LP Rounding and Combinatorial Algorithms for Minimizing Active and Busy Time *

Jessica Chang, Samir Khuller, and Koyel Mukherjee

University of Maryland, College Park
{jschang,samir,koyelm}@cs.umd.edu

Abstract. We consider fundamental scheduling problems motivated by energy issues. In this framework, we are given a set of jobs, each with release time, deadline and required processing length. The jobs need to be scheduled so that at most g jobs can be active on a machine at any given time. The duration for which a machine is active (i.e., "on") is referred to as its *active time*. The goal is to find a feasible schedule for all jobs, minimizing the total active time. When preemption is allowed at integer time points, we show that a minimal feasible schedule already yields a 3-approximation (and this bound is tight) and we further improve this to a 2-approximation via LP rounding. Our second contribution is for the non-preemptive version of this problem. However, since even asking if a feasible schedule on one machine exists is NP-hard, we allow for an unbounded number of virtual machines, each having capacity of g. This problem is known as the *busy time* problem in the literature and a 4-approximation is known for this problem. We develop a new combinatorial algorithm that gives a 3-approximation. Furthermore, we consider the preemptive busy time problem, giving a simple and exact greedy algorithm when unbounded parallelism is allowed, that is, where g is unbounded. For arbitrary g, this yields an algorithm that is 2-approximate.

1 Introduction

Scheduling jobs on multiple parallel or batch machines has received extensive attention in the computer science and operations research communities for decades. For the most part, these studies have focused primarily on "job-related" metrics such as minimizing makespan, total completion time, flow time, tardiness and maximizing throughput under various deadline constraints. Despite this rich history, some of the most environmentally (not to mention, financially) costly scheduling problems are those driven by a pressing need to reduce energy consumption and power costs, e.g. at data centers. In general, this need is not addressed by the traditional scheduling objectives. Toward that end, our work is most concerned with minimization of the total time that a machine is on [2, 5, 9, 12] to schedule a collection of jobs. This measure was recently introduced in an effort to understand energy-related problems in cloud computing contexts, and the busy/active time models cleanly capture many central issues in this space. Furthermore, it has connections to several key problems in optical network design, perhaps most notably in the minimization of the fiber costs of Optical Add Drop Multiplexers (OADMs) [5]. The application of busy time models to optical network design has been extensively outlined in the literature [5, 6, 8, 14].

With the widespread adoption of data centers and cloud computing, recent progress in virtualization has facilitated the consolidation of multiple virtual machines (VMs) into fewer hosts. As a consequence, many computers can be shut off, resulting in substantial power savings. Today, products such as Citrix XenServer an VMware Distributed Resource Scheduler (DRS) offer VM consolidation as a feature. In this sense, minimizing busy time (described next) is closely related to the basic problem of mapping VMs to physical hosts.

We first discuss the simple active time model [2]. In this model we have a collection $\mathcal{J}$ of n jobs $J_1, \ldots J_n$ that need to be scheduled on one machine. Each job J_j has release time r_j, deadline d_j and length p_j. We assume that time is slotted and that all job parameters are integral. The jobs need to be scheduled on a machine so that at most g jobs are running simultaneously. The goal is to minimize the *active time* of the machine, which is the duration for which the machine is on. If we are looking for a non-preemptive schedule, we can easily show that this problem is strongly NP-hard (even the feasibility question becomes NP-hard). In the special case that the jobs all have unit length, there is a fast algorithm [2] that yields an optimal solution. If we allow pre-emption at integer boundaries, the feasibility question is easy, but the exact complexity of minimizing active time is still open. In this work, we first show that taking

* This work has been supported by NSF Grants CCF-1217890 and CCF-0937865.

any minimal solution already gives a 3 approximation to this problem. We next show that this bound is tight. We then further improve the approximation ratio by showing that by considering a natural IP formulation and its relaxation to an LP, we can round the solution to obtain a solution that is within twice the integer optimum (again this bound is also tight). We conjecture that the problem itself is likely to be NP-hard.

We next consider a slight variant of the active time problem, called the *busy time* problem, which has been considered in the literature [5, 9] and is defined as follows. The main variation from the active time problem is that an unbounded number of virtual machines is available, and we would like a non-preemptive schedule. We are given a collection $\mathcal{J}$ of n jobs $J_1, \ldots J_n$ that need to be scheduled on a set of identical machines. Each job J_j has release time r_j, deadline d_j and length p_j. The jobs need to be partitioned into groups (each group of jobs will be scheduled non-preemptively on a machine) so that at most g jobs are running simultaneously on a given machine. We say that a machine is *busy* at time t if there is at least one job running on the machine at t; otherwise the machine is *idle*. The time intervals during which a machine M is busy is called its *busy time* and we denote its length by *busy(M)*. The objective is to find a feasible schedule of all the jobs on the machines (partitioning jobs into groups) to minimize the cumulative busy time over all the machines. We will call this the **busy time problem**. The schedule can potentially use an unbounded number of machines since each group is really a virtual machine.

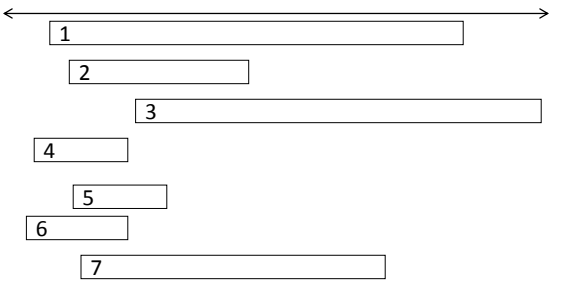
A well-studied special case of this model is one in which each job J_j is "rigid", i.e. $d_j = p_j + r_j$. Since each job's deadline is exactly its release time plus its processing time, there is no question about when it must start. Jobs of this particular form are called *interval jobs*. The busy time problem is NP-hard [14] even when $g = 2$ even for interval jobs. We say the interval $[r_j, d_j)$ is the *span* of job J_j. The *span* of a job set $\mathcal{J}'$ is the union of the spans of jobs in $\mathcal{J}'$. Since the problem is NP-hard we will be interested in approximation algorithms for this problem. What makes this special case particularly central is that one can convert an instance of the general busy time problem to an instance of interval jobs in polynomial time, by solving a dynamic program with unbounded g [9]. The dynamic program "fixes" the positions of the jobs to minimize their shadow (projection on the time-axis). The span of this solution with $g = \infty$ is the smallest possible span of any solution to the original problem and can be used as a lower bound on the optimal solution. Then, one can adjust the release times and deadlines to artificially "fix" the position of each job to where it was scheduled in the solution for unbounded g. This creates an instance of interval jobs, on which we can then apply an approximation algorithm for the case of interval jobs. Figure 1 shows a collection of jobs and the corresponding packing that yields an optimal solution, i.e., minimizing busy time.

Busy time scheduling in this form was first studied by Flammini et al. [5]. They present a very simple greedy algorithm FIRSTFIT for interval jobs and demonstrate that it always produces a solution of busy time at most 4 times that of the optimal solution. The algorithm considers jobs in non-increasing order by length, greedily packing each job in the first group in which it fits. In the same paper, they highlight an instance on which the cost of FIRSTFIT is three times that of the optimal solution. Closing this gap would be very interesting [1].

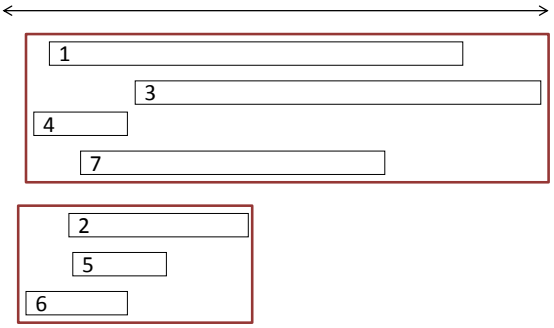
However, unknown to Flammini et al, earlier work by Alicherry and Bhatia [1] and Kumar and Rudra [11] already considered a problem in the context of wavelength assignment. Their algorithms immediately yield two different factor 2 approximations for the the problem of minimizing the busy time for scheduling interval jobs.

Khandekar et al. [9] consider the generalization in which each job has an associated width or "demand" on its machine. For any set of jobs assigned to the same machine, the cumulative demand of the active ones can be at most g at any time. The authors apply FIRSTFIT ideas to this problem and obtain a 5-approximation. The main idea involves partitioning jobs into those of "narrow" and "wide" demand. Each wide job is assigned to its own machine, while First Fit is applied to the set of narrow jobs. In addition, the authors give improved bounds for special cases of busy-time scheduling with jobs of unit demand. When the interval jobs form a clique, they provide a PTAS. They give a simple exact algorithm when the intervals of the jobs are laminar, i.e. two jobs' intervals intersect only if one interval is contained in the other. However, we note that for the case of unit width jobs the same approach gives a 4

[1] In an attempt to improve approximation guarantees, Flammini et al. [5] consider two special cases. The first case pertains to "proper intervals", where no job's interval is strictly contained in that of another. For instances of this type, they show that the greedy algorithm ordering jobs by release times is actually 2-approximate. The second special case involve instances whose corresponding interval graph is a clique - in other words, there exists a time t such that each interval $[r_j, d_j)$ contains it. In this case, a greedy algorithm also yields a 2-approximation. As with proper intervals, it is not obvious that minimizing busy time on clique instances is NP-hard. However, when the interval jobs are both proper *and* form a clique, a very simple dynamic program gives an optimal solution [12].



(A)



(B)

Fig. 1: (A) Collection of interval jobs with unit demand, numbered arbitrarily. (B) Optimal packing of the jobs on two machines with $g = 3$ minimizing total busy time.

approximation for flexible jobs, by solving a dynamic program for $g = \infty$. With little effort one can show that using the same approach the methods of Kumar and Rudra [11] and Alicherry and Bhatia [1] also yield a 4 approximation, and the bounds are tight. We develop a new improved algorithm with a bound of 3. Our approach is completely different from prior approaches.

1.1 Problem Definition

In this section we formally define the notions of *active time* and *busy time*. Both models are motivated by the total amount of time that a machine is actively working.

Active Time The input consists of a set of jobs $\mathcal{J}$, where each job j has a release time r_j, a deadline d_j, and a length p_j. This means that p_j units of job j must be scheduled within time slots $[r_j, d_j)$, which we sometimes refer to as the *window* of j. We have access to a single machine, which is either active ('on') at any instant or not. The machine can process only g jobs at any time instant. Since there is a single machine, we simply refer to the time axis henceforth in place of the machine. We consider time to be slotted, and hence the release times and deadlines of jobs are integral. Consequently, the jobs are integral in length, and additionally, we allow preemption at integer boundaries. In other words, we consider each job j of length p_j to be a *chain* of p_j unit jobs, with identical windows $[r_j, d_j)$ and the restriction that in any time slot, at most one of these unit jobs can be scheduled. Hence, in this model, $\sum_{j \in \mathcal{J}} p_j$ is polynomial. Let us denote by T the length of the time window, spanning the union of the windows of the entire job instance. In other words, $T = |\bigcup_{j \in \mathcal{J}} [r_j, d_j)|$. We assume without loss of generality that the earliest release time of any job $j \in \mathcal{J}$ is 0 and the latest deadline of any job in $j \in \mathcal{J}$ is T. Sometimes for ease of notation, we will be using t to denote any slot $[t-1, t)$. In this notation, let $\mathcal{T}$ denote the set of time slots $[1, \ldots, T]$.

When preemption is not allowed, determining whether there exists a feasible solution for non-unit length jobs becomes strongly NP-hard, by a reduction from 3-PARTITION, even for the special case when the windows of all the jobs are identical.

Busy Time Given that determining even the feasibility for the non-preemptive problem is hard in the active time model, we consider a relaxation of the model, which has been studied in literature as the busy-time problem. The key difference between busy time and active time is while active time assumes access to a single machine, busy time can open up an unbounded number of machines if necessary. The input for both models is the same: a set of jobs, $\mathcal{J}$, where each job j has a release time r_j, a deadline d_j, and a length p_j, and the machine capacity g. Note that every instance is feasible in the busy time model.

1.2 Our Results

For the active time problem when we are allowed pre-emption at integer time points and time is slotted, we first show that considering any minimal solution gives us a 3 approximation. We then consider a natural IP formulation for this problem and show that considering an LP relaxation allows us to convert a fractional schedule to an integral schedule by paying a factor of 2. As a by product, this yields a 2 approximation. We note that the integrality gap of 2 is tight [2].

Earlier work [2] was only able to address this problem when job lengths were unit, and for that case an optimal polynomial time algorithm was provided. Unfortunately, it does not appear at all easy to extend that framework for the case of non-unit length jobs.

Since the busy time problem for interval jobs is NP-hard [14], the focus in this paper is the development of a polynomial-time algorithm GREEDYTRACKING with a worst case approximation guarantee of 3, improving the previous bounds of 4 (as mentioned earlier, there seem to be several different routes to arrive at this bound). As before we use the dynamic program to first solve the problem for $g = \infty$ [**?**] and then reduce the problem to the case of interval jobs. The central idea is to iteratively identify a set of jobs whose spans are disjoint; we will reference this set as a "track". Then, the jobs assigned a particular machine is simply the union of g such tracks; we call the set of jobs assigned to the same machine a *bundle* of jobs. The *busy time* of that machine is the span of its bundle. The goal is to assign jobs to bundles so that at no time does a single bundle have more than g active jobs, and to do so in a way that minimizes the cumulative busy time. Intuitively, this approach is less myopic than FIRSTFIT, which schedules

jobs one at a time. We also give instances where GREEDYTRACKING yields a solution twice that of the optimum for interval jobs.

One important consequence of GREEDYTRACKING is an improved bound for the busy time problem on flexible jobs. Similar to Khandekar et al. [9], we first solve the problem assuming unbounded machine capacity to get a solution that minimizes the projection of the jobs onto the time-axis. Then, we can map the original instance to one of interval jobs, forcing each job to be done exactly as it was in the unbounded capacity solution. We prove that in total, this approach has busy time within thrice that of the optimal solution. In addition, we explore the preemptive version of the problem and provide a greedy 2 approximation.

1.3 Related Work

While in both busy time and active time models, we assign jobs to batch machines, the key difference between busy time and active time is that the former model assumes access to an unbounded number of machines, while the latter operates on a single machine. When jobs are unit in length, Chang et al. [2] present a fast linear time greedy algorithm. When the release times and deadlines can be real numbers, they give an $O(n^7)$ dynamic program to solve it; this result has since been improved to an $O(n^3)$-time algorithm in the work of Koehler and Khuller [10]. Chang et al. [2] also consider generalizations to the case where jobs can be scheduled in a union of time intervals (in contrast to the usual single release time and deadline). Under this generalization, once the capacity constraints exceeds two, minimizing active time becomes NP-hard via a reduction from 3-EXACT-COVER.

Mertzios et al. [12] consider a dual problem to busy time minimization, the *resource allocation maximization version*, where the goal is to maximize the number of jobs scheduled without violating a budget constraint given in terms of busy time and the parallelism constraint. They show that the maximization version is NP-hard whenever the (busy time) minimization problem is NP-hard. They give a 6 approximation algorithm for clique instances and a polynomial time algorithm for proper clique instances for the maximization problem.

