Polynomial Time Algorithms for Minimum Energy Scheduling

Philippe Baptiste* Marek Chrobak† Christoph Dürr*

Abstract

The aim of power management policies is to reduce the amount of energy consumed by computer systems while maintaining satisfactory level of performance. One common method for saving energy is to simply suspend the system during the idle times. No energy is consumed in the suspend mode. However, the process of waking up the system itself requires a certain fixed amount of energy, and thus suspending the system is beneficial only if the idle time is long enough to compensate for this additional energy expenditure. In the specific problem studied in the paper, we have a set of jobs with release times and deadlines that need to be executed on a single processor. Preemptions are allowed. The processor requires energy L to be woken up and, when it is on, it uses one unit of energy per one unit of time. It has been an open problem whether a schedule minimizing the overall energy consumption can be computed in polynomial time. We solve this problem in positive, by providing an $O(n^5)$-time algorithm. In addition we provide an $O(n^4)$-time algorithm for computing the minimum energy schedule when all jobs have unit length.

1 Introduction

Power management strategies. The aim of power management policies is to reduce the amount of energy consumed by computer systems while maintaining satisfactory level of performance. One common method for saving energy is a *power-down mechanism*, which is to simply suspend the system during the idle times. The amount of energy used in the suspend mode is negligible. However, during the wake-up process the system requires a certain fixed amount of *start-up* energy, and thus suspending the system is beneficial only if the idle time is long enough to compensate for this additional energy expenditure.

Scheduling to minimize energy consumption. The scheduling problem we study in this paper is quite fundamental. We are given a set of jobs with release times and deadlines that need to be executed on a single processor. Preemptions are allowed. We assume, without loss of generality, that,

*CNRS, LIX UMR 7161, Ecole Polytechnique 91128 Palaiseau, France. Supported by CNRS/NSF grant 17171 and ANR Alpage.

†Department of Computer Science, University of California, Riverside, CA 92521, USA. Supported by NSF grants OISE-0340752, CCR-0208856 and CCF-0729071.

when the processor is on, it uses one unit of energy per unit of time. The energy required to wake up the processor is denoted by L. The objective is to compute a feasible schedule that minimizes the overall energy consumption, or to report that no feasible schedule exists. Denoting by E the energy consumption function, this problem can be classified using Graham's notation as $1|r_j; \text{pmtn}|E$.

The question whether this problem can be solved in polynomial time was posed by Irani and Pruhs [9], who write that "... Many seemingly more complicated problems in this area can be essentially reduced to this problem, so a polynomial time algorithm for this problem would have wide application." Some progress towards resolving this question has already been reported. Chretienne [3] proved that it is possible to decide in polynomial time whether there is a schedule with no idle time. More recently, Baptiste [2] showed that the problem can be solved in time $O(n^7)$ for unit-length jobs and $L = 1$.

Our results. We solve the open problem posed by Irani and Pruhs [9], by providing a polynomial-time algorithm for $1|r_j; \text{pmtn}|E$. Our algorithm is based on dynamic programming and it runs in time $O(n^5)$. Thus not only our algorithm solves a more general version of the problem, but is also faster than the algorithm for unit jobs in [2]. For the case of unit jobs (that is, $1|r_j; p_j = 1|E$), we improve the running time further to $O(n^4)$.

The paper is organized as follows. First, in Section 2, we introduce the necessary terminology and establish some basic properties. Our algorithms are developed gradually in the sections that follow. We start with the special case of minimizing the number of gaps for unit jobs, that is $1|r_j; p_j = 1; L = 1|E$, for which we describe an $O(n^4)$-time algorithm in Section 3. Next, in Section 4, we extend this algorithm to jobs of arbitrary length ($1|r_j; \text{pmtn}; L = 1|E$), increasing the running time to $O(n^5)$. Finally, in Section 5, we show how to extend these algorithms to arbitrary L, without affecting their running times.

We remark that although our algorithms are based on dynamic programming, they are sensitive to the structure of the input instance and on typical instances they are likely to run significantly faster than their worst-case bounds.

Other relevant work. The non-preemptive version of our problem, that is $1|r_j|E$, can be easily shown to be $\mathbb{NP}$-hard in the strong sense, even for $L = 1$ (when the objective is to only minimize the number of *gaps*, see section 2), by reduction from 3-Partition [5, problem SS1].

More sophisticated power management systems may involve several sleep states with decreasing rates of energy consumption and increasing wake-up overheads. In addition, they may also employ a method called *speed scaling* that relies on the fact that the speed (or frequency) of processors can be changed on-line. As the energy required to perform the job increases quickly with the speed of the processor, speed scaling policies tend to slow down the processor while ensuring that all jobs meet their deadlines (see [9], for example). This problem is a generalization of $1|r_j; \text{pmtn}|E$ and its status remains open. A polynomial-time 2-approximation algorithm for this problem (with two power states) appeared in [7].

As jobs to be executed are often not known in advance, the on-line version of energy minimization is of significant interest. Online algorithms for power-down strategies with multiple power states were considered in [6, 8, 1]. In these works, however, jobs are critical, that is, they must be executed as soon as they are released, and the online algorithm only needs to determine the appropriate power-down state when the machine is idle. The work of Gupta, Irani and Shukla [7] on power-down with speed scaling is more relevant to ours, as it involves aspects of job scheduling. For the specific problem studied in our paper, $1|r_j; \text{pmtn}|E$, it is easy to show that no online algorithm can have a constant competitive ratio (independent of L), even for unit jobs. We refer the reader to [9] for a detailed survey on algorithmic problems in power management.

2 Preliminaries

Minimum-energy scheduling. Formally, an instance of the scheduling problem $1|r_j; \text{pmtn}|E$ consists of n jobs, where each job j is specified by its processing time p_j, release time r_j and deadline d_j. We have one processor that, at each step, can be on or off. When it is on, it consumes energy at the rate of one unit per time step. When it is off, it does not consume any energy. Changing the state from off to on (waking up) requires additional L units of energy.

The time is discrete, and is divided into unit-length intervals $[t, t+1)$, where t is an integer, called *time slots* or *steps*. For brevity, we often refer to time step $[t, t+1)$ as *time step* t. A preemptive schedule S specifies, for each time slot, whether some job is executed at this time slot and if so, which one. Each job j must be executed for p_j time slots, and all its time slots must be within the time interval $[r_j, d_j)$.

A *block* of a schedule S is a maximal interval where S is *busy*, that is, executes a job. The union of all blocks of S is called its *support*. A *gap* of S is a maximal interval where S is idle (does not execute a job).

Suppose that the input instance is feasible. Since the energy used on the support of all schedules that schedule all jobs is the same, it can be subtracted from the energy function for the purpose of minimization. The resulting function $E(S)$ is the "wasted energy" (when the processor is on but idle) plus L times the number of wake-ups. Formally, this can be calculated as follows. Let $[u_1, t_1), \ldots, [u_q, t_q)$ be the set of all blocks of S, where $u_1 < t_1 < u_2 < \ldots < t_q$. Then

$$E(S) \;=\; \sum_{i=2}^{q} \min\{u_i - t_{i-1}, L\}.$$

(We do not charge for the first wake-up at time u_1, since this term is independent of the schedule.) Intuitively, this formula reflects the fact that once the support of a schedule is given, the optimal suspension and wake-up times are easy to determine: we suspend the machine during a gap if and only if its length is at least L, for otherwise it would be cheaper to keep the processor on during the gap.

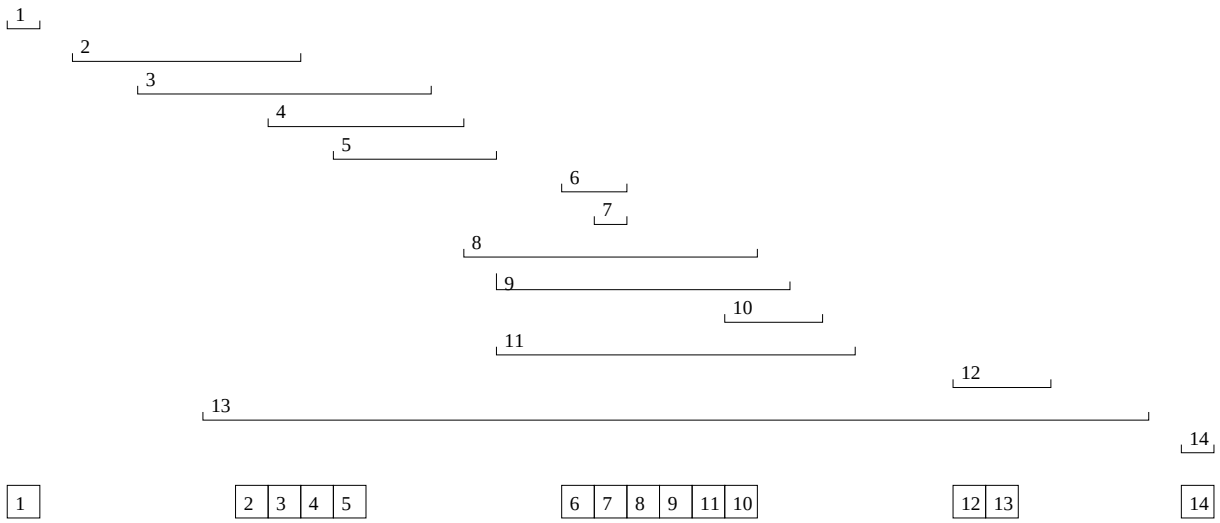

Figure 1: The release time – deadline intervals of unit jobs, and below an optimal schedule for $L = 1$ with 4 gaps.

Our objective is to find a schedule S that meets all job deadlines and minimizes $E(S)$. (If there is no feasible schedule, we assume that the energy value is $+\infty$.) Note that the special case $L = 1$ corresponds to simply minimizing the number of gaps.

By $C_j(S)$ (or simply C_j, if S is understood from context) we denote the completion time of a job j in a schedule S. By $C_{\max}(S) = \max_j C_j(S)$ we denote the maximum completion time of any job in S. We refer to $C_{\max}(S)$ as the *completion time of schedule* S.

Simplifying assumptions. Throughout the paper we assume that jobs are ordered according to deadlines, that is $d_1 \leq \ldots \leq d_n$. Without loss of generality, we also assume that all release times are distinct and that all deadlines are distinct. Indeed, if $r_i = r_j$ for some jobs $i < j$, since the jobs cannot start both at the same time r_i, we might as well increase by 1 the release time of j. A similar argument applies to deadlines.

To simplify the presentation, we will assume that the job indexed by 1 is a special job with $p_1 = 1$ and $d_1 = r_1 + 1$, that is job 1 has unit length and must be scheduled at its release time. (Otherwise, if job 1 does not satisfy these conditions, we can always add such an extra job, released $L + 1$ time slots before r_1. This increases each schedule's energy consumption by exactly L and does not affect the asymptotic running time of our algorithms.)

Without loss of generality, we can also assume that the input instance is feasible. A feasible schedule corresponds to a matching between units of jobs and time slots, so Hall's theorem gives us the following necessary and sufficient condition for feasibility: for all times $u < v$,

$$\sum_{u \leq r_j, d_j \leq v} p_j \leq v - u, \tag{1}$$

which in particular implies $d_j \geq r_j + p_j$ for all j.

We can also restrict our attention to schedules S that satisfy the following *earliest-deadline property* or *policy*: at any time t, either S is idle at t or it schedules a pending job with the earliest deadline. In other words, once the support of S is fixed, within the support we can schedule the jobs one by one, from left to right, in each slot of the support executing the pending job with minimum deadline. Using the standard exchange argument, any schedule can be converted into one that satisfies the earliest-deadline property and has the same support. Thus, throughout the paper, we will tacitly assume (unless explicitly noted otherwise) that all schedules we consider satisfy the earliest-deadline property.

We now make another observation concerning the number of gaps. We claim that, without loss of generality, we can assume that the optimal schedule has at most $n-1$ gaps. The argument is quite simple: if S is any schedule, consider a gap $[u, v)$ and the block that follows it, say $[v, w)$. If there is no release time in $[u, w)$, then all jobs executed in $[v, w)$ are released before u, so we can shift the whole block $[v, w)$ leftwards all the way to u, merging two blocks. If $[v, w)$ was the last block, this, clearly, decreases the cost. If $[v, w)$ is not the last block, this change merges two gaps into one, which can only decrease the cost. Therefore we can assume that $[u, w)$ contains a release time. As this is true for each gap in S, we conclude that the number of gaps is at most $n-1$, as claimed.

(k, s)-Schedules. We will consider certain partial schedules, that is schedules that execute only some jobs from the instance. For jobs k and s, a partial schedule S is called a (k, s)-*schedule* if it schedules all jobs $j \leq k$ with $r_s \leq r_j < C_{\max}(S)$ (recall that $C_{\max}(S)$ denotes the completion time of schedule S). From now on, unless ambiguity arises, we will omit the term "partial" and refer to partial schedules simply as schedules. When we say that a (k, s)-schedule S has g gaps, in addition to the gaps between the blocks we also count the gap (if any) between r_s and the first block of S. Note that the above bound of $n-1$ on the number of gaps in an optimal schedule applies to (k, s)-schedules as well, since the first job (with minimum release time) is tight and thus is not preceded by a gap.

