Skeletons and Minimum Energy Scheduling

Antonios Antoniadis
University of Twente, Enschede, The Netherlands

Gunjan Kumar
National University of Singapore, Singapore

Nikhil Kumar
Hasso Plattner Institute Potsdam, Germany

Abstract

Consider the problem where n jobs, each with a release time, a deadline and a required processing time are to be feasibly scheduled in a single- or multi-processor setting so as to minimize the total energy consumption of the schedule. A processor has two available states: a *sleep state* where no energy is consumed but also no processing can take place, and an *active state* which consumes energy at a rate of one, and in which jobs can be processed. Transitioning from the active to the sleep does not incur any further energy cost, but transitioning from the sleep to the active state requires q energy units. Jobs may be preempted and (in the multi-processor case) migrated.

The single-processor case of the problem is known to be solvable in polynomial time via an involved dynamic program, whereas the only known approximation algorithm for the multi-processor case attains an approximation factor of 3 and is based on rounding the solution to a linear programming relaxation of the problem. In this work, we present efficient and combinatorial approximation algorithms for both the single- and the multi-processor setting. Before, only an algorithm based on linear programming was known for the multi-processor case. Our algorithms build upon the concept of a *skeleton*, a basic (and not necessarily feasible) schedule that captures the fact that some processor(s) must be active at some time point during an interval. Finally, we further demonstrate the power of skeletons by providing a 2-approximation algorithm for the multiprocessor case, thus improving upon the recent breakthrough 3-approximation result. Our algorithm is based on a novel rounding scheme of a linear-programming relaxation of the problem which incorporates skeletons.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases scheduling, energy-efficiency, approximation algorithms, dynamic programming, combinatorial algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.51

Related Version *Full Version*: https://arxiv.org/abs/2107.07800

Funding *Gunjan Kumar*: This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13].

1 Introduction

Energy consumption is one of the most important aspects of computing environments as supported, for example, by the fact that data centers already account for more than 1% of global electricity demand, and are forecast to reach 8% by 2030 [10]. With that in mind, modern hardware increasingly incorporates power-management capabilities. However, in order to take full advantage of these capabilities algorithms in general, and scheduling algorithms in particular, must take energy consumption into consideration – on top of the classical algorithm complexity measures of time and space.

In this work we study *power-down mechanisms* which are one of the most popular power-management techniques available on modern hardware. In the most basic setting, the processor (or device) can reside either in an *active* state in which processing can take place, or in a *sleep* state of negligible energy-consumption in which no processing can take place. Since transitioning the processor from the sleep to the active state requires energy, the scheduler would like to satisfy all the processing requirements, while making use of the sleep state as efficiently as possible. To give some intuition, it is preferable to reside in the sleep state for fewer but longer time-intervals than frequently switching between the two states.

A bit more formally, consider a set of n jobs, each with a release time, a deadline and a processing-time requirement, to be feasibly scheduled on either a single or a multi-processor system that is equipped with a powerdown mechanism. When the (each) processor resides in the active state it consumes energy at a rate of one, whereas it does not consume any energy when in the sleep state. Transitioning from the sleep state to the active state requires a constant amount of energy (the *wake-up cost*), whereas transitioning from the active state to the sleep state is free of charge (this is w. l. o. g., since positive energy cost could be folded onto the wake-up cost). Jobs can be preempted and (in the multi-processor setting) migrated, but every job can be processed on at most one machine at any given time. The objective is to produce a feasible schedule (assuming that such a schedule exists) which consumes the minimum amount of energy. 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 was first stated in its *single-processor* version by Irani et al. [9] along with an $O(n \log n)$-time 2-approximation algorithm for it. The algorithm, called *Left-to-Right* greedily keeps the processor at its current state for as long as possible. The problem plays a central role in the area of energy efficient algorithms [8], and the exact complexity of it was only resolved by Baptiste et al. [5] (and a bit earlier for the special case of unit-processing-time jobs [4]), who gave an exact polynomial-time algorithm for the problem. Their algorithm is based on a dynamic programming approach, which at least for the arbitrary processing-time case is rather involved, and obtains a running time of $O(n^5)$.

When considering the *multi-processor* setting, it is unclear how to adapt the aforementioned dynamic programming approach to the *multi-processor* setting, while enforcing that a job does not run in parallel to itself. Additionally, several structural properties that had proven useful in the analysis of the single-processor setting do not carry over to the multiprocessor setting. Only very recently, an approximation algorithm for the problem that attains a non-trivial approximation guarantee was presented by Antoniadis et al. [2]. Their algorithm is based on carefully rounding a relaxation of a linear programming formulation for the problem and has an approximation ratio of 3. They also show that their approach gives an LP-based 2-approximation algorithm for the single-processor case. We note that whether the problem is NP–hard or not in the multiprocessor-setting remains a major open question.

1.1 Formal Problem Statement and Preliminaries

Consider a set of jobs $\{j_1, j_2, \ldots, j_n\}$; job j_i has release time r_i, deadline d_i and a processing requirement of p_i, where 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 $t' - t$. Finally, we use $t \in I = [a, b]$ to denote $a \leq t \leq b$ and call interval $[r_i, d_i]$ the *span* of job i.

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 processor can be in one of two states: the *active*, or the *sleep* state. For each time-slot that a processor is in the active state, it requires one unit of power whereas no power is consumed in the sleep state. However, q units of energy (called *wake up* energy) are expended when the processor transitions from the sleep to the active state. In its active state, the processor can either process a job (in which case we refer to it as being *busy*) or just be *idle*. On the other hand the processor cannot perform any processing while in the sleep state. Note that whenever a period of inactivity is at least q time-slots long then it is worthwhile to transition to the sleep state, whereas if it is less than q time-slots long then it is preferable to remain in the active state.

A processor can process at most one job in any time-slot and a job cannot be processed on more than one processor 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 processor. A job j_i must be processed for a total of p_i time-slots within the interval $[r_i, d_i]$. Any assignment of jobs to processors and time slots satisfying the above conditions is called a (feasible) *schedule*. We assume that the processor is initially in the sleep state. Therefore, the energy consumed by a schedule is the total length of the intervals during which the processor is active plus q times the number of intervals in which the processor is active. The objective of the problem is to find a schedule which consumes minimum energy.

We will use P to denote the sum of all the processing times, ie. $P = \sum_{i=1}^{n} p_i$. We will use OPT to denote a fixed optimal solution and Q to denote the total wake up cost incurred by OPT. Given a solution S (not necessarily feasible), by *maximal gaps* of S, we will refer to intervals $[a, b]$ such that there are no active time slots in $[a, b]$ and processor is active at time slots $a - 1$ and b. An interval $[a, b]$ of S is *active* if the processor is active in all time slots in $[a, b]$ and inactive in time slots $a - 1$ and b.

Given a single processor instance, a schedule is called a *skeleton* if for all jobs j_i, there is at least one active interval overlapping with the span of j_i. In other words, there must be at least one active time slot in $[r_i - 1, d_i + 1]$. Note that a skeleton need not be a feasible solution. A *minimum cost skeleton* is a skeleton of minimal energy consumption over all skeletons.

