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

Jessica Chang
University of Maryland
College Park, MD, USA
jschang@umiacs.umd.edu

Samir Khuller
University of Maryland
College Park, MD, USA
samir@cs.umd.edu

Koyel Mukherjee†
Xerox Research Centre India
Bangalore, India
koyel.mukherjee@xerox.com

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 running 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 is 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.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*; G.2 [**Discrete Mathematics**]: Combinatorics

Keywords

busy time; scheduling; packing

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

†This work was done while the author was a graduate student at the University of Maryland, College Park.

SPAA'14, June 23–25, 2014, Prague, Czech Republic.

http://dx.doi.org/10.1145/2612669.2612689.

1. INTRODUCTION

Scheduling jobs on multiple parallel or batch machines has received extensive attention in the computer science and operations research communities for decades [5, 15, 9]. 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. Energy-aware algorithmic efforts notwithstanding [1, 2, 4], the need to understand algorithmic design for energy efficiency remains largely unaddressed, particularly by techniques and approaches of traditional scheduling objectives. Toward that end, our work is most concerned with minimization of the total time that a machine is on [6, 15, 11, 14, 17] 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 and 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) [11]. The application of busy time models to optical network design has been extensively outlined in the literature [11, 12, 13, 18].

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 and VMware Distributed Resource Scheduler (DRS) offer VM consolidation as a feature. In this sense, minimizing busy time is closely related to the basic problem of mapping VMs to physical hosts.

We first discuss the active time model [6]. In this model we have a collection $\mathcal{J}$ of n jobs that need to be scheduled on one machine. Each job 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. For a job j, we need to schedule p_j units in the window $[r_j, d_j)$ and at most one unit can be scheduled in any time slot. The goal is to minimize the *active time* of the machine, that is, the total 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 [6, 15] that yields an optimal solution.

If we allow preemption at integer boundaries, the feasibility question is easily resolved by a network flow computation, as discussed by Chang and Khuller [7]. There is a node for every job and a node for every time slot. There is an edge from the source to each job node j with capacity p_j, and unit capacity edges from job node j to a slot node where j is feasible. Finally, all slot nodes are adjacent to the sink, with an edge capacitiy of g. Thus a flow of value $\sum_j p_j$ units corresponds to a feasible integral schedule for all the jobs.

Unfortunately, in such a construction, there is no control as to which slot nodes receive flow; to minimize active time suggests a shift from computing a maximum flow to computing a min-edge cost flow, where we only wish to send non-zero amount of flow through as few edges as possible that connect to the sink.

In this work, we first show that every minimal solution is a 3-approximation to this problem. We also show that this bound is tight. We then further improve the approximation ratio by considering a natural IP formulation and its LP relaxation to obtain a solution within twice the integer optimum (again this bound is tight). See Figure 1 for an example of an optimal solution, with preemption allowed at integral boundaries. This presents substantial progress on the problem left open earlier [6]. We also assume that the input is feasible as this can be easily verified by a simple network flow computation [7].

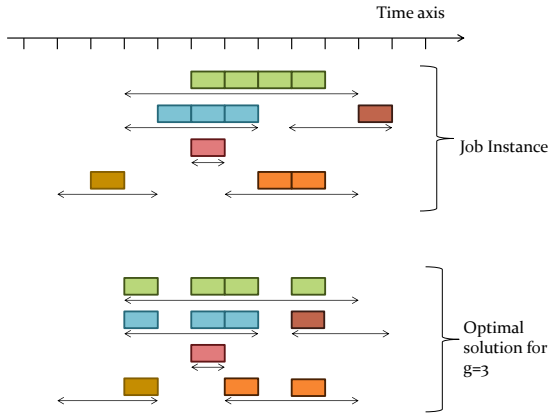

Figure 1: An optimal solution for the active time problem with integral preemption, for an instance of 6 jobs and $g = 3$.

We also consider a slight variant of the active time problem that has been considered previously in the literature [11, 14], and referred to as the *busy time* problem. 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 that need to be scheduled on a set of identical machines. Each job 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. The schedule can potentially use an unbounded number of machines since each group is really a virtual machine, and thus every input instance is feasible as we can simply create a virtual machine for each job.