For any k, s, the empty schedule is also considered to be a (k, s)-schedule. The completion time of an empty (k, s)-schedule is artificially set to r_s. (Note that, in this convention, empty (k, s)-schedules, for difference choices of k, s, are considered to be different schedules.)

Greedy schedules. For any k, s, and i such that $i \leq k$ and $r_i \geq r_s$, let $C^{\text{ED}}_{i,s}$ denote the minimum completion time of job i among all earliest-deadline (k, s)-schedules that schedule i. (As explained below, $C^{\text{ED}}_{i,s}$ does not depend on k.) Note that if $r_s \leq r_l \leq r_i$ then $C^{\text{ED}}_{i,s} \geq C^{\text{ED}}_{i,l}$ – simply because if we take an earliest-deadline (k, s)-schedule realizing $C^{\text{ED}}_{i,s}$ and remove all jobs released before r_l, we obtain an earliest-deadline (k, l)-schedule that schedules i.

By $G_{k,s}$ we denote the greedy (k, s)-schedule that, for each time step $t = r_s, r_s + 1, ...$, schedules the most urgent pending job. (Note that $G_{k,s}$ may not minimize the number of gaps.) In $G_{k,s}$, the schedule of a job i does not depend on any jobs $j > i$. Therefore $C_i(G_{k,s}) = C_i(G_{i,s}) = C_i(G_{i,l})$, for some $l \leq i$ such that $r_s \leq r_l \leq r_i$.

The duality lemma below establishes a relation between $C^{\text{ED}}_{i,s}$ and greedy schedules. In particular, it implies that greedy schedules are feasible (all deadlines are met). It also shows that $C^{\text{ED}}_{i,s}$ does not depend on k, justifying the omission of the subscript k in the notation $C^{\text{ED}}_{i,s}$. However, $C^{\text{ED}}_{i,s}$ may depend on s, as illustrated in Figure 2.

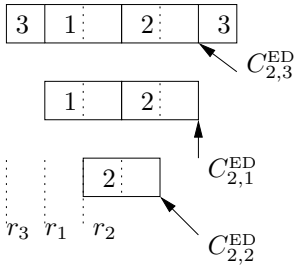

Figure 2: The minimum completion time of job i in a (k, s)-schedule may depend on s. In this example, $p_1 = p_2 = p_3 = 2$.

For any times $a < b$ and a job i, define

$$load_i(a, b) \quad = \quad \sum_{j \le i,\, a \le r_j < b} p_j.$$

Thus $load_i(a, b)$ is the total workload of the jobs released between a and b whose deadlines are at most d_i.

Lemma 1 (earliest completion) *For any k, s and $i \le k$ such that $r_i \ge r_s$, we have*

$$C^{\text{ED}}_{i,s} \quad = \quad C_i(G_{k,s}) \quad = \quad \max_{\substack{l \le i \\ r_s \le r_l \le r_i}} \min\{b \,:\, b > r_i \;\&\; b \ge r_l + load_i(r_l, b)\}. \tag{2}$$

Proof: Let RHS(2) stand for the expression on the right-hand side of (2). It is sufficient to show that $C^{\text{ED}}_{i,s} \le C_i(G_{k,s}) \le \text{RHS}(2) \le C^{\text{ED}}_{i,s}$. The inequality $C^{\text{ED}}_{i,s} \le C_i(G_{k,s})$ is trivial, directly from the definition of $C^{\text{ED}}_{i,s}$. Thus it is sufficient to show the two remaining inequalities.

We now show that $C_i(G_{k,s}) \le \text{RHS}(2)$. As we observed earlier, $C_i(G_{k,s})$ does not depend on k (as long as $k \ge i$, of course), by the earliest-deadline rule, so we can assume $k = i$. Write $C_i = C_i(G_{i,s})$. Let l be the first job scheduled in $G_{i,s}$ in the block containing slot r_i. It is sufficient to show that

$$C_i \quad \le \quad \min\{b \,:\, b > r_i \;\&\; b \ge r_l + load_i(r_l, b)\}. \tag{3}$$

Note that the minimum on the right-hand side of (3) is well defined, as this set contains any b that is large enough. Thus it remains to show that for any b such that $r_i < b < C_i$ we have $b < r_l + load_i(r_l, b)$. Indeed, consider schedule $G_{i,s}$. By the definition of l, the block containing r_i starts at time r_l. Also, there is no idle time between r_i and C_i. Therefore all slots $r_l, r_{l+1}, ..., b-1$ are filled with jobs $j \le i$ such that $r_l \le r_j < b$. Just after scheduling slot $b-1$, the greedy algorithm still has at least one unit of i pending (because i completes after b). This implies that $b < r_l + load_i(r_l, b)$, as claimed, completing the proof of the inequality $C_i(G_{k,s}) \le \text{RHS}(2)$.

Finally, we prove that RHS(2) $\leq C^{\text{ED}}_{i,s}$. Choose any $l \leq i$ with $r_s \leq r_l \leq r_i$. Recall that $C^{\text{ED}}_{i,l} \leq C^{\text{ED}}_{i,s}$ (see the comments following the definition of $C^{\text{ED}}_{i,s}$). Thus, if S is any earliest-deadline (i,l)-schedule that schedules i, it is sufficient to prove that

$$\min\{b : b > r_i \;\&\; b \geq r_l + \mathit{load}_i(r_l, b)\} \quad \leq \quad C_i(S). \tag{4}$$

All we need to do is to show that $C_i(S)$ is a candidate for b on the left-hand side of (4). That $C_i(S) > r_i$ is obvious. Further, in S, at time $C_i(S)$ the least urgent job i completes, so S has no pending jobs at time $C_i(S)$, which immediately implies that $C_i(S) \geq r_l + \mathit{load}_i(r_l, C_i(S))$. □

Fixed slots and segments. Let Q be a schedule and let t be some time slot in Q scheduling a job j. We call t *fixed* in Q if either (i) $t = r_j$ or, recursively, (ii) all time slots in $[r_j, t)$ are fixed. (In particular, of course, a fixed slot cannot be idle.) An interval $[t', t)$ is called a *fixed segment* of Q if all slots in $[t', t)$ are fixed and every job j scheduled in $[t', t)$ completes not later than t, and is released not before t'. By definition, if a fixed segment starts at time u and it executes a job l at time u, then $u = r_l$. See Figure 3 for illustration.

The following lemma relates fixed slots to earliest completion times.

Lemma 2 *Fix any k, s, and some arbitrary (k,s)-schedule S with $C_{\max}(S) = t$. Suppose that $[u, t)$ is a fixed segment in S. Then for every job $i \leq k$ that completes in this segments (that is, $u < C_i(S) \leq t$), we have $C_i(S) = C^{\text{ED}}_{i,s}$.*

Proof: Write $C_i = C_i(S)$. By definition, $C_i \geq C^{\text{ED}}_{i,s}$, so it is sufficient to show that $C_i \leq C^{\text{ED}}_{i,s}$.

By definition of fixed segments, $u \leq r_i$. Let l be the job executed in slot u. Then we must have $r_l = u$. Since $C^{\text{ED}}_{i,l} \leq C^{\text{ED}}_{i,s}$, it is sufficient now to show that $C_i \leq C^{\text{ED}}_{i,l}$.

The definition of fixed segments implies that all jobs executed in $[u, C_i)$ are released in $[u, C_i)$. Since, by our convention, S has the earliest-deadline property, S and $G_{i,l}$ are identical in $[u, C_i)$, and thus, by Lemma 1, we can conclude that $C_i = C_i(G_{i,l}) = C^{\text{ED}}_{i,l}$, completing the proof. □

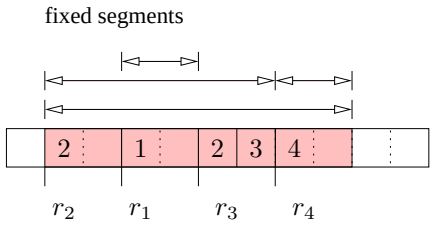

Figure 3: Illustration of fixed slots (in dark) and fixed segments. Here, $p_1 = 2$, $p_2 = 3$, $p_3 = 1$ and $p_4 = 2$.

An outline of the algorithms. For any $k = 0, ..., n$, $s = 1, ..., n$, and $g = 0, ..., n-1$, define $U_{k,s,g}$ as the maximum completion time of a (k,s)-schedule with at most g gaps, see Figure 4.

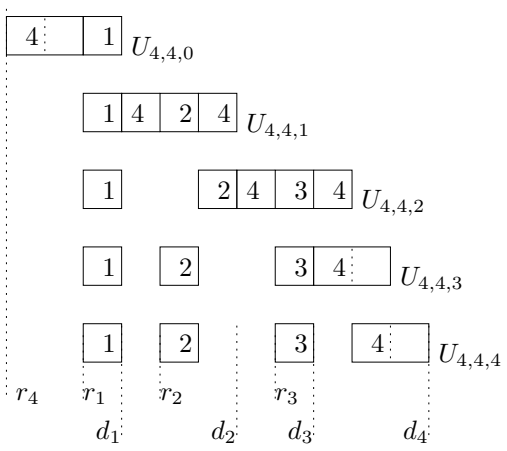

Figure 4: The value $U_{k,s,g}$ is non-decreasing in g. Here, $p_1 = p_2 = p_3 = 1$ and $p_4 = 2$.

Our algorithms consist of two stages. The first stage is to compute the table $U_{k,s,g}$, using dynamic programming. Note that from this table we can determine the minimum number of gaps in the (complete) schedule: the minimum number of gaps is equal to the smallest g for which $U_{n,1,g} > \max_j r_j$. The algorithm computing $U_{k,s,g}$ for unit jobs is called AlgA and the one for arbitrary length jobs is called AlgB.

In the second stage, described in Section 5 and called AlgC, we use the table $U_{k,s,g}$ to compute the minimum energy schedule. In other words, we show that the problem of computing the minimum energy reduces to computing the minimum number of gaps. This reduction, itself, involves again dynamic programming.

When presenting our algorithms, we will only show how to compute the minimum energy value. The algorithms can be modified in a straightforward way to compute the actual optimum schedule, without increasing the running time. (In fact, we explain how to construct such schedules in the correctness proofs.)

3 Minimizing the Number of Gaps for Unit Jobs

In this section we give an $O(n^4)$-time algorithm for minimizing the number of gaps for unit jobs, that is for $1|r_j; p_j = 1; L = 1|E$. Recall that we assume all release times to be different and all deadlines to be different. With this assumption, it is easy to see that there is always a feasible schedule, by scheduling every job at its release time.

As described in the previous section, the general idea of the algorithm is to compute all values of the function $U_{k,s,g}$ using dynamic programming. Before stating the algorithm, we establish some properties of (k, s)-schedules.

Some properties of (k, s)-schedules. A (k, s)-schedule S is called *frugal* if it satisfies the following properties:

(f1) There is no job $j \leq k$ with $r_j = C_{\max}(S)$, and

(f2) Suppose that S schedules job k and $C_{\max}(S) < d_k$. Then either (i) k is scheduled last in S (at time $C_{\max}(S) - 1$) and the last block contains at least one job other than k, or (ii) k is scheduled inside a block (that is, k is not the first nor the last job in a block).

Obviously, if $C_{\max}(S) = d_k$, then, by the assumption about different deadlines, k must be scheduled last in S. But in this case, even if S is frugal, the last block may or may contain jobs other than k.

Lemma 3 (frugality) *Fix some k, s, g, and let S be a (k, s)-schedule that realizes $U_{k,s,g}$, that is S has at most g gaps and $C_{\max}(S) = U_{k,s,g}$. Then S is frugal.*

Proof: The proof is quite simple. If S violates (f1) then we can extend S by scheduling j at $C_{\max}(S)$, obtaining a new (k, s)-schedule with at most g gaps and larger completion time, which contradicts the optimality of S.

Next, assume that S satisfies condition (f1), but not (f2). We have two cases. Suppose first that k is the last job in S. Then it is not possible that k is the only job in the last block of S, for then we could move k to $d_k - 1$, without increasing the number of gaps but increasing the completion time. The other case is that k is not last in S. If k were either the first or last job in its block, we could reschedule k at time $C_{\max}(S)$, without increasing the number of gaps and increasing the completion time. (By condition (f1), this is a correct (k, s)-schedule.) Thus in both cases we get a contradiction with the optimality of S. □

We now make some observations that follow from the lemma above. First, we claim that, for any fixed s and g, the function $k \to U_{k,s,g}$ is non-decreasing. Indeed, suppose that S is a (k, s)-schedule that realizes $U_{k,s,g}$. By the lemma above, we can assume that S is frugal. If $r_{k+1} \geq C_{\max}(S) = u$, then S is itself a valid $(k+1, s)$-schedule. If $r_{k+1} < u$, then we can extend S by scheduling job $k+1$ at time u, obtaining a new schedule S'. By the frugality of S, no job $j \leq k$ is released at time u. Also, $u \leq d_k < d_{k+1}$, so S' is a valid $(k+1, s)$-schedule, it has the same number of gaps as S, and $C_{\max}(S') > C_{\max}(S)$.