In the following we let $(x)^+$ to stand for x if $x \geq 0$ and 0 otherwise. Due to space constraints omitted proofs are deferred to the appendix.

1.2 Our Contribution

We study the problem in both the single-processor and the multi-processor setting. Our core technical contribution is that of introducing the concept of minimum cost skeletons. We then employ this concept to design combinatorial and efficient approximation algorithms for single and multi-processor setting. Finally we demonstrate how skeletons can also be useful in strengthening the LP-formulation of the problem with additional constraints by giving the first known $(2 + \epsilon)$-approximation algorithm for the multi-processor case. More formally, our contribution is based on the following.

Single Processor

We begin by introducing the notion of skeletons for single-processor instances, and presenting a simple dynamic programming algorithm for computing minimum cost skeletons in Section 2. Roughly speaking, a skeleton is a (not necessarily feasible) solution which overlaps with the span of every job. Apart from providing a lower bound for the cost of the optimal solution OPT, a skeleton has useful structural properties that allow us to convert it into a feasible solution without much overhead. The first result in which we demonstrate this, is the following.

▶ **Theorem 1.** *There exists an $O(n \log n)$-time algorithm that computes a solution of cost at most $OPT + P$.*

The algorithm produces a solution of cost at most $OPT + P$, where OPT is the cost of optimal solution and P is the sum of processing times. Since $P \leq OPT$ this implies another 2-approximation algorithm, thus matching algorithm Left-to-Right of [9] in both running time and approximation guarantee.

We further build upon the ideas of Section 2 in Section 3 to give a $O(n \log n)$-time algorithm that also builds upon a minimum cost skeleton in order to compute a solution of cost at most $OPT + P/(\alpha - 1) + Q(2\alpha + 1)$. Here Q is the total wake up cost incurred by the optimal solution and α is any real number greater than 1. We show how this result can be used in order to obtain a $O(n \log^2 n)$ algorithm that computes a near optimal solution when $P >> Q$ (which in most scenarios is the practically relevant case) or $Q >> P$:

▶ **Theorem 2.** *Let $t = \max\{P/Q, Q/P\}$. Then there exists a $O(n \log^2 n)$ time algorithm which computes a solution of cost at most $OPT(1 + 8t^{-1/2})$.*

Note, that this implies a $(1+\epsilon)$-approximation algorithm when $t \geq 8/\epsilon^2$, whereas Left-to-Right by Irani et al. [9] remains a 2-approximation algorithm even in that case.

Finally, we give an algorithm that is also an $35/18 \approx 1.944$-approximation algorithm in $O(n \log n)$-time, improving upon the greedy 2-approximation algorithm of Irani et al. [9]:

▶ **Theorem 3.** *There is a $\frac{35}{18}$ approximation algorithm for the single processor case in $O(n \log n)$ time.*

Although (as already mentioned) Baptiste et al. [5] present an exact algorithm for the problem, it is based on a rather involved dynamic program. In contrast, our algorithms have as their main advantages that they are combinatorial in nature, simple to implement, it has an improved approximation-ratio compared to all other non-exact algorithms for the problem and even obtains a near-optimal solution for the interesting and practically relevant case of $P >> Q$ in near linear time.

Multiple Processors

Although the notion of a skeleton does not naturally extend to the multi-processor setting, it is possible to define skeletons for that setting so as to capture the same intuition. In Section 4 we do exactly that before presenting a polynomial-time algorithm for computing minimum-cost multi-processor skeletons. In Section 5, we give an algorithm to convert any skeleton into a feasible solution while increasing the cost by a factor of at most 6:

▶ **Theorem 4.** *There exists a combinatorial 6-approximation algorithm for the multi-processor case of the problem.*

This implies the first combinatorial constant-factor approximation algorithm for the multiprocessor case. The arguments required in its analysis are however much more delicate and involved than the single processor case and heavily build upon the tools developed in [2].

Finally, in Section 6 we further demonstrate the power of skeletons by using them to develop a $(2+\epsilon)$-approximation algorithm for the multi-processor case. Thus we improve upon the recent breakthrough 3-approximation from [2]. We note, that obtaining any non-trivial approximation guarantee in the multiprocessor setting has been a long standing open problem (see [5]).

▶ **Theorem 5.** *There exists a $(2+\epsilon)$-approximation algorithm for the problem on parallel machines.*

More specifically, we are able to strengthen the linear program used in [2] with additional constraints that are guided by the definition of skeletons. We note that it is unclear whether the rounding scheme used in [2] can take advantage of these new constraints. To that end we devise a novel rounding technique, and also show that the considered linear program has an integrality gap of at most 2. In other words, we show the following result.

▶ **Theorem 6.** *There exists a pseudo-polynomial time 2-approximation algorithm for deadline scheduling on parallel processors.*

Theorem 6 when combined with standard arguments, which were already presented in [2], then implies Theorem 5.

1.3 Further Related Work

The single-processor setting has also been studied in combination with the other popular power-management mechanism of *speed-scaling* where the processor can additionally vary its speed while in the active state where the power consumption grows convexly in the speed. This allows for increased flexibility, as in some cases it may be beneficial to spend some more energy by increasing the speed in the active state in order to incur larger savings by transitioning the processor to the sleep state for longer periods of inactivity. This technique is commonly referred to as *race to idle* in the literature. The combined problem is known to be **NP**-hard [1, 11] and to admit a fully polynomial time approximation scheme (FPTAS) [3].

Finally, the problem of minimizing the number of gaps in the schedule, i. e., the number of contiguous intervals during which the processor is idle. Note that with respect to exact solutions our problem generalizes that of minimizing the number of gaps. Chrobak et al. [6] present a simple $O(n^2 \log n)$-time, 2-approximation algorithm for the problem of minimizing the number of gaps on single processor with unit-processing times. Demaine et al. [7] give an exact algorithm for the problem of minimizing the number of gaps in the multi-processor setting with unit-processing-times.

2 Computing Minimum Cost Skeletons for a Single Processor

The goal of this section is to prove Theorem 1. Any solution to the minimum cost skeleton problem can be seen as a (non-overlapping) set of active intervals separated by a set of maximal gaps. In this spirit one interpretation of a skeleton is that of a set of active intervals which overlaps with the span of every job. An equivalent definition for a skeleton is a set of maximal gaps such that the span of no job is properly contained inside any maximal gap: if there is at least one active time-slot in $[r_i - 1, d_i + 1]$ then the span of job i is not contained in a maximal gap and vice versa. As we will see, this alternative viewpoint will be useful in designing a near linear time algorithm for computing a minimum cost skeleton.

