Parallel Machine Scheduling to Minimize Energy Consumption

Antonios Antoniadis* Naveen Garg† Gunjan Kumar‡ Nikhil Kumar §

Abstract

Given n jobs with release dates, deadlines and processing times we consider the problem of scheduling them on m parallel machines so as to minimize the total energy consumed. Machines can enter a sleep state and they consume no energy in this state. Each machine requires L units of energy to awaken from the sleep state and in its active state the machine can process jobs and consumes a unit of energy per unit time. We allow for preemption and migration of jobs and provide the first constant approximation algorithm for this problem.

1 Introduction

Energy is an extremely important and scarce resource, and its consumption is progressively becoming a pivotal concern in modern societies. Computing environments account for a large fraction of the global energy consumption and alarmingly, this fraction is growing at a very high rate [1]. In response to this, modern hardware increasingly incorporates various energy-saving capabilities. To make use of these we need to design scheduling algorithms, not only for time and space considerations, but keeping energy consumption in mind as well.

We focus on one of the most common such power-management techniques called a *power-down mechanism*, which refers to the ability of the processor to transition into a sleep state where it consumes negligible energy. Since "waking-up" the processor requires a certain amount of energy, there is a trade-off to be had between the energy saved by residing in the sleep state and the energy expended in transitioning back to the active state. Intuitively, one should aim to keep the number of transitions to the sleep states low and once in a sleep state remain in it for as long as possible.

Consider a set of jobs with individual release times, deadlines and processing times, that are to be processed on either a single or a multiprocessor system equipped with a powerdown mechanism. The processor consumes one unit of energy per unit of time when in the active state and no energy when in the sleep state. Transitioning from the sleep state to the active state incurs a fixed energy cost. Preemption and migration of jobs is allowed but no job can be simultaneously processed on more than one machine. The goal is to produce a feasible schedule which consumes the minimum energy (or report that no feasible schedule exists). In Graham's notation, and with E being the appropriate energy function the problems we study can be denoted as $1|r_j; \overline{d}_j; \text{pmtn}|E$ and $m|r_j; \overline{d}_j; \text{pmtn}|E$ respectively.

The problem on a single machine was first stated in [12], where a greedy 2-approximation algorithm called *Left-To-Right* was presented. Roughly speaking, Left-To-Right tries to keep the machine at its current state (active or asleep) for as long as possible. However the computational complexity of the problem remained open and was repeatedly posed as an important open question, in particular because "many seemingly more complicated problems in this area can be essentially reduced to this problem" (c.f. [11]). The complexity question, for the single-machine setting, was eventually settled, initially by Baptiste [6], who gave a $O(n^7)$-time algorithm for the case of unit-size jobs and subsequently by Baptiste et. al. [7] who achieved a running time of $O(n^4)$ for unit sized jobs and $O(n^5)$ when jobs can have arbitrary processing times. Both algorithms are based on a rather involved dynamic programming approach.

The multiprocessor case turns out to be much more challenging than the single processor one, and obtaining any algorithm for it with a non-trivial performance guarantee has been a major open problem [7]. It is also an open problem whether the problem is **NP**-hard. The difficulty in obtaining a good approximation algorithm seems to arise from two aspects: First, it is not clear how to design a dynamic programming table of polynomial size when the jobs have arbitrary sizes, and a job is not allowed to run parallel to itself. Secondly, structural properties of an optimal schedule can be locally extracted in a single machine environment in contrast to the multi-machine case. As an example, we know that a single machine will be active for at least one time-point within the interval between the release time and the deadline of every job, but the

*Saarland University and Max-Planck-Institute for Informatics, Saarland University Campus, Saarbrücken, Germany. Supported by DFG grant AN 1262/1-1.

†Indian Institute of Technology Delhi. Supported by a Janaki and K.A. Iyer Chair.

‡Tata Institute of Fundamental Research, Mumbai.

§Indian Institute of Technology Delhi. Supported by IIT Delhi's CSE Research Acceleration Fund.

number of active machines at such a time-point in the multiprocessor setting could range from just one to all available machines. As a result, there has been only one previous result for the multiprocessor setting; by Demaine et al. [9] who extended the dynamic program of Baptiste [6] and showed an $O(n^7m^5)$-time algorithm for the special case of unit-size jobs and $m \geq 1$ machines.

1.1 Our Contribution In Section 3 we present a pseudo-polynomial time algorithm for single machines that produces a feasible schedule of total energy at most $\texttt{OPT} + P$ where $\texttt{OPT}$ is the minimum energy of any fractional solution and P the sum of processing times. The algorithm is based on an elegant linear programming relaxation which we extend to the multiprocessor case in a later section. We show that the solution of the linear program relaxation can be decomposed into a convex combination of integer solutions. Since the relaxation has a strictly positive integrality gap, none of the integer solutions in the decomposition may be feasible. We overcome this by showing how an (infeasible) integer solution can be extended into a feasible solution while increasing the total energy consumption by only an additive P. Note that P is also a lower-bound on the optimal energy consumption and hence our algorithm can be viewed as a 2-approximation. Let n be the number of jobs and D the maximum deadline. We prove the following theorem in Section 3.

THEOREM 1.1. *There is an algorithm with running time polynomial in n, D for single machines that produces a schedule of total energy at most $\texttt{OPT} + P$.*

Building upon ideas for the single machine case, we develop, in Section 6 the first constant-factor approximation algorithm for the multiple machines case. Checking the feasibility of an instance and formulating a linear program to minimize energy is much more involved in the setting of multiple machines. The intervals comprising the integer solutions in the convex decomposition of the optimum fractional solution are not disjoint anymore, and extending the intervals appropriately in order to obtain feasibility is much more challenging now. We overcome these obstacles and present a pseudo-polynomial time algorithm that produces a feasible schedule of total energy at most $2\texttt{OPT} + P$. We prove the following theorem in Section 6

THEOREM 1.2. *There is an algorithm with running time polynomial in n, D for m parallel machines that produces a schedule of total energy at most $2\texttt{OPT} + P$.*

Finally, in the Appendix, we show that the running time of our algorithms can be made polynomial in $n, 1/\epsilon$; we incur a $(1 + \epsilon)$ loss in the approximation factor in this process.

1.2 Further Related Work An important generalization of our problem would be speed scaling with a sleep state, where the processor can vary its speed when in the active state in order to further save energy. The power consumption of the processor when it is active depends on its speed. In a processor with only speed scaling (and no sleep state) one tries to keep the processor speed as low as possible (since power is a convex function of speed). However with both speed scaling and a sleep state it is often beneficial to run the processor at faster speeds in order to increase the length of the subsequent sleep states, a technique commonly referred to as *race to idle*. Speed scaling with a sleep state was first introduced in [12] who gave a 2-approximation algorithm for the problem. This result was later improved to a 4/3-approximation by Albers and Antoniadis [3], and eventually to a fully polynomial time approximation scheme (FPTAS) by Antoniadis et al. [4]. This is the best result one can hope for (unless $\mathbf{P} = \mathbf{NP}$), as the problem is known to be **NP**-hard [3, 14].

Another problem similar to ours is that of minimizing the number of gaps (a gap is a contiguous interval during which the processor is idle) in the schedule. If one is interested in exact solutions then this is a special case of our problem since by choosing a large value for energy consumed in the active state we can ensure that every idle period results in a transition to the sleep state; thus the optimal schedule also minimizes the number of gaps. Chrobak et al. [8] gave a simple 2-approximation algorithm for the gap minimization problem with a running time of $O(n^2 \log n)$ and memory just $O(n)$. Demaine et al. [9] gave an exact algorithm for the multiprocessor gap minimization problem with unit-size tasks. Several further generalizations - for example the set-cover-hard case when each job has several disjoint release time-deadline intervals to choose from - of the problem were considered in [9, 10].