Further, we also claim that, for any fixed k and s, the function $g \to U_{k,s,g}$ is strictly increasing as long as $U_{k,s,g} < d_k$. For suppose that S is a (frugal) schedule that realizes $U_{k,s,g} < d_k$. If there is a job $j \leq k$ with $U_{k,s,g} \leq r_j < d_k$, then in fact, by frugality, $U_{k,s,g} < r_j$. Choose such a j with minimum r_j and extend S by scheduling j at r_j. The new schedule S' is a (k, s)-schedule, it has one more gap than S, and $C_{\max}(S') > C_{\max}(S)$. Else, suppose that such j does not exist. In particular, $r_k < U_{k,s,g}$, so S schedules k. Let S' be the schedule obtained from S by moving k to time $d_k - 1$, so that $C_{\max}(S') = d_k > C_{\max}(S)$. S' is a (k, s)-schedule. By the frugality condition (f2) of S, either

k is the last job in the last block, or it is an internal job of another block. In both cases S' has only one more gap than S.

Lemma 4 (partitioning) *Let S be a (k,s)-schedule that realizes $U_{k,s,g}$ and schedules job k, but not as the last job. Let t be the time at which S schedules job k, and let h be the number of gaps in S in the interval $[r_s,t)$. Then $t = U_{k-1,s,h}$.*

Proof: By Lemma 3, S is frugal. Denote $v = U_{k-1,s,h}$. Clearly, by the earliest-deadline property, no jobs $j < k$ released in $[r_s,t)$ are pending at time t. So the segment of S in $[r_s,t)$ is a $(k-1,s)$-schedule with h gaps, implying that $v \geq t$. Thus it suffices now to show that $v \leq t$. Towards contradiction, suppose that $v > t$ and let R be a $(k-1,s)$-schedule that realizes $U_{k-1,s,h}$, that is R has at most h gaps and $C_{\max}(R) = v$. We consider two cases.

Case 1: R schedules all jobs $j < k$ with $r_s \leq r_j < t$ in the interval $[r_s,t+1)$. We can modify S as follows: Reschedule k at time $u = C_{\max}(S)$ and replace the segment $[r_s,t+1)$ of S by the same segment of R. Let S' be the resulting schedule. The earliest deadline property of S implies that there is no job $j < k$ released at time t. By this observation and the case condition, S' is a (k,s)-schedule. Also, no matter whether R is idle at t or not, S' has at most h gaps in the segment $[r_s,t+1)$, and therefore at most g gaps in total. We thus obtain a contradiction with the choice of S, because $C_{\max}(S') = u+1 > C_{\max}(S)$.

Case 2: R schedules some job $j < k$ with $r_s \leq r_j < t$ strictly after t. In this case, we claim that there is a $(k-1,s)$-schedule R' (not necessarily frugal) with at most h gaps and $C_{\max}(R') = t+1$. We could then again obtain a contradiction by proceeding as in Case 1.

Let $[w,v)$ be the last block of R. In Section 2 we defined the concept of fixed slots in a schedule. For unit jobs, the definition of fixed slots becomes very simple: a slot $z = w,...,v-1$ of R is *fixed* if the job scheduled at time z is released at z.

To obtain R', we gradually "compress" R, according to the procedure below (see Figure 5).

If the slot $v-1$ is fixed, then we simply remove it, replacing R by its segment in $[r_s,v-1)$. The result is still a $(k-1,s)$-schedule, even though it is not frugal. This schedule has completion time strictly smaller than v, but not less than $t+2$ because, by case assumption, strictly after time t it schedules a job $j < k$ with $r_s \leq r_j < t$, and this execution slot is not fixed.

The other case is when the slot $v-1$ is not fixed. Now for each non-fixed slot in $[w,v)$, move the job in this slot to the previous non-fixed slot. The job from the first non-fixed slot will move to slot $w-1$. By the assumption about distinct release times, this operation will not move a job before its release time. It also preserves fixed slots, while some non-fixed slots, including the empty slot $w-1$, might become fixed. The last block now ends one unit earlier, and either it starts one unit earlier or is merged with the second last block. After this operation, R remains a $(k-1,s)$-schedule with at most h gaps. If $C_{\max}(R) = t+1$, we let $R' = R$, otherwise we continue the process. □

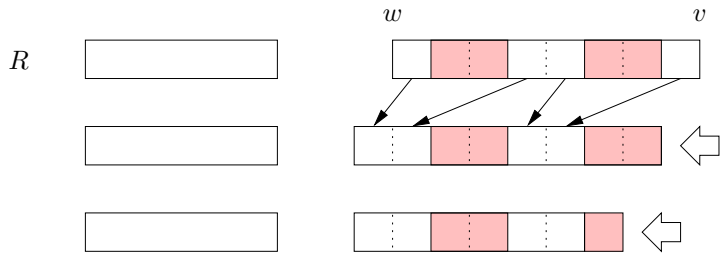

Figure 5: Illustration of the compression. Fixed slots are shown in dark.

Outline of the algorithm. As explained in the previous section, the algorithm computes the table $U_{k,s,g}$. The crucial idea here is this: Let S be a (k,s)-schedule that realizes $U_{k,s,g}$, that is S has g gaps and $C_{\max}(S)$ is maximized. If S does not schedule k, then S is a $(k-1,s)$-schedule, so $U_{k,s,g} = U_{k-1,s,g}$. If S schedules k as the last job, then either $U_{k,s,g} = C_{\max}(S) = d_k$ or the last block contains jobs other than k, in which case the part of S before k is a $(k-1,s)$-schedule with the same number of gaps g, implying that $U_{k,s,g} = U_{k-1,s,g} + 1$. The most interesting case is when S schedules k not as the last job, say at time t. By frugality, k is neither the first nor the last job in its block. Denote $u = U_{k,s,g}$. We show that, without loss of generality, there is a job l released and scheduled at time $t+1$. Further, the segment of S in $[r_s, t)$ is a $(k-1,s)$-schedule with completion time t, the segment of S in $[t+1, u)$ is a $(k-1,l)$-schedule with completion time $U_{k,s,g}$, and the total number of gaps in these two schedules equals g. Denoting by h the number of gaps of S in the interval $[r_s, t)$, we conclude that $U_{k,s,g} = U_{k-1,l,g-h}$, and by Lemma 4 we also have $t = U_{k-1,s,h}$, leading naturally to a recurrence relation for this case.

Algorithm ALGA**.** The algorithm computes all values $U_{k,s,g}$, for $k = 0, ..., n$, $s = 1, ..., n$ and $g = 0, ..., n-1$, using dynamic programming. The minimum number of gaps for the input instance is equal to the smallest g for which $U_{n,1,g} > \max_j r_j$.

The values $U_{k,s,g}$ will be stored in the table $\bar{U}_{k,s,g}$. To explain how to compute this table, we give the appropriate recurrence relation. For the base case $k = 0$ we let $\bar{U}_{0,s,g} \leftarrow r_s$ for all s and g. For $k \geq 1$, $\bar{U}_{k,s,g}$ is defined recursively as follows:

$$\bar{U}_{k,s,g} \quad \leftarrow \quad \max_{\substack{l<k \\ h \leq g}} \begin{cases} \bar{U}_{k-1,s,g} & \\ \bar{U}_{k-1,s,g} + 1 & \text{if } r_s \leq r_k \leq \bar{U}_{k-1,s,g} \\ d_k & \text{if } g \geq 1 \ \&\ (\, r_j < \bar{U}_{k-1,s,g-1} \ \forall j < k \,) \\ \bar{U}_{k-1,l,g-h} & \text{if } r_k < r_l = \bar{U}_{k-1,s,h} + 1 \end{cases} \tag{5}$$

Note that only the last option of the maximum depends on l and h, but we chose to express the recurrence in the above form to reduce clutter. Also, variables l and h are dependent: if we fix the value of one, then the other one's value is fixed as well (or it does not exist).

In the remainder of this section we justify the correctness of the algorithm and analyze its running

time. The first lemma establishes the feasibility and optimality of the values $\bar{U}_{k,s,g}$ computed by Algorithm ALGA. The main idea was explained earlier in this section and is quite simple, but the formal proof is rather involved. This is partially due to the fact that it does not seem possible to show the feasibility and optimality separately, because in some situations the feasibility of some (k,s)-schedules we construct depends on frugality (and thus also, indirectly, on optimality) of its $(k-1,s')$-sub-schedules.

Lemma 5 (correctness of ALGA) *Algorithm* ALGA *correctly computes the values $U_{k,s,g}$, that is $\bar{U}_{k,s,g} = U_{k,s,g}$ for all $k = 0,...,n$, $s = 1,...,n$ and $g = 0,...,n-1$.*

Proof: It is sufficient to show that the two following claims hold:

Feasibility: For any choice of indices k, s, g, there is a (k,s)-schedule $S_{k,s,g}$ with $C_{\max}(S_{k,s,g}) = \bar{U}_{k,s,g}$ and at most g gaps.

Optimality: For any choice of indices k, s, g, if Q is any (k,s)-schedule with at most g gaps then $C_{\max}(Q) \leq \bar{U}_{k,s,g}$.

The proof is by induction on k. Consider the base case first, for $k = 0$. To show feasibility, we take $S_{0,s,g}$ to be the empty (k,s)-schedule, which is trivially feasible and (by our convention) has completion time $r_s = \bar{U}_{0,s,g}$. The optimality condition follows from the fact that any $(0,s)$-schedule is empty and thus has completion time r_s.

Suppose now that the feasibility and optimality conditions hold for $k-1$. We will show that they hold for k as well.

Feasibility proof. By the inductive assumption, for any s' and g' we have a schedule $S_{k-1,s',g'}$ with completion time $\bar{U}_{k-1,s',g'} = U_{k-1,s',g'}$. By Lemma 3, $S_{k-1,s',g'}$ is frugal. The construction of $S_{k,s,g}$ depends on which expression realizes the maximum (5).

If $\bar{U}_{k,s,g} = \bar{U}_{k-1,s,g}$, we simply take $S_{k,s,g} = S_{k-1,s,g}$. Since we did not choose the second option in the maximum, either $r_k < r_s$ or $r_k > \bar{U}_{k-1,s,g}$. Therefore, directly from the inductive assumption, we get that $S_{k,s,g}$ is a (k,s)-schedule with completion time $\bar{U}_{k,s,g}$.

If $\bar{U}_{k,s,g} = \bar{U}_{k-1,s,g} + 1$, $r_s \leq r_k \leq \bar{U}_{k-1,s,g}$, then let $S_{k,s,g}$ be the schedule obtained from $S_{k-1,s,g}$ by adding to it job k scheduled at time $u = \bar{U}_{k-1,s,g}$. By the frugality of $S_{k-1,s,g}$, there is no job $j \leq k$ with $r_j = u$. We also have $u < d_k$, since $u \leq d_{k-1}$ and since we assumed that all jobs have distinct deadlines. Therefore $S_{k,s,g}$ is a (k,s)-schedule with completion time $u + 1 = \bar{U}_{k,s,g}$.

Next, suppose that $\bar{U}_{k,s,g} = d_k$, $g \geq 1$, and $\max_{j<k} r_j < \bar{U}_{k-1,s,g-1}$. Let $S_{k,s,g}$ be the schedule obtained from $S_{k-1,s,g-1}$ by adding to it job k scheduled at $d_k - 1$. The case condition implies that no jobs $j < k$ are released between $\bar{U}_{k-1,s,g-1}$ and $d_k - 1$. By the assumption about different deadlines, we also have $\bar{U}_{k-1,s,g-1} < d_k$. Therefore $S_{k,s,g}$ is a (k,s)-schedule with completion time $d_k = \bar{U}_{k,s,g}$ and it has at most g gaps, since adding k can add at most one gap to $S_{k-1,s,g-1}$.

Finally, suppose that $\bar{U}_{k,s,g} = \bar{U}_{k-1,l,g-h}$, for some $1 \le l < k$, $0 \le h \le g$, that satisfy $r_k < r_l = \bar{U}_{k-1,s,h} + 1$. The schedule $S_{k,s,g}$ is obtained by scheduling all jobs $j < k$ released between r_s and $r_l - 1$ using $S_{k-1,s,h}$, scheduling all jobs $j < k$ released between r_l and $\bar{U}_{k-1,l,g-h} - 1$ using $S_{k-1,l,g-h}$, and scheduling job k at $r_l - 1$. By the frugality of $S_{k-1,s,h}$, there is no job $j < k$ with $r_j = r_l - 1$. Thus $S_{k,s,g}$ is a (k, s)-schedule with completion time $\bar{U}_{k,s,g}$ and at most g gaps.

Optimality proof. For a given $k \ge 1$ assume that the lemma holds for $k - 1$ and any s' and g'. Let Q be a (k, s)-schedule with at most g gaps and completion time $u = C_{\max}(Q)$. We can assume that Q realizes $U_{k,s,g}$, that is, $u = U_{k,s,g}$. Without loss of generality, we can also assume that Q has the earliest-deadline property and is frugal. In particular, this implies that no job $j \le k$ is released at time u. We prove that $u \le \bar{U}_{k,s,g}$ by analyzing several cases.

Case 1: Q does not schedule job k. In this case Q is a $(k-1, s)$-schedule with completion time u, so, by induction, we have $u \le \bar{U}_{k-1,s,g} \le \bar{U}_{k,s,g}$.