A well-studied special case of this model is one in which each job j is "rigid", i.e., $d_j = p_j + r_j$, in which there is no question about when it must start. Jobs of this particular form are called *interval jobs*. Even in this case, the busy time problem is NP-hard for $g = 2$ [18]. We say the interval $[r_j, d_j)$ is the *span* of job 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 [14]. The dynamic program "fixes" the positions of the jobs to minimize their shadow, i.e., projection onto the time-axis. The span of the solution with g unbounded is the smallest possible span of any feasible 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 2 shows a collection of jobs and the corresponding packing that yields an optimal solution.

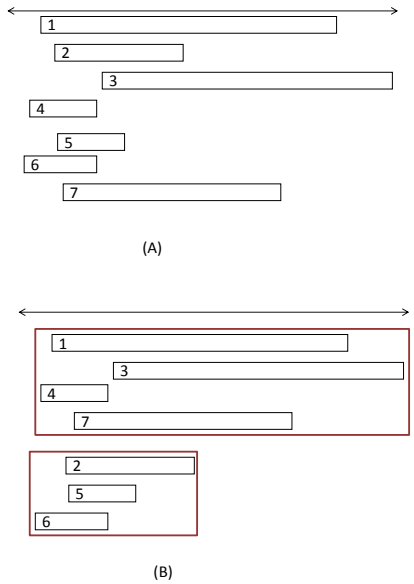

Figure 2: (A) Collection of interval jobs with unit demand, numbered arbitrarily. (B) Optimal packing of the jobs on two machines with $g = 3$ minimizing total busy time.

Busy time scheduling in this form was first studied by Flammini et al. [11]. They present a 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 [3] and Kumar and Rudra [16] already considered a problem in the context of wavelength assignment. One can show that their algorithms immediately yield two different 2-approximations for the the problem of minimizing the busy time for scheduling interval jobs.

Khandekar et al. [14] 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 principles to this problem to 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 FIRSTFIT 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 also give an 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-approximation for flexible jobs, by solving a dynamic program for unbounded g. With little effort [8] one can show via a similar extension that the methods of Kumar and Rudra [16] and Alicherry and Bhatia [3] also yield 4-approximations, and the bounds are tight. Details are in the full version of the paper [8]. We develop an improved algorithm with a bound of 3, using a completely different approach.

1.1 Our Results

In the active time problem, we are allowed preemption at integer time points and time is slotted. We show in Section 2 that any minimal feasible solution yields a 3-approximation. We then consider a natural IP formulation for this problem and show how its LP relaxation allows us to convert a fractional schedule to an integral one that is 2-approximate. We note that the integrality gap of 2 is tight. Earlier work [6] only addressed the special case where job lengths were unit, for which an optimal polynomial-time algorithm was given. Unfortunately, it is not clear how to extend that framework for the case of non-unit length jobs.

Since the busy time problem for interval jobs is NP-hard [18], the focus in this paper is on the development of a polynomial-time algorithm GREEDYTRACKING with a worst case approximation guarantee of 3. 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 set of jobs assigned to a particular machine is 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 a 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.

[1]In an attempt to improve approximation guarantees, Flammini et al. [11] 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 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 [17].

We also give instances of interval jobs where GREEDYTRACKING yields a solution twice that of an optimum in the full version [8].

One important consequence of GREEDYTRACKING is an improved bound for the busy time problem on instances in which jobs may not be interval jobs. In the spirit of Khandekar et al. [14], we first solve the problem assuming unbounded machine capacity g to get a solution that minimizes the projection of the jobs onto the time-axis. We use this to map the original instance to one of interval jobs, forcing each job to start exactly as it did in the solution for unbounded capacity, over which we compute a solution via GREEDYTRACKING. We prove that in total, this approach has busy time within thrice that of the optimal solution, and the bound is tight. In addition, we explore the preemptive version of the problem, providing a greedy 2-approximation. However, we omit the discussion of this result from the extended abstract; details can be found in the full version of this paper [8].

2. ACTIVE TIME SCHEDULING OF PREEMPTIVE JOBS

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. In this notation, let $\mathcal{T}$ denote the set of time slots $\{1, \ldots, T\}$.

DEFINITION 1. *A job j is* live *at slot t if $t \in [r_j, d_j)$.*

DEFINITION 2. *A slot is* active *or* open *if at least one job is scheduled in it. It is* inactive *or* closed *otherwise.*

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

A feasible solution σ is specified by a set $\mathcal{A} \subseteq \mathcal{T}$ of active time slots 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 $\mathcal{A}$ has been determined, a feasible integral assignment can be found via a max-flow computation [7].