Finally, one may consider the setting where one knows exactly when the processor (or how many processors at each point in time) need to be active in order to execute jobs, and has to decide about when to transition the processor(s) between the states. Although the offline version of the problem with a single processor equipped just with one active and one sleep state becomes trivial, the online version turns out to be a generalization of the well-known ski-rental problem. Additionally considering processor(s) with sleep states of various depths (each having an individual power consumption and an individual cost for transitioning back to the active state) leads to many interesting algorithmic problems both in the offline and in the online scenarios that have been studied by Albers [2], Augustine et al. [5], as well as Irani et al. [13].

2 Preliminaries

We are given a set of jobs $\{j_1, j_2, \ldots, j_n\}$; job j_i has release time r_i, deadline d_i and processing time p_i and we assume that all these quantities are non-negative integers. Let r_{min} and d_{max} be the earliest release time and furthest deadline of any job; it is no loss of generality to assume $r_{min} = 0$ and $d_{max} = D$. For $t \in \mathbb{Z}_{\geq 0}$, let $[t, t+1]$ denote the t^{th} *time-slot*. Let $I = [t, t'], t, t' \in \mathbb{Z}_{\geq 0}, t < t'$ be an *interval*. The length of I, denoted by $|I|$ is $t' - t$. We use $t \in I = [a, b]$ to denote $a \leq t \leq b$.

Two intervals $I_1 = [a_1, b_1]$ and $I_2 = [a_2, b_2]$ *overlap* if there is a t such that $t \in I_1$ and $t \in I_2$. Thus two intervals which are right next to each other would also be considered overlapping. Intervals which do not overlap are considered *disjoint*. I_1 is *contained* in I_2, denoted $I_1 \subseteq I_2$, if $a_2 \leq a_1 < b_1 \leq b_2$ and it is *strictly contained* in I_2, denoted $I_1 \subset I_2$, if $a_2 < a_1 < b_1 < b_2$.

At any time-slot, a machine can be in the *active* or the *sleep* state. For each time-slot that a machine is in the active state, one unit of power is required whereas no power is consumed in the sleep state. However, L units of energy (called *wake up* energy) are expended when the machine transitions from the sleep to the active state. In its active state, the machine can either process a job (in which case we refer to it as being *busy*) or just be *idle*. On the other hand the machine cannot perform any processing while in the sleep state. Note that if a machine is not required to do any processing for A consecutive time-slots, then it is advantageous to transition it to the sleep state when $A > L$ whereas for $A \leq L$ it is preferable to keep it active but idle.

A machine can process at most one job in any time-slot and a job cannot be processed on more than one machine in a time-slot. However, job preemption and migration are allowed, i.e., processing of a job can be stopped at any time and resumed later on the same or on a different machine. A job j_i must be processed for p_i time-slots in $[r_i, d_i]$. Any assignment of jobs to machines and time slots satisfying the above conditions is called a (feasible) *schedule*. We assume that the machine is initially in the sleep state. Therefore, the energy consumed by a schedule is the total length of the intervals during which the machine is active plus L times the number of intervals in which the machine is active. The objective of the problem is to find a schedule which consumes minimum energy.

3 An Additive P Approximation for Single Machine

We first show how to schedule jobs on a single machine so that the total energy consumption is at most P more than the optimum. For any $[a, b] \subseteq [0, D]$ (recall D is the furthest deadline of any job), let $V(a, b) = \sum_{i:[r_i,d_i]\subseteq[a,b]} p_i$ be the total processing time of jobs whose release and deadline are within $[a, b]$. For an instance to be feasible it is necessary that for all $0 \leq a < b \leq D$, $V(a, b) \leq b - a$. The Earliest Deadline First (EDF) algorithm for scheduling jobs with release dates and deadlines can also be used to establish the sufficiency of this condition.

Motivated by this necessary and sufficient condition for determining if an instance is feasible, we consider the following Integer Program for minimizing total energy consumed. For $I \subseteq [0, D]$ let x_I be a variable which is 1 if the machine becomes active at the start of I and remains so till its end when it transitions back to the sleep state; x_I is 0 otherwise. Since the machine uses L units of energy to wake-up at the start of I and $|I|$ units to run during this interval, the objective is to minimize $\sum_I x_I(L + |I|)$. We next discuss the constraints of this IP.

1. The intervals in which the machine is active are disjoint and hence for $0 \leq t \leq D$, $\sum_{I:t\in I} x_I \leq 1$.
2. To ensure that jobs can meet release dates and deadlines when scheduled within active intervals we add the constraint that for all $0 \leq a < b \leq D$, $V(a, b) \leq \sum_I x_I |I \cap [a, b]|$.
3. For any job j_i, the machine should be active at some point during $[r_i, d_i]$. Hence $\sum_{I:I\cap[r_i,d_i]\neq\varnothing} x_I \geq 1$

This gives us the following integer program.

$$
\begin{array}{rcll}
\text{minimize} & & \sum_I x_I(L + |I|) & \\
\text{subject to} & & & \\
\sum_{I:t\in I} x_I & \leq & 1 & 0 \leq t \leq D-1 \\
\sum_I x_I |I \cap [a,b]| & \geq & V(a,b) & 0 \leq a < b \leq D \\
\sum_{I:I\cap[r_i,d_i]\neq\varnothing} x_I & \geq & 1 & 1 \leq i \leq n \\
x_I & \in & \{0,1\} & I \subseteq [0, D]
\end{array}
$$

Consider a feasible solution to this IP and let $\mathcal{I} = \{I | x_I = 1\}$. A time-slot $[t, t+1]$ is *active* if it is contained in some interval of $\mathcal{I}$.

LEMMA 3.1. *Every job j_i can be assigned to p_i active time slots in $[r_i, d_i]$ such that each active time-slot is assigned to at most 1 job.*

Proof. Construct a bipartite graph $G = (U, V, E)$. For every job j_i we have p_i vertices in U and for every active time slot we have a vertex in V. E has an edge between a vertex corresponding to job j_i and a vertex corresponding to the active time-slot $[t, t+1]$ iff $[t, t+1] \subseteq [r_i, d_i]$. We want to find a matching in G which matches all vertices of U.

For contradiction assume that there is no such matching. By Hall's theorem there exists a Hall set $S \subseteq U$ such that $|\Gamma(S)| < |S|$ where $\Gamma(S)$ are the vertices in V adjacent to vertices in S. Let S be a minimal Hall set. Two vertices in U corresponding to the same job have identical neighbors in V and hence it is no loss of generality to assume that S contains all vertices corresponding to the same job. This allows us to view S as a set of jobs; $|S|$ then equals the total processing time of the jobs in S.

Consider the union of intervals $[r_i, d_i]$ where j_i is a job in S. The minimality of S implies that this union is a single interval, say $[a, b]$. Note that $V(a, b) \geq |S|$ and $|\Gamma(S)|$ is the number of active time slots in $[a, b]$. From the second set of constraints of the IP it follows that $|S| \leq V(a, b) \leq |\Gamma(S)|$ which contradicts our assumption that S is a Hall set. ∎

The above claim implies that an optimum solution to the integer program gives a feasible schedule which minimizes energy. We relax the integrality constraint on x_I to $0 \leq x_I \leq 1$ and solve the resulting linear program. Let x be the optimum fractional solution and let $\mathcal{I} = \{I | x_I > 0\}$. We will next show that x be decomposed into a convex combination of integer solutions.

Ordering intervals in $\mathcal{I}$: Let $[a, d], [b, c] \in \mathcal{I}$, $[b, c] \subset [a, d]$ and $x_{[a,d]} = x_{[b,c]} = \alpha$. We replace these intervals in $\mathcal{I}$ with intervals $[a, c], [b, d]$ and set $x_{[a,c]} = x_{[b,d]} = \alpha$. Doing so does not make x infeasible nor does it change the objective value. If $\beta = x_{[a,d]} > x_{[b,c]} = \alpha$ then we replace these intervals in $\mathcal{I}$ with three intervals $[a, d], [a, c], [b, d]$ and set $x_{[a,d]} = \beta - \alpha$ and $x_{[a,c]} = x_{[b,d]} = \alpha$. The case when $\beta = x_{[a,d]} < x_{[b,c]} = \alpha$ is handled similarly. We repeat this process whenever an interval in $\mathcal{I}$ strictly contains another interval in $\mathcal{I}$. Finally, order the intervals in $\mathcal{I}$ by their start-times; intervals which have the same start-time are ordered by their end-times. Let $\prec$ denote this total order on intervals of $\mathcal{I}$. Note that since no interval is strictly contained in another, we would get the same ordering if intervals were ordered by their end-times with intervals having the same end-time ordered by their start-times.