In all the remaining cases, we assume that Q schedules k. Obviously, this implies that $r_s \le r_k < u$.

Case 2: Q schedules k as the last job and k is not the only job in its block. Let $u' = u-1$, and define Q' to be Q restricted to the interval $[r_s, u')$. Then Q' is a $(k-1, s)$-schedule with completion time u' and at most g gaps, so $u' \le \bar{U}_{k-1,s,g}$, by induction. If $u' < \bar{U}_{k-1,s,g}$ then, trivially, $u \le \bar{U}_{k-1,s,g} \le \bar{U}_{k,s,g}$. Otherwise, assume $u' = \bar{U}_{k-1,s,g}$. Since k is executed at time u' in Q, we have $r_k \le \bar{U}_{k-1,s,g}$, so the second option of the maximum (5) is applicable. Therefore $u = u' + 1 = \bar{U}_{k-1,s,g} + 1 \le \bar{U}_{k,s,g}$.

Case 3: Q schedules k as the last job and k is the only job in its block. If $u = r_s + 1$ then $k = s$ and the condition in the second option of (5) is satisfied, so we have $u = r_s + 1 = \bar{U}_{s-1,s,g} + 1 \le \bar{U}_{s,s,g}$. Thus we can assume now that $u > r_s + 1$, which, together with the case condition, implies that $g > 0$.

We can also assume that $u = d_k$, for otherwise we could modify Q by rescheduling k at time u, thus obtaining a (k, s)-schedule Q' (by frugality of Q, no job $j < k$ is released at u) with at most g gaps and $C_{\max}(Q') = u + 1$ — contradicting the maximality of u.

Let u' be the earliest time $u' \ge r_s$ such that Q is idle in $[u', d_k - 1)$. Then, by the feasibility of Q and the case condition, $\max_{j<k} r_j < u'$ and the segment of Q in $[r_s, u')$ is a $(k-1, s)$-schedule with at most $g - 1$ gaps. So, by induction, we get $u' \le \bar{U}_{k-1,s,g-1}$. Thus the third option in (5) applies and we get $u = d_k = \bar{U}_{k,s,g}$.

Case 4: Q schedules k and k is not the last job. Suppose that k is scheduled at time t. By the frugality of Q, k is neither the first nor last job in its block. Since Q satisfies the earliest-deadline property, no job $j < k$ is pending at time t, and thus Q schedules at time $t + 1$ the job $l < k$ with release time $r_l = t + 1$ (see Figure 6).

By Lemma 4 and induction, $t = U_{k-1,s,h} = \bar{U}_{k-1,s,h}$ for some $h \le g$. Then the conditions of the last option in (5) are met: $l < k$, $h \le g$, and $r_k < r_l = \bar{U}_{k-1,s,h} + 1$. Let Q' be the segment of Q in $[r_l, u)$. Then Q' is a $(k-1, l)$-schedule with completion time u and at most $g - h$ gaps, so by induction we get $u \le \bar{U}_{k-1,l,g-h} \le \bar{U}_{k,s,g}$, completing the argument for Case 4. □

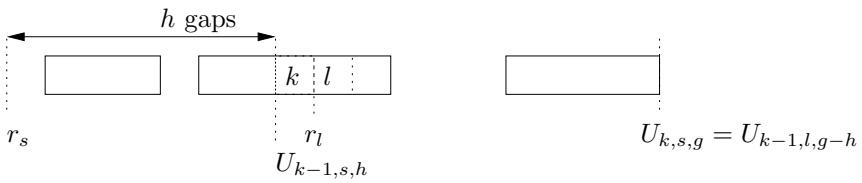

Figure 6: Illustration of Case 4.

Theorem 1 *Algorithm* ALGA *correctly computes the optimum solution for* $1|r_j; p_j = 1; L = 1|E$, *and it can be implemented in time* $O(n^4)$.

Proof: The correctness of Algorithm ALGA follows from Lemma 5, so it is sufficient to give the running time analysis. There are $O(n^3)$ values $\bar{U}_{k,s,g}$ to be computed. For fixed k, s, g, the first two choices in the maximum (5) can be computed in time $O(1)$ and the third choice in time $O(n)$. In the last choice we maximize only over pairs (l, h) that satisfy the condition $r_l = \bar{U}_{k-1,s,h} + 1$, and thus we only have $O(n)$ such pairs. Further, since the values of $\bar{U}_{k-1,s,h}$ increase with h, we can determine all these pairs in time $O(n)$ by searching for common elements in two sorted lists: the list of release times, and the list of times $\bar{U}_{k-1,s,h} + 1$, for $h = 0, 1, ..., n$. Thus each value $\bar{U}_{k,s,g}$ can be computed in time $O(n)$, and we conclude that the overall running time of Algorithm ALGA is $O(n^4)$. □

4 Minimizing the Number of Gaps for Arbitrary Jobs

In this section we give an $O(n^5)$-time algorithm for minimizing the number of gaps for instances with jobs of arbitrary lengths, that is for the scheduling problem $1|r_j; \text{pmtn}; L = 1|E$.

As in Algorithm ALGA, we focus on computing the function $U_{k,s,g}$. The new recurrence relations for $U_{k,s,g}$ are significantly more involved than in Algorithm ALGA, but the fundamental principle is quite intuitive (see Figure 7): Imagine a (k, s)-schedule S with at most g gaps that maximizes the completion time. If the last internal execution interval of k in S ends at v, then, by the earliest-deadline property we have $v = r_l$, for some job $l < k$. Further, the segment of S in $[r_s, v)$ must have a minimum number of units of k, for otherwise these units could be moved to the end of S increasing its completion time. We represent this minimum number of units of k in $[r_s, v)$ by another function $P_{k,s,l,h}$, where h is the number of gaps of S in $[r_s, v)$. On the other hand, the segment of S starting at v consists of a $(k-1, l)$-schedule followed by some number of units of k. This structure of S allows us to express $U_{k,s,g}$ in terms of $P_{k,s,l,h}$ and $U_{k-1,l,g-h}$.

The above intuition, although fundamentally correct, glosses over some important technical issues and ignores some special cases (for example, when S completes at d_k). To formalize this idea we need to establish some properties of optimal schedules. We proved some results about the structure of optimal schedules for unit jobs in the previous section; we now extend those results to jobs of arbitrary length.

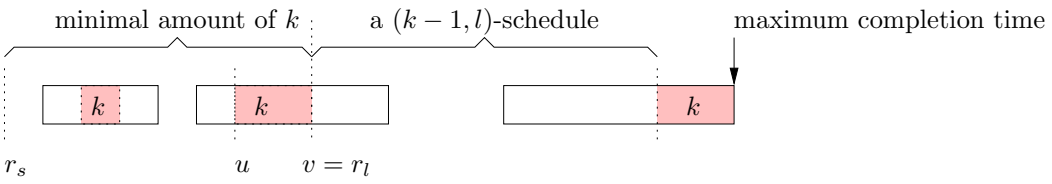

Figure 7: The fundamental idea of Algorithm AlgB.

Frugal (k,s)-schedules. Given a schedule S, by an *execution interval* $[u,v)$ of job k we mean an inclusion-wise maximal time interval where S executes k (that is, k is scheduled in each time unit inside $[u,v)$ but is not scheduled at times $u-1$ and v).

A (k,s)-schedule S is called *frugal* if it satisfies the following properties:

(f1) There is no job $j \le k$ with $r_j = C_{\max}(S)$, and

(f2) Suppose that $C_{\max}(S) < d_k$ and S schedules job k. Let $[u,v)$ be an execution interval of job k. Then the slot $u-1$ is not idle, and if v is idle then $v = C_{\max}(S)$.

Lemma 6 (frugality) *Fix some k, s, g, and let S be a (k,s)-schedule that realizes $U_{k,s,g}$, that is S has at most g gaps and $C_{\max}(S) = U_{k,s,g}$. Then S is frugal.*

Proof: If S violates (f1) then we can extend S as follows. Let $t = C_{\max}(S)$ and $w > t$ be the smallest time such that

$$w \ge t + load_k(t,w). \tag{6}$$

(Recall that $load_k(t,w) = \sum_{j \le k,\, t \le r_j < w} p_j$.) This time w can be found simply by setting initially $w = t+1$, and iteratively replacing w by the right-hand side of (6). Note that for this time w we have in fact equality in (6). We can extend S by the time interval $[t,w)$ in which we schedule all jobs $j < k$ with $t \le r_j < w$, according to the earliest-deadline policy. The result is a (k,s)-schedule with at most g gaps, contradicting the maximality of S.

Now assume that S satisfies (f1) but not (f2). Let $[u,v)$ be some execution interval of job k in S. If S is idle at time $u-1$, then we can move one unit of job k from u to $t = C_{\max}(S) < d_k$. If $v < t$ and S is idle at v, then we can proceed in the same manner, moving one unit of job k from $v-1$ to t. In both cases, by (f1), we obtain a (k,s)-schedule. This schedule has at most g gaps and completion time $t+1$, contradicting the maximality of S. □

Function $U_{k,s,g}(p)$. Now we extend the definition of $U_{k,s,g}$ as follows. First, for any integer $p \ge 0$, we define a (k,s,p)-*schedule* as a (k,s)-schedule for the modified instance where we change the processing time of k to p, that is $p_k \leftarrow p$. (All release times, deadlines, and the processing times of jobs other than k remain unchanged.) For $p = 0$, the notion of a $(k,s,0)$-schedule is equivalent to a $(k-1,s)$-schedule. Let $0 \le k \le n$, $1 \le s \le n$, $0 \le g \le n-1$ and $p \ge 0$. We then define $U_{k,s,g}(p)$ as

the maximum completion time of a (k, s, p)-schedule with at most g gaps. Naturally, for $p = 0$, we have $U_{k,s,g}(0) = U_{k-1,s,g}$.

The following lemma will be useful in the proof of correctness of our algorithm.

Lemma 7 (expansion) *Fix any k, s, g and $p < p_k$. If $U_{k,s,g}(p) < d_k$, then $U_{k,s,g}(p+1) > U_{k,s,g}(p)$ and if $U_{k,s,g}(p) = d_k$, then $U_{k,s,g}(p+1) = d_k$ as well.*

Proof: Let S be a schedule that realizes $U_{k,s,g}(p)$. By Lemma 6 we know that S is frugal. So in case $C_{\max}(S) < d_k$, appending one unit of job k at $C_{\max}(S)$ produces a (k, s)-schedule with at most g gaps, and shows that $U_{k,s,g}(p+1) > U_{k,s,g}(p)$.

Now consider the case $C_{\max}(S) = d_k$ and let $[u, d_k)$ be the last block of S. We extend the support of S by the time unit $[u-1, u)$. Set $p_k \leftarrow p+1$ and schedule jobs using the earliest-deadline rule inside this new support. This new schedule S' will be identical to S in $[r_s, u-1)$.

First we claim that in S' the unit $u-1$ will not remain idle. Indeed, otherwise we would have that all jobs scheduled in $[u, d_k)$ are released in that interval. These jobs include job k whose one unit is scheduled at $d_k - 1$, by the assumption about different deadlines. Since $p < p_k$, this would contradict the feasibility assumption (1) for $v = d_k$. (Note that it is not necessarily job k that is scheduled at $u-1$.) Second, in this new schedule no job will complete later than in S, so all deadlines are met. This shows that $U_{k,s,g}(p+1) = d_k$, as claimed. □

Schedule compression. In the previous section, in the proof of the partitioning lemma, at one point we were gradually compressing a unit-jobs schedule. We generalize this operation now to arbitrary-length jobs.

Fix any k', s, p. (We use notation k' now instead of k, to avoid confusion later in this section where the results derived below will be used with either $k' = k-1$ or $k' = k$. Later, in Section 5 we will use $k' = n$). Let T be some (k', s)-schedule and $[w, v)$ the last block in T, where $v = C_{\max}(T)$. The compression of T consists of reducing its completion time, without increasing the number of gaps. It is accomplished by applying one of the steps below, Truncate or ShiftBack, depending on whether the slot $v-1$ of T is fixed or not.

Truncate: Suppose that slot $v-1$ is fixed, and let $[r_i, v)$ be the fixed segment containing $v-1$, with maximal r_i. The job i can be found by a simple procedure: Initially, let i be the job scheduled at $v-1$. Then iteratively replace i with the job j scheduled in $[r_i, v)$ that minimizes r_j, until a fix-point is reached.

Now, remove $[r_i, v)$ from T and let T' be the resulting schedule. By definition of fixed segments, all jobs scheduled in $[r_i, v)$ are released in this segment. Therefore T' is a (k', s)-schedule, and if $r_i - 1$ is idle (and $i \neq s$), T' has one gap less than T, otherwise the number of gaps remains the same. By the definition of fixed schedules, T' schedules all jobs of T that are released before r_i.

ShiftBack: Suppose that slot $v-1$ of T is not fixed. In this case we modify T as follows: For each non-fixed slot in $[w, v)$, move the job unit in this slot to the previous non-fixed slot. The job unit scheduled in the first non-fixed slot in this block will move to $w-1$. Let T' be the resulting schedule.