The online version of both the busy time minimization and resource allocation maximization was considered by Shalom et al. [13]. They prove a lower bound of g where g is the parallelism parameter, for any deterministic algorithm for general instances and give an $O(g)$ competitive algorithm. Then they consider special cases, and show a lower bound of 2 and an upper bound of $(1 + \phi)$ for a one-sided clique instances (which is a special case of laminar cliques), where ϕ is the golden ratio. They also show that the bounds increase by a factor of 2 for clique instances. For the maximization version of the problem with parallelism g and busy time budget T, they show that any deterministic algorithm cannot be more than gT competitive. They give a 4.5 competitive algorithm for one-sided clique instances.

Flammini et al. [7] consider the problem of optimizing the cost of regenerators that need to be placed on light paths in optical networks, after every d nodes, to regenerate the signal. They show that the 4 approximation algorithm for minimizing busy time [5] solves this problem for a path topology and $d = 1$ and extend it to ring and tree topologies for general d.

Faigle et al. [4] consider the online problem of maximizing "busy time" but their objective function is totally different from ours. Their setting consists of a single machine and no parallelism. Their objective is to maximize the total length of intervals scheduled as they arrive online, such that at a given time, at most one interval job has been scheduled on the machine. They give a randomized online algorithm for this problem.

2 Active time scheduling of preemptive jobs

Definition 1. *A job j is said to be* live *at t if $t \in [r_j, d_j)$*

Definition 2. *A slot is* active *if at least one job is scheduled in it. It is* inactive *otherwise.*

Definition 3. *An active slot is* full *if there are g jobs assigned to it. It is* non-full *otherwise.*

A feasible solution σ is specified by a set of active time slots $\mathcal{A} \subseteq \mathcal{T}$, and a mapping or assignment of jobs to time slots in $\mathcal{A}$, such that at most g jobs are scheduled in any slot in $\mathcal{A}$, at most one unit of any job j is scheduled in any time slot in $\mathcal{A}$ and every job j has been assigned to p_j active slots within its window $[r_j, d_j)$. Once the set $\mathcal{A}$ of active slots has been determined, a feasible integral assignment can be found via a max-flow computation.

The cost of a feasible solution σ is the number of active slots in the solution, denoted by $|\mathcal{A}|$. Let $\mathcal{A}_f$ denote the set of active slots that are full, and let $\mathcal{A}_n$ denote the set of active slots which are non-full. Therefore, $|\mathcal{A}| = |\mathcal{A}_f| + |\mathcal{A}_n|$.

Definition 4. *A minimal feasible solution is one in which no active slot can be made inactive, and still feasibly satisfy the entire job set.*

Given a feasible solution, one can easily find a minimal feasible solution.

Definition 5. *A non-full-rigid job is one which is scheduled for one unit in every non-full slot where it is live.*

Lemma 1. *For any minimal feasible solution σ, there exists another solution σ' of same cost, where every active slot that is non-full, has at least one non-full-rigid job scheduled in it.*

Proof. Consider any non-full slot in a minimal feasible solution σ, which does not have any non-full-rigid job scheduled in it. Move any job in that slot to any other (non-full, active) slot that it may be scheduled in, and where it is not already scheduled. There must at least one such slot, otherwise this would be a non-full-rigid job. Continue this process for as long as possible. Note that in moving these jobs, we are not increasing the cost of the solution, as we are only moving jobs to already active slots. If we can do this till there are no jobs scheduled in this slot, then we would have found a smaller cost solution, violating our assumption of minimal feasibility. Otherwise, there must be at least one job left in that slot, which cannot be moved to any other active slots. This can only happen if all the slots in the window of this job are either full, or inactive, or non-full where one unit of this job has been scheduled, thus making this a non-full rigid job.

Continue this process till in all the non-full slots, there is at least one non-full-rigid job scheduled.

Corollary 1. *There exists a set of jobs $\mathcal{J}^*$ consisting of non-full-rigid jobs, such that at least one of these jobs is scheduled in every non-full slot.*

We say that such a set $\mathcal{J}^*$ covers all the non-full slots.

Lemma 2. *There exists a set $\mathcal{J}^*$ of non-full-rigid jobs covering all the non-full slots, such that no job window is completely contained within the window of another job.*

Proof. Let us consider a set $\mathcal{J}^*$ of non-full-rigid jobs that are covering all the non-full slots. Suppose it contains a pair of non-full-rigid jobs j and j', such that the $[r_j, d_j) \subseteq [r_{j'}, d_{j'})$. One unit of j' must be scheduled in every non-full slot in the window of j'. However, this also includes the non-full slots in the window of j, hence we can discard j from $\mathcal{J}^*$ without any loss.

We repeat this with every pair of non-full-rigid jobs in $\mathcal{J}^*$, such that the window of one is contained within the window of another, till there exists no such pair.

Let us call such a set $\mathcal{J}^*$ of non-full-rigid jobs whose windows are not contained within each other, and which covers all the non-full slots, as a minimal set $\mathcal{J}^*$.

Now, we prove that there exists a minimal set $\mathcal{J}^*$ such that at every time slot, at most two of the jobs in the set $\mathcal{J}^*$ are live. We will be charging the cost of the non-full slots to the set $\mathcal{J}^*$. The full slots can obviously be charged to the mass bound, which is a lower bound on the optimal solution.

Lemma 3. *There exists a minimal set $\mathcal{J}^*$ of non-full-rigid jobs such that at least one of these jobs is scheduled in every non-full slot, and at every time slot, at most two of the jobs in set $\mathcal{J}^*$ are live.*

Proof. Consider the first time slot t where 3 or more jobs of $\mathcal{J}^*$ are live. Let these jobs be numbered according to their deadlines ($j_1, j_2, j_3, \ldots j_\ell,\ \ell \geq 3$). By definition, the deadline of all of these jobs must be $\geq t$ since they are all live at t. Moreover, they are all non-full-rigid, being a part of $\mathcal{J}^*$, which means they are scheduled one unit in every non-full active slot in their window. Since the set $\mathcal{J}^*$ is minimal, no job window is contained within another, hence none of the jobs $j_2, \ldots, j_\ell$ have release time earlier than that of j_1. Therefore, all non-full slots before the deadline of j_1 must be charging either j_1 or some other job with an earlier release time. Consequently, discarding any of the jobs $j_2, \ldots, j_\ell$ will not affect the charging of these slots.

Let t' be the first non-full active slot after the deadline of j_1. t' therefore needs to charge one of $j_2, j_3, \ldots, j_\ell$. Among these, all jobs which have a deadline earlier than t', can be discarded from $\mathcal{J}^*$, without any loss, since no non-full slot needs to charge it. Hence, let us assume that all of these jobs $j_2, j_3, \ldots, j_\ell$ are live at t'. However, all of them being non-full-rigid, and t' being non-full and active, all of them must have one unit scheduled in t'. Therefore, if we discard all of the jobs $j_2, \ldots, j_{\ell-1}$ and keep j_ℓ alone, that would be enough since it can be charged all the non-full slots between t' and its deadline d_ℓ. Hence, after discarding these intermediate jobs from $\mathcal{J}^*$, there would be only two jobs j_1 and j_ℓ left which overlap at t.

Repeat this for the next slot t'' where 3 or more jobs of $\mathcal{J}^*$ are live, till there are no such time slots left.

The cost of the non-full slots of the minimal feasible solution σ' is $|\mathcal{A}_n| \leq \sum_{j \in \mathcal{J}^*} p_j$.

Theorem 1. *The cost of any minimal feasible solution is at most* 3 *times that of an optimal solution.*

Proof. It follows from Lemma 3 that $\mathcal{J}^*$ can be partitioned into two job sets $\mathcal{J}_1$ and $\mathcal{J}_2$, such that the jobs in each set have windows disjoint from one another. Therefore the sum of the processing times of the jobs in each such partition is a lower bound on the cost of any optimal solution. Let us denote the cost of the optimal solution as OPT. Hence, the cost of of the non-full slots is $|\mathcal{A}_n| \leq \sum_{j \in \mathcal{J}^*} p_j \leq \sum_{j \in \mathcal{J}_1} p_j + \sum_{j' \in \mathcal{J}_2} p_{j'} \leq 2OPT$. Furthermore, the full slots charge once to OPT, since they have a mass of g scheduled in them, which is a lower bound on OPT. $|\mathcal{A}_f| \leq \frac{\sum_{j \in \mathcal{J}} p_j}{g} \leq OPT$. Therefore, in total the cost of any minimal feasible solution $cost(\sigma) = cost(\sigma') = |\mathcal{A}| = |\mathcal{A}_f| + |\mathcal{A}_n| \leq 3OPT$. This proves the theorem.

The above bound is asymptotically tight as proved by the following example of a minimal feasible solution (see Figure 2)

There are two jobs each of length g, one has a window $[0, 2g)$ and the other one has a window $[g, 3g)$. There are $g-2$ rigid jobs, each of length $g-2$, with windows: $[g+1, 2g-1)$. There are $g-2$ unit jobs with window $[g+1, 2g)$ and another $g-2$ unit jobs with window $[g, 2g-1)$. An optimal solution schedules the two longest jobs from $[g, 2g)$, and one set of $g-2$ unit jobs on time slot g, and the other set of unit jobs on time slot $2g-1$. The total cost of the solution is g. However, a minimal feasible solution may schedule the two sets of $g-2$ unit jobs in the window $[g+1, 2g-1)$, with the rigid jobs of length $g-2$. Now, the two longest jobs cannot fit anywhere in the window $[g+1, 2g-1)$, since these slots have become full slots. Hence, it has to pay the cost of the g length jobs additionally. So, one feasible way to pack all the jobs would be to pack one of the longest jobs from $[1, g+1)$ and the other one from $[2g-1, 3g-1)$. The total cost would therefore be $3g-2$, which tends to 3 times the optimal solution as $g \to \infty$.

3 A 2 approximation algorithm based on LP rounding

Here we use LP-rounding to give a 2 approximation for the active time problem on non-unit length jobs with preemption allowed at integral boundaries. In this section onwards, we will be using t to denote any slot $[t-1, t)$ for ease of notation. Let us first write an LP for the problem. Let y_t denote the indicator variable for every time slot $t \in \mathcal{T}$. Let $x_{t,j}$ denote the indicator variable for unit job $j \in \mathcal{J}$ and every $t \in \mathcal{T}$. The LP is as follows:

$$
\begin{aligned}
&\min \sum_{t \in \mathcal{T}} y_t \\
\text{s.t. } &x_{t,j} \leq y_t \; \forall t \in \mathcal{T}, j \in \mathcal{J} \\
&\sum_{j \in \mathcal{J}} x_{t,j} \leq g y_t \; \forall t \in \mathcal{T} \\
&\sum_{t \in \mathcal{T}} x_{t,j} \geq p_j \; \forall j \in \mathcal{J} \\
&0 \leq y_t \leq 1 \; \forall t \in \mathcal{T} \\
&x_{t,j} \geq 0 \; \forall t \in \mathcal{T}, j \in \mathcal{J} \\
&x_{t,j} = 0 \; \forall t \notin [r_j, \ldots, d_j]
\end{aligned}
$$

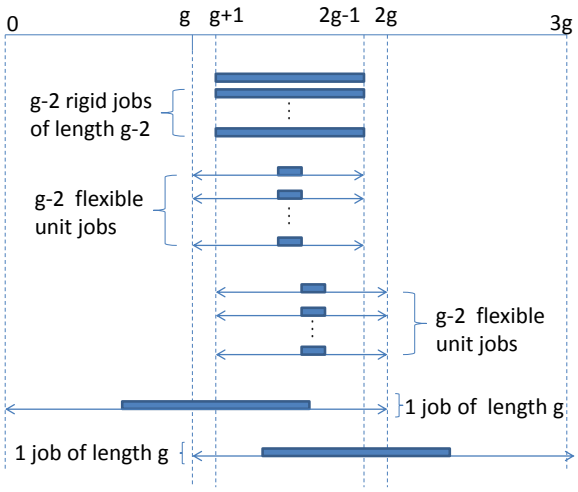


Fig. 2: Example of an instance where the minimal feasible solution is almost 3 times the optimal solution

We first solve the LP to optimality. Since any integral optimal solution is a feasible LP solution, the optimal LP solution is a lower bound on the cost of any optimal solution. Our goal is to round it to get a feasible integral solution within twice the cost of the optimal LP solution. However, before we do the rounding, we pre-process the optimal LP solution, to get a certain structure without increasing the cost of the solution. Then we solve another feasibility LP to get the fractional assignments of the jobs to this pre-processed LP solution. Note that this will not increase the cost of the solution, but will simply give a feasible fractional assignment of the jobs to the pre-processed optimal solution. Then we round this solution to get a feasible integral solution.

In the rounding, our goal will be to find a set of slots to open integrally, such that there exists a feasible fractional assignment for the jobs in the integrally open slots. An integral assignment can be found at the end of the procedure, when we have determined the integrally open slots, via a max-flow computation. We do not try to integrally assign jobs at any intermediate steps.

3.1 Pre-Processing

We sort the deadlines of the jobs to get the set of distinct deadlines in increasing order and then process the LP solution sequentially according to these deadlines. Let the set of distinct deadlines be $\mathcal{D} = [d_1, d_2, \ldots, d_\ell]$. We denote the set of jobs with deadline d_i as $\mathcal{J}_i$.

We define Y_1 as the sum of the y values for all time slots $\leq d_1$, that is, $Y_1 = \sum_{t \leq d_1} y_t$. Then we modify the optimal LP solution as follows. We open the slots $[d_1 - \lfloor Y_1 \rfloor + 1, \ldots, d_1]$ integrally and the slot $d_1 - \lfloor Y_1 \rfloor$ fractionally up to $Y_1 - \lfloor Y_1 \rfloor$, and close all the earlier slots.

Similarly, for the i^{th} deadline, define $Y_i = \sum_{d_{i-1} < t \leq d_i} y_t$. We want to pre-process the optimal solution to have the right-shifted structure, where for every deadline d_i, $\lfloor Y_i \rfloor$ slots are open integrally backwards from d_i, in other words, the slots $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ are fully open, and the slot $d_i - \lfloor Y_i \rfloor$ is open up to $Y_i - \lfloor Y_i \rfloor$.

By definition, the LP solution can be written as: $\sum_{t \in \mathcal{T}} y_t = \sum_{i \in [1,\ldots,\ell]} Y_i$.

For every job $j \in \mathcal{J}$, let us define $a_{j,1} = \sum_{t \leq d_1} x_{t,j}$, where $a_{j,1}$ the total assignment any job j is getting from slots $\leq d_1$ in the optimal LP solution. Similarly, for any deadline d_i, $i > 1$, we define $a_{j,i} = \sum_{d_{i-1} < t \leq d_i} x_{t,j}$, where $a_{j,i}$ is the total assignment any job j is getting from the slots $[d_{i-1}+1, \ldots, d_i]$.

The following lemma proves that there exists an optimal feasible LP solution in the modified instance up to d_1.

Lemma a: There exists an assignment of the jobs in the modified LP solution up to d_1 such that every job j in $\mathcal{J}$ can be accommodated up to $a_{j,1}$.