Decomposing x into a convex combination of integer solutions: For $I \in \mathcal{I}$ let s_I be the fractional part of $\sum_{I' \prec I} x_{I'}$; thus $0 \leq s_I < 1$. For k, $0 \leq k < 1$ construct $\mathcal{I}_k \subseteq \mathcal{I}$ as follows: $I \in \mathcal{I}_k$ iff either $s_I \leq k < s_I + x_I$ or $s_I \leq k + 1 < s_I + x_I$.

Claim 3.1. *For any $k \in [0, 1)$, the intervals in $\mathcal{I}_k$ are pairwise disjoint.*

Proof. Let $I_1, I_2 \in \mathcal{I}_k$, $I_1 \prec I_2$ and $I_1 \cap I_2 \neq \varnothing$. Since $I_1, I_2 \in \mathcal{I}_k$ and $I_1 \prec I_2$, we get $\sum_{I_1 \preceq I \preceq I_2} x_I > 1$. Since I_1, I_2 are not disjoint, all intervals I such that $I_1 \preceq I \preceq I_2$ have a common overlap, say at time t. But this violates the LP-constraint $\sum_{I:t \in I} x_I \leq 1$ and yields a contradiction. ∎

Let $0 = s_1 < s_2 < \cdots < s_m < 1$ be the distinct values in the set $\{s_I, I \in \mathcal{I}\}$; note that $m \leq |\mathcal{I}|$. Let $s_{m+1} = 1$. From our construction of $\mathcal{I}_k$ it follows that for all $k \in [s_j, s_{j+1}), 1 \leq j \leq m$ the set $\mathcal{I}_k$ are identical; let $\mathcal{C}_j$ denote this set and we assign it a weight $w_j = s_{j+1} - s_j$. By Claim 3.1, each "solution" $\mathcal{C}_j, 1 \leq j \leq m$ is a set of disjoint intervals.

Claim 3.2. *The solutions $\mathcal{C}_j$ and weights w_j, $1 \leq j \leq m$, form a convex decomposition of the fractional solution x.*

Proof. First note that for all $1 \leq j \leq m$, $w_j \geq 0$ and $\sum_{j=1}^{m} w_j = 1$. Now consider an interval $I \in \mathcal{I}$ and let $s_I = s_a$ and $s_I + x_I = s_b$, $b > a$. The interval I appears in solutions $\mathcal{C}_a, \mathcal{C}_{a+1}, \ldots, \mathcal{C}_{b-1}$ and these have a total weight $s_b - s_a = x_I$. ∎

Remark: An alternate procedure to construct this convex decomposition of x would be to replace each interval $I \in \mathcal{I}$ with x_I/ϵ intervals where ϵ is such that x_I/ϵ is an integer for all $I \in \mathcal{I}$. Let $\mathcal{I}'$ be the multiset of intervals obtained. Consider intervals in $\mathcal{I}'$ in the order $\prec$ and assign them to solutions $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{1/\epsilon}$ in a round robin manner. Although easy to present, this procedure has the disadvantage that the number of solutions in the convex decomposition is $1/\epsilon$ and ϵ, which is the granularity of the fractional solution x, could be exponentially small. One could round x to multiples of ϵ for a suitable choice of ϵ but this would then incur a multiplicative constant in the approximation guarantee. The procedure presented above is conceptually similar to this round-robin assignment.

Extending Intervals: Although $\mathcal{C}_j, 1 \leq j \leq m$ is a set of disjoint intervals it need not be a feasible solution, i.e. it could be that jobs cannot meet release dates and deadlines if they have to be scheduled within intervals of $\mathcal{C}_j$. This is illustrated by the example in Figure 1, the details of which can be found in the Appendix.

We next show that we can extend the intervals in any solution $\mathcal{C}_j, 1 \leq j \leq m$ by at most P units to get a feasible solution, $\mathcal{C}'_j$.

Lemma 3.2. *Let $\mathcal{C} = \mathcal{C}_j, 1 \leq j \leq m$ be a solution from the convex decomposition of x. $\mathcal{C}$ can be converted into a feasible solution $\mathcal{C}'$ by increasing the total length of intervals in $\mathcal{C}$ by at most P.*

Proof. A slot $[t, t+1]$ is *active* if it is contained in some interval in $\mathcal{C}$. Let $s(a, b)$ be the number of active slots in

Job	Release	Deadline
1	0	1
2	1	7
3	2	4
4	4	6
5	7	8

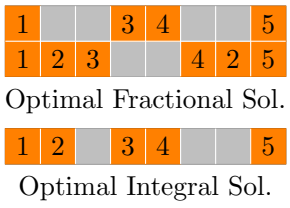

Figure 1: An instance where solutions in the convex decomposition are not all feasible. All jobs are unit size. The top right shows the two solutions $\mathcal{C}_1, \mathcal{C}_2$ in the convex decomposition of the optimum fractional solution. The total length of intervals in $\mathcal{C}_1$ is 4 which is less than the total processing time of jobs and implies $\mathcal{C}_1$ is infeasible.

the interval $[a, b] \subseteq [0, D]$ and $\delta(a, b) = \max(0, V(a, b) - s(a, b))$ its *deficiency*.

If $\mathcal{C}$ is infeasible there exists $[a, b]$ such that $\delta(a, b) > 0$. Among all intervals with positive deficiency consider those whose end-time is the least and let these be $[a_1, t], [a_2, t], \ldots, [a_k, t]$ where $t > a_1 > a_2 > \cdots > a_k$. Let P_t be the total processing time of jobs whose deadline is t. For $1 \leq i \leq k$, $V(a_i, t) \leq V(a_i, t-1) + P_t$ and since $\delta(a_i, t-1) = 0$ we have $\delta(a_i, t) \leq P_t$.

We now show how to extend intervals in $\mathcal{C}$ by P_t time-slots so that deficiency of intervals $[a_i, t], 1 \leq i \leq k$ becomes 0.

Claim 3.3. *$\mathcal{C}$ contains an interval which overlaps $[a_1, t]$.*

Proof. $\delta(a_1, t) > 0$ implies $V(a_1, t) > 0$ which in turn implies that there exists a job j_i such that $[r_i, d_i] \subseteq [a_1, t]$. The third set of constraints of the integer program ensure that the sum of x_I where $I \in \mathcal{I}$ and $I \cap [r_i, d_i] \neq \phi$ is at least 1. By our procedure for building the convex decomposition it follows that at least one of these intervals is in $\mathcal{C}$. Since this interval overlaps $[r_i, d_i]$ it also overlaps $[a_1, t]$ proving the claim. ∎

Let $I \in \mathcal{C}$ overlap $[a_1, t]$. We first extend I to the right till we have included time-slot $[t-1, t]$ and continue by extending I to the left, perhaps combining with other intervals of $\mathcal{C}$ in this process. We stop when P_t time-slots have been added or when all time-slots before t have been included. Consider the interval $[a_i, t]$. Either we have added P_t time slots in this interval or extended I to include all time-slots in this interval. In the former case the deficiency of $[a_i, t]$ is reduced to 0. In the later case $s(a_i, t) = t - a_i \geq V(a_i, t)$, where the second inequality follows from the fact that the instance is feasible. Hence $\delta(a_i, t) = 0$.

After having reduced to zero the deficiency of all intervals ending at t, we find the next set of intervals with positive deficiency whose end-time is the least. The process continues till all intervals have zero deficiency. Note that the intervals of $\mathcal{C}$ are extended by at most $\sum_t P_t = P$ time-slots. ∎

Since the number of intervals in $\mathcal{C}'_j$ equals the number of intervals in $\mathcal{C}_j$ and the total length of intervals in $\mathcal{C}'_j$ exceeds the total length of intervals in $\mathcal{C}_j$ by at most P, the energy consumed by the solution $\mathcal{C}'_j$ is at most P more than the energy consumed by $\mathcal{C}_j$. Since this is true for all solutions $\mathcal{C}'_j, 1 \leq j \leq m$, the solution of minimum cost among these has cost at most P more than the optimum fractional solution.