▶ **Definition 7.** ***Gap Skeleton Problem****: Find a set of maximal gaps $G_1, G_2, \ldots, G_k$ such that for each job $j_i, [r_i - 1, d_i + 1] \not\subseteq G_p$ for $p = 1, 2, \ldots, k$ and the quantity $\sum_{i=1}^{k} (|G_i| - q)^+$ is maximized.*

Observe that we may w. l. o. g. restrict ourselves to input instances with at least two jobs, and may only consider minimum cost skeletons, the leftmost interval of which begins at d_{min} and the rightmost active interval ends at r_{max} (or else we can transform them to a skeleton satisfying the property without increasing their cost). We call such skeletons *nice*. For the purpose of computing a minimum cost skeleton, we shall restrict our attention to nice skeletons only. The maximal gaps of a nice skeleton form a feasible solution to the gap skeleton problem. By construction, the sum of costs of a nice skeleton and the corresponding gap skeleton is exactly equal to $r_{max} - d_{min} + q$. Hence, the problem of finding a minimum cost skeleton is in fact equivalent to finding a maximum cost gap skeleton. We now show how to compute a maximum gap skeleton by using dynamic programming.

Without loss of generality, we may also assume that maximal gaps in any maximum cost gap skeleton start and end at one of the points in $T = \cup_{i=1}^{n} \{r_i, d_i\}$ (otherwise we could increase the length of maximal gaps, without thereby decreasing the cost of the solution). A maximal gap $[x, y]$ is called *right maximal* if there exists a j_i such that $d_i = y$ and $r_i > x$. Without loss of generality, we may assume that there exists an optimal solution to maximum gap skeleton problem in which all the gaps are right maximal (we can always convert any given optimal solution to one containing only right maximal gaps). Let us rename $T = \cup_{i=1}^{n} \{r_i, d_i\}$ to $T = \{t_1, t_2, \ldots, t_{2n}\}$ such that $t_1 \leq t_2 \leq \ldots \leq t_{2n}$. Observe that for any $t \in T$, at most one right maximal gap can have its left endpoint at t and hence there can be a total of at most $2n$ right maximal gaps.

We can list all the right maximal gaps in $O(n \log n)$ time as follows: we sort all the r_i's in $O(n \log n)$ time and then for each $t \in T$, we compute the first r_i to the right of t. Then $[t, d_i]$ is the unique right maximal gap starting at t. Since for each t, the above can be done in $O(\log n)$ time (by using binary search), all the right maximal gaps can be computed in $O(n \log n)$ time.

Let $T_i = \{t_i, t_{i+1}, \ldots, t_{2n}\}, i = 1, \ldots 2n$ and $A[i]$ be the maximum value of the gap skeleton problem when restricted to T_i. By the discussion above, we may restrict our attention to only the right maximal gaps. Observe that $A[2n] = 0$ and $A[1]$ gives the value of maximum cost gap skeleton. A satisfies the following recurrence: let $g = [t_i, t_j]$ be the right maximal gap starting at t_i. In the optimal solution, either a right maximal gap starts at t_i or it doesn't, giving $A[i] = \max\{(t_j - t_i - q)^+ + A[j + 1], A[i + 1]\}$. Using the above recurrence, $A[1], \ldots, A[2n]$ can be computed in $O(n)$ time. Hence, by the equivalence of the two problems, a skeleton with minimum cost can be computed in $O(n \log n)$ time.

▶ **Theorem 8.** *A minimum cost skeleton can be computed in $O(n \log n)$ time.*

Since any feasible solution to the minimum energy scheduling problem is also a skeleton, the following follows:

▶ **Observation 9.** *The minimum cost skeleton has value at most OPT.*

In Lemma 10 we show how to convert a skeleton into a feasible solution in $O(n \log n)$ time with an additional cost of at most P. Along with Theorem 8 and Observation 9, this completes the proof of Theorem 1

▶ **Lemma 10.** *Let S be any feasible skeleton and P_S be the maximum total volume of jobs that can be feasibly processed in it. Then we can convert S into a feasible solution S' with an additional cost of $P - P_S$ in $O(n \log n)$ time.*

3 Improved Approximation Algorithms

We further develop the ideas introduced in the last section to give fast and improved approximation algorithm for the minimum energy scheduling problem. The main insight is to compute a minimum cost skeleton after scaling the wake up cost and then using Lemma 10 to find a feasible solution. As we will show, this leads to near optimal solutions in case $P >> Q$ or $P << Q$.

Let $\alpha > 1$ be a real number and S_α be the minimum cost skeleton obtained by scaling the wake up cost by α. S_α can be computed in $O(n \log n)$ time by Theorem 8. Let F_α be the solution obtained by converting S_α into a feasible solution using Lemma 10. The following theorem bounds the cost of F_α in terms of P, Q and α.

▶ **Theorem 11.** *The cost of F_α is at most $OPT + 2(\alpha+1)Q + P/(\alpha-1)$.*

Proof. Let OPT be a fixed optimal solution to the original instance. We abuse notation and also use OPT to denote the cost of the solution. To bound the cost of F_α, we will first convert OPT into S_α in a series of steps, while carefully accounting for changes in the cost, and the total volume of jobs that can be processed. Since F_α can be obtained from S_α by using Lemma 10 with a further increase in cost equal to the missing volume, this will allow us to obtain the desired bound. Let $g = [a, b]$ be any maximal gap in OPT. We first show that at most two active intervals of S_α overlap with g. The proof of this claim follows a similar proof found in Irani et al. [9].

▷ Claim 12. At most two active intervals of S_α overlap with any maximal gap g of OPT.

Let g be a maximal gap of OPT and I_1, I_2 be the two intervals of S_α that overlap with g (it is of course possible that either one or both of these intervals are empty). Recall that $|I_1 \cap g|$ and $|I_2 \cap g|$ denote the length of overlap between g and I_1, I_2 respectively.

▷ Claim 13. $|I_1 \cap g| \leq \alpha q$ and $|I_2 \cap g| \leq \alpha q$.

Proof. Let $I_1 \cap g = [a, b]$. For the sake of contradiction, suppose that $|I_1 \cap g| > \alpha q$. Observe that span of no job is strictly contained inside $I_1 \cap g$, ie. for no j_k, $[r_k, d_k] \subset I_1 \cap g$ (otherwise job j_k would not be scheduled in OPT at all). This implies that S_α remains a valid skeleton even if we make all the time slots in $[a, b]$ inactive. This operation decreases the total length of active intervals in S_α by $|a - b| > \alpha q$ while introducing at most one new active interval. This implies that S_α is not the optimal skeleton with wake up cost equal to αq, a contradiction (as making all time slots in $[a, b]$ will give a lower cost skeleton). Hence, $|I_1 \cap g| \leq \alpha q$. A similar argument shows that $|I_2 \cap g| \leq \alpha q$. ◁

We first transform OPT so that its active intervals contain the active intervals of S_α. More formally:

▷ Claim 14. We can modify OPT while increasing the cost by at most $2(\alpha+1)Q$ so that for any active interval $I_\alpha \in S_\alpha$, there exists an active interval $I_o \in OPT$ such that $I_\alpha \subseteq I_o$.