Proof. This is a proof by construction. Let t be the latest slot $\leq d_1$ for which $y_t > 0$ in the optimal LP solution. Note that, without loss of generality, we can make $y_{d_1} = y_t$ and $y_t = 0$, and move all the job assignments intact from t to d_1. This is because, all jobs which are feasible at t are also feasible at d_1 by definition of d_1. Now, let t' be the latest slot $\leq d_1$ for which $y_{t'} > 0$. If $y_{d_1} + y_{t'} \leq 1$, we merge $y_{t'}$ with y_{d_1}, in other words, set $y_{d_1} = y_{d_1} + y_{t'}$ and transfer all the job assignments from t' to d_1, without violating feasibility or increasing the cost. Otherwise, we set $y_{d_1-1} = y_{d_1} + y_{t'} - 1$ and $y_{d_1} = 1$. However, we need to maintain the constraint $x_{t,j} \leq y_t$ to get a feasible LP solution. Suppose $J_{t'}$ is the set of jobs that were originally assigned to the t'. Set $\delta = g \cdot y_{d_1-1}$ initially. As long as there exists some job in $j \in J_{t'}$, we assign j up to $x_{d_1-1,j} = \min(x_{t',j}, y_{d_1-1}, \delta)$ to the slot $d_1 - 1$, and decrement both δ and $x_{t',j}$ by $x_{d_1-1,j}$. If $x_{t',j}$ becomes 0, then we remove that job from $J_{t'}$. Once δ goes to 0 on adding some job to slot $d_1 - 1$, we assign all the remaining jobs (portions) in $J_{t'}$ to d_1, without violating feasibility, since 1) $y_{d_1} = 1$, 2) we have assigned up to $g \cdot y_{d_1-1}$ in $d_1 - 1$, there is space in d_1 up to g, and we know $y_{d_1} + y_{t'} = 1 + y_{d_1-1}$, hence there is enough space in d_1 to accommodate the remaining jobs. If δ is not 0, but $J_{t'}$ is not empty, then every job $j \in J_{t'}$ have been assigned either up to $x_{t',j}$ to $d_1 - 1$, or up to y_{d_1-1}, and the remaining jobs (portions) are assigned to d_1. This is also feasible because of the following reasons. The space available in d_1 before merging with t' was $g \cdot y_{d_1}$. Let the space occupied by jobs with $x_{t',j} \leq y_{t'}$ be δ' in $d_1 - 1$. The jobs with $x_{t',j} > y_{d_1-1}$ must have all been assigned up to y_{d_1-1}, since δ is still not 0. Therefore, the number of such jobs is $< \frac{g \cdot y_{d_1-1} - \delta'}{y_{d_1-1}}$. These jobs could have been assigned to at most $y_{t'}$ in t' before merging. Hence, the remaining portion of these jobs is $< \frac{g \cdot y_{d_1-1} - \delta'}{y_{d_1-1}} \cdot (y_{t'} - y_{d_1-1})$. The space available for accommodating them in d_1 is $(y_{t'} - y_{d_1-1}) \cdot g$. However, $\frac{g \cdot y_{d_1-1} - \delta'}{y_{d_1-1}} \cdot (y_{t'} - y_{d_1-1}) \leq (y_{t'} - y_{d_1-1}) \cdot g$, therefore, the remaining portions of the jobs in $J_{t'}$ can be accommodated in d_1, without violating LP feasibility. If δ does not go to 0, while there are no jobs left in $J_{t'}$, then we have feasibly assigned all the jobs in $y_{t'}$, respecting all the constraints.

Now, we repeat the above procedure with the latest slot $t'' < d_1 - 1$, merging $y_{t''}$ with y_{d_1-1}, if $y_{d_1-1} > 0$, otherwise with y_{d_1}. If there exists no such t'' we stop. At the end, we have a feasible LP solution where all jobs $j \in \mathcal{J}$ have been assigned up to $a_{j,1}$ in the right shifted structure.

Let us assume by induction hypothesis that the above holds for all deadlines d_k, where $k \leq i-1$. The next lemma proves that the property holds for the deadline d_i, assuming it holds for all earlier deadlines.

Lemma b: There exists an assignment of the jobs in the modified LP solution up to d_i ($i \geq 1$) such that every job j in $\mathcal{J}$ can be accommodated up to $\sum_{1 \leq k \leq i} a_{j,i}$.

Proof. We prove this by induction. The base case is proved in Lemma a. We assume by induction hypothesis, that the claim holds for all iterations $1 \leq k < i$ and now do the same construction as in Lemma a for proving the claim for iteration i. Now, we only consider slots $d_{i-1}+1, \ldots d_i$, starting with the closest slot t to d_i with $y_t > 0$ and repeat the merging and reassignment procedure as described in Lemma a. Again the key observation is that any job j which is feasible at some $t \leq d_i$, is also feasible in all slots $[t, t+1, \ldots, d_i]$.

Theorem 2. *There exists an optimal fractional LP solution that is right-shifted.*

Proof. As already noted, the LP solution can be written as the sum of the Y_i values, as $\sum_{t \in \mathcal{T}} y_t = \sum_{i \in [1,\ldots,\ell]} Y_i$. Therefore, the right shifted solution has the same cost as the optimal solution. The theorem follows from the Lemmas a and b by induction.

We can use the construction process highlighted in the Lemmas a and b to get a feasible fractional assignment for all the jobs.

Henceforth, we work with this feasible, right-shifted optimal LP solution.

3.2 Overview of Rounding

We process the deadlines one after another in at most ℓ iterations, where we process the d_i in iteration i. At the end of every iteration i, we have a set of integrally open slots $\mathcal{O}_i$. We maintain the invariant that at the end of the i^{th} iteration, the number of integrally open slots up to d_i is $|\mathcal{O}_i| \leq 2\sum_{j\leq i} Y_j$, and there exists a feasible fractional assignment of $\mathcal{J}_i$ in $\mathcal{O}_i$. This gives us the 2 approximation by the end of the ℓ^{th} iteration, when we have processed the last deadline.

We refer to a slot t with $y_t = 1$ as "fully open", that with $1 > y_t \geq \frac{1}{2}$ as "half-open", that with $0 < y_t < \frac{1}{2}$ as "barely open" and that with $y_t = 0$ as "closed".

Obviously, fully open slots do not charge anything extra to the LP solution. Half-open slots can be opened at a cost of at most 2, charging themselves. For opening a barely open slot, we need to charge it to an fully open slot. We say that a barely open slot is "dependent" on the fully open slot that it charges to. In this case, the y value of the barely open slot is not charged at all. We will sometimes allow two barely open slots on either side of a fully open slot to open up along with a fully open slot, when the total sum of the y values of the barely open slot and the fully open slot is $\geq \frac{3}{2}$. We will refer to such slots as a "trio". Note that here we *do* charge the y value of the barely open slot.

We will additionally maintain the invariant that at every iteration, every barely open slot that we have opened is either a dependent on a fully open slot, or is part of a trio, and every fully open slot has at most one dependent or it is part of at most one trio. Half open slots charge themselves. This will ensure that we have charged the LP solution at most twice.

Every time we open a barely open slot as a dependent, we make it a dependent on the earliest fully open slot that does not have a dependent or is not part of a trio.

Sometimes, in the rounding process we close a barely open slot $d_i - t$, ($t \geq 0$), while processing deadline d_i. However, if jobs of later deadline were assigned by the LP in this barely open slot, then we need to accommodate them. We make sure that when we close a slot, we are not charging its y value at all. Hence we create a proxy copy of the slot that we closed, and carry it over to the next iteration. The y value of this proxy slot is set to be the y value of the slot we have just closed, without any double counting. The proxy also carries a pointer to the actual slot that it is a proxy for. In any iteration i, when we have a proxy, we treat it as a regular fractionally open slot (though there may be no actual slot at that point). If this slot remains closed after the rounding, the proxy gets carried over to the next iteration, whereas if it does get opened by the rounding, the actual slot which it points to gets opened. However, now the cost of opening it will be accounted for by the current solution. This is outline in details in Section 3.4. There can be at most one proxy slot at any iteration.

3.3 Processing d_1

Claim. $Y_1 \geq 1$

Proof. This is obvious as otherwise the LP solution is not feasible: none of the chains with deadline d_1 or otherwise, could have been assigned one unit before the first deadline.

Slots $[d_1 - \lfloor Y_1 \rfloor + 1, \ldots, d_1]$ are fully open. In the following, we outline how we deal with the slot $d_1 - \lfloor Y_1 \rfloor$.

Case 1. $Y_1 - \lfloor Y_1 \rfloor \geq \frac{1}{2}$.

We open $d_1 - \lfloor Y_1 \rfloor$ as half open and charge it to itself.

Case 2. $Y_1 - \lfloor Y_1 \rfloor < \frac{1}{2}$.

We first try to close $d_1 - \lfloor Y_1 \rfloor$ and find a feasible assignment of $\mathcal{J}_1$ in $\lfloor Y_1 \rfloor$ slots fully opened, using max-flow. If successful, then we keep it closed, and move to the next deadline, after passing over a proxy slot with y value $Y_1 - \lfloor Y_1 \rfloor$ to iteration 2. The proxy slot has the same job assignments as in the right-shifted solution LP solution, except those in $\mathcal{J}_1$, which are removed from the proxy slot. If we do not find a feasible assignment for $\mathcal{J}_1$ by closing $d_1 - \lfloor Y_1 \rfloor$, we keep it barely opened and make it dependent on $d_1 - \lfloor Y_1 \rfloor + 1$. We are guaranteed to find a fully open slot on which to make it dependent since $Y_1 > 1$.

3.4 Processing deadline d_i, $i > 1$

Now, we proceed to the next deadline d_i.

Dealing with a proxy slot While processing a deadline d_i, suppose there is a proxy of value y_p carried over from iteration $(i-1)$. Note that we are working with a right shifted solution. and the slots $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ are fully open. Before we do any rounding in the iteration i, we do the following.

Case 3. $y_p + Y_i - \lfloor Y_i \rfloor \leq 1$.

In this case, we merge the proxy with the slot $d_i - \lfloor Y_i \rfloor$. Specifically, we add y_p to $y_{d_i - \lfloor Y_i \rfloor}$, and transfer all the job assignments from the proxy slot to the slot $(d_i - \lfloor Y_i \rfloor)$ without violating the LP solution feasibility. (This maintains all the LP constraints). Now, we proceed with the rounding, treating $d_i - \lfloor Y_i \rfloor$ as a regular fractional slot in the right shifted solution (in other words, consider Y_i to be $Y_i + y_p$). Note that the jobs originally assigned to the proxy must have deadlines $\geq d_i$, since they were passed over from iteration $i-1$. Hence, the pointer to the proxy can now be safely changed to $d_i - \lfloor Y_i \rfloor$. If this gets opened, then no proxy is carried over. Otherwise, the new proxy carried over will be of value $y_p + Y_i - \lfloor Y_i \rfloor$ and the pointer will be to $d_i - \lfloor Y_i \rfloor$. If however, $d_{i-1} = d_i - \lfloor Y_i \rfloor$, then $\lfloor Y_i \rfloor = Y_i$, and in this case, we consider an imaginary slot $d_{i-1} = d_i - \lfloor Y_i \rfloor$ in between d_{i-1} and $d_i - \lfloor Y_i \rfloor + 1$, to which we assign y_p, consider Y_i to be $Y_i + y_p$ and process it for the time being as a regular fractional slot. If this remains closed, we pass over a proxy of value y_p with the original pointer, otherwise, we open up the actual slot to which the proxy points.

Case 4. $y_p + Y_i - \lfloor Y_i \rfloor > 1$.

In this case, let $y_p{}' = y_p + Y_i - \lfloor Y_i \rfloor - 1$. We make $d_i - \lfloor Y_i \rfloor$ fully open and create a new proxy with y value equal to $y_p{}'$. If $d_{i-1} < d_i - \lfloor Y_i \rfloor - 1$, we set y of the slot $d_i - \lfloor Y_i \rfloor - 1$ to $y_p{}'$, since all jobs in the proxy are feasible here, and treat it as a regular fractional slot. In other words, assume Y_i to be $Y_i + y_p$, and process it. If this fractional slot gets opened, then we are fine, otherwise, this new proxy of value $y_p{}'$ gets carried over to iteration $i+1$, with the pointer changed to $d_i - \lfloor Y_i \rfloor - 1$. Otherwise, $d_{i-1} = d_i - \lfloor Y_i \rfloor - 1$; in this case, consider an imaginary slot $d_{i-1} = d_i - \lfloor Y_i \rfloor - 1$ in between d_{i-1} and $d_i - \lfloor Y_i \rfloor$, to which we assign $y_p{}'$, consider Y_i to be $Y_i + y_p$ and process it for the time being as a regular fractional slot. If this imaginary slot remains closed, we pass over a proxy slot of value $y_p{}'$ with the original pointer, otherwise, we open up the actual slot to which the proxy points. For the case $y_p + Y_i - \lfloor Y_i \rfloor > 1$, since we create a new proxy $y_p{}'$, we need to specify how to change the assignments of the actual jobs, in order to create a new LP feasible solution. The procedure is essentially the same as in Lemma a. For completeness, it is outlined below.

We can assign a mass up to $g y_p{}'$ in the new proxy slot. Let $\delta = g \cdot y_p{}'$ initially. As long as there exists some job in $j \in J_p$, we assign j up to a $x_{p',j} = \min(x_{p,j}, y_p{}', \delta)$, where $x_{p,j}$ was the original assignment of j to y_p. We decrement both δ and $x_{p,j}$ by $x_{p',j}$. If $x_{p,j}$ becomes 0, then we remove that job from J_p. Once δ goes to 0 on adding some job to the proxy slot we assign all the remaining jobs (portions) in J_p to $d_i - \lfloor Y_i \rfloor$, without violating any feasibility. The slot $d_i - \lfloor Y_i \rfloor$ is now fully open, hence constraint $x_{t,j} \leq y_t$ won't be violated. Moreover, $y_p{}'$ being fully utilized, the remaining mass to be accommodated must be $\leq (y_p - y_p{}') \cdot g$, hence there is enough space in $d_i - \lfloor Y_i \rfloor$ to accommodate the remaining jobs. If δ is not 0, but J_p is not empty, then every job $j \in J_p$ have been assigned either up to $x_{p,j}$ to the proxy slot, or up to $y_p{}'$, and the remaining jobs (portions) are assigned to $d_i - \lfloor Y_i \rfloor$. This is feasible because of the following reasons. The space available in $d_i - \lfloor Y_i \rfloor - 1$ originally was $g y_p{}'$. Let the space occupied by jobs with $x_{p,j} \leq y_p{}'$ be δ'. The jobs with $x_{p,j} > y_p{}'$ must have all been assigned up to $y_p{}'$, since δ is still not 0. Therefore, the number of such jobs is $< \frac{g y_p{}' - \delta'}{y_p{}'}$. These jobs could have been assigned to at most y_p in the proxy slot originally. Hence, the remaining portion of these jobs is $< \frac{g y_p{}' - \delta'}{y_p{}'} \cdot (y_p - y_p{}')$. The space available for accommodating them in $d_i - \lfloor Y_i \rfloor$ is $(y_p - y_p{}') \cdot g$. However, $\frac{g y_p{}' - \delta'}{y_p{}'} \cdot (y_p - y_p{}') \leq (y_p - y_p{}') \cdot g$, therefore, the remaining portions of the jobs in J_p can be accommodated in $d_i - \lfloor Y_i \rfloor$, without violating LP feasibility. If δ does not go to 0, while there are no jobs left in J_p, then we have feasibly assigned all the jobs in y_p, respecting all the constraints.

Clearly by the above procedure, there can be at most one proxy in an iteration. Note that when we pass over a proxy from an iteration i, all job assignments of jobs in $\bigcup_{1 \leq k \leq i} \mathcal{J}_k$ are removed from the proxy (since they have already been accounted for), without violating any feasibility.