Note that if t, $w \le t < v$, is a non-fixed slot executing some job i and $t' < t$ is the previous non-fixed slot (that is, all slots between $t'+1$ and t are fixed), then, by the definition of fixed slots, we have $r_i \le t'$. Therefore shifting the schedule, as above, will not violate release times, and we conclude that T' is a (k', s)-schedule with $C_{\max}(T') = C_{\max}(T) - 1$. If $w-2$ is not idle, T' has one gap less than T, otherwise the number of gaps remains the same. Also, T' schedules all jobs of T.

Both operations, Truncate and ShiftBack, convert T into another (k', s)-schedule T' with $C_{\max}(T') < C_{\max}(T)$, and with the number of gaps in T' not exceeding the number of gaps in T. In what follows, we will also use the fact that ShiftBack reduces the completion time only by 1.

Lemma 8 (compression lemma) *Fix any k', s, and consider a time step $t \ge r_s$ that satisfies the following condition: for each job $j \le k'$, if $r_s \le r_j < t$ then $C^{\mathrm{ED}}_{j,s} \le t$. Suppose that there is a (k', s)-schedule Q with completion time $C_{\max}(Q) > t$ and at most g gaps. Then there is a (k', s)-schedule R that schedules all jobs $j \le k'$ with $r_s \le r_j < t$ and satisfies the following properties:*

(a) *$C_{\max}(R) \le t$ and the number of gaps in R is at most g, and*

(b) *if $C_{\max}(R) < t$ then the number of gaps in R is strictly less than g.*

Proof: Starting from Q, we repeatedly apply the compression steps Truncate and ShiftBack described above, until we obtain a schedule R with $C_{\max}(R) \le t$. As explained above, the compression steps do not increase the number of gaps and R schedules all jobs of Q released before t. Thus (a) holds.

To prove (b), suppose $C_{\max}(R) < t$. Since ShiftBack reduces the completion time by 1 only, this is possible only if the compression process ended with a Truncate step. Denote by T the schedule right before this step and let $[r_i, v)$ be the fixed segment truncated from T in this step, where $C_{\max}(T) = v > t$.

If $r_i \ge t$ then, since $C_{\max}(R) < t$, T had a gap $[C_{\max}(R), r_i)$ that will be eliminated in the last step. So the number of gaps in R is strictly less than g.

Thus, to complete the proof, it is sufficient to show that we must have $r_i \ge t$. Towards contradiction, suppose that $r_i < t$. All slots of T in $[t, v)$ are fixed, so, by the assumptions of the lemma and by Lemma 2, they cannot contain any jobs released before t. But then the choice of r_i in procedure Truncate implies that $r_i < t$ is not possible, as claimed. □

Function $P_{k,s,l,g}$. We now extend somewhat the notion of gaps. Let S be a (k, s)-schedule and $t \ge C_{\max}(S)$. A *gap of S with respect to* $[r_s, t)$ is either a gap of S (as defined before) or the interval

$[C_{\max}(S), t)$, if $C_{\max}(S) < t$.

For any job k' and time t, let $prevr_{k'}(t)$ be the latest release of a job $j \leq k'$ before t, that is

$$prevr_{k'}(t) \quad = \quad \max\{r_j : j \leq k' \,\&\, r_j < t\}.$$

If there is no such job j, we take $prevr_{k'}(t) = -\infty$.

We define another table $P_{k,s,l,g}$, where the indices range over all $k = 1, \ldots, n$, $s = 1, \ldots, n$, $g = 0, \ldots, n-1$ and $l = 1, \ldots, k-1$ for which $r_l \geq r_s$. $P_{k,s,l,g}$ is the minimum amount $p \geq 0$ of job k for which there is a (k, s, p)-schedule S that satisfies $prevr_{k-1}(r_l) < C_{\max}(S) \leq r_l$ and has at most g gaps with respect to $[r_s, r_l)$. By convention, $P_{k,s,l,g} = +\infty$ if there is no such p. In particular, for $r_l = r_s$ we have $P_{k,s,s,g} = 0$ (this value is realized by the empty schedule). Note also that $P_{k,s,l,g}$ is defined when $r_k < r_s$ or $r_k \geq r_l$, although in those cases its value can only be 0 or $+\infty$, depending on whether there exists or not a $(k-1, s)$-schedule S that satisfies the condition above.

Lemma 9 (extremal values of P) (a) *If there is a job $j < k$ released in $[r_s, r_l)$ with $C^{\mathrm{ED}}_{j,s} > r_l$, then $P_{k,s,l,g} = +\infty$.*

(b) *$P_{k,s,l,g} = 0$ if and only if $U_{k-1,s,g} \geq r_l$ and every job $j < k$ released in $[r_s, r_l)$ satisfies $C^{\mathrm{ED}}_{j,s} \leq r_l$.*

Proof: To show (a), suppose that for some (finite) p there is a (k, s, p)-schedule S with $P_{k,s,l,g} = p$. Then, by the definition of $P_{k,s,l,g}$, every job $j \leq k$ released in $[r_s, r_l)$ is scheduled by S and therefore $C^{\mathrm{ED}}_{j,s} \leq r_l$.

We now show (b). Suppose that $P_{k,s,l,g} = 0$. By part (a), every job $j < k$ released in $[r_s, r_l)$ satisfies $C^{\mathrm{ED}}_{j,s} \leq r_l$. Let S be a $(k-1, s)$-schedule that realizes $P_{k,s,l,g}$. In particular, S schedules all jobs $j < k$ released in $[r_s, r_l)$. Let T be the $(k-1, l)$-schedule with completion time $U_{k-1,l,0}$ and no gaps. Note that T is not empty, since it schedules r_l. Then the union of S and T is a $(k-1, s)$-schedule with at most g gaps and completion time at least r_l, which shows $U_{k-1,s,g} \geq r_l$.

To show the reverse implication, assume that $U_{k-1,s,g} \geq r_l$ and that every job $j < k$ released in $[r_s, r_l)$ satisfies $C^{\mathrm{ED}}_{j,s} \leq r_l$. Let S be a $(k-1, s)$ schedule that realizes $U_{k-1,s,g}$, that is, S has at most g gaps and completion time $U_{k-1,s,g} \geq r_l$. If we have equality we are done. Otherwise, S satisfies the assumptions of the compression lemma, Lemma 8 (with $k' = k-1$ and $t = r_l$). By applying this lemma, we obtain a $(k-1, s)$-schedule R with $C_{\max}(R) \leq r_l$. The conditions (a) and (b) of Lemma 8 imply that R has at most g gaps with respect to $[r_s, r_l)$. $\square$

Let S be a (k, s, p)-schedule. An execution interval $[u, v)$ of job k in S is called an *internal execution interval of k* if (i) v is not idle and (ii) $u - 1$ is not idle or $u = r_s$. By extension, if $C_{\max}(S) \leq t$, we call $[u, v)$ an *internal execution interval of k with respect to* $[r_s, t)$ if (i) v is not idle or $v = t$, and (ii) $u-1$ is not idle or $u = r_s$. Intuitively, an execution interval is internal if its removal creates a gap.

Lemma 10 (internal execution intervals) *Let $p = P_{k,s,l,g}$ and assume $p < +\infty$. Let S be a (k,s,p)-schedule that realizes $P_{k,s,l,g}$. Then*

(a) *Every execution interval of k in S is an internal execution interval with respect to $[r_s, r_l)$. Moreover, if $p > 0$ then S contains exactly g gaps with respect to $[r_s, r_l)$.*

(b) *Let $[u,t)$ be the some execution interval of k, h be the number of gaps before u in S, and q the amount of k scheduled in $[r_s, u)$ by S. Then $u = U_{k,s,h}(q)$.*

Proof: Part (a) of the lemma follows simply from the minimality of p. If S had an non-internal execution interval of k, we can remove this interval, reducing p, without increasing the number of gaps. Similarly, if the number of gaps is more than g, we can remove any execution interval of k.

We now show part (b). By (a), $[u,t)$ is an internal execution of k. By the earliest deadline property, all jobs $j < k$ with $r_s \le r_j < u$ are completed before u. So the segment of S between r_s and u is a (k,s,q)-schedule with h gaps and completion time u (because either $u = r_s$ or slot $u-1$ is not idle), so $U_{k,s,h}(q) \ge u$.

To show equality, we consider the modified instance where $p_k \leftarrow q$. For this modified instance, $C^{\mathrm{ED}}_{k,s} \le u$. Also, by the earliest deadline policy, every job $j < k$ released in $[r_s, t)$ completes not later than at u in S (in particular, no job $j < k$ is released in $[u,t)$). Therefore $C^{\mathrm{ED}}_{j,s} \le u$. Now, we proceed by contradiction and assume $U_{k,s,h}(q) > u$. Let Q be a (k,s,q)-schedule with at most h gaps and completion time $U_{k,s,h}(q)$. By the compression lemma, Lemma 8(b) (with $k' = k$), there is a (k,s,q)-schedule R scheduling all jobs $j < k$ released in $[r_s, u)$ that satisfies condition (a) and (b). of that lemma. We distinguish two cases.

If $u < C_{\max}(R) \le t$ and R has at most h gaps, then let S' be the result of replacing in S the portion between r_s and $C_{\max}(R)$ by R. Then S' is a (k,s,p')-schedule with at most g gaps with respect to $[r_s, r_l)$, where $p' = p - t + C_{\max}(R) < p$, contradicting the minimality of p.

If $C_{\max}(R) \le u$ then R has at most $h-1$ gaps. Let S' be the union of R and the portion of S between t and r_l. Then S' has at most h gaps in $[r_s, t)$. Therefore S' is a (k,s,p')-schedule with at most g gaps with respect to $[r_s, r_l)$, where $p' = p - t + u < p$, again contradicting the minimality of p. □

Outline of the algorithm. The algorithm in this section computes both functions $U_{k,s,g}$ and $P_{k,s,l,g}$. The intuition is this. Let S be a (k,s)-schedule that realizes $U_{k,s,g}$, that is S has at most g gaps and completion time $u = C_{\max}(S) = U_{k,s,g}$. If S does not schedule k then $u = U_{k-1,s,g}$.

So assume that S schedules job k. There are several cases. Consider, for example, the case when $u < d_k$ and when k has an execution interval $[t', t)$ with $t < u$. (See the third case in Figure 8.) Take $[t', t)$ to be the last such interval. Since S is frugal, we know that S is not idle at $t' - 1$ and at t. Then, by the earliest-deadline policy, S schedules at t some job $l < k$ with $r_l = t$. Now, the part of

S up to r_l has some number of gaps, say h. The key idea is that, roughly, the amount q of job k in this part is minimal among all (k, s, q)-schedules with completion time r_l and at most h gaps, so this amount is equal to $P_{k,s,l,h}$. Otherwise, if it were not minimal, then we could replace the part of S before t by a (k, s, q')-schedule for some $q' < q$ and this would imply $U_{k,s,g}(p) \geq U_{k,s,g}(p_k)$ for $p = p_k + q' - q < p_k$, contradicting Lemma 7. By the choice of $[t', t)$, and induction, the interval $[t, u)$ of S consists of a $(k-1, l)$-schedule with at most $g - h$ gaps followed by $p_k - P_{k,s,l,h}$ units of k, and thus $U_{k,s,g}$ can be expressed as $U_{k-1,l,g-h} + p_k - P_{k,s,l,h}$.

If $u < d_k$ and k has just one execution segment ending at u, then there is no segment $[t', t)$ considered above. But then the formula $U_{k-1,l,g-h} + p_k - P_{k,s,l,h}$ applies as well, since we can take $l = s$ and $h = 0$, and then $P_{k,s,l,g} = 0$, so in this case $U_{k,s,g}$ will be equal to $U_{k-1,s,g} + p_k$.

The remaining case, when $u = d_k$, breaks into two sub-cases depending on whether the last block contains only units of k or not. In order to determine whether it is possible to achieve $u = d_k$ with only g gaps, we proceed in a similar manner, by partitioning the schedule using the second last execution interval $[t', t)$ of k (if it exists).

The idea behind the recurrence for $P_{k,s,l,g}$ is similar – essentially, it consists of partitioning the schedule realizing $P_{k,s,l,g}$ into disjoint sub-schedules, with the first one ending at a release time of some job j.

Algorithm ALGB**.** The algorithm computes the values of $U_{k,s,g}$ and $P_{k,s,l,g}$ in order of increasing k and stores these values in tables $\bar{U}_{k,s,g}$ and $\bar{P}_{k,s,l,g}$.

First, for $k = 0$, we initialize $\bar{U}_{0,s,g} \leftarrow r_s$ for all $s = 1, ..., n$ and $g = 0, ..., n-1$. Then, for $k = 1, ..., n$ we do the following:

- Compute $\bar{P}_{k,s,l,g}$ for all $s = 1, ..., n$, $g = 0, ..., n-1$, and for $l = 1, ..., k-1$ such that $r_l \geq r_s$. The indices l are processed in order of increasing r_l.
- Compute $\bar{U}_{k,s,g}$ for all $s = 1, ..., n$ and $g = 0, ..., n-1$.