Theorem 3.1. *Given n jobs with release dates, processing times and deadlines in $[0, D]$, there is an algorithm with running time polynomial in n, D which schedules these jobs on a single machine such that the total energy consumption is at most $\texttt{OPT} + P$ where P is the sum of processing times.*

4 Deadline Scheduling on Parallel Machines

In this section we prove a necessary and sufficient condition for scheduling jobs on m parallel machines so that all release dates and deadlines are met. While this is a standard problem in an undergraduate Algorithms course we repeat the argument here since it will be useful in developing the linear program for minimizing energy consumption in the next section.

Recall we are given n jobs. Job $j_i, 1 \leq i \leq n$ requires p_i units of processing, is released at time r_i and has deadline d_i. The jobs are to be scheduled on m identical machines and we allow for preemption and migration. An instance is feasible iff for every job $j_i, 1 \leq i \leq n$ we can assign p_i distinct time-slots during $[r_i, d_i]$ such that no time-slot is assigned to more than m jobs.

For reasons that will become clear later, we consider a minor generalization of the above problem which we refer to as `deadline-scheduling-on-intervals`. Instead of m machines, we are given k supply-intervals, $\mathcal{I} = \{I_1, I_2, \ldots, I_k\}$ and are required to schedule the given jobs within these intervals. Let s_j, t_j denote the start and end-times of interval I_j. The intervals in $\mathcal{I}$ need not be disjoint; however any point in time is contained in at most m intervals. Note that if each interval in $\mathcal{I}$ was $[0, D]$ then we would recover the problem of scheduling on parallel machines. An instance of this problem is thus specified by the processing time, release date and deadline of each of the n jobs and the start and end-times of the k supply-intervals. The feasibility of an instance can be checked by formulating it as a problem of finding a flow in a suitable network.

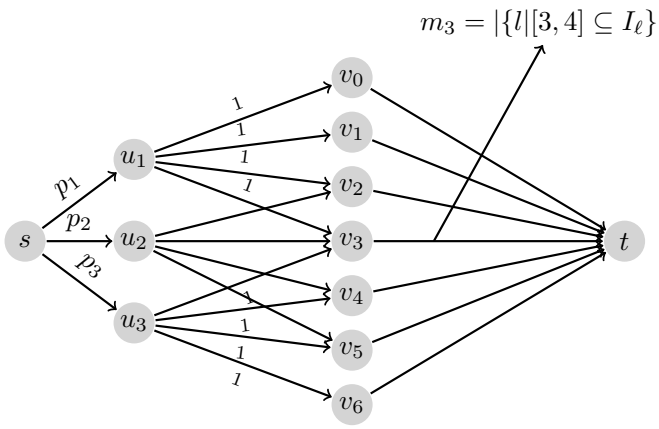

Figure 2: Network $G = (V, E)$ for checking feasibility of an instance

Construct a network $G = (V, E)$ with source s, sink t, a vertex u_i for each job j_i and a vertex v_t for each time-slot $[t, t+1], 0 \leq t \leq D-1$. Vertex u_i has edges to vertices $\{v_t | [t, t+1] \subseteq [r_i, d_i]\}$ of capacity 1 and an edge from s of capacity p_i. Let m_t be the number of intervals in $\mathcal{I}$ which contain the time-slot $[t, t+1]$. Vertex v_t has an edge to the sink t of capacity m_t. Let $c : E \to \mathbb{R}^+$ denote the capacity function on the edges.

The s-t cut $(\{s\}, V - \{s\})$ has capacity $P = \sum_{j=1}^{n} p_j$ and so the maximum flow between s and t cannot exceed P.

LEMMA 4.1. *An instance of* `deadline-scheduling-on-intervals` *is feasible iff P units of flow can be sent from s to t in the network G with capacities given by c.*

Proof. Let $f : E \to \mathbb{Z}_{\geq 0}$ be an s-t flow of value P. Since edge capacities are integral f can also be assumed to be integral. We use f to determine an assignment of jobs to time-slots. If $f(u_i, v_t) = 1$ then we assign job j_i to the time-slot $[t, t+1]$. Since $f(v_t, t) \leq m_t$ the number of jobs assigned to time-slot $[t, t+1]$ cannot exceed the number of intervals in $\mathcal{I}$ containing this time-slot. Since f has value P which is the capacity of the cut $(\{s\}, V - \{s\})$, all edges incident to s are saturated. Hence $f(s, u_i) = p_i$ which implies that job j_i is assigned to exactly p_i time-slots in $[r_i, d_i]$. This assignment of jobs to time-slots is therefore a feasible schedule.

For the converse, consider a schedule, $\mathcal{S}$, which respects release dates and deadlines. We build a flow f from s to t of value P. If job j_i is processed in time-slot $[t, t+1]$ in $\mathcal{S}$ then $f(u_i, v_t) = 1$; since $[t, t+1] \subseteq [r_i, d_i]$, the edge (u_i, v_t) is in E and has capacity 1. The flow on edges entering t and leaving s is determined by conservation. Note that at most m_t jobs could be scheduled in the time-slot $[t, t+1]$ and hence the flow on edge (v_t, t) does not exceed its capacity. Since in schedule $\mathcal{S}$, job j_i is processed for p_i units, the flow on edge (s, u_i) equals p_i which implies that the total flow from s to t is P. ∎

Let $(S, \overline{S})$ be an s-t cut and $c(S)$ denote its capacity.

CLAIM 4.1. *If $c(S) < P$ then $S \cap \{v_0, v_1, \ldots, v_{D-1}\} \neq \varnothing$.*

Proof. For contradiction assume that S does not contain any vertex from the set $\{v_0, v_1, \ldots, v_{D-1}\}$. Then the capacity of the cut $(S, \overline{S})$ is $\sum_{i:u_i \in S}(d_i - r_i) + \sum_{i:u_i \notin S} p_i$. If for job j_i, $d_i - r_i < p_i$ then the instance is trivially infeasible. Hence we assume that $d_i - r_i \geq p_i, 1 \leq i \leq n$, and this implies that the capacity of the cut $(S, \overline{S})$ is at least $\sum_{i=1}^{n} p_i = P$. ∎

We aggregate the time-slots corresponding to vertices in $S \cap \{v_0, v_1, \ldots, v_{D-1}\}$ into a minimal set of intervals, $Q(S)$. No two intervals in $Q(S)$ are overlapping since we could combine them and obtain a smaller set of intervals. Recall that if two intervals share an end-point then we consider them overlapping.

DEFINITION 4.1. *The* forced volume *of a job j_i with respect to an interval $[a, b]$, denoted by* `fv`$(j_i, [a, b])$*, is the minimum volume of j_i that must be processed during $[a, b]$ in any feasible schedule. Let Q be a set of disjoint intervals. The* forced volume *of job j_i with respect to Q denoted by* `fv`(j_i, Q)*, is the minimum volume of j_i that must be processed during the intervals in Q in any feasible schedule.*

If I_1, I_2 are disjoint intervals then `fv`(j_i, I_1) + `fv`$(j_i, I_2) \leq$ `fv`$(j_i, I_1 \cup I_2)$. For instance suppose $I_1 = [0, 3]$, $I_2 = [5, 8]$, $r_1 = 2$, $d_1 = 6$ and $p_1 = 3$. Then `fv`(j_1, I_1)=`fv`(j_1, I_2)=0 but `fv`$(j_1, I_1 \cup I_2)$=1. Note that the forced volume of a job j_i with respect to an interval $[a, b]$ is independent of the supply-intervals and depends only a, b, p_i, r_i and d_i. For instance, if $r_i < a < d_i < b$ then `fv`$(j_i, [a, b]) = \max(0, r_i + p_i - a)$. Similarly, if $r_i \leq a < b \leq d_i$ then `fv`$(j_i, [a, b]) = \max(0, p_i - (a - r_i) - (d_i - b))$. Note that if the total length of intervals in $Q \cap [r_i, d_i]$ is Q_i, then `fv`$(j_i, Q) = \max\{0, p_i - |d_i - r_i| + Q_i\}$.