Processing Y_i In the following discussion, we assume Y_i already takes into account any proxy from iteration $i-1$ and the processing outlined above for a proxy slot has already been done.

Case 5. $1 > Y_i \geq \frac{1}{2}$.

We open d_i and charge it to itself as half-open.

Case 6. $Y_i > 1$ and $Y_i - \lfloor Y_i \rfloor \geq \frac{1}{2}$.

We open $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ as fully open and $d_i - \lfloor Y_i \rfloor$ as half open and charge it to itself.

Case 7. $Y_i > 1$ and $Y_i - \lfloor Y_i \rfloor < \frac{1}{2}$.

We open $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ as fully open. $d_i - \lfloor Y_i \rfloor$ is barely open. We first close $d_i - \lfloor Y_i \rfloor$, and try to find a feasible assignment of all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$, using max-flow. If successful, then we keep it closed and move on to the next deadline. If d_i is not the last deadline, we pass over a proxy for $d_i - \lfloor Y_i \rfloor$ to iteration $i+1$. This proxy has the y value of $Y_i - \lfloor Y_i \rfloor$, and all the job assignments from $(d_i - \lfloor Y_i \rfloor)$ except those in $\mathcal{J}_i$. Moreover, we have already adjusted for any proxy from iteration $i-1$, as outlined in the previous section.

Otherwise, we need to open $d_i - \lfloor Y_i \rfloor$, and have to account for its cost. We charge $d_i - \lfloor Y_i \rfloor$ as a dependent on the earliest fully open slot that has no dependents. If all the fully open slots $< d_i - \lfloor Y_i \rfloor$ have dependents, we charge it to $d_i - \lfloor Y_i \rfloor + 1$, which is fully open since $Y_i > 1$.

Case 8. $Y_i < \frac{1}{2}$.

We will first try to close d_i. Suppose the closest deadline open before d_i is d_k. Since d_i is barely open, all jobs in $\mathcal{J}_i$ must have release time $\leq d_k$, for a feasible LP solution. We try to find a feasible assignment of all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$, using max-flow in the open slots earlier than d_i that are already accounted for. If successful, then we keep d_i closed and move on to the next deadline. If d_i is not the last deadline, we create a proxy for d_i which we pass over to iteration $(i+1)$, assign it y value of Y_i, transfer all the job assignments from d_i to the proxy, except those of $\mathcal{J}_i$. If there was a proxy already in iteration i, carried over from the previous iteration, then note that this new proxy value already adjusts for the previous proxy value as per the rounding outlined in Section 3.4 and since all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$ are already accounted for, without loss of generality we change the pointer of the proxy to d_i.

Suppose closing d_i and finding a feasible assignment of jobs in $\bigcup_{j \leq i} \mathcal{J}_j$ was not successful. Let the earliest open deadline before d_i be d_k. If d_k was barely open, then flow would have certainly been successful, because all jobs feasible at d_i must be feasible at d_k, and both d_k and d_i being barely open with all intermediate deadlines closed in a feasible LP solution, there is space to accommodate all the jobs of $\bigcup_{k \leq x \leq i} \mathcal{J}_x$. Hence, d_k must be half open or fully open. If half-open, we first try to merge d_k and d_i. If $y_{d_k} + y_{d_i} \leq 1$, then flow would have found a feasible assignment. Therefore, $y_{d_k} + y_{d_i} > 1$. In this case, we make d_k fully open (now barely open slots can be made dependent on it) and set the new $Y_i = y_{d_k} + y_{d_i} - 1$. The job assignments are altered to make the resultant solution LP feasible in the same manner as outlined earlier for modifying assignments for proxy slot in Section 3.4. The new Y_i, which is still barely open, is then processed similar to the previous Y_i until we have either closed d_i locally, or opened it as part of a trio or as a dependent on a fully open slot. Once we have done either of that, we can move to d_{i+1}.

The final possibility is that d_k is fully open. We then try to charge d_i as a dependent on the earliest fully open slot that has no dependents. If successful, we move on to d_{i+1}. Otherwise, all of the fully open slots up to time slot d_k have dependents. We then try to charge d_i as a trio with the fully open d_k and its dependent. If we are successful, we move on to iteration $(i+1)$.

We will next argue that we will always be able to charge a barely open slot d_i that the rounding needs to open.

Lemma 4. *If we need to open a barely open slot d_i in an iteration, then we will always find a fully open slot to charge it as a dependent or as a trio.*

In order to prove the above, let us first assume that it is not true, and we are in the situation where we could not close d_i which is barely open, the closest open slot to d_i is fully open, and we could not charge d_i as a trio or as a dependent on any of the fully open earlier slots. That necessary means all the fully open slots before d_i must have dependents.

The following lemma argues that in case all the fully open slots before d_i have dependents, then the structure of the solution must be of the following form. There must be a deadline d_z, with the closest open slot $< d_z$ being d_w (there may be no d_w if d_z is the first fully open deadline), such that either 1) there are $\geq g+1$ jobs in $\mathcal{J}_z$ with release time later than d_w; or, 2) the sum of the number of rigid (unit) jobs with release time and d_z and the length 2 jobs with release time $\geq d_w$ is $\geq g+1$. Let us call such a d_z a *stopping deadline*. In other words, any integral solution would have to open at least one slot in $[d_w+1, \ldots, d_z-1]$, along with d_z, since there are $g+1$ job units to be scheduled between d_w+1 and d_z. Between stopping deadline d_z and d_i, the fully open slots will be a subset of the set of deadlines, and for each deadline d_x that is fully open, the slot d_x-1 is barely open and dependent on d_x. (Note that d_x-1 can be another deadline itself). Let us call this structure an *alternating* structure.

Lemma 5. *If the closest open slot to deadline d_i is fully open, and all the fully open slots before some deadline d_i have dependents, then without loss of generality, there is a stopping deadline d_z and between d_z and d_i, the structure of the solution must be alternating.*

Proof. Let d_k be the closest open slot to d_i, and d_k is fully open. Given the structure of the solution that we start out with and the rounding process, this must be a deadline. All the fully open slots before d_i have dependents. Since no slot between d_k and d_i are open, the dependent on d_k has to be some earlier slot. Either it is a barely open slot t such that $d_{k-1}+1 \leq t \leq d_k-1$. Or it must be a proxy slot carried over from an earlier deadline than d_k. Only one barely open slot is opened in any iteration by the rounding process. Therefore, if we open a proxy slot in an iteration, then there can be no other local barely open slots open in this iteration. Any barely open slot is dependent on the earliest fully open slot without a dependent. If $Y_k \geq 2$, d_k would not have got charged in iteration k, and hence would have no dependents even at iteration i. Therefore, $Y_k < 2$ and only d_k is fully open in the slots after d_{k-1}.

If d_k is not charged by a proxy slot, then necessarily d_k-1 is barely open. However, even if d_k is charged by a proxy slot, we show that any such proxy slot can be considered to be a local barely open slot without any loss of generality. Let us suppose that the dependent on d_k is a proxy slot. In that case, the alternating structure may not hold when we open the actual slot for the proxy. That means, the actual slot must be occurring in at some t', $d_{j-1} \leq t' \leq d_j$, where $j < k$. No barely open or half open slots could have opened between j and k as otherwise it would have accounted for the proxy slot. If there is a fully open slot between d_j (inclusive of d_j) and d_k then the proxy can charge this slot as it must have been uncharged so far. Since the proxy is a dependent on d_k, d_k must be the first fully open slot from iteration j onwards. Moreover, all the jobs in $\bigcup_{x<k} J_x$ do not need the proxy value for a feasible assignment. Hence, we can change the pointer of the proxy slot to d_k-1 without any loss of generality and consider d_k-1 as dependent on d_k. Note that d_k-1 may also be equal to d_j. Therefore, even if d_k is charged by proxy slot, we can convert it to a local barely open slot d_k-1.

Now, consider the rounding process in the iteration k. We must have first tried to close d_k-1 and find a feasible assignment using max flow. Clearly that must have failed. Also, no job in $\mathcal{J}$ with release time $> d_{k-1}$ can be of length > 1, because $Y_k < 2$. Therefore, one reason can be that there are $\geq g+1$ unit jobs in $\mathcal{J}_k$ with release time $\geq d_{k-1}$. In that case, d_k is the stopping deadline, and we have the alternating structure trivially.

If that is not the case, then that necessarily means the closest earlier open slot, say d_p,was half-open or fully open. The closest open slot cannot be barely open in a feasible LP solution, otherwise max flow would have been able to find a feasible assignment of the jobs in $\bigcup_{p \leq x \leq k} \mathcal{J}_k$ even after closing the barely open slot d_k-1. If half-open, then clearly, $y_{d_p} + y_{d_k-1} > 1$, otherwise, an assignment could be found by max-flow. However, in this case, the rounding would have made d_p fully open, and charged the new $y_{d_k-1} = y_{d_p} + y_{d_k-1} - 1$ as a dependent to it, if no other fully open slots were available for charging. Therefore, the only possibility is that d_p is fully open, and has a dependent already. Then the same argument can be be repeated for d_p and d_p-1. We repeat this argument for next closest open slot (which must be fully open with a dependent) till we come to a stopping deadline. We are guaranteed to find a stopping deadline because, if we ultimately come d_1, then that must also have a dependent d_1-1, (which means no jobs in $\mathcal{J}_1$ can be of length > 1) and we know from our rounding rule for d_1, that d_1-1 is opened only when max flow failed, which implies there are $\geq g+1$ unit jobs in $\mathcal{J}_1$. Hence, without loss we can convert our LP solution to the alternating form between d_i and the stopping deadline d_z.

Lemma 6. *Suppose d_z is the latest stopping deadline, in the alternating structure going backwards from d_i. Then for every intermediate fully open deadline $d_x \notin d_z, d_i$, at least $\geq 2g+1$ job units in $\mathcal{J}_u \cup \mathcal{J}_x$ must have release time $\geq d_u$, where d_u is the latest open deadline before d_x.*

Proof. We shall prove this by induction. Let the closest open slot before d_z be d_w. There are $\geq g+1$ job units in d_z which need to be scheduled in slots $[d_w+1,\ldots,d_z]$ due to release time constraints. This follows from the definition of a stopping deadline. Let the next fully open deadline after d_z in the alternating structure be d_a ($d_a > d_z$). Note that the total mass scheduled by the LP in $[d_z-1, d_z, d_a-1, d_a]$ is $\leq \frac{5g}{2}$, by the definition of the alternating structure. (The barely open slots could not form trio with each other). We want to prove that there are $2g+1$ job units in $\mathcal{J}_z \cup \mathcal{J}_a$ with release time $\geq d_z$. Let us assume there are $\leq 2g$ units of jobs in $\mathcal{J}_z \cup \mathcal{J}_a$ with release time $\geq d_z$ for contradiction. No job in $\mathcal{J}_a$ can be ≥ 2 in length for a feasible LP solution since $Y_a < 2$. Let n_z denote the rigid jobs in $\mathcal{J}_x$ (those releasing at d_z), n'_z denote the flexible jobs in $\mathcal{J}_z$ which need to be assigned before d_w (if there is any d_w), $n_{a,2}$ denote the number of length 2 jobs in $\mathcal{J}_a$, $n_{a,1}$ denote the unit length jobs in $\mathcal{J}_a$ with release time d_a and $n'_{a,1}$ denote the unit length jobs in $\mathcal{J}_a$ with release time $\geq d_z$. We know that $n_z + 2n_{a,2} + n_{a,1} + n'_{a,1} \leq 2g$ by assumption, $n_z + n'_z \geq g+1$ by definition of d_z, and since d_z is the latest stopping deadline, $n_{a,1} + n_{a,2} \leq g$. Since max flow failed, the only possibility is that $n_z + n_{a,2} \geq g+1$. For LP feasibility, $n_z + \frac{n'_z}{2} + \frac{n_{a,2}}{2} \leq g$. However, $n_z + n'_z + n_z + n_{a,2} > 2g$. Hence we get a contradiction. Therefore, there must be $\geq 2g+1$ job units $\mathcal{J}_z \cup \mathcal{J}_a$ with release time $\geq d_z$.

For ease of notation, without loss of generality, assume that the deadlines are consecutive. Therefore, all the deadlines $[d_z, d_{z+1}, \ldots, d_k]$ are fully open (here, $d_a = d_{z+1}$). Now, assume by induction hypothesis, that the claim is true for all deadlines up to d_k in the alternating structure, and the next fully open slot is d_{k+1}. For any deadline d_p which is fully open in the alternating structure between d_z and d_{k+1}, let us denote by $n_{p,1}$ the unit length rigid jobs in $\mathcal{J}_p$, $n_{p,2}$ the length 2 jobs of release time $\geq d_{p-1}$, and $n'_{p,1}$ the unit length jobs of release time $\geq d_{p-1}$ in $\mathcal{J}_p$. By induction hypothesis, for any two adjacent open deadlines d_{p-1} and d_p, where $p \geq 2$, in the alternating structure, there are $\geq 2g+1$ job units in $\mathcal{J}_{p-1} \cup \mathcal{J}_p$ with release time $\geq d_{p-1}$, i.e., $n_{(p-1),1} + n_{p,1} + n'_{p,1} + 2n_{p,2} \geq 2g+1$. As in the base case, assume for contradiction, that there are $\leq 2g$ job units in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ with release time $\geq d_k$. Therefore, $n_{k,1} + n_{(k+1),1} + n'_{(k+1),1} + 2n_{(k+1),2} \leq 2g$. Since the latest stopping deadline $d_z < d_{k+1}$, it also holds that $n_{(k+1),1} + n_{(k+1),2} \leq g$. Therefore, for max-flow to fail, it must be that $n_{k,1} + n_{k+1,2} \geq g+1$. For LP feasibility, the $\sum_{z\leq p\leq k} n_{p,1} + \sum_{z\leq p\leq k} \frac{n'_{p,1}}{2} + \sum_{z\leq p\leq k} \frac{3n'_{p,2}}{2} + \frac{n_{(k+1),2}}{2} \leq g(k-z+1)$. However, from the induction hypothesis, $2\sum_{z\leq p\leq k} n_{p,1} + \sum_{z\leq p\leq k} n'_{p,1} + 2\sum_{z\leq p\leq k} n'_{p,2} + n_{(k+1),2} > 2g(k-z+1)$. Hence this case is also not possible. Therefore, max-flow can fail only if there are $\geq 2g+1$ job units (including flexible, unit length and non-unit length) jobs in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ with release time $\geq d_k$.

Therefore, we have proved the claim by induction.

Proof of Lemma 4 continued:
When all the fully open slots before d_i have dependents, we try making d_i a trio with the closest fully open slot d_k and its dependent (which must necessarily be $d_k - 1$ according to the above lemmas). If we can, then we move to d_{i+1}. Suppose we cannot. Therefore, the total mass scheduled by the LP feasible solution in slots d_i, d_k and $d_k - 1$ is $< \frac{3g}{2}$, whereas opening d_k and $d_k - 1$ would give a space of $2g$. However, we cannot close d_i since flow did not find a feasible assignment. Therefore, there must be $\geq g+1$ jobs with release time d_k in $\mathcal{J}_i \cup \mathcal{J}_k$ and these must be unit length jobs for LP feasibility.