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

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 as follows. Assume that all the slots are initially active. Now (in any order) make slots inactive, if one can feasibly do so (this might change the actual slots to which jobs are assigned at every iteration).

The cost $|\mathcal{A}_f|$ of the full slots can be charged to OPT. On the other hand, to bound the number of non-full active slots requires a concept specific to minimal feasible solutions.

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

LEMMA 1. *For a minimal feasible solution σ, there exists a solution σ' of the same cost in which each active non-full slot has at least one non-full-rigid job scheduled in it.*

PROOF. Consider any non-full slot of a minimal feasible solution σ that does not have any non-full-rigid job scheduled in it. Move any job j in that slot to any other (non-full and active) slot that it may be scheduled in, and where it is not already scheduled. There must at least one such slot; otherwise j 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: we only move jobs to slots that are already active. If this process continues, eventually there will be no more jobs scheduled in this slot, we would have found a solution of smaller cost, violating our assumption of minimal feasibility. Thus, there must be at least one job j' scheduled in that slot that cannot be moved to any other active slot. This can only happen if all the slots in the window of j' are already assigned one unit of j', or are full or inactive, i.e., if j' is a non-full rigid job. Continue this process until each non-full slot has 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 of σ'.*

We say that such a set $\mathcal{J}^*$ covers 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. $\mathcal{J}^*$ is called a minimal set.*

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. □

It can be proven that there exists a minimal set $\mathcal{J}^*$ such that at every time slot, at most two of the jobs in $\mathcal{J}^*$ are live. We charge the cost of the non-full slots to $\mathcal{J}^*$.

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. $\mathcal{J}^*$ 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 an optimal solution as OPT. Hence, the cost 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. This is also a lower bound on OPT. Thus, $|\mathcal{A}_f| \leq \frac{\sum_{j \in \mathcal{J}} p_j}{g} \leq OPT$ and 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 (see Figure 3).

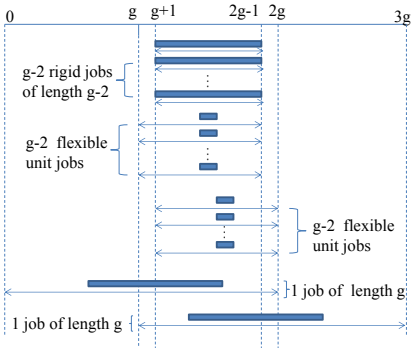

Figure 3: Instance where the minimal feasible solution is almost 3 times the optimal solution. The optimal solution keeps slots $g+1$ and $2g$ open, assigning to them all of the flexible unit-length jobs. However, the minimal solution that forces slots $g+1$ and $2g$ to inactive will necessarily schedule the two jobs of length g by themselves, thus incurring a total cost of $3g-2$.

3. AN LP-ROUNDING 2-APPROXIMATION

In this section, we develop a 2-approximation for the active time problem via LP-rounding techniques. Recall that jobs may be non-unit in length and that preemption is permitted only at integral boundaries. In this section onwards, we will be using t to denote the slot $[t-1, t)$ for ease of notation. Let $y_t \in \{0, 1\}$ denote the indicator variable for every time slot $t \in \mathcal{T}$. Let $x_{t,j} \in \{0, 1\}$

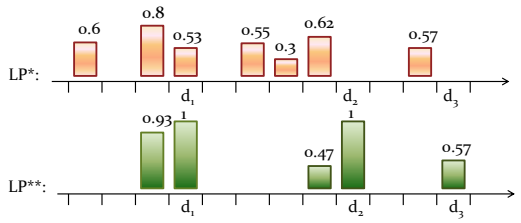

Figure 4: LP* is an optimal LP solution, and LP** is the right-shifted solution of the same cost.

denote the indicator variable for job $j \in \mathcal{J}$ and slot $t \in [r_j, d_j]$. The natural LP relaxation is as follows:

$$\min \sum_{t \in \mathcal{T}} y_t$$

$$x_{t,j} \leq y_t \qquad \forall\, j \in \mathcal{J}, t \in [r_j, d_j]$$

$$\sum_{j \in \mathcal{J}} x_{t,j} \leq g y_t \qquad \forall\, t \in \mathcal{T}$$

$$\sum_{t \in \mathcal{T}} x_{t,j} \geq p_j \qquad \forall\, j \in \mathcal{J}$$

$$x_{t,j}, y_t \geq 0 \qquad \forall\, t \in \mathcal{T}, j \in \mathcal{J}$$

Our approach computes an optimal fractional solution and rounds it to a feasible integral solution within twice the cost of the fractional one. Before we can round, we pre-process the fractional solution to get a certain structure without increasing the cost of the solution.