The values of $\bar{P}_{k,s,l,g}$ and $\bar{U}_{k,s,g}$ are computed using the recurrence relations described below. Once all these values are computed, the algorithm determines the minimum number of gaps as the smallest g for which $U_{n,1,g} > \max_j r_j$.

Computing $\bar{P}_{k,s,l,g}$. If there is a job $j < k$ such that $r_s \leq r_j < r_l$ and $C^{\text{ED}}_{j,s} > r_l$, then $\bar{P}_{k,s,l,g} \leftarrow +\infty$. Otherwise, we have that every job $j < k$ such that $r_s \leq r_j < r_l$ satisfies $C^{\text{ED}}_{j,s} \leq r_l$. If $r_s = r_l$ or $\bar{U}_{k-1,s,g} \geq r_l$ then $\bar{P}_{k,s,l,g} \leftarrow 0$. In the remaining case, we have $r_s < r_l$ and $\bar{U}_{k-1,s,g} < r_l$. We then compute $\bar{P}_{k,s,l,g}$ recursively as follows:

$$\bar{P}_{k,s,l,g} \leftarrow \min_{\substack{0 \leq h \leq g \\ j < k \\ r_s \leq r_j < r_l}} \left\{ r_j - \bar{U}_{k-1,s,h} + \bar{P}_{k,j,l,g-h} : prevr_{k-1}(r_j) < \bar{U}_{k-1,s,h} < r_j \;\&\; r_k \leq \bar{U}_{k-1,s,h} \right\} \quad (7)$$

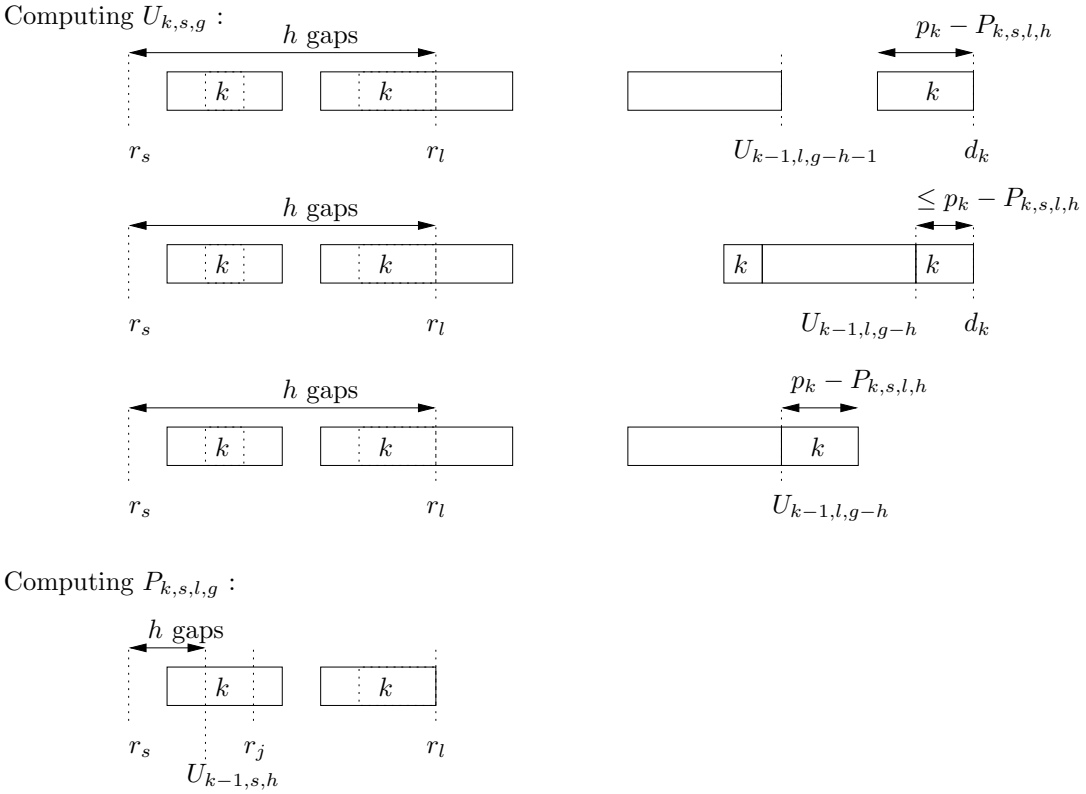

Figure 8: Idea of the proof of Lemma 11.

As usual, by default, if the conditions in the minimum are not satisfied by any h, j, then $\bar{P}_{k,s,l,g}$ is assumed to be $+\infty$.

Computing $\bar{U}_{k,s,g}$. $\bar{U}_{k,s,g}$ is computed recursively as follows. If $r_k < r_s$ or $r_k > \bar{U}_{k-1,s,g}$ then we let $\bar{U}_{k,s,g} \leftarrow \bar{U}_{k-1,s,g}$. Otherwise, for $r_s \leq r_k \leq \bar{U}_{k-1,s,g}$, we let

$$\bar{U}_{k,s,g} \leftarrow \max_{\substack{l<k \\ h\leq g}} \begin{cases} d_k & \text{if } \bar{P}_{k,s,l,h} < p_k, \\ & d_k - \bar{U}_{k-1,l,g-h-1} > p_k - \bar{P}_{k,s,l,h} \text{ and} \\ & \bar{U}_{k-1,l,g-h-1} > prevr_{k-1}(d_k) \\ d_k & \text{if } \bar{P}_{k,s,l,h} < p_k, \\ & d_k - \bar{U}_{k-1,l,g-h} \leq p_k - \bar{P}_{k,s,l,h} \text{ and} \\ & \bar{U}_{k-1,l,g-h} > prevr_{k-1}(d_k) \\ \bar{U}_{k-1,l,g-h} + p_k - \bar{P}_{k,s,l,h} & \text{if } \bar{P}_{k,s,l,h} \leq p_k, \\ & d_k - \bar{U}_{k-1,l,g-h} > p_k - \bar{P}_{k,s,l,h} \text{ and} \\ & \bar{U}_{k-1,l,g-h} > prevr_{k-1}(\bar{U}_{k-1,l,g-h} + p_k - \bar{P}_{k,s,l,h}) \end{cases} \tag{8}$$

Lemma 11 (correctness of ALGB**)** *Algorithm* ALGB *correctly computes the values of* $U_{k,s,g}$ *and* $P_{k,s,l,g}$. *More specifically, for all* $k = 0, ..., n$, $s = 1, ..., n$, *and* $g = 0, ..., n-1$ *we have* $\bar{U}_{k,s,g} = U_{k,s,g}$ *and* $\bar{P}_{k,s,l,g} = P_{k,s,l,g}$ *for* $k > 0$ *and all* $l = 1, ..., k-1$.

Proof: We show that there are schedules that realize the values $\bar{U}_{k,s,g}$ and $\bar{P}_{k,s,l,g}$ (the feasibility condition) and that these values are indeed optimal. More specifically, we prove the following four properties.

Feasibility of $\bar{P}_{k,s,l,g}$: For each $k > 0$, s, l and g for which $\bar{P}_{k,s,l,g} = p < +\infty$ there is a (k, s, p)-schedule $T_{k,s,l,g}$ with $prevr_{k-1}(r_l) < C_{\max}(T) \leq r_l$ and at most g gaps with respect to $[r_s, r_l)$.

Optimality of $\bar{P}_{k,s,l,g}$: $\bar{P}_{k,s,l,g} \leq P_{k,s,l,g}$, for all $k > 0$, s, l and g.

Feasibility of $\bar{U}_{k,s,g}$: For each k, s and g there is a (k, s)-schedule $S_{k,s,g}$ with completion time $\bar{U}_{k,s,g}$ and at most g gaps.

Optimality of $\bar{U}_{k,s,g}$: $\bar{U}_{k,s,g} \geq U_{k,s,g}$, for all k, s and g.

The proof is by induction on k. Consider first $k = 0$. In this case we only need to prove the feasibility and optimality of $\bar{U}_{0,s,g}$ (since $P_{k,s,l,g}$ and $\bar{P}_{k,s,l,g}$ are not defined for $k = 0$). We take $S_{0,s,g}$ to be the empty schedule, which is trivially feasible and has completion time $r_s = \bar{U}_{0,s,g}$. On the other hand, there is only one $(0, s)$-schedule, namely the empty schedule, which has completion time r_s, proving the optimality of $\bar{U}_{0,s,g}$.

Now fix some k, s, l, g with $k \geq 1$. Assume that the feasibility and optimality condition for $\bar{U}_{k-1,s,g}$ is true for any s', g'. We show the feasibility and optimality of $\bar{P}_{k,s,l,g}$.

Feasibility of $\bar{P}_{k,s,l,g}$: We prove the existence of $T_{k,s,l,g}$ by induction on $r_l - r_s$. If $r_l = r_s$ then we take $T_{k,s,s,g}$ to be the empty schedule.

So assume now that $r_r < r_l$. We can also assume that every job $j < k$ released in $[r_s, r_l)$ satisfies $C^{\text{ED}}_{j,s} \leq r_l$ (for otherwise $\bar{P}_{k,s,l,g} = +\infty$). If $U_{k-1,s,g} \geq r_l$ then by Lemma 9 we have $P_{k,s,l,g} = 0$; in other words, there is a $(k-1, s)$-schedule T with at most g gaps with respect to $[r_s, r_l)$. Thus in this case we can take $T_{k,s,l,g} = T$.

Consider now the case $U_{k-1,s,g} < r_l$, when the algorithm will compute $\bar{P}_{k,s,l,g}$ using the recurrence (7). Let h, j be the values that realize the minimum in (7) and denote $u = U_{k-1,s,h}$. By the case condition, $\bar{P}_{k,j,l,g-h}$ is finite, $p = r_j - u + \bar{P}_{k,j,l,g-h}$, $r_k \leq u$ and $prevr_{k-1}(r_j) < u < r_j$. The last inequality means that there are no jobs $i < k$ released in $[u, r_j)$. We let $T_{k,s,l,g}$ be the union of schedules $S_{k-1,s,h}$ and $T_{k,j,l,g-h}$ (that both exist, by induction), with additional $r_j - u$ units of k scheduled in the interval $[u, r_j)$. Then $T_{k,s,l,g}$ is a feasible (k, s, p)-schedule with at most g gaps with respect to $[r_s, r_l)$.

Optimality of $\bar{P}_{k,s,l,g}$: The proof is by induction on $r_l - r_s$. For the base case $r_s = r_l$ we have $\bar{P}_{k,s,s,g} = 0 \leq P_{k,s,s,g}$. Now assume $r_l > r_s$.

We can assume $P_{k,s,l,g} < +\infty$, since otherwise $\bar{P}_{k,s,l,g} \leq P_{k,s,l,g}$ is trivial. Then, by Lemma 9(a), every job $j < k$ released in $[r_s, r_l)$ satisfies $C^{\text{ED}}_{j,s} \leq r_l$. If $\bar{P}_{k,s,l,g} = 0$ then $\bar{P}_{k,s,l,g} \leq P_{k,s,l,g}$ is trivial

again, so we can assume that $\bar{P}_{k,s,l,g} > 0$. By the algorithm, this implies that $\bar{U}_{k-1,s,g} < r_l$ (because the value of recurrence (7) cannot be 0). Therefore by Lemma 9 we have $P_{k,s,l,g} > 0$.

Let T be a schedule that realizes $P_{k,s,l,g} = p$, that is T is a (k, s, p)-schedule with $prevr_{k-1}(r_l) < C_{\max}(T) < r_l$ and at most g gaps with respect to $[r_s, r_l)$. Let $[u, t)$ be the first execution interval of k in T and h the number of gaps before u. By Lemma 10(a), $[u, t)$ is an internal execution interval of T with respect to $[r_s, r_l)$, so there is a job $j < k$ with $r_j = t$. By the minimality of p, the segment of T in $[r_j, r_l)$ schedules $P_{k,j,l,g-h}$ units of k and, by the induction hypothesis, this equals $\bar{P}_{k,j,l,g-h}$. By Lemma 10(b) we have $u = U_{k-1,s,h}$ which by the induction hypothesis equals $\bar{U}_{k-1,s,h}$. The earliest-deadline policy applied to T implies there is no job $i < k$ released in $[u, r_j)$, that is $prevr_{k-1}(r_j) < u < r_j$. Therefore h, j are a valid choice for the recurrence (7), and $\bar{P}_{k,s,l,g} \leq p$ follows.

At this point we can assume the feasibility and optimality condition for $\bar{P}_{k,s',l',g'}$ and $\bar{U}_{k-1,s',g'}$, for any s', l' and g'. Thus in the rest of the proof we will interchangingly use notations $\bar{P}_{k,s',l',g'}$ and $P_{k,s,l,g}$, as well as $\bar{U}_{k-1,s',g'}$ and $U_{k-1,s,g}$, without an explicit reference to the inductive assumption. We show the feasibility and optimality of $\bar{U}_{k,s,g}$.

Feasibility of $\bar{U}_{k,s,g}$: Here we will show how we can construct $S_{k,s,g}$ using the recurrence for $\bar{U}_{k,s,g}$. We consider cases corresponding to those in the algorithm.