Proof. We transform every maximal gap g of OPT as follows. Let I_1, I_2 be two active intervals of S_α that overlap with g. We add (active) intervals $I_1 \cap g$ and $I_2 \cap g$ to OPT. The additional cost incurred is $2q + |I_1 \cap g| + |I_2 \cap g| \leq 2q + \alpha q + \alpha q = 2(\alpha+1)q$. The condition of the claim clearly holds after we perform the above transformation for every maximal gap of OPT. The total cost incurred over all maximal gaps is $2(\alpha+1)Q$ and the claim follows. ◁

Let OPT_1 be the solution obtained after the modification in Claim 14. Let $OPT_1 \setminus S_\alpha$ denote the set of active intervals formed by taking the difference of active time slots in OPT_1 and S_α. Let G_α be the set of all maximal gaps of S_α. By Claim 14, for every active interval $I \in OPT_1 \setminus S_\alpha$, there exists a maximal gap $g_\alpha \in G_\alpha$ such that $I \subseteq g_\alpha$. We convert OPT_1 into S_α by removing all the active intervals in $OPT_1 \setminus S_\alpha$. To bound the cost of OPT_1, we need to consider the following possibilities:

- $I \subset g_\alpha$: In this case, removing I doesn't create any new active intervals in OPT_1. Hence, decrease in the cost of OPT_1 is equal to the length of I. Also, the reduction in the total volume of jobs that can be scheduled in OPT_1 is at most $|I|$.
- $I = g_\alpha$: In this case, removing I creates a new interval in OPT_1. Hence, total cost of OPT_1 decreases by $|I| - q$. Also, reduction in the total volume of jobs that can be scheduled in OPT_1 is at most $|I|$.

Let A the set of intervals such that $I \in OPT_1 \setminus S_\alpha = g_\alpha$. We need the following bound on the cardinality of A to finish the proof.

▷ Claim 15. $q|A| \leq P/(\alpha - 1)$.

Proof. Recall that we compute S_α by scaling the wake up cost by α, hence any maximal gap of S_α has length at least αq. Since for any interval $I \in A$, there exists a maximal gap $g_\alpha \in S_\alpha$ such that $I = g_\alpha$, we have $|I| \geq \alpha q$ for any $I \in A$. We now show that there can't be more than q idle time slots in I in any feasible schedule of OPT.

Fix a feasible schedule of jobs in OPT. Since $I = g_\alpha$ and S_α is a feasible skeleton, there is no k such that span of j_k is strictly contained in I. Let J_1 be the set of jobs which overlap with the left end point of I and J_2 be the set of jobs which overlap only with the right end point of I. Note it is possible that some job is included in both J_1 and J_2. By shifting the jobs in J_1 to as far left as possible and those in J_2 to as far right as possible, we may assume that all the idle slots in I appear contiguously. If the length of this idle interval is more than q, then its removal decreases the cost of the solution without affecting the feasibility, thus contradicting the fact that OPT is a minimum cost solution. Hence, there cannot be more than q idle slots in any $I \in A$ in any feasible schedule of OPT. This implies that at least $|I| - q$ volume of jobs is processed in I in any feasible schedule of OPT. Hence, $P \geq \sum_{I \in A}(|I| - q) \geq (\alpha - 1)q|A|$ and the statement of the claim follows. ◁

We are now ready to bound the cost of the final solution. By the case analysis above, the cost of S_α is at most $OPT_1 - \sum_{I \in OPT_1 \setminus S_\alpha} |I| + q|A| \leq OPT + 2(\alpha+1)Q - \sum_{I \in OPT_1 \setminus S_\alpha} |I| + q|A|$. The total volume of jobs that can be processed in S_α is at least $P - \sum_{I \in OPT_1 \setminus S_\alpha} |I|$. Hence, using Lemma 10 to convert S_α into a feasible solution gives that the total cost of S_α is at most $OPT + 2(\alpha+1)Q - \sum_{I \in OPT_1 \setminus S_\alpha} |I| + q|A| + \sum_{I \in OPT_1 \setminus S_\alpha} |I| = OPT + 2(\alpha+1)Q + q|A| \leq OPT + 2(\alpha+1)Q + P/(\alpha - 1)$. This completes the proof of the theorem. ◀

We are now ready to prove the two main theorems of this section. We start with Theorem 2.

Proof of Theorem 2. We construct a series of solutions, $F_0, F_1, F_2, F_4, \ldots, F_{2^{\log \lceil n \rceil}}$ as follows: F_0 is the solution given by Theorem 1 of cost at most $OPT + P$. For each $i \geq 1$, we construct F_i by setting $\alpha = \sqrt{P/iq}$ in Theorem 11. Our solution F is obtained by taking the minimum cost solution among all the $F_i's$. Since, each of $F_i's$ can be constructed in $O(n \log n)$ time, F can be constructed in $O(n \log^2 n)$ time. We now show that F has the desired approximation guarantee. First note that $OPT \geq P + Q$ and $t \geq 1$. If $P \leq Q$, then F_0 has cost at most $OPT + P = OPT(1 + \frac{P}{OPT}) \leq OPT(1 + \frac{P}{P+Q}) \leq$

$OPT(1+t^{-1}) \leq OPT(1+8t^{-1/2})$. Now consider the case when $P > Q$. One can verify that for $\alpha \geq 1, f(\alpha) = Q(2\alpha+2) + P/(\alpha-1)$ has a unique minimum at $\alpha^* = \sqrt{P/2Q}+1$. We must have used exactly one $\alpha \in [\alpha^*, \sqrt{2}\alpha^*]$ to construct one of the F_i's. Since, $f(\alpha)$ has a unique minimum in $\alpha \geq 1$, it follows that it is increasing in $[\alpha^*, \sqrt{2}\alpha^*]$. Hence, there exists an F_i with cost no more than guaranteed by setting $\alpha = 2\alpha^*$ in Theorem 11. A straightforward calculation shows that $f(2\alpha^*) \leq 8\sqrt{PQ}$. Hence, F has cost at most $OPT + 8\sqrt{PQ} \leq OPT(1 + \frac{8\sqrt{PQ}}{P+Q}) \leq OPT\left(1 + \frac{8\sqrt{t}}{t+1}\right) \leq OPT(1+8t^{-1/2})$. ◀

Finally, we give a $O(n \log n)$ algorithm that has a performance guarantee better than 2.

Proof of Theorem 3. We construct two solutions: first one of cost at most $OPT + P$ (by using Theorem 1) and second one of cost at most $OPT + 8Q + P/2$ (by using Theorem 11). In case $P \leq 17Q$, the first solution has cost at most $OPT + P \leq OPT + 17(P+Q)/18 \leq OPT \cdot 35/18$. In case $P > 17Q$, the second solution has cost at most $OPT + 8Q + P/2 \leq OPT + 8(1/18) \cdot OPT + 1/2 \cdot OPT = (1 + 8/18 + 1/2) \cdot OPT = OPT \cdot 35/18$. ◀

4 Skeletons for Parallel Processors

In this section, we extend the idea of skeletons from the single processor setting to the multi-processor one. We design an efficient and combinatorial algorithm for finding the minimum cost skeleton. In the next section we then show how this skeleton can be used to design a combinatorial approximation algorithm for the multi-processor setting. Let T be as defined in the last section, i. e. $T = \cup_{i=1}^{n}\{r_i, d_i\}$. For any $t_i, t_j \in T$ such that $t_i < t_j$, let $l(t_i, t_j)$ be the maximum number of processors that can be blacked out in $[t_i, t_j]$. More formally, $l(t_i, t_j)$ is the maximum number of processors so that there exists a feasible schedule using at most on $m - l(t_i, t_j)$ many processors at any timeslot $t \in [t_i, t_j)$. Equivalently, at some time $t \in [t_i, t_j]$, at least $m - l(t_i, t_j)$ processors must be active in any feasible solution. We are now ready to define skeleton for the multi-processor case.