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_\ell\}$ be the distinct deadlines of the instance, sorted in increasing order. The pre-processing step iteratively transforms the fractional solution to have a *right-shifted* structure: for deadline d_i, if any slot $t \in (d_{i-1}, d_i]$ is open to any extent, then every subsequent slot $t' \in (t, d_i]$ has the property that $y_{t'} = 1$. Informally, this structure is obtained by iteratively "pushing" all y_t values between d_{i-1} and d_i toward the right until it runs into d_i; details can be found in the full version of the paper [8].

See Figure 4 for an example of a right-shifted LP solution.

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

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

Overview of Rounding

We iteratively process the deadlines of $\mathcal{D}$ in increasing order. We denote the set of jobs with deadline d_i as $\mathcal{J}_i$. At the end of iteration i, we have a set of integrally open slots $\mathcal{O}_i$. Let the cumulative y-value between consecutive deadlines be defined as $Y_i = \sum_{d_{i-1} < t \leq d_i} y_t$ ($d_0 = 0$). The rounding algorithm maintains the invariant that at the end of the i^{th} iteration, the number of integrally open slots up to d_i is $|\mathcal{O}_i| \leq 2 \sum_{j \leq i} Y_j$. Furthermore, there exists a feasible fractional assignment of $\bigcup_{k \leq i} \mathcal{J}_k$ in $\mathcal{O}_i$. This will yield the 2-approximation by the end of the ℓ^{th} iteration. We refer to slots t with $y_t = 1$ as "fully open". Those with $\frac{1}{2} \leq y_t < 1$ are denoted as "half-open", and those with $0 < y_t < \frac{1}{2}$ are"barely open". Finally, slots with $y_t = 0$ are "closed".

Fully open slots do not charge anything extra to the LP solution. Half-open slots will be opened at a cost of at most 2, charging themselves. To open a barely open slot, we need to charge it to a fully open slot. We say that a barely open slot is "dependent" on the fully open slot that it charges. In this case, the y-value of the barely open slot is not charged at all. In other cases, we allow two barely open slots on either side of a fully open slot to open up along with a fully open slot; this is permissible only when the sum of the y-values of the barely open slots and the fully open slot is at least $\frac{3}{2}$. We refer to such slots as a "trio". Note that in the case of a trio, we charge the y-value of the barely open slots.

The algorithm maintains the invariant that in each iteration, every barely open slot 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 and is not part of a trio.

Sometimes while processing a deadline d_i, the algorithm may choose to close a barely open slot $t \in (d_{i-1}, d_i]$. In such a case, it must also schedule elsewhere the job segments that were initially assigned to t. In particular, the algorithm accommodates jobs of later deadline by creating a *proxy* copy of the slot t and carrying it over to the next iteration. The y-value of this proxy slot is the y-value of the slot t we have just closed; note that we do not charge the y-value of t in this iteration, so the proxy slot may charge its y-value in the future.

Intuitively, proxy slots permit the algorithm to delay the scheduling of job segments, in the hope that the proxy slot can be merged with fractional slots of future iterations. Unlike methods for conventional scheduling problems, algorithms that minimize active time must favor delaying early jobs in the hopes that they can be batched with later jobs. The difficulty of the active time objective lies in balancing the tension between this bias and feasibility constraints: what should the algorithm do if it delays jobs only to discover later that the number of jobs needing to be scheduled (including those that were delayed) exceeds the time and resources available? To handle this, proxy slots maintain a pointer initialized to the actual slot from which it was derived; in this way, the proxy slot keeps track of a "safe" slot to which it can fall back in such cases of excess demand. As a proxy slot is propagated from one iteration to the next, it may update its y-value or the pointer to an actual slot.

If a proxy slot is passed to an iteration, the algorithm treats 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 slot is propagated to the next iteration. If the algorithm determines that the proxy slot should be "opened", the actual slot to which the proxy slot points is opened. It is at this point that the cost of the proxy is charged to its y-value.