Suppose first that $r_k < r_s$ or $r_k > \bar{U}_{k-1,s,g}$, in which case $\bar{U}_{k,s,g} = \bar{U}_{k-1,s,g}$. In this case we take $S_{k,s,g} = S_{k-1,s,g}$. By induction, $S_{k,s,g}$ is a feasible $(k-1, s)$-schedule with completion time $\bar{U}_{k,s,g}$, and the condition on r_k implies that $S_{k,s,g}$ is also a feasible (k, s)-schedule.

For the rest of the feasibility proof, assume that $r_s \leq r_k \leq \bar{U}_{k-1,s,g}$. We now show the following claim:

($*$) There is a choice for h and l for which at least one of the options in the maximum (8) applies.

To show ($*$), we distinguish some cases. Let $v = \min\{\bar{U}_{k-1,s,g} + p_k, d_k\}$. If $\bar{U}_{k-1,s,g} > prevr_{k-1}(v)$ then we can choose $h = 0$ and $l = s$. In this case we have $\bar{P}_{k,s,s,0} = 0 < p_k$ and either the second or the third case applies (depending on whether $v = d_k$ or $v = \bar{U}_{k-1,s,g} + p_k$).

Otherwise, $\bar{U}_{k-1,s,g} \leq prevr_{k-1}(v)$, that is, there is $j < k$ such that $\bar{U}_{k-1,s,g} \leq r_j < v$. Choose such j with smallest r_j. By induction, $u = \bar{U}_{k-1,s,g} = U_{k-1,s,g}$ is maximum, so by frugality (Lemma 6) we cannot have $r_j = u$. Since $0 < r_j - u \leq p_k$, we could extend $S_{k-1,s,g}$ by scheduling $r_j - u$ units of k in the interval $[u, r_j)$. We can thus conclude that there is a job $j < k$ released in $[r_s, d_k)$ for which we have $\bar{P}_{k,s,j,g} \leq p_k$.

Now choose $l < k$ to be the job with maximum r_l for which $\bar{P}_{k,s,l,g} \leq p_k$. If $\bar{P}_{k,s,l,g} = p_k$, the the third option in (8) applies. If $\bar{P}_{k,s,l,g} < p_k$, let $v' = \min\{\bar{U}_{k-1,l,0} + p_k - \bar{P}_{k,s,l,g}, d_k\}$. By the choice of l, there are no jobs $i < k$ released in $[\bar{U}_{k-1,l,0}, v')$, since otherwise we could choose an l with larger r_l. In other words, $\bar{U}_{k-1,l,0} > prevr_{k-1}(v')$. Then in (8) we can choose this l and $h = g$, and either

option two or three will apply, depending on whether $v' = d_k$ or $v' = \bar{U}_{k-1,l,0} + p_k - \bar{P}_{k,s,l,g}$. This completes the proof of $(*)$.

Continuing the feasibility proof of $\bar{U}_{k,s,g}$ we need to construct $S_{k,s,g}$ for the three cases in the maximum (8). Let h and l be the values from Claim $(*)$.

If $\bar{U}_{k,s,g}$ is realized by the first option, we have $\bar{U}_{k,s,g} = d_k$, $\bar{P}_{k,s,l,h} < p_k$, $d_k - \bar{U}_{k-1,l,g-h-1} > p_k - \bar{P}_{k,s,l,h}$, and $\bar{U}_{k-1,l,g-h-1} > prevr_{k-1}(d_k)$. Let $p = p_k - \bar{P}_{k,s,l,h}$. Then we take $S_{k,s,g}$ to be the union of $T_{k,s,l,h}$ and $S_{k-1,l,g-h-1}$, with additional p units of k scheduled in the interval $[d_k - p, d_k)$. The union of $T_{k,s,l,h}$ and $S_{k-1,l,g-h-1}$ contains at most $g-1$ gaps, and it schedules $\bar{P}_{k,s,l,h} < p_k$ amount of job k. Scheduling the remaining units of k in $[d_k - p, d_k)$ will create one more gap. Overall then, $S_{k,s,g}$ is a feasible (k,s)-schedule with completion time d_k and at most g gaps.

If $\bar{U}_{k,s,g}$ is realized by the second option, we have $\bar{U}_{k,s,g} = d_k$, $\bar{P}_{k,s,l,h} < p_k$, $d_k - \bar{U}_{k-1,l,g-h} \leq p_k - \bar{P}_{k,s,l,h}$, and $\bar{U}_{k-1,l,g-h} > prevr_{k-1}(d_k)$. Let $u = \bar{U}_{k-1,l,g-h}$, and $p = \bar{P}_{k,s,l,h} + d_k - u$. Now let S be the union of the $T_{k,s,l,h}$ and $S_{k-1,l,g-h}$ followed by $d_k - u$ units of job k. Then S is a (k,s,p)-schedule with completion time d_k and at most g gaps. By Lemma 7, there is a (k,s)-schedule $S_{k,s,g}$ (scheduling all p_k units of job k) that has the same properties as S.

Finally, suppose that $\bar{U}_{k,s,g}$ is realized by the last option. Then $\bar{U}_{k,s,g} = \bar{U}_{k-1,l,g-h} + p_k - \bar{P}_{k,s,l,h}$, $\bar{P}_{k,s,l,h} \leq p_k$, $d_k - \bar{U}_{k-1,l,g-h} > p_k - \bar{P}_{k,s,l,h}$, and $u = \bar{U}_{k-1,l,g-h} > prevr_{k-1}(t)$ for $t = \bar{U}_{k,s,g}$. Then we define $S_{k,s,g}$ to be a union of $T_{k,s,l,h}$ and $S_{k-1,l,g-h}$, with additional $t-u$ units of k scheduled in the interval $[u,t)$. By induction, and since there are no jobs $j < k$ released in $[u,t)$, $S_{k,s,g}$ is a feasible (k,s)-schedule with completion time t and at most g gaps.

Optimality of $\bar{U}_{k,s,g}$: We start this proof with the following observation: For $p = 0, 1, ..., p_k$, let $\bar{U}_{k,s,g}(p)$ be the value computed by the algorithm for the modified instance where $p_k \leftarrow p$. We claim that $\bar{U}_{k,s,g}(p) \leq \bar{U}_{k,s,g}(p+1)$. To justify this, we start with the value of $\bar{U}_{k,s,g}(p)$ and see how the choice of the algorithm in (8) will be affected by increasing p to $p+1$. Except for p_k itself, all values in the right hand side of (8) — for example $\bar{P}_{k,s,l,h}$, $\bar{U}_{k-1,l,g-h-1}$ or $prevr_{k-1}(d_k)$ — do not depend on p_k. (In particular, although $\bar{P}_{k,s,l,h}$ involves subscript k, its value is actually independent of p_k.) If $\bar{U}_{k,s,g}(p)$ is realized by option two, then $\bar{U}_{k,s,g}(p+1)$ will be also realized by option two, so its value remains d_k. If $\bar{U}_{k,s,g}(p)$ is realized by option one, then $\bar{U}_{k,s,g}(p+1)$ will be realized either by option one or option two (this uses the fact that $U_{k-1,l,g-h} \geq U_{k-1,l,g-h-1}$), and thus its value remains d_k as well. Finally, suppose $\bar{U}_{k,s,g}(p)$ is realized by option three. The value of this option is increasing with p_k, and the other two options are larger, so in this case it does not matter which option realizes $\bar{U}_{k,s,g}(p+1)$. Thus we have $\bar{U}_{k,s,g}(p) \leq \bar{U}_{k,s,g}(p+1)$, as claimed.

Let $t = U_{k,s,g}$. We now want to show that $t \leq \bar{U}_{k,s,g}$. Define $p^* \leq p_k$ to be the minimum amount of job k for which $U_{k,s,g}(p^*) = t$. By our earlier claim, we have $\bar{U}_{k,s,g}(p^*) \leq \bar{U}_{k,s,g}$, so it is enough to show that $t \leq \bar{U}_{k,s,g}(p^*)$. In other words, we can simply assume from now on that $p_k = p^*$.

If $p^* = 0$ then $t = U_{k-1,s,g} = \bar{U}_{k-1,s,g} \leq \bar{U}_{k,s,g}$, where the inequality follows from the algorithm, by using $l = s$ and $h = 0$ in (8).

So assume now $p^* > 0$. This, of course, implies that $r_s < r_k < U_{k-1,s,g}$, so the algorithm will apply (8).

Let S be a (k, s)-schedule that realizes $U_{k,s,g}$, that is, S schedules $p_k = p^*$ units of k, has at most g gaps and completion time t. By the minimality of p^*, S can have at most one non-internal execution interval of k, and, if it has one, this interval ends at t.

For the rest of the proof we have to identify two numbers h, l and show that we can find a corresponding decomposition of S that would allow us to apply one of the cases in (8) and induction, yielding $t \leq \bar{U}_{k,s,g}$. We choose these numbers as follows. If S does not have an internal execution interval of k, then we choose $h = 0$ and $l = s$. Otherwise, let $[u, v)$ be the last internal execution interval of k of S. We let $l < k$ to be the job released and scheduled at v (this job l exists by the definition of internal execution intervals and the earliest-deadline policy), and we let h be the number of gaps of S in the segment of S in $[r_s, v)$.

Let q be the number of units of k scheduled by S in $[r_s, v)$. The segment of S in $[r_s, v)$ is a (k, s, q)-schedule with h gaps with respect to $[r_s, v)$, thus $q \geq P_{k,s,l,h}$. In fact, we claim that $q = P_{k,s,l,h}$. For suppose, towards contradiction, that $q > P_{k,s,l,h}$. Let Q be the schedule that realizes $P_{k,s,l,h}$. Then we could replace the segment of S in $[r_s, v)$ by Q, reducing the number of units of k in S, without changing the number of gaps and the completion time of S – a contradiction with the minimality of p^*. Therefore S must schedule exactly $P_{k,s,l,h}$ units of k in $[r_s, v)$, as claimed.

We now examine three cases.

Case 1: $t = d_k$ and k is the only job in the last block. By the minimality of p^*, and the previous paragraph, $p^* = q + 1$ and the last block is $[d_k - 1, d_k)$. Let $[t', d_k - 1)$ be the last gap in S. Then $prevr(d_k) < t'$. Since the segment of S in $[r_l, t')$ is a $(k-1, l)$-schedule with at most $g - h - 1$ gaps, we also have $t' \leq U_{k-1,l,g-h-1}$, so $prevr(d_k) < U_{k-1,l,g-h-1}$. Obviously, $P_{k,s,l,h} = q < p^*$. If $d_k - U_{k-1,l,g-h-1} > p^* - q$, option one in (8) will apply. Otherwise, $d_k - U_{k-1,l,g-h-1} \leq p^* - q$, in which case option two will apply, because $U_{k-1,l,g-h} \geq U_{k-1,l,g-h-1}$. (In this particular case, we in fact we would have equality, since $d_k - U_{k-1,l,g-h-1} = p^* - q \leq 1$ implies $U_{k-1,l,g-h-1} = d_{k-1} = d_k - 1$.) In both of these cases we obtain $\bar{U}_{k,s,g} = d_k = t$.

Case 2: $t = d_k$ and k is not the only job in the last block. Let $[z, d_k)$ be the last execution interval of k in S. We have $z > prevr_{k-1}(d_k)$. Since the segment of S in $[r_l, z)$ is a $(k-1, l)$-schedule with completion time z and at most $g - h$ gaps, we also have $z \leq U_{k-1,l,g-h}$. We can thus conclude that $d_k - U_{k-1,l,g-h} \leq p^*$ and $U_{k-1,l,g-h} > prevr_{k-1}(d_k)$. Therefore the second option in (8) applies, yielding $\bar{U}_{k,s,g} = d_k = t$.

Case 3: $t \neq d_k$. As in the previous case, let $[z, t)$ be the execution interval of k at the end of S. (It is possible here that $z = t$.) In this case, the last block contains jobs other thank k. Thus the segment of S in $[r_l, z)$ is a $(k-1, l)$-schedule with at most $g - h$ gaps, so $z \leq U_{k-1,l,g-h}$.

We claim that, in fact, we have $z = U_{k-1,l,g-h}$. Indeed, towards contradiction, suppose that

$z < z' = U_{k-1,l,g-h}$. Note that $z' \leq t$. Let Q be a $(k-1,l)$ schedule with at most $g-h$ gaps that realizes $U_{k-1,l,g-h}$. We can modify S as follows: replace the segment $[r_l, z')$ of S by Q and append to it a segment of $t - z' < p^* - q$ units of k, obtaining a $(k, s, p^* - t + z')$-schedule with at most g gaps and completion time t, contradicting the minimality of p^*.

We now have $d_k - U_{k-1,l,g-h} = d_k - z > t - z \geq p^* - q$ and $U_{k-1,l,g-h} = z > prevr_{k-1}(t)$, for $t = U_{k-1,l,g-h} + p^* - q$. Thus the third option in (8) will apply, and we obtain $\bar{U}_{k,s,g} \geq U_{k-1,l,g-h} + p^* - q = t$.