DEFINITION 4.2. *Let Q be a set of disjoint intervals. The deficiency of Q, denoted by* `def`(Q)*, is the non-negative difference between the sum of the forced volume of all jobs with respect to Q and the total volume of jobs that can be processed in Q. Thus*

$$\texttt{def}(Q) = \max\left(0, \sum_{i=1}^{n} \texttt{fv}(j_i, Q) - \sum_{t:[t,t+1] \subseteq Q} m_t\right).$$

Note that deficiency of Q also depends on the supply intervals in the instance. From the above definition it follows that if a set of disjoint intervals, Q, has positive deficiency then the instance is infeasible. The following lemma will help us argue the converse.

LEMMA 4.2. *Let $(S, \overline{S})$ be a s-t cut in G. Then* $\texttt{def}(Q(S)) + c(S) \geq P$. *The inequality holds with an equality if $(S, \overline{S})$ is a minimum s-t cut.*

Proof. We consider each vertex in S and count the total capacity of edges in the cut $(S, \overline{S})$ incident to this vertex.

1. For the source s, this quantity is $\sum_{i:u_i \notin S} p_i$.
2. Let $u_i \in S$ and c_i be the number of edges from u_i to vertices in $\overline{S}$. If $c_i > p_i$ then $\texttt{fv}(j_i, Q(S)) = 0$ and if $c_i \leq p_i$ then $\texttt{fv}(j_i, Q(S)) = p_i - c_i$. Hence $c_i \geq p_i - \texttt{fv}(j_i, Q(S))$.
3. If $v_t \in S$ then the edge (v_t, t) of capacity m_t is in $(S, \overline{S})$.

Combining these we get

$$\begin{aligned} c(S) &\geq \sum_{i:u_i \notin S} p_i + \sum_{i:u_i \in S} (p_i - \texttt{fv}(j_i, Q(S))) \\ &\quad + \sum_{t:[t,t+1] \subseteq Q(S)} m_t \\ &\geq \sum_{i=1}^{n} p_i - \sum_{i=1}^{n} \texttt{fv}(j_i, Q(S))) + \sum_{t:[t,t+1] \subseteq Q(S)} m_t \\ &\geq P - \texttt{def}(Q(S)) \end{aligned}$$

which proves the first part of the lemma.

Let $(S, \overline{S})$ be a minimum s-t cut.

1. If $u_i \in S$ then $c_i \leq p_i$ or else we would have moved u_i to $\overline{S}$ to obtain a cut of smaller capacity. Hence $c_i = p_i - \texttt{fv}(j_i, Q(S))$.
2. In a maximum s-t flow, flow on edge (u_i, v_t), $u_i \notin S, v_t \in S$, is 0. Since p_i units enter u_i, this implies that $\texttt{fv}(j_i, Q(S)) = 0$ and hence $\sum_{i:u_i \in S}(p_i - \texttt{fv}(j_i, Q(S))) = \sum_{i=1}^{n} \texttt{fv}(j_i, Q(S)))$.

The above two observations imply that $c(S) = P - \texttt{def}(Q(S))$ which proves the second part of the Lemma. ∎

By Lemma 4.1 an infeasible instance has a cut $(S, \overline{S})$ such that $c(S) < P$. Lemma 4.2 then implies that $\texttt{def}(Q(S)) > 0$ which proves the following theorem.

THEOREM 4.1. *An instance of* `deadline-scheduling-on-intervals` *is feasible iff no set of disjoint intervals has positive deficiency.*

Making an instance feasible: Given an infeasible instance of `deadline-scheduling-on-intervals`, we would like to extend the intervals of the instance to make it feasible. We need some additional tools to do this and shall take this up in a later section. Let $F < P$ be the maximum s-t flow in the network G corresponding to this instance. We now show that an s-t flow of value P can be routed in G by increasing capacities of edges incident to the sink such that the total increase in capacities is $P - F$.

By submodularity of the cut-function it follows that if $(S_1, \overline{S_1})$, $(S_2, \overline{S_2})$ are minimum s-t cuts then $(S_1 \cap S_2, \overline{S_1 \cap S_2})$ is also a minimum s-t cut. Hence a minimum s-t cut in which the side containing the source is minimal is unique; let $(S, \overline{S})$ be this cut. Since the capacity of this cut is less than P, by Claim 4.1 it follows that $S \cap \{v_0, v_1, \ldots, v_{D-1}\} \neq \varnothing$.

CLAIM 4.2. *Increasing the capacity of* any *edge $(v_i, t), v_i \in S$ by 1 increases the s-t max-flow in G by 1.*

Proof. For contradiction assume that increasing the capacity of edge (v_i, t) does not increase the s-t max-flow in G. Hence there is a minimum s-t cut, $(X, \overline{X})$, such that $v_i \notin X$. Since $v_i \in S$, this means $S \not\subseteq X$ which implies that S is not minimal. ∎

Claim 4.2 gives us an algorithm for increasing capacities. At each step we find a minimum s-t cut in which the side containing the source is minimal and increase the capacity of any edge in this cut which is also incident to the sink by 1. Since with every step, we increase the s-t flow in G by 1, the number of steps, and the total increase in edge capacities, equals $P - F$.

Claim 4.2 also implies that $(S, \overline{S})$ remains a minimum s-t cut in G even after we increase the capacity of edge $(v_i, t), v_i \in S$, by 1; however S need not be minimal. Let $(S', \overline{S'})$ be the new s-t minimum cut in which the side containing the source is minimal. The fact that $(S, \overline{S})$ is a minimum s-t cut implies that $S' \subseteq S$. Thus with every step the s-side of the cut under consideration shrinks. This is an important property of this process and shall find use later.

5 Linear Programming Relaxation

We are now ready to give a linear programming relaxation for the problem of scheduling jobs on parallel machines so as to minimize total energy consumed. A solution to the problem is completely specified by the set of time intervals in which each machine is active; let $\mathcal{I}$ be this multiset. The energy consumed by this solution equals $\sum_{I \in \mathcal{I}}(|I| + L)$. Note that at most m intervals in $\mathcal{I}$ can overlap at any point in time. Further, $\mathcal{I}$ forms a feasible solution if the corresponding instance of `deadline-scheduling-on-intervals` is feasible.

With every interval $I \subseteq [0, D]$ we associate a variable $x(I), 0 \leq x(I) \leq m$ which indicates the number of times I is picked in a solution. The objective is to minimize $\sum_I x(I)(|I| + L)$. We now list the constraints of this linear program.

1. Let $m_t = \sum_{I:[t,t+1]\subseteq I} x(I)$. Since at most m intervals overlap at any time t we get that for all $t, 0 \leq t \leq D-1$, $m_t \leq m$.

2. Let $f(i,t)$ be a variable denoting the flow in the edge $(u_i, v_t), 0 \leq i \leq n, 0 \leq t \leq D-1$ in the flow network G corresponding to this instance. Then $0 \leq f(i,t) \leq 1$.

3. The conservation constraint on vertex v_t and the capacity constraint on edge (v_t, t) together give: for all $t, 0 \leq t \leq D-1$, $\sum_{i:[t,t+1]\subseteq[r_i,d_i]} f(i,t) \leq m_t$.

4. Since P units of flow have to be routed, all edges incident to the source are saturated. This together with the conservation constraint at vertex u_i yields: for all i, $\sum_{t=r_i}^{d_i-1} f(i,t) = p_i$.