Thus, in each iteration i of the rounding algorithm, the y-value of any proxy slot (if it exists) is considered along with Y_i. In a single iteration, slots are opened from right to left, i.e., starting with d_i, then $d_i - 1$, and so forth. The algorithm first opens as many fully open slots as possible. If the remaining y-value is at least half, then the next slot (denote t'') is opened and charged to itself. If the remaining y-value is positive but less than half, the algorithm attempts to close t'' and find a feasible assignment of all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$, using max-flow. If such an assignment exists, then t'' will remain closed and any remaining jobs that are not in $\mathcal{J}_i$ are passed on via a proxy slot. (Notice that there can exist at most one proxy slot at any time.) If such an assignment does not exist, then the algorithm is forced to open t'' as barely open. The cost of opening t'' will be charged to the earliest fully open slot that does not yet have a dependent; if no such fully open slot exists, then t'' forms a trio with the preceding fully open slot and its dependent. See Figure 5 for an example.

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

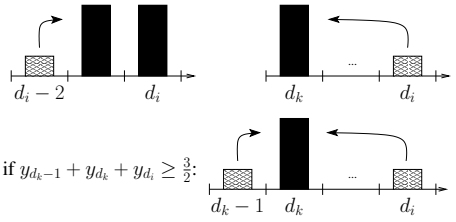

Figure 5: Three possible ways by which we can charge a barely open slot when max-flow cannot close it.

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, and there are no fully open slots that we can charge it to. The rounding procedure (described in details in the full version [8]) ensures that this can only happen if the closest open slot to d_i is fully open. Since we could not charge d_i as a trio or as a dependent on any of the fully open earlier slots, all the fully open slots before d_i must have dependents.

The following lemma argues that if all the fully open slots before d_i have dependents, then the structure of the solution must have a specific form. There must be a deadline d_z, with the closest open slot before 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 at least $g+1$ 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 d_z and the length 2 jobs with release time at least d_w is at least $g+1$. 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). This structure is called an *alternating* structure.

LEMMA 5. *If the closest open slot to deadline d_i is fully open, and all the fully open slots before 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 (so 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. □

The following lemma shows that for any pair of intermediate deadlines, d_u and d_x in the alternating structure, there are $2g+1$ units of jobs that need to be scheduled in the window $(d_u, d_x]$.

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 $2g+1$ job units in $\mathcal{J}_u \cup \mathcal{J}_x$ must have release time at least 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 that 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 at least d_z. Let us assume there are at most $2g$ units of jobs in $\mathcal{J}_z \cup \mathcal{J}_a$ with release time at least d_z, by way of contradiction. No job in $\mathcal{J}_a$ can be greater than or equal to

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. Then, 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, by 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 at least $2g+1$ job units (including flexible, unit length and non-unit length) in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ whose jobs are released by d_k.

Therefore, we have proved the claim by induction. □

Proof of Lemma 4 continued:
Since we cannot charge d_i as a dependent or a trio, by arguments similar to those proving Lemma 6, it can be shown that there must be at least $g+1$ unit jobs with release time d_k in $\mathcal{J}_i \cup \mathcal{J}_k$.

Without loss of generality, we assume the deadlines are open in consecutive order. The alternating structure is over some set of deadlines $\{d_z, d_{z+1}, \ldots, d_i\}$. From Lemmas 5 and 6, the following hold: for d_z, we have that $n'_z + n_z \geq g+1$, where d_z is the stopping deadline, and 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 $d_z < d_p < d_i$, $n_{p-1} + n_{p,1} + 2n_{p,2} + n'_{p,1} \geq 2g+1$, where $n_{p,1}$ 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 at least d_{p-1} and $n'_{p,1}$ denotes the number of flexible unit length jobs with deadline at least d_{p-1}. Therefore, $\sum_{z \leq p \leq i} n'_{p,1} + 2\sum_{z \leq p \leq i} n_{p,1} + 2\sum_{z \leq p \leq i} n_{p,2} > 2(i-z)g$. However, for LP feasibility, jobs must be assigned to an extent of strictly less than $\frac{1}{2}$ 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. □

THEOREM 3. *There exists an algorithm whose active time is within twice the optimum, on non-unit length jobs that can be preempted on the integer time boundary.*

PROOF. From Lemma 4 and the rounding procedure, it follows that at the end of iteration i, the total number of integrally open slots $|\mathcal{O}_i| \leq 2\sum_{1 \leq k \leq i} Y_k$, and there exists a LP feasible fractional assignment of jobs in $\bigcup_{x \leq i} \mathcal{J}_x$ in $\mathcal{O}_i$. Hence the proof follows. An integral feasible assignment of all jobs can be found in the slots $\mathcal{O}_\ell$ via max-flow. □

4. BUSY TIME: NOTATIONS AND PRELIMINARIES

DEFINITION 6. *Job j is an* interval job *means that its length p_j is equal to $d_j - r_j$.*

A job j is *active* on machine m at some time $t \in [r_j, d_j)$ if 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 ([3], [16]).