We have now proved that in all cases we obtain $t \leq \bar{U}_{k,s,g}$, completing the proof of optimality of $\bar{U}_{k,s,g}$, and the lemma. □

Theorem 2 *Algorithm* ALGB *correctly computes the optimum solution for* $1|r_j; \text{pmtn}; L = 1|E$*, and it can be implemented in time* $O(n^5)$*.*

Proof: The correctness follows from Lemma 11. The running time analysis is similar to the analysis of Algorithm ALGA. The table $\bar{U}_{k,s,g}$ is computed in time $O(n^5)$ since there are $O(n^3)$ variables and each requires minimization over $O(n^2)$ values. The table $\bar{P}_{k,s,l,g}$ has size $O(n^4)$. For each entry $\bar{P}_{k,s,l,g}$, the job j in the recurrence is uniquely determined by h (if it exists at all), so the minimization requires time $O(n)$. Thus the total running time is $O(n^5)$. □

5 Minimizing the Energy

We now show how to solve the general problem of minimizing the energy for an arbitrary given value L. This new algorithm consists of computing the table $U_{k,s,g}$ (using either Algorithm ALGA or ALGB) and an $O(n^2 \log n)$-time post-processing. Thus we can solve the problem for unit jobs in time $O(n^4)$ and for arbitrary-length jobs in time $O(n^5)$.

Recall that for this general cost model, the cost (energy) is defined as the sum, over all gaps, of the minimum between L and the gap length. Call a gap *small* if its length is at most L and *large* otherwise. The idea of the algorithm is this: We show first that there is an optimal schedule where the short gaps divide the instance into disjoint sub-instances (in which all gaps are large). For those sub-instances, the cost is simply the number of gaps times L. To compute the overall cost, we add to this quantity the total size of short gaps.

Given two schedules S, S' of the input instance, we say that S *dominates* S' if there is a time point t such that the supports of S and S' in the interval $(-\infty, t)$ are identical and S schedules a job at time t while S' is idle. This relation defines a total order on all schedules. The correctness of the algorithm relies on the following separation lemma.

Lemma 12 *There is an optimal schedule S with the following property: For any small gap $[u,t)$ of S and job j, if $C_j(S) \geq t$ then $r_j \geq t$.*

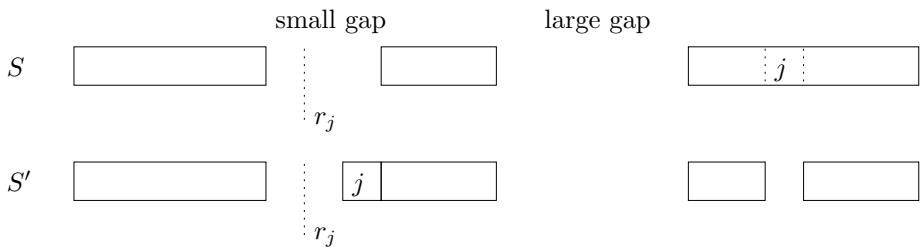

Figure 9: Idea of the proof of Lemma 12. Schedule S' dominates S.

Proof: Among all optimal schedules, choose S to be one not dominated by another optimal schedule, and let $[u, t)$ be a small gap in S (see Figure 9). If there is a job j with $r_j < t$ such that a unit of j is scheduled at some time $t' \geq t$, then we can move this execution unit to the time unit $t - 1$. This will not increase the overall cost, since the cost of the small gap decreases by one, and the idle time unit created at t' increases the cost at most by 1. The resulting schedule, however, dominates S – contradiction. □

For any job s, define an *s-schedule* to be a (partial) schedule that schedules all jobs j with $r_j \geq r_s$. We use notation E_s to represent the minimum cost (energy) of an s-schedule, including the cost of the possible gap between r_s and its first block.

Lemma 13 (partitioning) *There exists an optimal s-schedule S with the following property: Either S does not have any small gap, or if $[u, t)$ is the first small gap in S and h the number of gaps in $[r_s, u)$, then $u = U_{n,s,h}$.*

Proof: Let S be an optimal schedule. If S does not have any small gaps, we are done. Otherwise, let $[u, t)$ be the first small gap in S and let $\mathcal{J}$ be the set of jobs released in $[r_s, u)$. By Lemma 12, we can assume that all jobs from $\mathcal{J}$ are completed in S no later than at time u. This means that the segment of S in $[r_s, u)$ is an (n, s)-schedule, and thus $u \leq U_{n,s,h}$.

Towards a proof by contradiction, assume that this inequality is strict, that is $u < U_{n,s,h}$. We now use Lemma 8 (the compression lemma). First we show that the assumptions of this lemma are satisfied. By Lemma 12, no job is released in $[u, t)$, and every job j released before u is completed in S not later than at u, so $C^{\text{ED}}_{j,n} \leq u$. Let Q be the (n, s)-schedule with at most h gaps and completion time $U_{n,s,h}$. Now, applying Lemma 8 with $k' = n$, we obtain that there is an (n, s)-schedule R, scheduling all jobs from $\mathcal{J}$, with completion time $v = C_{\max}(R) \leq t$ and at most h gaps. Moreover, if $v \leq u$ then R has in fact at most $h - 1$ gaps.

We replace the segment of S in $[r_s, t)$ by R, obtaining an s-schedule S'. To complete the proof, it is sufficient to show that the cost of S' is strictly smaller than that of S, as this will contradict the optimality of S. Schedules S and S' are identical in $[t, \infty)$. The cost of the gaps of S in $[r_s, t)$ is $Lh + t - u$. If $v > u$, then the gaps in S' in $[r_s, t)$ cost at most $Lh + t - v$, and if $v \leq u$, they cost at most $L(h - 1) + L$, since the gap between v and t can cost at most L. Thus in both cases the cost

of these gaps is strictly smaller than $Lh + t - u$. $\square$

Algorithm ALGC**.** The algorithm first computes the table $U_{k,s,g}$, for all $k = 0, ..., n$, $s = 1, ..., n$, and $g = 0, 1, ..., n-1$, using either Algorithm ALGA or ALGB, whichever applies. Then we use dynamic programming to compute all values E_s. These values will be stored in table $\bar{E}_s$ and computed in order of decreasing release times r_s:

$$\bar{E}_s \leftarrow \min_{0 \le g \le n-1} \begin{cases} Lg & \text{if } U_{n,s,g} > \max_j r_j \\ Lg + r_l - u + \bar{E}_l & \text{otherwise, where } u = U_{n,s,g}, r_l = \min\{r_j : r_j > u\} \end{cases} \quad (9)$$

The algorithm outputs $\bar{E}_1$ as the minimum energy of the whole instance, where r_1 is the first release time. (Recall that the job 1 is assumed to be tight, so the schedule realizing E_1 will not have a gap at the beginning.)

Note that the minimum (9) is well-defined, for if $u = U_{n,s,g} \le \max_j r_j$, then the frugality of the schedule realizing $U_{n,s,g}$ implies that we have, in fact, $u < \max_j r_j$, and therefore there is l with $r_l > u$.

We now prove the correctness of Algorithm ALGC and analyze its running time.

Lemma 14 (feasibility of ALGC**)** *For each job $s = 1, 2, ..., n$, we have $\bar{E}_s \ge E_s$.*

Proof: We need to show that for each s there is an s-schedule S_s of cost at most $\bar{E}_s$. The proof is by backward induction on r_s. In the base case, when s is the job with maximum release time, then we take S_s to be the schedule that executes s at r_s. The cost of S_s is 0, so the lemma holds.

Assume now that for any $s' > s$ we have already constructed an s'-schedule $S_{s'}$ of cost at most $\bar{E}_{s'}$. Let g be the value that realizes the minimum in (9). We distinguish two cases, depending on which option realizes the minimum.

Suppose first that $\bar{E}_s = Lg$ and $U_{n,s,g} > \max_j r_j$. Then there is a schedule of all jobs released at or after r_s with at most g gaps. Let S_s be this schedule. Since each gap's cost is at most L, the total cost of S_s is at most Lg.

The second case is when $\bar{E}_s = Lg + r_l - u + \bar{E}_l$, where $u = U_{n,s,g} \le \max_j r_j$ and $r_l = \min\{r_j : r_j > u\}$. Choose an (n, s)-schedule Q with at most g gaps and completion time u. As explained right after the algorithm, the frugality of Q implies that there is no job released at u, and thus l is well-defined.

By induction, there exists an l-schedule S_l of cost at most $\bar{E}_l$. We then define S_s as the disjoint union of Q and S_l. The cost of Q is at most Lg. If $v \ge r_l$ is the first start time of a job in S_l, write $\bar{E}_l$ as $\bar{E}_l = \min\{v - r_l, L\} + E'$. In other words, E' is the cost of the gaps in S_l excluding the gap $[r_l, v)$ (if $r_l < v$). Then the cost of S_s is at most $Lg + \min\{v - u, L\} + E' \le Lg + (r_l - u) + \min\{v - r_l, L\} + E' = Lg + r_l - u + \bar{E}_l = \bar{E}_s$. $\square$

Lemma 15 (optimality of ALGC**)** *For each job $s = 1, 2, ..., n$, we have $\bar{E}_s \le E_s$.*

Proof: For any job s, we now prove that any s-schedule S has cost at least $\bar{E}_s$. The proof is by backward induction on r_s. In the base case, when s is the job that is released last, then $U_{n,s,0} > r_s = \max_j r_j$, so we have $\bar{E}_s = 0$, and the lemma holds.

Suppose now that s is a job that is not released last and let S be an optimal s-schedule. Without loss of generality, we can assume that S satisfies Lemma 12 and Lemma 13.

If S does not have any small gaps then, denoting by g the number of gaps in S, the cost of S is exactly Lg. The existence of S implies that $U_{n,s,g} > \max_j r_j$, so $\bar{E}_s \leq Lg$, completing the argument for this case.

Otherwise, let $[u, t)$ be the first small gap in S. Denote by S' the segment of S in $[r_s, u)$ and by S'' the segment of S in $[t, C_{\max}(S))$. By Lemma 12, S'' contains only jobs j with $r_j \geq t$. In particular the job l to be scheduled at t is released at $r_l = t$. Therefore S'' is an l-schedule, and, by induction, we obtain that the cost of S'' is at least $\bar{E}_l$.

Let g be number of gaps in S'. By Lemma 13 we have $u = U_{n,s,g}$. So the cost of S is $Lg + r_l - u + \bar{E}_l \geq \bar{E}_s$, where the inequality holds because u, g and l satisfy the condition in the second option of (9). This completes the proof. □

Theorem 3 *Algorithm* AlgC *correctly computes the optimum solution for* $1|r_j|E$*, and it can be implemented in time* $O(n^5)$*. Further, in the special case* $1|r_j; p_j = 1|E$*, it can be implemented in time* $O(n^4)$*.*

Proof: The correctness of AlgC follows from Lemma 14 and Lemma 15, so it is sufficient to justify the time bound. By Theorem 1 and Theorem 2, we can compute the table $U_{k,s,g}$ in time $O(n^4)$ and $O(n^5)$ for unit jobs and arbitrary jobs, respectively. The post-processing, that is computing all values E_s, can be easily done in time $O(n^2 \log n)$, since we have n values E_s to compute, for each s we minimize over $n - 1$ values of g, and for fixed s and g we can find the index l in time $O(\log n)$ with binary search. (Finding this l can be in fact reduced to amortized time $O(1)$ if we process g in increasing order, for then the values of $U_{n,s,g}$, and thus also of l, increase monotonically as well.) □

6 Final Comments

We presented an $O(n^5)$-time algorithm for the minimum energy scheduling problem $1|r_j; \text{pmtn}|E$, and an $O(n^4)$ algorithm for $1|r_j; p_j = 1|E$.

Many open problems remain. Can the running times be improved further? In fact, fast — say, $O(n \log n)$-time — algorithms with low approximation ratios may be of interest as well.

For the multiprocessor case, we are given m parallel machines, and every job j has to be assigned to p_j time slots in $[r_j, d_j)$ which may belong to different machines. At any time a job can be scheduled on at most one machine. The goal is to minimize the total energy usage over all machines. In [4] an $O(n^7 m^5)$-time algorithm was given for this problem, for the special case when $L = 1$ and the jobs

have unit length. It would be interesting to extend the results of this paper to the multiprocessor case, improving the running time and solving the general case for arbitrary L.

Another generalization is to allow multiple power-down states [9, 8, 10]. Can this problem be solved in polynomial-time? In fact, the SS-PD problem discussed by Irani and Pruhs in their survey [9] is even more general as it involves speed scaling in addition to multiple power states, and its status remains open as well.

References

[1] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *Proc. 45th Symp. Foundations of Computer Science (FOCS'04)*, pages 530–539, 2004.

[2] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proc. 17th Annual ACM-SIAM symposium on Discrete Algorithms (SODA'06)*, pages 364–367, 2006.

[3] P. Chretienne. On the no-wait single-machine scheduling problem. In *Proc. 7th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2005.

[4] Erik D. Demaine, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. In Phillip B. Gibbons and Christian Scheideler, editors, *SPAA*, pages 46–54. ACM, 2007.

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Co., 1979.

[6] S. Irani, R. Gupta, and S. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *Proc. Conf. on Design, Automation and Test in Europe (DATE'02)*, page 117, 2002.

[7] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 37–46, 2003.

[8] S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *Trans. on Embedded Computing Sys.*, 2(3):325–346, 2003.

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

[10] Minming Li and F. Frances Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM J. Comput.*, 35(3):658–671, 2005.