5. Consider an interval $[a,b] \subseteq [0,D]$. The total forced volume of all jobs with respect to $[a,b]$ equals $\sum_{i=1}^{n} \texttt{fv}(j_i, [a,b])$. If this quantity equals $\alpha(b-a)$ then the number of intervals overlapping $[a,b]$ should be at least $\lceil \alpha \rceil$. This yields the constraint: for all $0 \leq a < b \leq D$,

$$\sum_{I:[a,b]\cap I \neq \varnothing} x_I \geq \left\lceil \frac{\sum_{i=1}^{n} \texttt{fv}(j_i,[a,b])}{b-a} \right\rceil.$$

Thus our linear program for scheduling on multiple machines to minimize energy is as follows.

$$\begin{array}{rcll} \text{minimize} & & \sum_I x(I)(|I|+Q) & \\ \text{subject to} & & & \\ m_t & = & \sum_{I:[t,t+1]\in I} x(I) & 0 \leq t < D \\ m_t & \geq & \sum_{i:r_i \leq t \leq d_i-1} f(i,t) & 0 \leq t < D \\ p_i & = & \sum_{t=r_i}^{d_i-1} f(i,t) & 0 \leq i \leq n \\ \sum_{I:[a,b]\cap I\neq\varnothing} x_I & \geq & \lceil \sum_{i=1}^{n} \frac{\texttt{fv}(j_i,[a,b])}{(b-a)} \rceil & 0 \leq a < b \leq D \\ f(i,t) & \in & [0,1] & \forall i,t \\ x(I), m_t & \in & [0,m] & \forall t, I \subseteq [0,D] \end{array}$$

6 Minimizing Energy on Parallel Machines

Our algorithm for the case of parallel machines is along the lines of the one for single machines. We begin by solving the linear program from Section 5 and let x be the optimum fractional solution and $\texttt{OPT}$ the cost of this solution. Our algorithm will produce a solution of cost at most $\texttt{2OPT} + P$.

Let $\mathcal{I} = \{I | x_I > 0\}$. After ensuring that no interval of $\mathcal{I}$ is strictly contained in another, we order the intervals by increasing start-times (breaking ties using end-times) and let $\prec$ be this order. As in Section 3, we construct r integral solutions, $\mathcal{C}_i, 1 \leq i \leq r$ and associate weights w_i with solutions $\mathcal{C}_i$ such that this forms a convex decomposition of x. Note that $\mathcal{C}_i$ is no more a disjoint set of intervals as in the single machine case. However at most m intervals of $\mathcal{C}_i$ could overlap at any point in time.

For the rest of this section we will consider one of the integral solutions in the convex decomposition and refer to it as $\mathcal{C}$. The arguments of this section will apply to all r solutions. Note that $\mathcal{C}$ need not be a feasible instance of `deadline-scheduling-on-intervals` and we will modify the intervals in $\mathcal{C}$ to make it a feasible solution. Let $I_1 \prec I_2 \prec \cdots \prec I_N$ be the intervals in $\mathcal{C}$.

LEMMA 6.1. *Suppose $[a,b] \subseteq [0,D]$ overlaps l intervals of $\mathcal{C} = \mathcal{C}_i$. Then $[a,b]$ overlaps at most $l+1$ intervals of $\mathcal{C}_k, k \neq i$.*

Proof. From our round-robin procedure for assigning intervals to solutions in the convex decomposition it follows that for any $1 \leq i \leq N-1$, $\mathcal{C}_k$ contains exactly one interval I between I_i and I_{i+1} i.e. $I_i \prec I \prec I_{i+1}$. Suppose $[a,b]$ overlaps intervals $I_j, I_{j+1}, \ldots, I_{j+l-1}$ of $\mathcal{C}$. Then $[a,b]$ would definitely overlap the $l-1$ intervals of $\mathcal{C}_k$ between I_j and I_{j+l-1}. In addition $[a,b]$ could possibly overlap the two intervals of $\mathcal{C}_k$ between I_{j-1} and I_j and between I_{j+l-1} and I_{j+l}. Thus $[a,b]$ could overlap at most $l+1$ intervals of $\mathcal{C}_k$. ∎

Modifying intervals: Let s_j, e_j denote the start and end times of interval $I_j \in \mathcal{C}$. We consider the intervals in the order $\prec$ and modify them as follows:

> If I_j, I_{j+1} overlap then replace I_j with the interval $[s_j, \min\{e_{j+1}, s_{j+m}\}]$. Else create a copy of I_{j+1} if it does not overlap I_{j+m}.

For $j = 0$ we add a copy of I_1 if it does not overlap I_m. The set of intervals formed through this modification continue to have the property that no interval is strictly contained in another although now we could have two copies of some intervals. Let $I'_1 \prec I'_2 \prec \cdots \prec I'_M$ be the new (multi)set of intervals which we denote by $\mathcal{C}'$.

CLAIM 6.1. *The sets $\mathcal{C}$ and $\mathcal{C}'$ relate as:*

1. *The total length of the intervals in $\mathcal{C}'$ is at most twice the total length of intervals in $\mathcal{C}$.*

2. *The number of intervals in $\mathcal{C}'$ is at most twice the number of intervals in $\mathcal{C}$.*

3. *If $[a,b] \subseteq [0,D]$ overlaps $0 < l < m$ intervals of $\mathcal{C}$ then it overlaps at least $l+1$ intervals of $\mathcal{C}'$.*

4. *At most m intervals of $\mathcal{C}'$ overlap at any point in time.*

Proof. The first 2 statements follow from our procedure for constructing $\mathcal{C}'$. The final statement of the claim follows from the fact that we add a copy of interval I_{j+1} only if it does not overlap I_{j+m}, and never extend I_j further than s_{j+m}.

To prove the third statement, suppose $[a,b]$ overlaps intervals $I_j, I_{j+1}, \ldots, I_{j+l-1}$ of $\mathcal{C}$. Then $[a,b]$ would also overlap the corresponding intervals of $\mathcal{C}'$. Since $l < m$, I_j does not overlap I_{j+m-1}.

Let $j > 1$. If I_{j-1} overlaps I_j then $[a,b]$ would also overlap the interval in $\mathcal{C}'$ that replaced I_{j-1}. If I_{j-1} does not overlap I_j then $\mathcal{C}'$ would contain a copy of I_j which $[a,b]$ would overlap. Finally if $j = 1$ then we would have created a copy of I_1 in $\mathcal{C}'$ which $[a,b]$ would overlap. Thus $[a,b]$ would overlap at least $l+1$ intervals of $\mathcal{C}'$. ∎

Extending Intervals: We will now extend intervals in $\mathcal{C}'$, without creating any new ones, to obtain a feasible instance of `deadline-scheduling-on-intervals`. We begin by running the feasibility test of Section 4 on the instance whose supply-intervals are the intervals of $\mathcal{C}'$. Suppose the test fails and returns a set of intervals $Q = \{Q_1, Q_2, \ldots Q_k\}$ of maximum deficiency. Let $I' \in \mathcal{C}'$ be such that it overlaps Q_i without containing Q_i i.e. $Q_i \not\subseteq I'$. Then a time-slot in Q_i can be used to extend I' and doing this decreases the deficiency of Q by 1. Recall that this also decreases the maximum deficiency of any set of intervals by 1. We modify the intervals in $\mathcal{C}'$ in this manner, always extending an interval of $\mathcal{C}'$ by a time-slot contained in one of the intervals comprising the set of intervals with maximum deficiency. We stop when it is not possible to extend an interval of $\mathcal{C}'$ in this manner and will now argue that the $\mathcal{C}'$ thus obtained is a feasible instance of `deadline-scheduling-on-intervals`.

The intervals comprising Q shrink during the above procedure and let $\{Q_1, Q_2, \ldots, Q_k\}$ be the set of intervals with maximum deficiency when we stop. Let $Q_i = [a_i, b_i]$ and c_i be the number of intervals of $\mathcal{C}'$ which overlap Q_i.

LEMMA 6.2. *The number of intervals in $\mathcal{C}_j, 1 \le j \le r$ which overlap Q_i is at most c_i.*

Proof. If $c_i = 0$ then no interval in $\mathcal{C}'$ overlaps $Q_i = [a_i, b_i]$. Since in going from $\mathcal{C}$ to $\mathcal{C}'$ we have only extended intervals or introduced new intervals, this implies that no interval in $\mathcal{C}$ overlaps $[a_i, b_i]$. By our convex decomposition procedure this implies that $\sum_{I: I \cap [a_i,b_i] \neq \varnothing} x_I < 1$. Since x is a feasible solution to the linear program (Section 5) we conclude that $\sum_{k=1}^{n} \texttt{fv}(j_k, [a_i, b_i]) = 0$.