Similar to Lemma 6, we assume the deadlines are open in consecutive order. The alternating structure consists of $[d_z, d_{z+1}, \ldots, d_i]$. We assume the same notation for the jobs as in Lemma 6. For d_z, $n'_z + n_z \geq g+1$, where d_z is the stopping deadline, where n'_z denotes the number of length 2 jobs plus the flexible unit length jobs which must be scheduled before the closest earlier open deadline. For any $d_i > d_p >$, $n_{p,1}$ denotes the number of unit length rigid jobs with release time d_p, $n_{p,2}$ denotes the number of length 2 jobs with release time $\geq d_{p-1}$ and $n'_{p,1}$ denotes the number of flexible unit length jobs with deadline $\geq d_{p-1}$. By Lemma 6, for any $d_z < d_p < d_i$, $n_{p-1} + n_{p,1} + 2n_{p,2} + n'_{p,1} \geq 2g+1$. For d_i, we have argued above that $n_{i-1,1} + n'_{i,1} \geq g+1$, since for d_i, $n_{i,1} = n_{i,2} = 0$ for LP feasibility.

Adding the above, $\sum_{z\leq p\leq i} n'_{p,1} + 2\sum_{z\leq p\leq i} n_{p,1} + 2\sum_{z\leq p\leq i} n_{p,2} > 2(i-z)g$.

However, for LP feasibility, jobs can be assigned to an extent $< \frac{1}{2}$ in the barely open slots. Hence, $\sum_{z\leq p\leq i} \frac{n'_{p,1}}{2} + \sum_{z\leq p\leq i} \frac{3n_{p,2}}{2} + \sum_{z\leq p\leq i} n_{p,1} \leq (i-z)g$, which is a contradiction. Therefore we will always be able to charge a barely open slot which cannot be closed by max-flow assignment.

The proof of Lemma 4 follows from the above discussion.

Theorem 3. *There exists a polynomial time algorithm which gives a solution of cost at most twice that of any optimal solution to the active time problem on non-unit length jobs with integral preemption.*

Proof. From the above discussion, it follows, that at the end of every iteration i, we have a set of integrally open slots $\mathcal{O}_i$, such that there is a LP feasible fractional assignment of jobs in $\bigcup_{x \leq i} \mathcal{J}_x$ in $\mathcal{O}_i$. Furthermore, $|\mathcal{O}_i| \leq 2 \sum_{1 \leq k \leq i} Y_k$. We do this till the last deadline d_ℓ. At the end, we are assured of an integral feasible assignment on the set of opened slots via max-flow, while the number of open slots is at most twice the optimal LP objective function value. Hence, we get a 2 approximation.

3.5 LP Integrality gap

We show here that the natural LP for this problem has an integrality gap of 2. Hence, a 2 approximation is the best possible using LP rounding. Consider, g pairs of adjacent slots. In each pair, there are $g+1$ jobs which can only be assigned to that pair of slots. An integral optimal solution will have cost $2g$, where as LP optimal solution, will open each such pair up to 1 and $\frac{1}{g}$, assign all the $g+1$ jobs up to $\frac{g}{g+1}$ to the fully open slot, and up to $\frac{1}{g+1}$, to the barely open slot, thus maintaining all the constraints. Therefore, optimal LP solution has cost $g+1$ and $\frac{g+1}{2g} \to 2$ as $g \to \infty$.

4 Busy Time: Notations and Preliminaries

Definition 6. *A job j is said to be an* interval job *when $p_j = d_j - r_j$.*

A job J_j is *active* on machine m at some time $t \in [r_j, d_j)$ if J_j is one of the jobs being processed by machine m at time t.

Definition 7. *The length of a time interval $I = [a, b)$ is denoted $\ell(I) = b - a$, For a single contiguous interval, the length is the same as its span, and hence may be referred to interchangeably as the span of I, $|Sp(I)|$. For a set of intervals $\mathcal{I}$, the length of $\mathcal{I}$ is $\ell(\mathcal{I}) = \sum_{I \in \mathcal{I}} \ell(I)$. The span of $\mathcal{I}$ is $Sp(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} I$.*

For the special case of interval jobs, we need to find a partition of the jobs into groups or bundles, such that in every bundle, there are at most g jobs active at any time t. We then schedule each bundle on a single machine. Let $\mathcal{B}_\kappa$ be the set of interval jobs assigned to bundle κ by some partitioning scheme. Then, the busy time of the machine on which the bundle κ will be scheduled is given by $|Sp(\mathcal{B}_\kappa)|$. Suppose we have partitioned all jobs into k feasible bundles (the feasibility respects the parallelism bound g as well as the release times and deadlines). Then the total cost of the solution is given by the cumulative busy time $\sum_{\kappa=1}^{k} |Sp(\mathcal{B}_\kappa)|$. The objective is to minimize this cost. We consider both the variants where g is unbounded and where $g < \infty$. For the *preemptive* version of the problem, the problem definition remains the same, the only difference being that the jobs can be processed preemptively across various machines.

To minimize busy time in the general case, the difficulty lies not just in finding a partition of jobs, but also in deciding when each job j should be scheduled. We study both the preemptive and non-preemptive versions of this problem.

We denote the cost of the optimal solution of an instance $\mathcal{J}$ by $OPT(\mathcal{J})$. We denote by $OPT_\infty(\mathcal{J})$ the cost of the optimal solution for the instance $\mathcal{J}$ when unbounded parallelism is allowed.

Without loss of generality, the busy time of a machine is contiguous. If it is not, we can break it up into disjoint periods of contiguous busy time, assigning each of them to different machines, without increasing the total busy time of the solution.

The following lower bounds on any optimal solution for a given instance $\mathcal{J}$ were introduced earlier ([1], [11]).

Observation 1 $OPT(\mathcal{J}) \geq \frac{\ell(\mathcal{J})}{g}$*, where $g \geq 1$ and $\ell(\mathcal{J})$ denotes the sum of the processing lengths of the jobs in $\mathcal{J}$, interchangeably referred to as the mass of the set $\mathcal{J}$.*

This holds because in any machine, we can have at most g jobs active simultaneously.

Observation 2 $OPT(\mathcal{J}) \geq OPT_\infty(\mathcal{J})$*, where $OPT_\infty(\mathcal{J})$.*

The above observation follows from the fact that if a lower cost solution exists for bounded g, then it is a feasible solution for unbounded g as well. If the jobs in $\mathcal{J}$ are interval jobs, then, $OPT_\infty(\mathcal{J}) = |Sp(\mathcal{J})|$.

However, the above lower bounds, individually, can be arbitrarily bad. For example, consider an instance of g disjoint unit length interval jobs. The mass bound would simply give a lower bound of 1, whereas the optimal solution pays g. Similarly, consider an instance of g^2 identical unit length interval jobs. The span bound would give a lower bound of 1, whereas the optimal solution has to open up g machines for unit intervals, paying g.

We introduce a stronger lower bound, which we call the Demand Profile. In fact, the algorithm of Alicherry and Bhatia [1] as well as that of Kumar and Rudra [11] implicitly charge the demand profile. This lower bound holds for the case of interval jobs.

Definition 8. *Let $A(t)$ be the set of interval jobs that are active at time t. In other words, $A(t) = \{j : t \in [r_j, d_j)\}$. We say that $|A(t)|$ is the* raw demand *at time t, and define the* demand *at time t as $D(t) = \lceil \frac{|A(t)|}{g} \rceil$.*

Definition 9. *An interval within which no job begins or ends is called an* interesting *interval.*

By definition, the raw demand, and hence the demand, is uniform over an interesting interval because no job begins or ends within the interval. Let us represent the raw demand over an interesting interval I_i, as $A(I_i)$ and the demand as $D(I_i)$. Let $\mathcal{I}$ be the set of interesting intervals, $\mathcal{I} = I_1, I_2, \ldots, I_\ell$, where $\ell \leq 2n$, and $D(I_i) = D(t),\ \forall t \in I_i$. $\bigcup_{I_i \in \mathcal{I}} Sp(I_i) = Sp(\mathcal{J})$.

Definition 10. *We define the Demand Profile of an instance of interval jobs $\mathcal{J}$, $DeP(\mathcal{J})$ as the set of tuples $(I_i, D(I_i))$, where $I_i \in \mathcal{I}$.*

Note that the above definition expresses the Demand Profile in a linear number of tuples, even when the release times and deadlines of jobs, as well as their processing lengths are arbitrary (not polynomial).

We denote the cost of the demand profile $DeP(\mathcal{J})$ as $|DeP(\mathcal{J})|$. Specifically, $|DeP(\mathcal{J})| = \sum_{I_i \in \mathcal{I}} D(I_i)$.

Observation 3 *The demand profile of an instance is a lower bound on the cost of any feasible solution. Therefore $OPT(\mathcal{J}) \geq |DeP(\mathcal{J})|$.*

Proof. There are $|A(I_i)|$ jobs active within an interesting interval I_i. Then any feasible solution would have $\lceil \frac{|A(I_i)|}{g} \rceil$ machines busy during the interval I_i. Moreover, $Sp(\mathcal{J}) = Sp(\bigcup_{I_i \in \mathcal{I}} I_i)$. Hence, the proof follows.

4.1 2 approximation for Interval Jobs

In this section, we briefly outline how the work of Alicherry and Bhatia [1] and that of Kumar and Rudra [11] for related problems imply 2 approximation algorithms for the busy time problem for interval jobs, improving the best known factor of 4 [5]. A detailed description can be found in the Appendix of this full version.

Both the above mentioned works consider request routing problems on interval graphs, motivated by optical design systems. The requests need to use links on the graphs for being routed from source to destination, and the number of requests that can use a link is bounded. The polynomial time complexity of the algorithms crucially depend on the fact that the request (job) lengths are linear, ,which does not hold for the busy time problem for interval jobs with arbitrary release times, deadlines and processing lengths. However, the key observation is that even if the release times and deadlines of jobs are not integral, there can be at most $2n$ *interesting* intervals, such that no jobs begin or end within the interval. The demand profile is uniform over every interesting interval. Therefore, their algorithms can be applied to the busy time problem, with this simple modification, still maintaining the polynomial complexity. In order to bound the performance of their algorithms applied to the busy time problem, one would additionally need to assume that the demand everywhere is a multiple of g. However, note that for an arbitrary instance, we can add dummy jobs spanning any interesting interval I_i where the raw demand $|A(I_i)|$ is not a multiple of g without changing the demand profile. Specifically, if $cg < |A(I_i)| < (c+1)g$, for some $c \geq 0$, then $DeP(I_i) = c+1$, hence adding $(c+1)g - |A(I_i)|$ jobs spanning I_i does not change the demand profile. Hence, applying their algorithms to a suitably modified busy time instance of interval jobs, will cost at most twice the demand profile.

Theorem 4. *There exist 2 approximation polynomial time algorithms for the busy time problem on interval jobs. The approximation factor is tight.*

5 A 3 approximation algorithm for the non-preemptive general busy time problem

The general busy time problem was studied by Khandekar et al. [9], who refer to this problem as the real-time scheduling problem. They gave a 5 approximation for this problem when the interval jobs can have arbitrary widths. For the unit width interval job case, their analysis can be modified to give a 4 approximation. As a first step towards proving the 5-approximation for the general problem with flexible windows and non-unit width, Khandekar et al. [9] prove that if g is unbounded, then this problem is polynomial-time solvable. The output of their dynamic program essentially converts an instance of interval jobs with flexible windows to an instance of interval jobs (with rigid windows), by fixing the start and end times of every job.

Theorem 5. *[9] If g is unbounded, the real-time scheduling problem is polynomial-time solvable.*

From Theorem 10, the busy time of the output of the dynamic program on the set of flexible interval jobs $\mathcal{J}$ is equal to $OPT_\infty(\mathcal{J})$.

Once Khandekar et al. obtain the modified interval instance, they apply their 5 approximation algorithm for non-unit width interval jobs to get the final bound. However, for jobs with unit width, their algorithm and analysis can be modified without loss to apply the 4 approximation algorithm of Flammini et al. [5] for interval jobs with bounded g to get the final bound of 4. Moreover, extending the algorithms of Alicherry and Bhatia [1] and Kumar and Rudra [11] to the general busy time problem, by converting a general instance to an interval instance (similar to Khandekar et al. [9]) also gives a 4 approximation[2].

In this section, we give a 3 approximation algorithm for the general problem, improving the existing 4 approximation. Analogous to Khandekar et al. [9], we first convert this instance to an instance of interval jobs by running the dynamic program on $\mathcal{J}'$. Let $\mathcal{J}$ be the resultant set of interval jobs on fixing the job windows according to the output of the dynamic program, and let $OPT_\infty(\mathcal{J}')$ denote the cost or busy time of the output of the dynamic program. From Observation 2, we know that $OPT_\infty(\mathcal{J}') \leq OPT(\mathcal{J}')$.

On the interval job instance $\mathcal{J}$, we run our algorithm, which we call GREEDYTRACKING. Note that, now the span of the effective windows of every job becomes equal to its processing length. For an interval job j, we denote its window $[r_j, d_j)$ as the span of j: $Sp(j)$. For the rest of the section, we assume that our input consists of interval jobs.

Before we describe the algorithm, we define the notion of a *track*.

Definition 11. *A* track *of interval jobs is a set of interval jobs with disjoint spans.*

Given a feasible solution, one can think of each bundle $\mathcal{B}$ as the union of g individual tracks of jobs. The main idea behind the algorithm is to identify such tracks iteratively, bundling the first g tracks into a single bundle, the second g tracks into the second bundle, etc. FIRSTFIT [5] suffers from the fact that it greedily considers jobs one-by-one; GREEDYTRACKING is less myopic in that it identifies jobs whole tracks at a time.

In the i^{th} iteration, $i \geq 1$, the algorithm identifies a track $\mathcal{T}_i \subseteq \mathcal{J} \setminus \bigcup_{k=1}^{i-1} \mathcal{T}_k$ of maximum length $\ell(\mathcal{T}_i)$ and assigns it to bundle B_p, where $p = \lceil \frac{i}{g} \rceil$. One can find such a track efficiently via weighted interval scheduling algorithms [3]. We consider the lengths of the interval jobs as their weights, and find the maximum weight set of interval jobs with disjoint spans. If the final solution has κ bundles, the algorithm's total cost is $\sum_{i=1}^{\kappa} |Sp(\mathcal{B}_i)|$. The pseudocode for GREEDYTRACKING is provided in Algorithm 1.

Algorithm 1 GREEDYTRACKING. Inputs: $\mathcal{J}$, g.

1: $\mathcal{S} \leftarrow \mathcal{J}$, $i \leftarrow 1$.
2: **while** $\mathcal{S} \neq \emptyset$ **do**
3: Compute the longest track $\mathcal{T}_i$ from $\mathcal{S}$ and assign it to bundle $B_{\lceil \frac{i}{g} \rceil}$.
4: $\mathcal{S} \leftarrow \mathcal{S} \setminus \mathcal{T}_i$, $i \leftarrow i + 1$.
5: **end while**
6: Return bundles $\{\mathcal{B}_p\}_{p=1}^{\lceil \frac{i-1}{g} \rceil}$

[2] The bound of 4 for all these algorithms is tight, as shown in the Appendix of this full version.

We next prove a key property of GREEDYTRACKING: the span of any track is at least half that of the remaining unscheduled jobs. In particular, the span of any bundle is at most twice that of the first track to be assigned to it.