▶ **Definition 16.** *A set of active intervals (not necessarily feasible) is called a* ***skeleton*** *if for any time interval $[t_i, t_j]$, there exists a $t \in [t_i, t_j)$ such that at least $m - l(t_i, t_j)$ processors are active at timeslot t.*

By the definition of $l(t_i, t_j)$ and the definition of multi-processor skeletons, it directly follows that every feasible solution is also a skeleton. Hence, the cost of the optimal skeleton is a lower bound on the cost of an optimal solution.

We note that all $l(t_i, t_j)$'s can be computed in polynomial time: there are $O(n^2)$ possible pairs (t_i, t_j) and for each pair, the value $l(t_i, t_j)$ can be computed in $O(F \log m)$ time by using binary search (F here denotes the time needed to check feasibility of an instance, which can also be done in polynomial time as we will see in the next section). Therefore, the total time required to compute all $l(t_i, t_j)'s$ is $O(Fn^2 \log m)$.

4.1 Computing Minimum Cost Skeleton for Parallel Processors

We show how an optimal multi-processor skeleton can be computed by combining up to m many distinct single-processor skeletons. To that end, let $\mathcal{I}_k$ be the set of all tuples (t_i, t_j) such that $m - l(t_i, t_j) \geq k$. Each of $\{I_k\}_{k=1}^{m}$ can be thought of as defining an instance of the minimum skeleton problem for a single processor as follows: for each $I = [a, b] \in \mathcal{I}_k$, we have a job with release time a, deadline b and a unit processing requirement. We can compute the minimum skeleton S_k for each $k = 1, 2, \ldots, m$ using Theorem 8. It remains to show that $\{S_k\}_{k=1}^{m}$ is indeed the desired optimal skeleton.

▶ **Lemma 17.** *$\{S_k\}_{k=1}^m$ is an optimal skeleton for the multi-processor case.*

Proof. Let O be the optimal multi-processor skeleton for an arbitrary given instance. Without loss of generality, we may assume that O has a laminar structure, i. e. each active interval on processor $k+1$ is a subset of some active interval on processor k. Let O_k be the set of active intervals on processor k in the optimal solution and l_i be the total length of active intervals on processor i. We consider $O = \{O_k\}_{k=1}^m$ such that it is lexicographically maximal with respect to the m-tuple $(l_1, l_2, \ldots, l_m)$, and it differs from $\{S_k\}_{k=1}^m$ in least number of processors among those lexicographically maximal ones. If $O_k = S_k$ for $1 \leq k \leq m$, then the lemma follows. For the sake of contradiction, let us assume that $O_k \neq S_k$, for some k.

▷ Claim 18. O_r is a feasible skeleton for the single processor instance $\mathcal{I}_r, 1 \leq r \leq m$.

Proof. Suppose the statement of the claim doesn't hold. Then there exists an interval $[t_i, t_j]$ such that $m - l(t_i, t_j) \geq r$ but there is no active interval overlapping $[t_i, t_j]$ in O_r. Since the optimal solution O is laminar, there is no active interval overlapping $[t_i, t_j]$ for any $O_l, r \leq l \leq m$. This implies that the number of active interval at any time in $[t_i, t_j]$ in O is at most $r - 1 < m - l(t_i, t_j)$. This contradicts the feasibility of the optimal solution and the claim follows. ◁

▷ Claim 19. The solution obtained by replacing the intervals on processor k in the optimal solution, i. e. O_k by S_k is a feasible multi-processor skeleton.

Proof. Suppose the statement of the claim doesn't hold. Then there exists an interval $[t_i, t_j]$ such that some active interval in O_k overlaps with $[t_i, t_j]$ but no active interval in S_k overlaps with $[t_i, t_j]$. Since the optimal solution O is laminar, this implies that some active interval in O_ℓ overlaps with $[t_i, t_j]$ for any $1 \leq \ell \leq k$. Hence, $m - l(t_i, t_j) \geq k$, which implies that some active interval in S_k must overlap with $[t_i, t_j]$. This contradicts our assumption and the claim follows. ◁

Since O_k is a feasible solution to $\mathcal{I}_k$ and S_k is an optimal solution for $\mathcal{I}_k$, cost of S_k is at most the cost of O_k. Hence, replacing O_k by S_k gives a feasible skeleton without increasing the cost. The new solution as constructed above is laminar as well, otherwise we could move active time slots from a higher numbered processor to a lower numbered processor, contradicting our assumption that the optimal solution is the largest in lexicographical ordering $(l_1, l_2, \ldots, l_m)$. Thus we have obtained a different optimal solution which matches $\{S_i\}_{i=1}^m$ on more processors. This contradicts our choice of the optimal skeleton and the lemma follows. ◀

5 Converting a Minimum Cost Skeleton into a Feasible Solution

As argued in Lemma 17, the cost of the obtained optimal skeleton is at most the cost of an optimal solution. However, the optimal skeleton may not be a feasible solution. In this section we show how to overcome this by transforming the optimal skeleton $\{S_k\}_{k=1}^m$ into a feasible solution while increasing its energy cost by at most a factor 6. This transformation consists of two phases: the *extension phase* and the *tripling phase*. In the following subsections we describe each one of them in more detail.

5.1 Extension Phase

The extension phase of the transformation is inspired by a similar transformation performed in [2]. For the sake of completeness we give a brief and high-level description of the required terminology and results and refer the interested reader to [2] for the details.

We begin by introducing the notions of *forced volume* and of *deficiency*:

▶ **Definition 20** ([2]). *The* forced volume *of a job j_i with respect to an interval $[a,b]$, is defined as $\mathtt{fv}(j_i,[a,b]) := \max\{0, p_i - (|[r_i,d_i] \setminus [a,b]|)\}$. Let $\mathcal{D}$ be a set of disjoint intervals. The* forced volume *of job j_i with respect to $\mathcal{D}$ is defined as $\mathtt{fv}(j_i,\mathcal{D}) := \max\{0, p_i - |[r_i,d_i]| - \sum_{D\in\mathcal{D}} |D \cap [r_i,d_i]|\}$.*

Intuitively, $\mathtt{fv}(j_i,[a,b])$ is the minimum volume of j_i that must be processed during $[a,b]$ in any feasible schedule, and $\mathtt{fv}(j_i,\mathcal{D})$ is the amount of volume that must be processed within the intervals of $\mathcal{D}$ in any feasible schedule.