Since Q is a minimal set of intervals with maximum deficiency, $\texttt{def}(Q \setminus Q_i) < \texttt{def}(Q)$. Since no interval of $\mathcal{C}'$ overlaps Q_i this implies $\sum_{k=1}^{n} \texttt{fv}(j_k, Q \setminus Q_i) < \sum_{k=1}^{n} \texttt{fv}(j_k, Q)$. Hence there exists a job j_k such that $\texttt{fv}(j_k, Q \setminus Q_i) < \texttt{fv}(j_k, Q)$. This implies that $[r_k, d_k] \cap Q_i \neq \varnothing$. Further $[r_k, d_k] \not\subseteq Q_i$ as that would imply $\texttt{fv}(j_k, Q_i) > 0$. Hence either $r_k < a_i < d_k$ or $r_k < b_i < d_k$; note that both conditions could also be true.

If $r_k < a_i < d_k$ then expanding Q_i to $[a_i - 1, b_i]$ would increase $\texttt{fv}(j_k, Q)$ by 1. Since Q is a set of intervals with maximum deficiency, some interval of $\mathcal{C}'$ must include the time-slot $[a_i - 1, a_i]$. Similarly, if $r_k < b_i < d_k$ then by expanding Q_i to $[a_i, b_i + 1]$ we conclude that an interval of $\mathcal{C}'$ contains $[b_i, b_i + 1]$. In either case, we have an interval of $\mathcal{C}'$ overlapping $[a_i, b_i]$ which implies $c_i > 0$.

By the third statement of Claim 6.1 the number of intervals in $\mathcal{C}$ overlapping Q_i is at most $c_i - 1$. Then by Lemma 6.1 the number of intervals in $\mathcal{C}_j$ overlapping Q_i is at most c_i and this proves the lemma. ∎

Consider x, the optimum solution to the LP. For all t such that $[t, t+1] \subseteq Q_i$ we have,

$$\begin{aligned} m_t &= \sum_{I: [t,t+1] \subseteq I} x_I = \sum_{j=1}^{r} w_j |\{I \in \mathcal{C}_j, [t, t+1] \subseteq I\}| \\ &\le \sum_{j=1}^{r} w_j c_i = c_i, \end{aligned}$$

where the inequality follows from Lemma 6.2.

Consider the cut $(S, \overline{S})$ where

$$S = \{s\} \cup \{v_t | [t, t+1] \in Q\} \cup \{u_i | \texttt{fv}(j_i, Q) > 0\}.$$

The capacity of this cut is

$$\begin{aligned} &\sum_{t: [t,t+1] \subseteq Q} m_t + \sum_{i=1}^{n} (p_i - \texttt{fv}(j_i, Q)) \\ \le\ & P - \left(\sum_{i=1}^{n} \texttt{fv}(j_i, Q) - \sum_{i=1}^{k} c_i |Q_i| \right) = P - \texttt{def}(Q), \end{aligned}$$

where $\texttt{def}(Q)$ is the deficiency of the set of intervals Q for an instance of `deadline-scheduling-on-intervals` defined by intervals of $\mathcal{C}'$. If $\texttt{def}(Q) > 0$ then $c(S) < P$ which contradicts the feasibility of x. Thus $\texttt{def}(Q) = 0$ and so the intervals of $\mathcal{C}'$ form a feasible solution.

By Claim 6.1, the total energy consumption of intervals in $\mathcal{C}'$ is at most twice that of the intervals in $\mathcal{C}$. Our procedure for extending intervals in $\mathcal{C}'$ increases their total length, and hence the total energy, by at most P. Hence the solution of minimum cost among $\mathcal{C}'_j, 1 \leq j \leq r$ has cost at most $2\texttt{OPT} + P$ where $\texttt{OPT}$ is the cost of the optimum fractional solution.

THEOREM 6.1. *Given n jobs with release dates, processing times and deadlines in $[0, D]$, there is an algorithm with running time polynomial in n, D which schedules these jobs on m machines such that the total energy consumption is at most $2\texttt{OPT} + P$ where P is the sum of processing times.*

7 Conclusions

The two algorithms with running times polynomial in n and D can be converted to polynomial time algorithms by limiting the number of intervals we consider in the linear program and by suitably modifying our procedure for extending the intervals in the integral solutions of the convex decomposition (see Appendix). We believe that our approach of formulating this problem of minimizing energy as a linear program and the tools we develop in this paper for rounding the fractional solutions, hold much promise and can be applied to more general machine models and power management techniques.

References

[1] How Dirty is your Data? `https://www.greenpeace.org/international/publication/7196/how-dirty-is-your-data/`, 2011.

[2] Susanne Albers. On energy conservation in data centers. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 35–44, 2017.

[3] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms*, 10(2):9:1–9:31, 2014.

[4] Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1102–1113, 2015.

[5] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. *SIAM J. Comput.*, 37(5):1499–1516, 2008.

[6] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 364–367. Society for Industrial and Applied Mathematics, 2006.

[7] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial time algorithms for minimum energy scheduling. In *European Symposium on Algorithms*, pages 136–150. Springer, 2007.

[8] Marek Chrobak, Uriel Feige, Mohammad Taghi Hajiaghayi, Sanjeev Khanna, Fei Li, and Seffi Naor. A greedy approximation algorithm for minimum-gap scheduling. *Journal of Scheduling*, 20(3):279–292, 2017.

[9] Erik D Demaine, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Amin S Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 46–54. ACM, 2007.

[10] Erik D. Demaine and Morteza Zadimoghaddam. Scheduling to minimize power consumption using submodular functions. In *SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, Thira, Santorini, Greece, June 13-15, 2010*, pages 21–29, 2010.

[11] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[12] Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4):41, 2007.

[13] Sandy Irani, Sandeep K. Shukla, and Rajesh K. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embedded Comput. Syst.*, 2(3):325–346, 2003.

[14] Gunjan Kumar and Saswata Shannigrahi. On the NP-hardness of speed scaling with sleep state. *Theor. Comput. Sci.*, 600:1–10, 2015.

Appendix

A From Pseudopolynomial to Polynomial Time

In this section we prove that it is sufficient to limit ourselves to intervals with start and endpoints from a set W of polynomially many time-slots in $[0, D]$, with the loss of a small factor in the approximation ratio.

DEFINITION A.1. *Let $T := \cup_i \{r_i, d_i\}$ and $W := T \cup \{w | w \in [0, D], \exists t \in T, k \in \mathbb{N}$ with $|t - w| = \lceil (1+\epsilon)^k \rceil\} \cup \{0, D\}$.*

CLAIM A.1. *$|W|$ is polynomial in the input size, and therefore so is the number of possible intervals that start and end at time-slots of W.*

Proof. Consider some $t \in T$. We argue about the number of distinct $w \in [0, D]$, so that $|t - w| = \lceil (1+\epsilon)^k \rceil$ for some $k \in \mathbb{N}$. Since $t, w \in [0, D]$, we have that for any such w, $|t - w| \leq D + 1$ and therefore for the corresponding k, $k = O(\log D)$. In turn $|W| = O(n \log D)$, and the number of possible intervals starting and ending at W is $|W|^2 = O(n^2 \log^2 D)$. ∎

LEMMA A.1. *Considering only intervals that start and end at time-slots of W, does not increase the cost of being in the active state by more than a factor of* $(1+\epsilon)$.

Proof. Consider an optimal solution OPT. We will transform OPT to a solution that satisfies the lemma property while increasing its active cost by at most an $(1+\epsilon)$ factor. We can associate intervals in OPT with processors as follows. Recall that the intervals are ordered by their start-time and ties are broken by end-times. Go through the intervals in this order and associate each interval to the smallest-index processor so that it does not overlap with any other interval already there. We will use the following claim to prove the lemma:

CLAIM A.2. *Assuming that* $I \cap T \neq \varnothing$ *holds for any interval* $I \in$ OPT *is without loss of generality.*

Consider some interval $I \in$ OPT, associated with a processor m_I and let $t \in I$ be a time slot of T whose existence is guaranteed by Claim A.2.