Lemma 7. *Let $\mathcal{T}_i$ be the ith track found by* GREEDYTRACKING*, $i \geq 1$. Let $\mathcal{J}'_i \subseteq \mathcal{J}$ denote the set of unscheduled jobs $\mathcal{J} \setminus \bigcup_{k=1}^{i-1} \mathcal{T}_k$. Then $|Sp(\mathcal{J}'_i)| \leq 2 \cdot |Sp(\mathcal{T}_i)|$.*

Proof. In order to prove this, we first prove the following. There exists two tracks $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$, such that $\mathcal{T}_1^* \subseteq \mathcal{J}'_i$ and $\mathcal{T}_2^* \subseteq \mathcal{J}'_i$, $\mathcal{T}_1^* \cap \mathcal{T}_2^* = \emptyset$ and $Sp(\mathcal{T}_1^*) \cup Sp(\mathcal{T}_2^*) = Sp(\mathcal{J}'_i)$. Let us assume, by way of contradiction, that the above is not true. In other words, for every pair of disjoint tracks from the set of yet unscheduled jobs $\mathcal{J}'_i$, the union of their spans does not cover $Sp(\mathcal{J}'_i)$.

Let $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$ be two disjoint tracks from $\mathcal{J}'_i$, such that the union of their spans is maximum among all such tracks. By our assumption, $|Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)| < |Sp(\mathcal{J}'_i)|$. This implies that there exists an interval $I \in Sp(\mathcal{J}'_i)$, such that $I \notin Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. Let I be $[t_I, t'_I)$.

Clearly, no job $j \in \mathcal{J}'_i$ has a window $\subseteq [t_I, t'_I)$, by the maximality of $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. In fact, all jobs intersecting I, must intersect with some job in both $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$, because of the same reason.

In the following we prove that *no* such interval I can exist given our assumptions on $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$.

Let us first define the notion of *minimum replaceable set*.

Definition 12. *Consider a track $\mathcal{T}$ and an interval job j with window $[r_j, d_j)$. Let $j_f \in \mathcal{T}$ have the earliest deadline $d_{j_f} > r_j$ such that $r_{j_f} \leq r_j$. Let $j_\ell \in \mathcal{T}$ have the latest release time $r_{j_\ell} < d_j$, such that $d_{j_\ell} \geq d_j$. Then the set of jobs in $\mathcal{T}$ with windows in $[r_{j_f}, d_{j_\ell})$ is the* minimum replaceable set *$MRS(j, \mathcal{T})$ for j in $\mathcal{T}$. In other words, it is the set of jobs whose union has the minimum span, such that $\{\mathcal{T} \cup j\} \setminus MRS(j, \mathcal{T})$ is a valid track. If there exists no such job j_f (respectively, j_ℓ), then $MRS(j, \mathcal{T})$ would consist of jobs in the interval $[r_j, d_{j_\ell})$ (respectively, $[r_{j_f}, d_j)$). If there exists no such job j_f as well as j_ℓ, then $MRS(j, \mathcal{T}) = \emptyset$. (See Figure 3 for an example).*

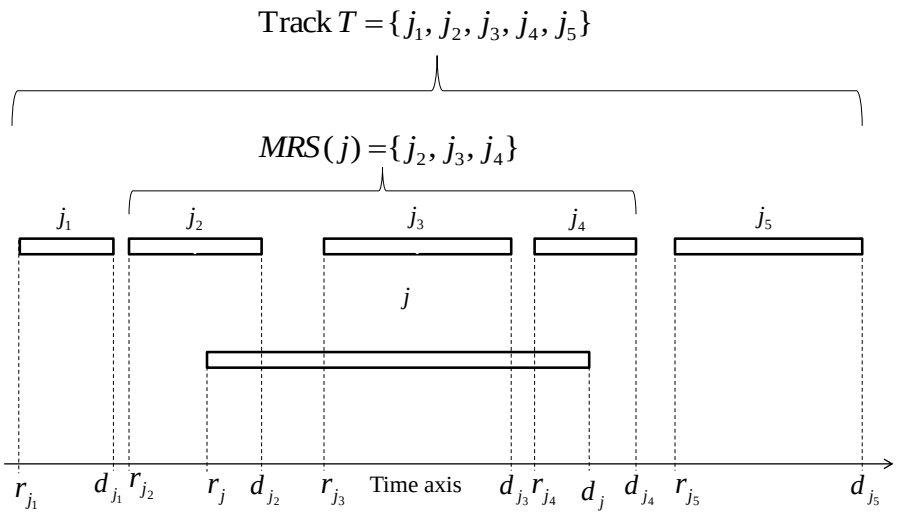


Fig. 3: An example showing the minimum replaceable set of a job j, i.e., $MRS(j)$ with respect to a track T.

Now we proceed with the proof.

Case 9. There exists a job j in $\mathcal{J}'_i \setminus \{\mathcal{T}_1^* \cup \mathcal{T}_2^*\}$, such that $r_j < t_I$ and $t_I < d_j < t'_I$.

Consider $MRS(j, \mathcal{T}_1^*)$ and $MRS(j, \mathcal{T}_2^*)$. Without loss of generality, they cannot be empty, as otherwise, by adding j to the corresponding track, we could have increased $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. Let j_e be the job with the earliest release time r_e in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$, and without loss of generality, suppose it belongs to $\mathcal{T}_1^*$. Replacing $MRS(j, \mathcal{T}_2^*)$ with j will increase $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. This is because, whereas $Sp(MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)) < [r_e, t_I)$, $Sp(j) \geq [d_e, t_I]$, and hence $Sp(MRS(j, \mathcal{T}_1^*) \cup j) > [r_e, t_I]$. See Figure 4 for an example.

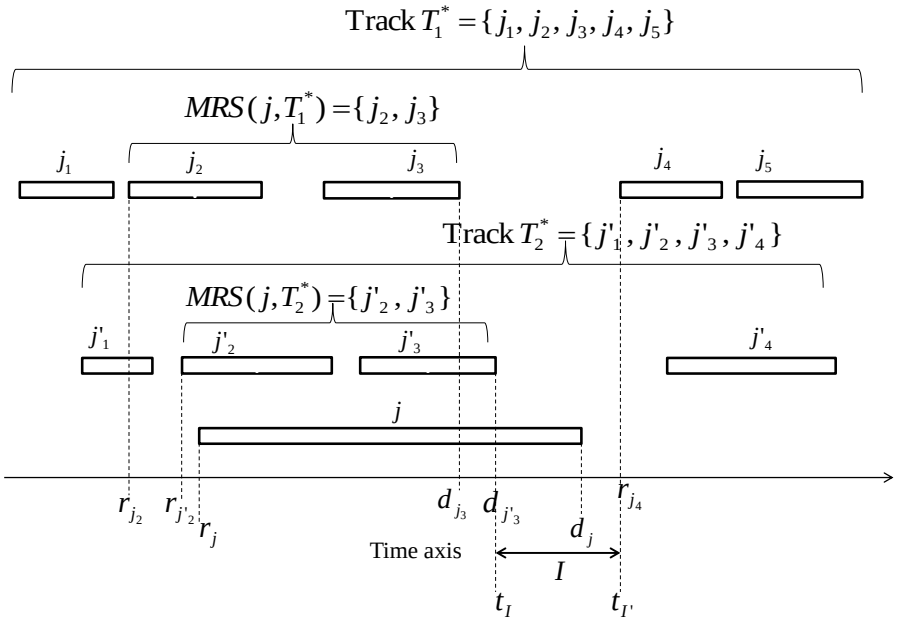


Fig. 4: An example for Case 9 of Lemma 7. Here replacing $MRS(j, \mathcal{T}_2^*)$ by j will increase the span of the union of the tracks: $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$.

Hence, this case is not possible.

Case 10. There exists a job j in $\mathcal{J}_i' \setminus \{\mathcal{T}_1^* \cup \mathcal{T}_2^*\}$, such that $t_I < r_j < t_I'$ and $d_j > t_I'$.

Consider $MRS(j, \mathcal{T}_1^*)$ and $MRS(j, \mathcal{T}_2^*)$. Without loss of generality, they cannot be empty sets as argued in Case 1. Let the job j_ℓ have the latest deadline d_ℓ in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$. Without loss of generality, suppose j_ℓ belongs to $\mathcal{T}_1^*$. Then we can replace $MRS(j, \mathcal{T}_2^*)$ with j, thereby increasing $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. This is because, $Sp(MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)) \leq [t_I', d_\ell)$, whereas $Sp(MRS(j, \mathcal{T}_1^*) \cup j) > [t_I', d_\ell)$, since $Sp(j) > [t_I', d_j)$.

Hence, this case is also not possible.

Case 11. There exists a job j, such that $[r_j, d_j) \supset [t_I, t_I')$.

Let the earliest release time of any job in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$ be r_e (the corresponding job is j_e) and the latest deadline of any job in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$ be d_ℓ (the corresponding job is j_ℓ). Once again we assume WLOG, that these sets are not empty. If both the jobs j_e and j_ℓ belong to the same track, say, $\mathcal{T}_1^*$, we can replace $MRS(j, \mathcal{T}_2^*)$ with j in $\mathcal{T}_2^*$ and increase the union of the span of $\mathcal{T}_1^* \cup \mathcal{T}_2^*$. This is because, $Sp(j) \geq [d_e, r_\ell)$ and includes $I = [t_I, t_I')$, whereas $Sp(MRS(j, \mathcal{T}_2^*) \setminus MRS(j, \mathcal{T}_1^*))$ is at most $[d_e, r_\ell) \setminus [t_I, t_I')$. Therefore, j_e and j_ℓ must belong to different tracks.

Without loss of generality, let $j_e \in \mathcal{T}_1^*$ and $j_\ell \in \mathcal{T}_2^*$. Let us replace $MRS(j, \mathcal{T}_2^*)$ with j. Next, we put j_ℓ in $\mathcal{T}_1^*$ replacing $MRS(j_\ell, \mathcal{T}_1^*)$. Note that $d_e \leq t_I$, $r_\ell \geq t_I'$, and $t_I' - t_I > 0$ by our assumptions. Therefore, $j_e \notin$

$MRS(j_\ell, \mathcal{T}_1^*)$. In fact, none of the jobs in $\mathcal{T}_1^*$ with release time $< t'_I$ are included in $MRS(j_\ell, \mathcal{T}_1^*)$, and hence none of them are discarded. Therefore, the loss of coverage by $\mathcal{T}_1^*$ after putting j_ℓ in place of $MRS(j_\ell, \mathcal{T}_1^*)$ is at most the interval $[t'_I, r_\ell)$. However, we have added j to $\mathcal{T}_2^*$, and not only does j span $[t_I, t'_I)$, but also the interval $[t'_I, r_\ell]$, since $d_j \geq r_\ell$ for j_ℓ to be originally a part of $MRS(j, \mathcal{T}_2^*)$. Hence, we would increase $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$, which is a contradiction. Therefore, this case is also not possible,

Since no job window in $\mathcal{J}'_i$ can intersect I, there exists no such I in $Sp(\mathcal{J}'_i)$. Therefore, $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*) = Sp(\mathcal{J}'_i)$. Furthermore, $|Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)| \leq |Sp(\mathcal{T}_1^*)| + |Sp(\mathcal{T}_2^*)|$, in other words, the longer of $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$ is $\geq \frac{|Sp(\mathcal{J}'_i)|}{2}$. Since, $\mathcal{T}_i$ is the longest track in $\mathcal{J}'_i$, therefore, $|Sp(\mathcal{J}'_i)| \leq 2|Sp(\mathcal{T}_i)|$.

We next prove that our algorithm generates a solution within 3 times the cost of an optimal solution via the following lemmas.

Lemma 8. *For any $i > 1$, the span of bundle $\mathcal{B}_i$ can be bounded by the mass of the bundle $\mathcal{B}_{i-1}$ as follows: $|Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{B}_{i-1})}{g}$.*

Proof. Let $\mathcal{T}_i^1$ denote the first track of the bundle $\mathcal{B}_i$. From Lemma 7, it follows that $|Sp(\mathcal{B}_i)| \leq 2|Sp(\mathcal{T}_i^1)|$. The jobs in $\mathcal{T}_i^1$ are disjoint by definition of a track, hence $|Sp(\mathcal{T}_i^1)| = \ell(\mathcal{T}_i^1)$, and $|Sp(\mathcal{B}_i)| \leq 2\ell(\mathcal{T}_i^1)$. Since $\mathcal{T}_i^1$ started the ith bundle, bundle $\mathcal{B}_{i-1}$ must already have had g tracks in it. Furthermore, the lengths of these tracks are longer than that of $\mathcal{T}_i^1$ since GREEDYTRACKING chooses tracks in non-increasing order of length. Therefore, $\ell(\mathcal{B}_{i-1}) = \sum_{p=1}^{g} \ell(\mathcal{T}_{i-1}^p) \geq g \cdot \ell(\mathcal{T}_i^1)$. It follows that $|Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{B}_{i-1})}{g}$.

Lemma 9. *The total busy time of all the bundles except the first one is at most twice that of an optimal solution for the entire instance. Specifically, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$.*

Proof. This proof follows from Lemma 8. For any $i > 1$, $|Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{B}_{i-1})}{g}$. Summing over all $i > 1$, we get the following: $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2\frac{\sum_{i>1} \ell(\mathcal{B}_{i-1})}{g} = 2\frac{\sum_{i>1} \sum_{j \in \mathcal{B}_{i-1}} \ell(j)}{g}$. Therefore, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{J})}{g}$. Note that $\ell(\mathcal{J}) = \sum_{j \in \mathcal{J}'} p_j$, where $\mathcal{J}'$ is the original flexible interval job instance. This is true because the dynamic program converting a flexible instance to an interval instance, does not reduce the processing length of any job. Hence, from Observation 1, $OPT(\mathcal{J}') \geq \frac{\ell(\mathcal{J}')}{g}$. It follows that $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$.

Theorem 6. *The cost of the algorithm is at most 3 times the cost of an optimal solution. Specifically, $\sum_i |Sp(\mathcal{B}_i)| \leq 3OPT(\mathcal{J})$.*

Proof. From Lemma 9, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$. Furthermore, $|Sp(\mathcal{B}_1)| \leq OPT_\infty(\mathcal{J}')$. From Observation 2, $OPT_\infty(\mathcal{J}') \leq OPT(\mathcal{J}')$. Therefore, $\sum_i |Sp(\mathcal{B}_i)| \leq 3OPT(\mathcal{J}')$.

Figure 5 shows that the approximation factor of 3 achieved by GREEDYTRACKING is tight. In the instance shown, a gadget of $2g$ interval jobs is repeated g times. In this gadget, there are g identical unit length interval jobs which overlap for ϵ amount with another g identical unit length interval jobs. The g gadgets are disjoint from one another, which means, there is no overlap among the jobs of any two gadgets. There are $2g$ flexible jobs, whose windows span the windows of all the g gadgets. These jobs are of length $1 - \frac{\epsilon}{2}$. An optimal packing would pack each set of g identical jobs of each gadget in one bundle, and the flexible jobs in 2 bundles, giving a total busy time of $2g + 2 - \epsilon$. However, the dynamic program minimizing the span does not take capacity into consideration, hence in a possible output, the flexible jobs may be packed 2 each with each of the g gadgets, in a manner such that they intersect with all of the jobs of the gadget. Hence, the flexible jobs cannot be considered in the same track as any unit interval job in the gadget it is packed with. Due to the greedy nature of GREEDYTRACKING, the tracks selected would not consider the flexible jobs in the beginning, and the interval jobs may also get split up as in Figure 6, giving a total busy time of $4(1 - \epsilon)g + (2 - o(\epsilon))g = (6 - o(\epsilon))g$, hence it approaches a factor 3 asymptotically.