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 $\mathcal{J}$.*

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

OBSERVATION 2. *$OPT(\mathcal{J}) \geq OPT_\infty(\mathcal{J})$.*

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, $OPT_\infty(\mathcal{J})$ is equal to $|Sp(\mathcal{J})|$.

The following theorem follows from the works of Alicherry and Bhatia [3] and Kumar and Rudra [16].

THEOREM 4. *There exists a factor* 2 *approximation algorithm for the busy time problem on interval jobs. The approximation factor is tight.*

5. A 3-APPROXIMATION ALGORITHM FOR NON-PREEMPTIVE BUSY TIME

The busy time problem was studied by Khandekar et al. [14], who referred to it as the real-time scheduling problem. In fact, they gave a 5-approximation for a slight generalization, in which jobs can have arbitrary widths. (The generalized constraint is that at no point may the sum of widths of "live" jobs in a given bundle exceed g.) In the busy time problem as defined in this work, widths are all unit; under such assumptions, their analysis yields a 4-approximation. As a first step towards proving this, Khandekar et al. [14] show that if g is unbounded, then the problem is polynomial-time solvable via a dynamic program. Recall that an interval job j is defined to have the property $p_j = d_j - r_j$. The output of their dynamic program essentially converts a given busy time instance to one of interval jobs by fixing the start and end times of every job.

THEOREM 5. *[14] If g is unbounded, the real-time scheduling problem is polynomial-time solvable.*

By Theorem 5, the span of the output of the dynamic program is $OPT_\infty(\mathcal{J})$.

Once Khandekar et al. [14] obtain the modified interval instance, they apply the 5-approximation 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 get a final bound of 4. Moreover, extending the algorithms of Alicherry and Bhatia [3] and Kumar and Rudra [16] to the busy time problem by converting a given instance to an interval instance (similar to the approach of Khandekar et al. [14]) also gives a 4-approximation[2].

In this section, we give a 3-approximation for the busy time problem, i.e., for unit width jobs, improving the existing 4-approximation. Analogous to Khandekar et al. [14], we first convert the instance $\mathcal{J}'$ to an instance $\mathcal{J}$ of interval jobs by temporarily removing the assumption that g is bounded, applying a dynamic program on $\mathcal{J}'$ and fixing the job windows according to the output of the dynamic program. Let $OPT_\infty(\mathcal{J}')$ denote the busy time of the output of the dynamic program. By Observation 2, we know that $OPT_\infty(\mathcal{J}') \leq OPT(\mathcal{J}')$.

Then, on the interval job instance $\mathcal{J}$, we will run our algorithm GREEDYTRACKING. For an interval job j, its window $[r_j, d_j)$ is denoted as the span $Sp(j)$ of j. For the remainder of the section, we assume that the input consists of interval jobs.

To describe GREEDYTRACKING requires the notion of a *track*.

DEFINITION 8. *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 [11] suffers from the fact that it greedily considers jobs one-by-one; GREEDYTRACKING is less myopic in that it identifies sets of jobs, entire 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 [10]. 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 $\{\mathcal{B}_p\}_{p=1}^{\lceil \frac{i-1}{g} \rceil}$

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*, for $i \geq 1$. 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 $|Sp(\mathcal{J}'_i)| \leq 2|Sp(\mathcal{T}_i)|$.*

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 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 $\mathcal{T}_2^*$.

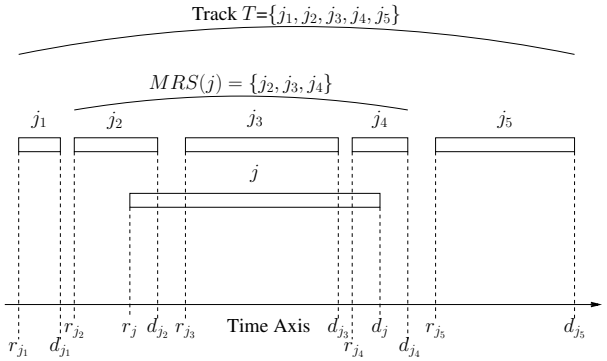

Figure 6: An example showing the minimum replaceable set of a job j, i.e., $MRS(j)$ with respect to a track T.