▶ **Definition 21** ([2]). *Let $\mathcal{D}$ be a set of disjoint intervals, and $\mathcal{I} = \{I_1, I_2, \dots I_k\}$ be a set of not necessarily disjoint intervals with the property, that for any time-point t, $m_t := |\{i \in \mathcal{I} : I_i \cap t \neq \emptyset\}| \leq m$ holds. Furthermore let $\mathcal{J}$ be a set of jobs. The* deficiency *of $\mathcal{D}$* with respect to *$\mathcal{I}$ and $\mathcal{J}$, denoted by $\mathtt{def}(\mathcal{D},\mathcal{I},\mathcal{J})$, is the non-negative difference between the sum of the forced volume of all jobs of $\mathcal{J}$ with respect to $\mathcal{D}$ and the total volume that can be processed in $\mathcal{D}$ within $\mathcal{I}$. Thus*

$$\mathtt{def}(\mathcal{J},D,\mathcal{I}) = \max\left(0, \sum_{j\in\mathcal{J}} \mathtt{fv}(j,\mathcal{D}) - \sum_{t:[t,t+1]\subseteq\mathcal{D}} m_t\right).$$

In [2], a decision problem called `deadline-scheduling-on-intervals` was introduced. More formally, problem `deadline-scheduling-on-intervals` takes as input k (not necessarily disjoint) supply-intervals $\mathcal{I} = \{I_1, I_2, \dots I_k\}$ and the set $\mathcal{J}$ of jobs (each with a release-time, a deadline and processing volume), and asks whether the jobs of $\mathcal{J}$ can be feasibly scheduled on $\mathcal{I}$. In [2] a polynomial-time algorithm *DSI-ALG* was presented that decides `deadline-scheduling-on-intervals`. Furthermore, in case the input instance is infeasible, DSI-ALG returns a *minimal* set of intervals $\mathcal{D} = \{D_1, D_2, \dots D_\ell\}$ of maximum deficiency with respect to $\mathcal{I}$. The following theorem follows from Section 4 in [2] (more specifically the first statement appears in [2] as Theorem 4.1, the last one as Claim 4.2, and the polynomial-time algorithm is described and analyzed throughout Section 4):

▶ **Theorem 22** ([2]). *An instance of `deadline-scheduling-on-intervals` is feasible iff no set of disjoint intervals has positive deficiency. Additionally, there is a polynomial-time algorithm that decides if a given instance to `deadline-scheduling-on-intervals` is feasible and if it is not, then a minimal set of disjoint intervals of maximum deficiency with respect to the instance is returned. Furthermore, increasing the volume of supply intervals at any time point in the minimal set of maximum deficiency by one unit decreases the maximum deficiency by one unit.*

Finally, [2] presents a polynomial time algorithm *EXT-ALG* which extends a supply interval $I' \in \mathcal{I}$ (a property that we will use later, is that I' is chosen so that it overlaps some $D' \in \mathcal{D}$ without containing D', i.e, $D' \not\subseteq I'$ but $D' \cap I' \neq \emptyset$) by one time slot so as to decrease the total deficiency of the set of intervals of maximum deficiency $\mathcal{D}$ by one. This is repeated until the resulting set of supply intervals becomes feasible. Since the maximal deficiency at the beginning was at most the total processing time P of all jobs, the total increase in energy consumption by extending the supply intervals could also only have been at most P.

The extension phase consists of repeatedly extending the intervals of $\{S_k\}_{k=1}^m$ via algorithm EXT-ALG until this is not possible anymore. Assume that at this point $\{S_k\}_{k=1}^m$ has been extended to an interval set $\mathcal{I}$. The extension phase thus terminates either because $\mathcal{I}$ is a feasible instance for $\mathcal{J}$ (and by Theorem 22 no set of disjoint intervals has positive deficiency with respect to $\mathcal{I}$ and $\mathcal{J}$) or because the minimal set $\mathcal{D}$ of maximum deficiency returned by DSI-ALG does not contain any interval D' such that $I' \cup D' \neq \emptyset$ and $I' \not\supseteq D'$ holds for some $I' \in \mathcal{I}$. In the later case $\mathcal{I}$ is still not feasible, and a further transformation (described in the next subsection) is required. The extension phase as stated now is pseudo-polynomial but can be carried out in polynomial time by using standard techniques (see [2] for more details). From the argument from [2] as well as the discussion above, the following lemma follows:

▶ **Lemma 23.** *The energy-cost of the schedule $\mathcal{I}$ differs from that of $\{S_k\}_{k=1}^m$ by at most an additive factor of P. Furthermore $\mathcal{I}$ is either feasible, or contains no interval $I' \in \mathcal{I}$ that overlaps but does not contain an interval from the minimal set of intervals of maximum deficiency $\mathcal{D}$.*

5.2 Tripling Phase

In case the extension phase terminated with an infeasible solution, then, by Lemma 23, there is no interval $I' \in \mathcal{I}$ such that I' overlaps some interval $D' \in \mathcal{D}$ without containing it. In that case, we need to perform the tripling phase, in which we carefully power on further machines at specific times so as to make the instance feasible. Let m_t be the number of machines active at time t in $\mathcal{I}$. We create a new solution $\mathcal{I}'$ by setting $m'_t = \min\{3m_t, m\}$. In Lemma 24, we show that $\mathcal{I}'$ is a feasible solution to the original instance. By construction, the total cost of intervals in $\mathcal{I}'$ is at most thrice that of $\mathcal{I}$. From the above discussion and Lemma 23, it follows that the algorithm consisting of the tripling and the extension phase is a 6-approximation algorithm. In other words Theorem 4 follows.

▶ **Lemma 24.** *$\mathcal{I}'$ is a feasible solution to the original instance.*

6 A 2-Approximation Algorithm for Multiple Processors

In this section we prove Theorem 5 thus improving upon the recent 3-approximation algorithm of [2]. To achieve this, we introduce additional constraints to the linear programming (LP) relaxation of [2] and devise a new rounding scheme to harness the power of new constraints. In the remainder of the section, we prove that our rounding technique gives a pseudo-polynomial time 2-approximation algorithm – thus also showing an upper bound of 2 on the integrality-gap of the LP. By using standard arguments, this directly implies $(2+\epsilon)$-approximation algorithm in polynomial time (see [2] for more details).

We now give a brief description of the LP relaxation of [2]. For every possible interval $I \subseteq [0, D]$, there is an associated variable x_I, $0 \leq x_I \leq m$ which indicates the number of times I is picked in the solution. The objective is to minimize the total energy consumption, ie. $\sum_I x_I(|I|+q)$. m_t denotes the number of processors that are active during time slot t (or equivalently total capacity of active intervals in time slot t) and $f(i,t)$ denotes the volume of job i that is processed in time slot t. The constraints of the LP are self explanatory; the interested reader is referred to [2] for the details.