6 Preemptive busy time problem

In this section, we remove the restriction, a job needs to be assigned to a single machine. A job j needs to be assigned a total of p_j time units within the interval $[r_j, d_j)$ and at most one machine may be working on it at any given time.

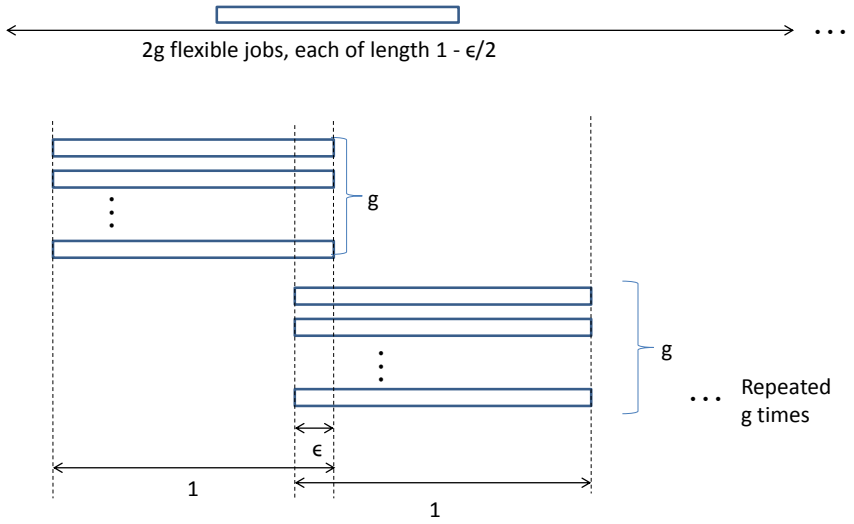


Fig. 5: Gadget for factor 3 for GREEDYTRACKING

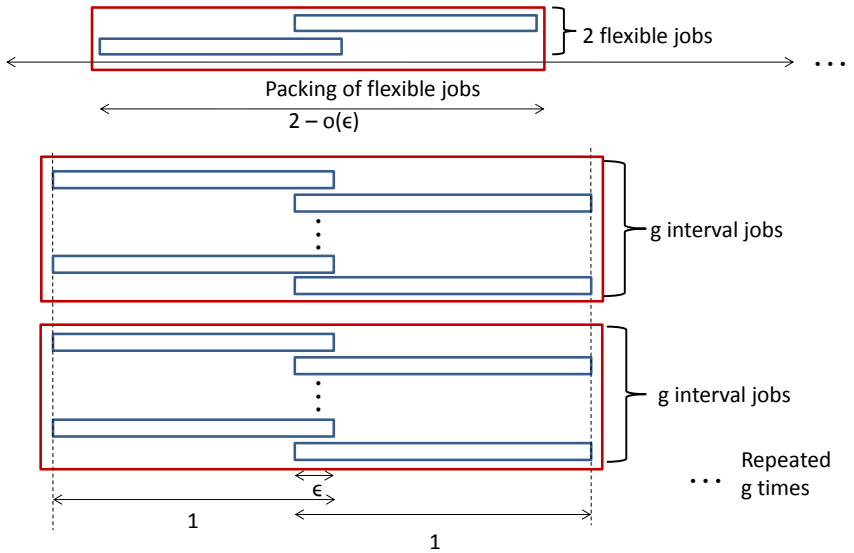


Fig. 6: Possible packing by GREEDYTRACKING

Theorem 7. *For unbounded g and preemptive jobs, there is an exact algorithm to minimize busy time.*

Proof. The algorithm is a simple greedy one. Let $\mathcal{J}_1$ be the set of jobs of earliest deadline d_1 and let the longest job $j_{max,1}$ in $\mathcal{J}_1$ have length $\ell_{max,1}$. We open the interval $[d_1 - \ell_{max,1}, d_1)$, and for every the job $j \in \mathcal{J}$ such that $[r_j, d_j) \cap [d_1 - \ell_{max,1}, d_1) \neq \emptyset$, we schedule it up to $d_1 - r_j$ in the interval $[r_j, d_1)$. Then we shrink the interval $[d_1 - \ell_{max,1}, d_1)$ and adjust the windows and remaining processing lengths of the jobs in $\mathcal{J}$ and then repeat till all jobs in $\mathcal{J}$ have been completely scheduled.

In the first iteration, without loss of generality, the optimal solution will also open the interval $[d_1 - \ell_{max,1}, d_1)$; $j_{max,1}$ has to be scheduled completely d_1 and since d_1 is the earliest deadline, opening this length of interval as late as possible ensures that we can schedule the maximum length of any job in the instance $\mathcal{J}$ with $j_{max,1}$. The correctness follows by induction on the remaining iterations.

As a consequence, one can approximate preemptive busy time scheduling for bounded g. First, solve the instance under the assumption that g is unbounded; denote by $\mathcal{S}_\infty$ this (possibly infeasible) solution. The busy time of $\mathcal{S}_\infty$ is $OPT_\infty(\mathcal{J})$, and is a lower bound on the optimal solution for bounded g. The algorithm for bounded g will commit to working on job j precisely in the time intervals where $\mathcal{S}_\infty$ had scheduled it. Partition the busy time of $\mathcal{S}_\infty$ into the set of interesting intervals $\{I_1, \ldots, I_k\}$, where $k = \theta(n)$.

For every interesting interval I_i, assign the jobs scheduled in I_i to $\lceil \frac{n(I_i)}{g} \rceil$ machines in arbitrary order, filling the machines greedily such that there is at most one machine with strictly less than g jobs.

For each I_i, at most one machine contains less than g jobs, which we charge to $OPT_\infty(\mathcal{J})$ All other machines are at capacity, i.e., have exactly g jobs and hence we charge them to $\frac{\ell(\mathcal{J})}{g}$. This implies an approximation of 2.

Theorem 8. *There is a preemptive algorithm whose busy time is at most twice that of the optimal preemptive solution, for bounded g.*

References

1. Mansoor Alicherry and Randeep Bhatia. Line system design and a generalized coloring problem. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, pages 19–30, 2003.
2. Jessica Chang, Harold Gabow, and Samir Khuller. A model for minimizing active processor time. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 289–300, 2012.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2001.
4. Ulrich Faigle, R. Garbe, and Walter Kern. Randomized online algorithms for maximizing busy time interval scheduling. *Computing*, 56(2):95–104, 1996.
5. Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Hadas Shachnai, Mordechai Shalom, Tami Tamir, and Shmuel Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *Proceedings of the IEEE 23rd International Parallel and Distributed Processing Symposium*, pages 1–12, 2009.
6. Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Approximating the traffic grooming problem with respect to adms and oadms. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 920–929, 2008.
7. Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Optimizing regenerator cost in traffic grooming. *Theoretical Computer Science*, 412(52):7109–7121, 2011.
8. Michele Flammini, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Approximating the traffic grooming problem. In *Proceedings of the 16th international conference on Algorithms and Computation (ISAAC)*, pages 915–924, 2005.
9. Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami Tamir. Minimizing busy time in multiple machine real-time scheduling. In *Proceedings of the 30th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 169 – 180, 2010.
10. Frederic Koehler and Samir Khuller. Optimal batch schedules for parallel machines. In *Proceedings of the 13th Algorithms and Data Structures Symposium (WADS)*, 2013.
11. Vijay Kumar and Atri Rudra. Approximation algorithms for wavelength assignment. In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 152–163, 2005.
12. George B. Mertzios, Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, and Shmuel Zaks. Optimizing busy time on parallel machines. In *Proceedings of the IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 238–248, 2012.

13. Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, Fencol C.C. Yung, and Shmuel Zaks. Online optimization of busy time on parallel machines. *Theory and Applications of Models of Computation. Lecture notes in Computer Science*, 7287:448–460, 2012.
14. Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 830 – 831, 2003.

A Appendix of full version: Interval Jobs

A.1 Kumar and Rudra's Algorithm

Here we provide an overview of the algorithm of Kumar and Rudra [11] for fiber minimization problem which implies a 2 approximation for the busy time problem on interval jobs. The fiber minimization problem is as follows. An optical fiber network needs to satisfy the given set of requests, that need to be assigned to consecutive links or edges connected in a line. There are n of these links. Each request needs some links $[i, i+1, \ldots, j]$, where $1 \leq i < j \leq n$. Each segment of an optical fiber can support μ wavelengths over the consecutive links that it spans, and no two requests can be assigned the same wavelength on the same fiber, if they need to use the same link. We want a feasible assignment of the requests such that total length of optical fiber used is minimized. Notice, this is very similar to the busy time problem on interval jobs. Think of the requests as interval jobs. If a request needs the consecutive links $[i, i+1, \ldots, j]$, where $1 \leq i < j \leq n$, then this can be equivalently thought of as an interval job with release time i and deadline j, i.e., with a window $[i, j)$, with processing length $j - i$, in a discrete setting where time is slotted. The total number of links being n, the processing lengths of the jobs here is linear. In this case, we can think of each slot as an interesting interval (since jobs begin and end only at slots) and define the demand profile as the tuples $(i, D(i))$, where i is a time slot $\in [1, \ldots, n]$. Their algorithm proceeds in two phases. In the first phase they assign the jobs to levels within the demand profile (where the total number of levels equals the maximum raw demand at any point), and potentially allow for a limited infeasibility in this packing. Specifically, at most two jobs can be assigned to the same level anywhere within the demand profile. In the second phase, they give a feasible packing of the jobs, considering μ levels at a time, removing the infeasibility introduced earlier, but without exceeding the cost by more than a factor of 2. This is done as follows. For levels $\{(i-1)\mu + 1, \ldots, i\mu\}$, $(i \in \{1, \ldots, D_{max}\})$ where D_{max} is the maximum height of the demand profile), Kumar and Rudra [11] open two fibers, instead of one, and assign jobs to fibers, such that two jobs which were assigned to the same level in the demand profile, get assigned to separate fibers according to a simple parity based assignment. Their analysis assumes that the raw demand at every time slot t, $|A(t)|$ is a multiple of μ and charges to such a demand profile. It is clear that the demand profile gets charged at most twice, respecting the μ capacity constraint of the fibers.

The polynomial time complexity of the algorithm crucially depends on the fact that we have n links, and hence the job lengths being linear, we need to consider only a linear number of slots. The above does not hold for the busy time problem for interval jobs with arbitrary release times, deadlines and processing lengths. The number of time instants to consider may not be polynomial. However, the key observation is that even if the release times and deadlines of jobs are not integral, there can be at most $2n$ *interesting* intervals, such that no jobs begin or end within the interval. The demand profile is uniform over every interesting interval. Therefore, their algorithm can be applied to the busy time problem, with this simple modification, still maintaining the polynomial complexity. The assumption regarding multiple of μ (in the busy time case, this would be g) at every slot, would translate as a multiple of g jobs over every interesting interval. However, note that for an arbitrary instance, we can add dummy jobs spanning any interesting interval I_i where the raw demand $|A(I_i)|$ is not a multiple of g without changing the demand profile. Specifically, if $cg < |A(I_i)| < (c+1)g$, for some $c \geq 0$, then $DeP(I_i) = c + 1$, hence adding $(c+1)g - |A(I_i)|$ jobs spanning I_i does not change the demand profile. Thus we can apply their algorithm on the busy time instance, where the demand profile is defined only interesting intervals and the demand everywhere is a multiple of g. The assignments to the fibers as done by their algorithm in Phase 2, will give the bundles for the busy time problem.

A.2 Alicherry and Bhatia's Algorithm

Now, we describe how the work of Alicherry and Bhatia [1], implies another, elegant algorithm with a 2 approximation for interval jobs. Alicherry and Bhatia study a generalized coloring and routing problem on interval and circular

graphs, motivated by optical design systems. Though the problems they consider are not directly related to the busy time problem, we can use their techniques to develop the 2 approximation algorithm. Similar to Kumar and Rudra's work, their goal is to route certain requests, which require to be assigned to consecutive links or edges in the interval or circular graph. At each link, we color the requests assigned to that link. The colors are partitioned into sets, which are ordered, such that colors in the higher numbered sets cost more. The total cost of the solution is the sum of the costs of the highest colors used at all the links, and the objective is to minimize this cost. Though this problem seems quite different from the busy time problem on interval jobs, the one of the key observations is that the cost needs to be a monotonically non-decreasing function respecting the set order. It need not be a strictly increasing function. Hence, we can think of the sets numbered in a linear order, and give each set g colors. We set the number of all the colors in a set i as i. If $c \cdot g + k$ requests use a link, the cost of that link would be the cost of the highest color used at the link, which is $c + 1$. Hence, what we are really summing is the total cost of the demand profile defined on a set of interval jobs, which have integral release times, and deadlines, and linear processing lengths, since the number of links is n (part of the input). Therefore, a 2 approximation algorithm minimizing the cost is really providing a solution that costs at most twice the demand profile of this restricted instance. The technique used involves setting up a flow graph with a certain structure, depending on the current demands or requests as yet unassigned. It can be easily proved that the graph has a cut of size at least 2 everywhere if the demand everywhere is at least 2. Now, we find a flow of size two in this graph from the source to the sink. Each flow path will consist of a set of disjoint requests or demands (where the disjointness refers to the links they need to use), and the union of the two flows will reduce a demand of at least unity from every link. This is repeated till the demand is 0 or 1 everywhere.

As in Section A.1, we use the following observation: the time slots can be considered to be interesting intervals for a set of interval jobs. The busy time instance with non-polynomial job lengths and arbitrary release times and deadlines has a linear number of interesting intervals, and hence we can think of our instance in this discretized setting. Therefore, we can apply their algorithm, modified accordingly, to our problem to get an algorithm within twice of the optimal solution. The algorithm will consider a busy time instance with the demand profile defined on interesting intervals and with a multiple of g jobs everywhere without any loss of generality. It will first open up two bundles. The flow graph is then set up as defined by Alicherry and Bhatia. For the first g iterations, the algorithm will find $2g$ flow paths (each consisting of disjoint interval jobs), the union of which removes at least a demand of g from everywhere. We assign g of these paths to one bundle and the remaining g to the other. Each flow path consists of disjoint jobs, hence, each bundle will have at most g jobs at time instant. Moreover, together, these bundles have removed a demand g from everywhere in the demand profile, hence they have charged the lowermost level (which is also the widest level) of the demand profile at most twice. The demand profile is now suitable modified after removing the jobs already assigned. Once again two bundles are opened, and the same procedure is performed for the next g iterations. This continues till the demand profile becomes empty everywhere, in other words, all jobs are assigned. The resultant bundles are feasible and charge the demand profile at most twice.

A.3 Lower bound

Though the upper bound of 2 was shown by Kumar and Rudra [11] and Alicherry and Bhatia [1] for their algorithms, a lower bound on the performance of the algorithms was not provided. Here we show that for both these algorithms, the approximation ratio obtained can be arbitrarily close to 2. Figure 7 shows an instance of interval jobs, for which both the algorithms implied by the work of Kumar and Rudra and Alicherry and Bhatia approach a factor of 2 of the optimal solution. In this example, $g = 2$ and there are two interval jobs of length 1, one interval job of length ϵ, one of length $\epsilon' < \epsilon$, and one of length $\epsilon - \epsilon'$. As required by the analysis of Kumar and Rudra and Alicherry and Bhatia, the demand everywhere is a multiple of g. A possible output by both algorithms (adapted to the busy time problem as described) has cost $2 + \epsilon$, whereas the optimal solution has cost $1 + \epsilon$. For $\epsilon \to 0$, the approximation factor approaches 2.