We will expand I towards the left and the right respectively until we hit either some time slot in W or we hit another interval associated with this processor. In the second case we merge the two intervals. We repeat this for every interval, and the process will terminate since in each step we either "snap" one of the endpoints to a point in W or reduce the number of intervals by one. Note that eventually all interval endpoints will be slots in W (W includes 0 and D).

The total increase in length of an interval I is at most $(1+\epsilon)\cdot|I|$, because we expand towards the left by at most a factor $(1+\epsilon)\cdot|t-s_I|$, and similarly towards the right by at most a factor $(1+\epsilon)\cdot|e_I-t|$. This is because by construction there are points in W at every $(1+\epsilon)$ multiple distance away from t, and we never expand more than that.

We conclude the proof of the lemma by proving Claim A.2.

Proof. [Claim A.2] Assume for the sake of contradiction that there exists an $I \in$ OPT such that $I \cap T = \varnothing$. Then we move I towards an adjacent point $t \in T$. Without loss of generality assume that we move I leftwards. So consider moving I leftwards one slot at a time. We break up this moving of I one slot leftwards into consecutively moving all units of I one slot leftwards: We first move the leftmost unit, then the next one etc. The following could potentially happen:

- Interval I reaches t. In this case $I \cap T \neq \varnothing$ and we stop.
- Interval I meets the endpoint e_I of some other interval I' on the same or a different processor. This cannot happen since it would contradict the optimality of OPT. The reason is that one can either merge I with I', or use part of I to close the gap following I' on its processor. Either requires one wake-up operation less but has otherwise identical costs to OPT.
- We are not able to move some unit of I one more slot leftwards without producing an infeasible schedule. Since there is still no point in T intersecting I this must be because some job j running in this unit of I would run in parallel to itself if we move the interval one more slot leftwards. Let ℓ be the slot on which j runs in I, and assume that it runs in some slot $\ell-1$ on some other processor. If there is some interval I' on one of the other processors ending at slot $\ell-1$, we simply move the unit of j to that processor continue shifting the remaining slots of I to the left. Thus we may assume that slot $\ell-1$ contains strictly less jobs than slot ℓ. By the pigeon hole principle there exists some job that we can swap with j in slot ℓ so that we can move one more unit of I one slot leftwards.

Since in each step we move one unit of I one slot leftwards, the process will eventually terminate with $I \cap T \neq \varnothing$. Note that the process does not increase the number of intervals, nor the sum of interval lengths (although it may change individual interval lengths), and hence does not affect the cost of the solution. ∎

∎

Modifying the Flow Network and Linear Program. We first show how to modify the network for checking the feasibility of `deadline-scheduling-on-intervals`. Let $W = \{a_0, a_1, \ldots, a_k\}$, with $a_0 < a_1 < \ldots < a_k$. The consecutive points in W partition $[0, D]$ into k time intervals, ie. $I_W = \{[a_0, a_1], \ldots, [a_{k-1}a_k]\}$. We refer to the interval $[a_{k-1}, a_k]$ as the k^{th} time slot. We next discuss how to adapt the maximum flow formulation. Firstly, instead of nodes v_t for each time $t, 1 \leq t \leq D-1$, we now have a node $v_t, 1 \leq t \leq k$ for each time slot in I_W. The capacity of edge (u_i, v_t) is the length of interval $[a_{t-1}, a_t]$. Let n_t be the number of intervals crossing time slot t. The capacity of edge (v_i, t) is m_t, where m_t is defined as the product of n_t and the length of time slot t. Note that size of the network after doing the above modification is $O(n|W|)$. As in Lemma 4.1, we can again argue that the given instance is feasible iff P units of flow can be routed in the network. If the instance is feasible, then P units of flow can clearly be routed. Suppose P units of flow can be routed in the network. Fix a time slot t. We have to schedule

$f(i,t)$ units of job i in the t^{th} time slot such that $f(i,t) \leq |a_t - a_{t-1}|$ and $\sum_i f(i,t) \leq m_t = n_t|a_t - a_{t-1}|$. Consider a schedule of all jobs (active in time slot t) on a single machine such that job i is processed for $f(i,t)$ units, every job is processed contiguously and there is no gap in the schedule. The machine runs continuously in $[0, \sum_i f(i,t)]$. We replicate the schedule of this machine in time $[(i-1)|a_t - a_{t-1}|, i|a_t - a_{t-1}|]$ on the i^{th} interval crossing time slot t. No job is processed in two intervals at the same time as no job has length more than $|a_t - a_{t-1}|$. We modify appropriate constraints in the Linear Program to reflect changes made in the network.

Modifying the Rounding Procedure. We now argue that the rounding procedure of Section 6 can be carried out in polynomial time. The algorithm works in iterations. In each iteration, the rounding procedure finds a minimal set of intervals of maximum deficiency and increases the length of an interval in this set by 1. This results in reduction of maximum deficiency by 1 and there can be at most P such iterations. We make the following minor modification to this algorithm. Assume that we extend an interval I of some solution $\mathcal{C}_j$ in an iteration. Instead of extending it by one unit, we extend it by δ, where δ is the maximum number such that extending I by δ also reduces the maximum deficiency of this solution by δ. We can find such a δ by binary search, and thus in time polynomial in the input.

Recall that minimal maximum deficiency set shrinks after every iteration. Suppose Q is the minimal maximum deficiency set after I was extended by δ. Since I was not extended any further, either it does not overlap with Q or none of the endpoints of I are inside Q. In either case, and because Q only shrinks, I will never be extended in any further iteration. Hence, the total number of iterations for each solution $\mathcal{C}_j$ is bounded by the maximum number of intervals in a solution, which is $O(m|W|)$. Furthermore, the total number of solutions is at most the total number of s_I's which is upper bounded by the total number of intervals in the support $\mathcal{I}$. This is in turn upper bounded by the total number of all possible intervals which is $|W|^2$. Overall, the total number of iterations required for constructing all the solutions is at most $O(m|W|^3)$, and each iteration requires time polynomial in the input size.

Also, the total length of intervals added to a solution is equal to the maximum deficiency, which is at most P and hence the rounding procedure does not further affect the approximation guarantee of the algorithm. After extending the intervals, each solution has a maximum deficiency of zero and hence feasible (by discussion in the last section).

B Integrality Gap Example

Consider an instance on a single machine with 5 jobs, $j_1, \ldots, j_5$ (see Figure 1). Let $r_1 = 0, d_1 = 1, r_2 = 1, d_2 = 7, r_3 = 2, d_3 = 4, r_4 = 4, d_4 = 6, r_5 = 7, d_5 = 8$. All jobs have unit processing time and the wake up cost of the machine is 1. Since the wake up cost is 1 we may assume that the machine transitions to the sleep state whenever it is idle, in other words there exists an optimal (integral) solution with no active but idle periods. We claim that the aforementioned instance requires at least three contiguous active time intervals: First note that j_1 and j_5 have to be done in time slots $[0,1]$ and $[7,8]$ respectively and since there are only three units of work to be done in $[1,7]$, j_1 and j_5 must be processed in two different intervals. Let I_1 and I_2 be these respective intervals. If j_2 is processed in I_1, then j_4 cannot be processed in I_1 or I_2. Similarly, if j_2 is processed in I_2, then j_3 cannot be processed in I_1 or I_2. Hence, the optimal solution must incur wake up energy of at least 3 and the total energy of the optimal solution is at least 8.

We now show a fractional solution with value strictly smaller than 8. Let $I_1 = [0,1], I_2 = [0,3], I_3 = [4,6], I_4 = [5,8], I_5 = [7,8]$ (see Figure 1). Consider a fractional solution with $x_{I_1} = x_{I_2} = x_{I_3} = x_{I_4} = x_{I_5} = 1/2$. It can easily be verified that this is a feasible fractional solution with energy $15/2$. Hence, the integrality gap of the LP is at least $16/15$.