In the following we describe the additional constraints (in bold). Recall that for any interval $[a,b]$, $l(a,b) \in \mathbb{Z}_{\geq 0}$ is the maximum number of machines that can remain inactive throughout interval $[a,b]$ without affecting the feasibility of the instance (see Section 4). This implies that at least $m - l(a,b)$ active intervals are overlapping with $[a,b]$ in any feasible schedule. We add a constraint capturing this fact for every $[a,b] \subseteq [0,D]$.

$$
\begin{array}{rcll}
\text{minimize} & & \sum_I (|I|+q)x_I & \\
\text{subject to} & & & \\
m_t & = & \sum_{I:[t,t+1]\in I} x_I & 0 \le t < D \\
m_t & \ge & \sum_{i:r_i \le t \le d_i-1} f(i,t) & 0 \le t < D \\
p_i & = & \sum_{t=r_i}^{d_i-1} f(i,t) & 0 \le i \le n \\
\boldsymbol{\sum_{I:[a,b]\cap I\neq\varnothing} x_I} & \boldsymbol{\ge} & \boldsymbol{m - l(a,b)} & \boldsymbol{0 \le a < b \le D} \\
f(i,t) & \in & [0,1] & \forall i,t \\
x_I, m_t & \in & [0,m] & \forall t, I \subseteq [0,D]
\end{array}
$$

Suppose the optimum fractional solution to the linear program has value f. For reasons that will become apparent later, we would like to use an optimum solution maximizing the value $\sum_t \min(m_t, 1)$. In order to compute such a solution we solve a second linear program that is based on the previous one, as follows: we introduce a new set of variables y_t and additional constraints $y_t \le m_t$ and $0 \le y_t \le 1$ for each time slot t. By adding constraint $\sum_I x_I(|I|+q) = f$ we enforce that the resulting solution has energy cost equal to f and is therefore also optimal. Finally we set the objective function to maximize $\sum_t y_t$.

Let $F = \{I : x_I > 0\}$ be the optimal fractional solution after solving the second LP. Let $\epsilon = \gcd_{i\in F}(x_I)$. We create x_I/ϵ copies of each $I \in F$ to assume that all intervals in F have the same x_I value (note that F is a multiset). If there exist $[a,b],[c,d], a < c < b < d$ with $x_{[a,b]}, x_{[c,d]} = \epsilon$, we replace them by $[a,d],[b,c]$ with $x_{[a,d]}, x_{[b,c]} = \epsilon$. It is easily verified that this process doesn't affect the feasibility of the solution. This process is repeated until no such pair of intervals remain in the instance. We therefore assume from now on, that the intervals in F are non-crossing. We would like to stress that the above is done only for the ease of analysis and during the course of the proof, it will be clear that we don't actually need to do this.

We partition the time slots in $[0,D]$ into *blocks* and *non-blocks* as follows. A duration $[a,b]$ is called a block iff $m_t \in [0,1)$ for all $a \le t \le b-1$ and $m_{a-1}, m_b \ge 1$. A duration $[a,b]$ is called a non-block iff $m_t \in [1,m]$ for all $a \le t \le b-1$ and $m_{a-1}, m_b < 1$. The following lemma leverages the new constraints and lower bounds the total weight of intervals of F contained in a non-block. The proof of this lemma crucially uses the fact that F is an optimum solution maximizing the value $\sum_t \min(m_t, 1)$.

▶ **Lemma 25.** *Let $N = [a,b]$ be a non-block. If there exists a $t \in [a, b-1]$ such that $m_t > 1$, then $\sum_{I:I\cap[a,b]\neq\emptyset} x_I \ge 2$.*

Proof. By noting that a non-block is a maximal contiguous set of time intervals with $m_t \ge 1$, intervals in F are laminar and $m_t > 1$ for some $t \in [a,b]$, there must exist an $I \in F$ such that $I \subseteq [a,b]$. Let F' be the solution obtained by replacing $I = [l,r]$ by $I' = [l+1,r]$ in F. Since F is a fractional solution with minimum cost, F' must be infeasible. Let $\mathcal{D} = \{D_1, D_2, \dots D_\ell\}$ be the *minimal* set of disjoint intervals of maximum deficiency (with respect to $\mathcal{I}$) as returned by Theorem 22 and J' be the set of jobs such that $fv(j_i, \mathcal{D}) > 0$ for all $j_i \in J'$. Note that the deficiency of $\mathcal{D}$ with respect to the current solution is ϵ. Let m'_t be defined with respect to F'.

Suppose there exists a t such that $[t,t+1] \subseteq D \in \mathcal{D}$, $[t-1,t]$ is part of a non-block and $[t,t+1]$ is part of a block (or vice versa). This implies that $m'_t < 1$ and $m'_{t-1} \ge 1$. Since $m'_{t-1} > m'_t$, there exists an $I' \in F'$ which ends at t. If we extend I' to the right by 1 unit, deficiency would decrease by ϵ (by Theorem 22) and we will obtain a feasible

solution different from F, with greater value of $\sum_t \min(m_t, 1)$ (as extending I' implies setting $m'_t = m_t + \epsilon > m_t$). Hence, no $D \in \mathcal{D}$ overlaps with a block and non-block simultaneously. Thus we can partition the intervals in $\mathcal{D}$ depending on whether they are contained in a block or a non-block. Let $\mathcal{D}_{NB} \subseteq \mathcal{D}$ and $\mathcal{D}_B \subseteq \mathcal{D}$ be the intervals of $\mathcal{D}$ contained in blocks and non-blocks respectively.

Let $D_1, D_2 \in \mathcal{D}$ be such that D_1, D_2 do not belong to the same block or the same non-block. Suppose there exists a $j_i \in J'$ such that $[r_i, d_i] \cap D_1 \neq \emptyset$ and $[r_i, d_i] \cap D_2 \neq \emptyset$. Then there must exist a t such that $[t, t+1] \subseteq [r_i, d_i]$ and $m_t < 1$. Since $fv(j_i, \mathcal{D} \cup [t, t+1]) = fv(j_i, \mathcal{D}) + 1$ and $m_t < 1$, we have that the deficiency of $\mathcal{D} \cup [t, t+1]$ is strictly more than the deficiency of $\mathcal{D}$. This contradicts the fact that $\mathcal{D}$ has maximum deficiency and hence no job in J' overlaps with distinct D_i, D_j. Therefore jobs in J' can be partitioned according to the block or non-block they overlap with. Let $J'_N \subseteq J'$ be the set of jobs with positive forced volume overlapping with the non-block N and $\mathcal{D}_N \subseteq \mathcal{D}_{NB}$ be the set of intervals of $\mathcal{D}$ overlapping with N.