Theorem 9. *There exist* 2 *approximation polynomial time algorithms for the busy time problem on interval jobs. The approximation factor is tight.*

Proof. The proof follows from the discussions of Sections A.1, A.2, and A.3.

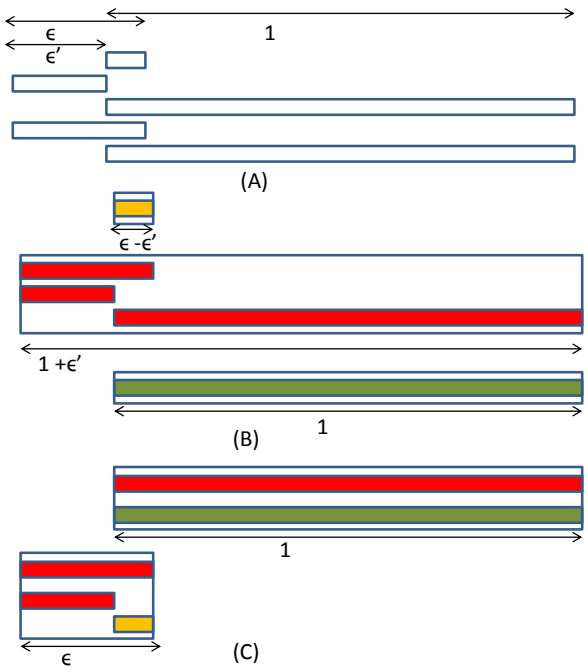


Fig. 7: (A) An instance of interval jobs and $g = 2$. (B) A possible output by the algorithms of Kumar and Rudra [11] and Alicherry and Bhatia [1], of cost = $2 + \epsilon$. (C) The optimal solution of cost $1 + \epsilon$.

B Appendix of full version: Flexible Jobs

B.1 Prior 4 approximation

In this section we discuss the more general problem of flexible jobs. This problem was studied by Khandekar et al. [9], who refer to this problem as the real-time scheduling problem. They gave a 5 approximation for this problem when the jobs can have arbitrary widths. For the unit width jobs, their analysis can be modified to give a 4 approximation.

As a first step towards proving the 5-approximation for the general problem with flexible windows and non-unit width, Khandekar et al. [9] prove that if g is unbounded, then this problem is polynomial-time solvable. The output of their dynamic program essentially converts an instance of jobs with flexible windows to an instance of interval jobs (with rigid windows), by fixing the start and end times of every job.

Theorem 10. *[9] If g is unbounded, the real-time scheduling problem is polynomial-time solvable.*

From Theorem 10, the busy time of the output of the dynamic program on the set of flexible interval jobs $\mathcal{J}$ is equal to $OPT_\infty(\mathcal{J})$.

Once Khandekar et al. obtain the modified interval instance, they apply their 5 approximation algorithm for non-unit width interval jobs to get the final bound. However, for jobs with unit width, their algorithm and analysis can be modified without loss to apply the 4 approximation algorithm of Flammini et al. [5] for interval jobs with bounded g to get the final bound of 4.

The 2 approximation algorithm [9] for interval instance charges the demand profile, hence it is immediately not clear how to extend it to handle flexible jobs since the demand profile cannot be defined analogous to the interval case. One possible natural extension is to follow the approach of Khandekar et al., to convert a flexible instance to an interval instance, and then apply the algorithm to this modified instance. Furthermore, the algorithm of Kumar and Rudra assumes that the demand profile everywhere is a multiple of g. Hence, after modifying the instance to an interval instance, we need to add dummy jobs accordingly to interesting intervals to bring up their demands to multiples of

g. However, there exists an instance where this algorithm will approach a factor of 4 of the optimal solution. This is the worst that it can do, since we prove in the following lemma that the demand profile of the modified instance of interval jobs is at most twice the demand profile of the optimal solution (note that once the jobs have been assigned in the optimal solution, their positions get fixed, and hence the demand profile can now be computed easily).

Lemma 10. *The demand profile of the output of the dynamic program converting the flexible jobs to interval jobs is at most* 2 *times the demand profile of an optimal solution structure.*

Proof. The objective function of the dynamic program (Theorem 10) is to minimize the total busy time of a flexible job instance assuming g is unbounded. Since the dynamic program is optimal, it will pack as many jobs and as much length as possible together. Hence, if a job has a choice of being assigned to a spot where other jobs need to be assigned as well, then it will be assigned at that spot instead of at some other spot where no jobs need to be assigned. Therefore, at any level of the demand profile, we can charge it to the mass of the level below, and if it is the first level, we charge it to OPT_∞ bound. Hence, in total the optimal solution gets charged twice, once by the mass bound, and once by the span bound, giving a 2 approximation.

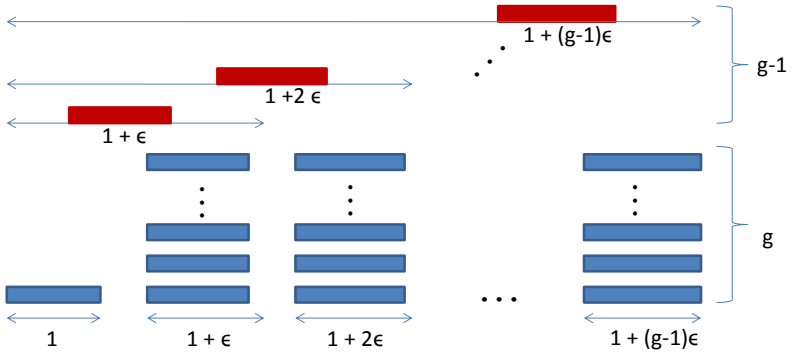
There exists an instance of flexible jobs for which the demand profile output by the dynamic program of Khandekar et al. approaches 2 times the cost of the demand profile of the optimal solution structure. We have shown such an instance in Figure 8. The instance consists of the following types of jobs: one interval job of unit length, followed by $(g-1)$ disjoint sets of identical g interval jobs, where in the i^{th} set, each job is of length $1+i\epsilon$, $(i \in [1,\ldots,(g-1)])$. Apart from these, there are $g-1$ flexible jobs, where the i^{th} job is of length $1+i\epsilon$, where $i \in [1,\ldots,(g-1)]$ and has a feasible window spanning the the windows of the first $i+1$ disjoint sets of interval jobs, as shown in the figure. An optimal solution would pack the $g-1$ flexible jobs with the first interval job, and the remaining $(g-1)$ disjoint sets of identical g interval jobs in their respective windows, with a total busy time of $g+\left(\frac{g(g+1)}{2}-1\right)\epsilon$. The dynamic program however disregards capacity constraints of the machines, and simply tries to minimize the span of the solution. Hence, with a little effort it can be seen that the unique output of the dynamic program (as shown in Figure 8) would have a span of $g+\frac{g(g-1)}{2}\epsilon$, and the demand profile on imposing a capacity of g is of cost $2g-1+g(g-1)\epsilon$, which approaches 2 the cost of the optimal solution when $\epsilon \to 0$.

Theorem 11. *A natural extension of the* 2 *approximation algorithm of Kumar and Rudra [11] (or the algorithm of Alicherry and Bhatia [1]) for the interval jobs problem, to the flexible jobs problem, gives an approximation of* 4*. This factor is tight.*

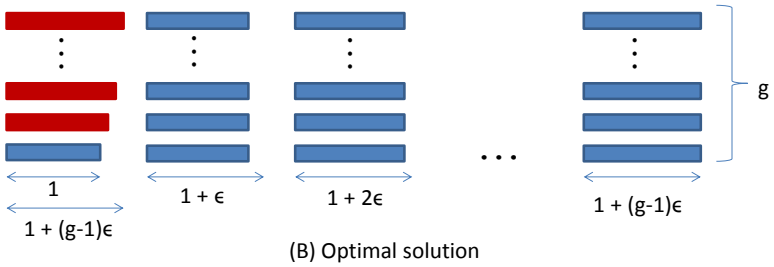
Proof. The approximation upper bound of 4 follows from Lemma 10 and Theorem 9. However, there is a tight example as well. In this example, we have an instance of interval and flexible jobs. The instance consists of a unit length interval job, followed by $g-1$ disjoint occurrences of the gadget shown in Figure 9. The gadget consists of g unit length interval jobs, $2g-2$ interval jobs of length ϵ, 2 interval jobs of length ϵ' and 2 jobs of length $\epsilon-\epsilon'$, as shown in the figure. There are $g-1$ unit length flexible jobs, each with windows spanning the windows of the union of all of the interval jobs.

On running the dynamic program to minimize span, a possible output is when each of $g-1$ flexible jobs are packed along with the $g-1$ gadgets. For applying the algorithms of Kumar and Rudra (or Alicherry and Bhatia), we need to make sure the demand everywhere is a multiple of g. Hence we add $g-1$ dummy jobs of unit length coincident with the first unit length interval job, as well as with each of the $g-1$ gadgets with a flexible job. This is shown in Figure 10.

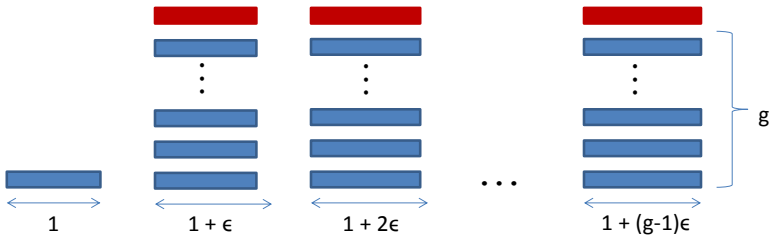
Now, one possible run of the algorithm of Kumar and Rudra (or Alicherry and Bhatia) may result in the packing shown in Figure 11, of cost $1+4(g-1)+O(\epsilon)$, whereas the optimal solution is to pack the flexible jobs with the first unit length interval job, and pack all the identical unit length jobs together, with a total cost of $g+O(\epsilon)$. Hence the ratio approaches 4 for large g and small ϵ.



(A) Instance of jobs



(B) Optimal solution



(C) Possible output generated by the dynamic program

Fig. 8: (A) An instance of interval and flexible jobs. (B)The optimal solution of busy time $g + \frac{g^2+g-2}{2}\epsilon$. (C)The output of the dynamic program of Khandekar et al. [9] of busy time = $2g - 1 + g(g-1)\epsilon$.

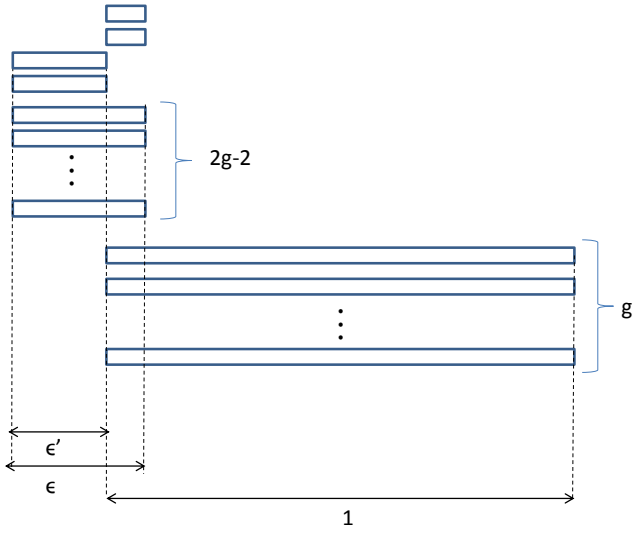


Fig. 9: Here we show the gadget for the factor 4 example.

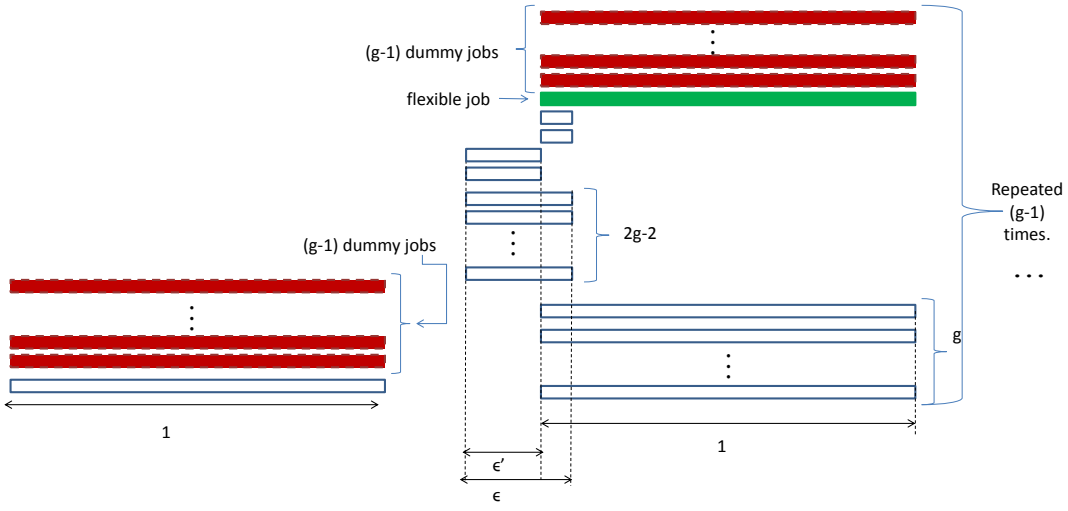


Fig. 10: Output of the dynamic program on the instance of interval and flexible jobs for the factor 4 example.

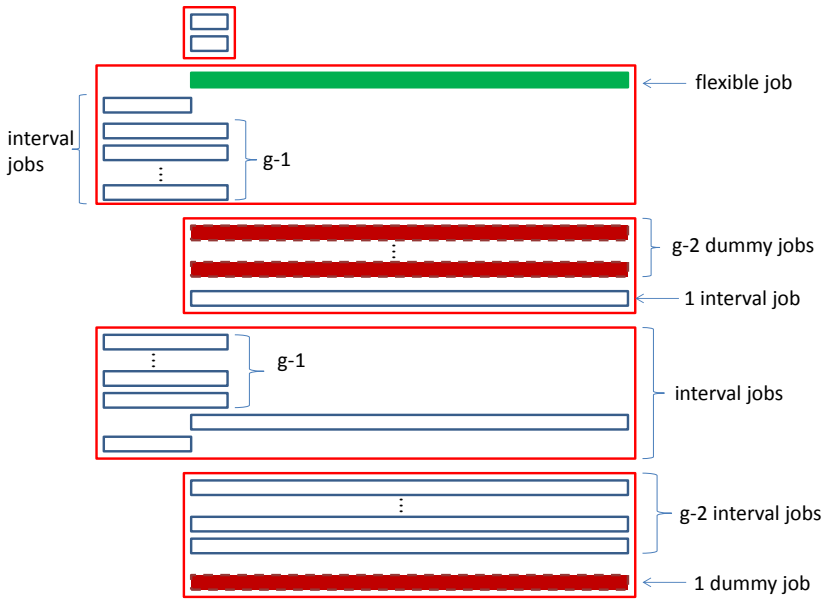


Fig. 11: Here we show the busy time bundling produced by a possible run of Kumar and Rudra or Alicherry and Bhatia's algorithms on one gadget along with the flexible job and dummy jobs.