[2]The bound of 4 for all these algorithms is tight [8].

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

DEFINITION 9. *Consider track $\mathcal{T}$ and 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 $\mathcal{T}$, i.e., 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 $MRS(j, \mathcal{T})$ consists of jobs in $[r_j, d_{j_\ell})$ (respectively, $[r_{j_f}, d_j)$). If neither j_f nor j_ℓ exists, then $MRS(j, \mathcal{T}) = \emptyset$. (See Figure 6 for an example.)*

CASE 1. *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 $MRS(j, \mathcal{T}_2^*)$. well as that in $MRS(j, \mathcal{T}_2^*)$ They cannot be empty, since 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 $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$: $Sp(MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)) < [r_e, t_I)$, but $Sp(j) \geq [d_e, t_I]$, so $Sp(MRS(j, \mathcal{T}_1^*) \cup j) > [r_e, t_I]$. See Figure 7 for an example. Hence, this case is not possible.

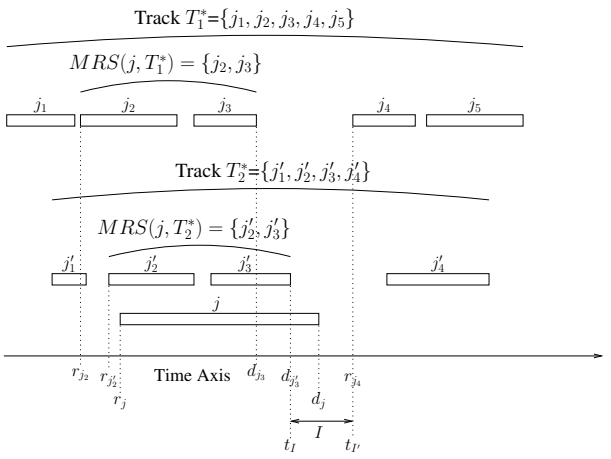

Figure 7: An example for Case 1 of Lemma 7. Replacing $MRS(j, \mathcal{T}_2^*)$ by j increases $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$.

CASE 2. *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 3. *There exists a job j, such that $[r_j, d_j) \supset [t_I, t'_I)$.*

Let the earliest release time (latest deadline, respectively) of any job in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$ be r_e (d_ℓ, respectively) and the corresponding job be j_e (j_ℓ, respectively). Without loss, these sets are not empty. If j_e and j_ℓ belonged to the same track, say $\mathcal{T}_1^*$, we could have replaced $MRS(j, \mathcal{T}_2^*)$ with j in $\mathcal{T}_2^*$ and increased the union of the span of $\mathcal{T}_1^* \cup \mathcal{T}_2^*$: $Sp(j) \geq [d_e, r_\ell)$ and would include $I = [t_I, t'_I)$, whereas $Sp(MRS(j, \mathcal{T}_2^*) \setminus MRS(j, \mathcal{T}_1^*))$ would be 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, $\ell(\mathcal{B}_{i-1}) = \sum_{p=1}^{g} \ell(\mathcal{T}_{i-1}^p) \geq g\ell(\mathcal{T}_i^1)$. And so we conclude 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, $OPT(\mathcal{J}') \geq \frac{\ell(\mathcal{J}')}{g}$. Therefore, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$. □

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, $\sum_i |Sp(\mathcal{B}_i)| \leq 3OPT(\mathcal{J}')$. □

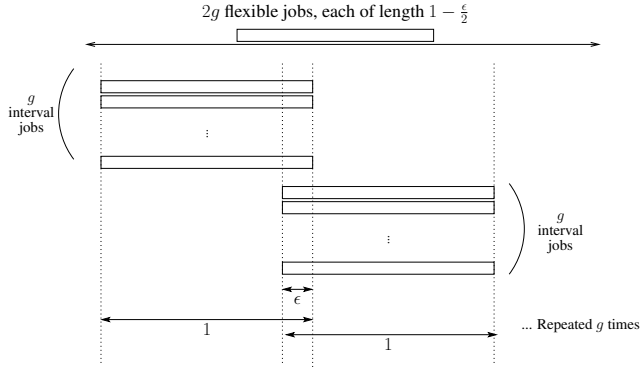

Figure 8: Construction for factor 3 for GREEDYTRACKING.