Observe that $\mathcal{D}_N$ must contain the time slot $[l, l+1]$ and hence is non-empty (recall that $[l, r]$ was replaced by $[l+1, r]$ in F to obtain F'). Also, $J'_N \neq \emptyset$, otherwise $\mathcal{D} \setminus \mathcal{D}_N$ would have been the minimal set with maximum deficiency. Hence, $def(J'_N, \mathcal{D}_N, F') > 0$. Let x be the left end point of leftmost interval in $\mathcal{D}_N$ and y be the right end point of the rightmost interval in $\mathcal{D}_N$. Since $m'_t \geq 1$ for all $[t, t+1] \in [x, y] \subseteq N$ and $def(J'_N, \mathcal{D}_N, F) > 0$, there must exist a time slot $t \in [x, y-1]$ such that $m_t \geq 2$ in any integer feasible solution. This implies that $m - l(x, y) \geq 2$ and hence $\sum_{I : I \cap [a,b] \neq \emptyset} x_I \geq m - l(a, b) \geq 2$. ◀

Rounding Scheme

We next convert/round F into an integral solution in a series of steps. We will denote the three intermediate solutions by F_1, F_2, F_3 and the corresponding $m'_t s$ as m^1_t, m^2_t, m^3_t. Let T_N, T_B be the set of time slots in non-blocks and blocks respectively. Let P_F be the total available capacity in F (ie. $P = \sum_t m_t$) and P_B, P_N be the total available capacity in the blocks and non-blocks respectively. Let Q_F be the total wake up cost incurred by F, ie. $Q_F = q \sum_I x_I$. Then $Cost(F) = P_F + Q_F$ and $P_F = P_B + P_N$. For each non-block $N = [a, b]$ such that $m_t > 1$ for some $t \in [a, b-1]$, we add an additional supply interval $[a, b]$ with $x_{[a,b]} = 1$ to F and call this solution F_1.

▷ Claim 26. $Cost(F_1) \leq Cost(F) + P_N + Q_F$.

Proof. F_1 is constructed by adding an interval $[a, b]$ with $x_{[a,b]} = 1$ for a non-block $[a, b]$ if $m_t > 1$ for some $t \in [a, b]$. For each time slot t in a non-block, we have $m_t \geq 1$ and hence the total length of new intervals added is at most $\sum_{[t,t+1] \subseteq T_N} m_t \leq P_N$. If we add an additional interval for a non-block $[a, b]$, then $\sum_{I : I \cap [a,b] \neq \emptyset} x_I \geq 2$ (by Lemma 25). Since the intervals in F are non-crossing, the above implies that the total weight of intervals which are completely contained in $[a, b]$ is at least 1. Thus the wakeup cost of each new interval can be charged to the wakeup cost of intervals completely contained inside the corresponding non-block, and the total additional wake up cost incurred is no more than Q_F. ◁

We convert F_1 into F_2 by deleting the portions of existing intervals in F such that for each time slot $t \in T_N$, we have the property $m^2_t = \lfloor m^1_t \rfloor$. This operation might increase the total wake up cost, but since the processing cost gets decreased by at least as much, the overall cost of the solution does not increase. This allows us to state the following.

▷ Claim 27. $Cost(F_2) \leq Cost(F) + P_N + Q_F$. Also, m^2_t is an integer and $m^2_t \geq m_t$ for each time slot $t \in T_N$.

We now describe our third transformation. As discussed earlier, we may again assume that all the intervals in F_2 have a weight of exactly ϵ and are non-crossing. Consider the single machine instance $J_S = J_N \cup J_B$, where $J_B = \{j_i | j_i \in J, [r_i, d_i] \subseteq T_B\}$ consists of jobs in J which are completely contained inside some block and J_N consists of additional jobs defined as follows: for each time slot $t \in T_N$, there is a job j_t with release time t, deadline $t+1$ and a processing requirement of 1. We now pick a subset F_2' of intervals in F_2 which form a feasible solution to the following LP relaxation of the minimum cost skeleton.

$$\min \quad \sum_I x_I(q + |I|)$$

$$\sum_{I:t\in I} x_I \leq 1 \quad \forall t \in [0, D]$$

$$\sum_{I:I\cap[r_i,d_i]\neq\emptyset} x_I \geq 1 \forall i \in [1, n]$$

$$x_I \geq 0 \qquad \forall I \subseteq [a, b]$$

$F_2' \subseteq F_2$ is the set of intervals of maximum total length such that the total weight of intervals containing any particular time slot is at most 1. It is worth noting that any interval containing a time slot of some block is a part of F_2'. The proof of the following claim is deferred to the full version.

▷ Claim 28. F_2' is a feasible fractional skeleton for J_S.

In the full version, we also show that the LP relaxation for the minimum cost skeleton is exact. Hence, there exists an integer skeleton J_S of cost no more than F_2'. Then our solution F_3 is $(F \setminus F_2') \cup S$. Observe that $Cost(F_3) \leq Cost(F_2)$ and m_t^3 is an integer for every time slot t. We may therefore assume that $x_I = 1$ for each $I \in F_3$. We now describe the final phase our algorithm, where we convert F_3 into a feasible solution by extending some existing intervals using *EXT-ALG* (see Section 5). If F_3 is a feasible solution, our algorithm terminates. Otherwise we find a disjoint minimal set of intervals of maximum deficiency $\mathcal{D} = \{D_1, D_2, \ldots, D_k\}$ guaranteed by Theorem 22. In each subsequent iteration, we use *EXT-ALG* to extend an interval of F_3 by 1 unit, thereby reducing the maximum deficiency by 1. Claim 29 shows that if the current solution is infeasible and it is not possible to extend an interval to reduce the deficiency, then the original instance is infeasible. Hence the extension phase of the algorithm terminates with a feasible solution.

▷ Claim 29. Let F_{curr} be the current solution. If F_{curr} is infeasible and for all $I \in F_{curr}$ the following is true: if $I \cap D_i \neq \varnothing$, then $D_i \subseteq I$, then the original instance is infeasible.

The deficiency at the start of the extension phase can be at most P_B as $m_t^3 \geq m_t$ for $t \in T_N$. Since we decrease the deficiency by 1 in each iteration, there can be at most P_B iterations of the extension phase. In each step we increase the cost of the solution by 1, hence cost of the final solution is at most $Cost(F_3)+P_B \leq Cost(F)+P_N+Q_F+P_B = Cost(F)+P+Q_F \leq 2Cost(F)$. This shows that the integrality gap of the LP relaxation is at most 2. To compute F_1, F_2, F_3, we only need the value of m_t's and don't need to create multiple copies of intervals in our solution. Thus our rounding algorithm can be implemented in pseudo-polynomial time and this completes the proof of Theorem 6.

References

1. 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.
2. Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *SODA*, pages 2758–2769. SIAM, 2020.
3. 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.
4. 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.
5. Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Trans. Algorithms*, 8(3):26:1–26:29, 2012.
6. 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.
7. Erik D. Demaine, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. *J. Sched.*, 16(2):151–160, 2013.
8. Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
9. Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4):41, 2007.
10. Nicola Jones. How to stop data centres from gobbling up the world's electricity. `https://www.nature.com/articles/d41586-018-06610-y`, 2018.
11. Gunjan Kumar and Saswata Shannigrahi. On the NP-hardness of speed scaling with sleep state. *Theor. Comput. Sci.*, 600:1–10, 2015.