the instance $\mathcal{J}$ forms a clique, there is a time t_0 such that $t_0 \in I(J)$ for all $J \in \mathcal{J}$. 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. Note that the original intervals are sub-intervals of the stretched ones, therefore, any feasible schedule of the stretched intervals induces a feasible schedule.

Approximation Scheme for a Clique:

1. Let $c > 1$ be a constant.
2. Let t_0 be such that $t_0 \in J$ for all $J \in \mathcal{J}$. 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 $\ell \in \{1, \dots, c\}$, the class ℓ consists of all jobs for which $\texttt{sh}(J)/\texttt{lo}(J) = \ell/c$ and $\texttt{left}(J) \ge \texttt{right}(J)$. For $\ell \in \{c+1, \dots, 2c-1\}$, the class ℓ 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 $C_i(n'_1, \dots, n'_{2c-1})$ 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 $C_m(n_1, \dots, n_{2c-1})$.

We can show the next result – the proof is omitted due to lack of space.

▶ **Theorem 27.** *For any $\varepsilon \in (0, 1]$, the scheme with $c = 1/\varepsilon$ is a PTAS for any clique.*

The case $g = 2$: In this case we can solve the problem optimally, using a reduction to the minimum-weight perfect matching in a complete graph. We first ensure that the number of jobs is even by adding a dummy job with an empty interval. We next 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 $\texttt{span}(J_i \cup J_j)$. Use Edmond's algorithm [8] to find a minimum-weight perfect matching in the graph. It is easy to see that this matching gives the optimum solution to our problem.

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. Improved approximation algorithms for resource allocation. In *9th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2002.

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

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

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

9 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. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2009.

10 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.

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

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

13 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.

14 E. Lawler, J.K. Lenstra, A.H.G.R. 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.

15 J. Leung, L. Kelly, and J. H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press, Inc., Boca Raton, FL, USA, 2004.

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

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

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

19 M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, 2008.

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

21 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.

22 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.

23 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.

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

25 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.