The bound for this algorithm is tight.

Figure 8 shows that the approximation factor of 3 achieved by GREEDYTRACKING is tight. In the instance shown, a small construction of $2g$ interval jobs is repeated g times. For each copy of the construction, there are g identical unit length interval jobs which overlap for ϵ amount with another g identical unit length interval jobs. The g copies are disjoint from one another, i.e., there is no overlap among the jobs of any two constructions. There are also $2g$ flexible jobs, whose windows span the windows of all g constructions. These jobs have length $1-\frac{\epsilon}{2}$. An optimal packing would pack each set of g identical jobs of each copy in one bundle, and the flexible jobs in two bundles, yielding a total busy time of $2g+2-\epsilon$. However, since the dynamic program minimizing the span is oblivious to capacity, it may pack the flexible jobs 2 each with each of the g constructions, in a manner such that they intersect with all of the jobs of the construction. Hence, the flexible jobs cannot be considered in the same track as any unit interval job in the construction 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 9, giving a total busy time of $4(1-\epsilon)g + (2-o(\epsilon))g = (6-o(\epsilon))g$, hence it approaches a factor of 3 asymptotically.

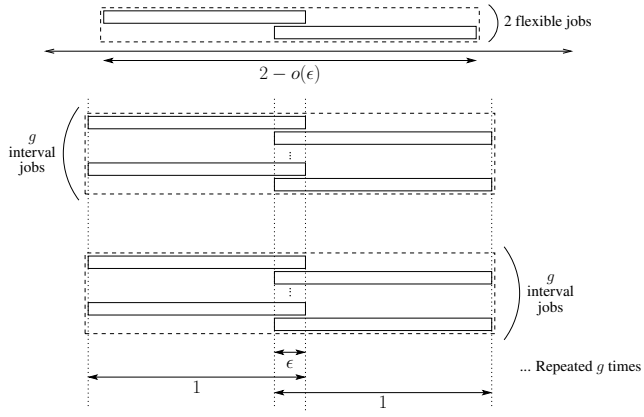

Figure 9: Possible packing produced by GREEDYTRACKING on the instance of Figure 8.

6. REFERENCES

[1] S. Albers. Algorithms for energy saving. *Efficient Algorithms*, 5760:173–186, 2009.

[2] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

[3] M. Alicherry and R. Bhatia. Line system design and a generalized coloring problem. In *Proceedings of ESA*, pages 19–30, 2003.

[4] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the 17th Annual ACM-SIAM SODA*, pages 364–367, 2006.

[5] P. Brucker. *Scheduling algorithms*. Springer, 2007.

[6] J. Chang, H. Gabow, and S. Khuller. A model for minimizing active processor time. In *Proceedings of ESA*, pages 289–300, 2012.

[7] J. Chang and S. Khuller. A min-edge cost framework for capacitated covering problems. In *Proceedings of the 15th Annual ALENEX*, pages 14–25, 2013.

[8] J. Chang, S. Khuller, and K. Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. `http://www.cs.umd.edu/~samir/fv.pdf`.

[9] A. Condotta, S. Knust, and N. V. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13(5):463–477, 2010.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, 2001.

[11] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *Proceedings of the IPDPS*, pages 1–12, 2009.

[12] M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem with respect to adms and oadms. In *Proceedings of Euro-Par*, pages 920–929, 2008.

[13] M. Flammini, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem. In *Proceedings of ISAAC*, pages 915–924, 2005.

[14] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Minimizing busy time in multiple machine real-time scheduling. In *Proceedings of FSTTCS*, pages 169–180, 2010.

[15] F. Koehler and S. Khuller. Optimal batch schedules for parallel machines. In *Proceedings of WADS*, pages 475–486, 2013.

[16] V. Kumar and A. Rudra. Approximation algorithms for wavelength assignment. In *Proceedings of FSTTCS*, pages 152–163, 2005.

[17] G. B. Mertzios, M. Shalom, A. Voloshin, P. W. Wong, and S. Zaks. Optimizing busy time on parallel machines. In *IPDPS*, pages 238–248, 2012.

[18] P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the 14th Annual ACM-SIAM SODA*, pages 830 – 831, 2003.