Polynomial Time Algorithms for Minimum Energy Scheduling

Philippe Baptiste[1], Marek Chrobak[2], and Christoph Dürr[1]

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

[2] Department of Computer Science, University of California, Riverside, CA 92521, USA. Supported by NSF grants OISE-0340752 and CCR-0208856.

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 the energy at a rate of R units per 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 extra energy expenditure. The intuition is that we can reduce energy consumption if we schedule the work to performed so that we reduce the weighted sum of two quantities: the total number of busy periods and the total length of "short" idle periods, when the system is left on.

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. The processor requires energy L to be woken up and, when it is on, it uses the energy at a rate of R units per unit of time. The objective is to compute a feasible schedule that minimizes the overall energy consumption. 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 [8], 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.

Our results. We solve the open problem posed by Irani and Pruhs [8], 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 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 increasing their running times.

We remark that our algorithms 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), by reduction from 3-Partition [4, 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 [8], for example). This problem is a generalization of $1|r_j|E$ and its status remains open. A polynomial-time 2-approximation algorithm for this problem (with two power states) appeared in [6].

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 [5, 7, 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 [6] 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, $1|r_j|E$, 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 [8] 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 R units 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. Without loss of generality, we assume that $R = 1$.

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). 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*.

Since the energy used on the support of all schedules 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 more than L, for otherwise it would be cheaper to keep the processor on during the gap.

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.

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 assume that the job indexed 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 we can always add such an extra job, released $L + 1$ time slots before r_1. This increases each schedule's energy 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 $u < v$,

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

We can also restrict our attention to schedules S that satisfy the following *earliest-deadline property*: 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, the jobs in the support are scheduled according to the earliest deadline policy. Using the standard exchange argument, any schedule can be converted into one that satisfies the earliest-deadline property and has the same support.

(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 $j \le k$ with $r_s \le r_j < C_{\max}(S)$ (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 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. 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.)

The following "compression lemma" will be useful in some proofs.

Lemma 1. *Let Q be a (k,s)-schedule with $C_{\max}(Q) = u$, and let R be a (k,s) schedule with $C_{\max}(R) = v > u$ and at most g gaps. Suppose that there is a time t, $u < t \le v$, such that there are no jobs $i \le k$ with $u \le r_i < t$, and that R executes some job $m < k$ with $r_m \le u$ at or after time t. Then there is a (k,s)-schedule R' with completion time t and at most g gaps.*

Proof. We can assume that R has the earliest-deadline property. We convert R into R' by gradually reducing the completion time, without increasing the number of gaps.

Call a time slot z of R *fixed* if R executes some job j at time z and either $z = r_j$ or all times $r_j, r_{j+1}, ..., z-1$ are fixed as well. Let $[w,v]$ be the last block of R and let j be the job executed at time $v-1$. If $v = t$, we are done. For $v > t$ we show that we can reduce $C_{\max}(R)$ while preserving the assumptions of the lemma.

Suppose first that the slot $v-1$ is not fixed. In this case, execute the following operation Shift: for each non-fixed slot in $[w,v]$ move the job unit in this slot to the previous non-fixed slot in R. Shift reduces $C_{\max}(R)$ by 1 without increasing the number of gaps. We still need to justify that R is a feasible (k,s)-schedule. To this end, it is sufficient only to show that no job will be scheduled before its release time. Indeed, if a job i is executed at a non-fixed time z, where $w \le z < v$, then, by definition, $z > r_i$ and there is a non-fixed slot in $[r_i, z-1]$, and therefore after Shift z will be schedule at or after r_i.

The other case is when the slot $v-1$ is fixed. In this case, we claim that there is a job l such that $w \le r_l < v$ and each job i executed in $[r_l, v]$ satisfies $r_i \ge r_l$. This l can be found as follows. If $v-1 = r_j$, let $l = j$. Otherwise, from all jobs executed in $[r_j, v-1]$ pick the job j' with minimum $r_{j'}$. Suppose that j' executes at v', $r_j \le v' \le v-1$. Since, by definition, the slot v' is fixed, we can apply this argument recursively, eventually obtaining the desired job l. We then perform the following operaiton Truncate: replace R by the segment of R in $[r_s, r_l]$. This decreases $C_{\max}(R)$ to r_l, and the new R is a feasible (k,s)-schedule, by the choice of l.

We repeat the process described above as long as $v > t$. Since the schedule at each step is a (k,s)-schedule, we end up with a (k,s)-schedule R'. Let $C_{\max}(R') = t' \le t$. It is thus sufficient to prove that $t' = t$. Indeed, consider the last step, when $C_{\max}(R)$ decreases to t'. Operation Truncate reduces $C_{\max}(R)$ to a completion time of a job released after t, so it cannot reduce it to t'. Therefore the last operation applied must have been Shift that reduces $C_{\max}(R)$ by 1. Consequently, $t' = t$, as claimed.

The $U_{k,s,g}$ function. For any $k = 0, ..., n$, $s = 1, ..., n$, and $g = 0, ..., n$, define $U_{k,s,g}$ as the maximum completion time of a (k,s)-schedule with at most g gaps. Our algorithms will compute the function $U_{k,s,g}$ and use it to determine a minimum energy schedule.

Clearly, $U_{k,s,g} \le d_k$ and, for any fixed s and g, the function $k \mapsto U_{k,s,g}$ is increasing (not necessarily strictly). For all k and s, the function $g \mapsto U_{k,s,g}$ increases as well. We claim that in fact it increases strictly as long as $U_{k,s,g} < d_k$. Indeed, suppose that $U_{k,s,g} = u < d_k$ and that $U_{k,s,g}$ is realized by a (k,s)-schedule S with at most g gaps. We show that we can extend S to a schedule S' with $g+1$ gaps and $C_{\max}(S') > C_{\max}(S)$. If there is a job $j \le k$ with $r_j \ge u$, take j to be such a job with minimum r_j. We must have $r_j > u$, since otherwise we could add j to S scheduling it at u without increasing the number of gaps, and thus contradicting the maximality of $C_{\max}(S)$. We thus obtain S' by scheduling j at r_j. The second case is when $r_j \le u$ for all jobs $j \le k$. In particular, $r_k < u$. We obtain S' by rescheduling k at u. (This creates an additional gap at the time slot where k was scheduled, for otherwise we would get a contradiction with the maximality of $C_{\max}(S)$.)

An outline of the algorithms. Our algorithms are based on dynamic programming, and they can be thought of as consisting of two stages. First, we compute the table $U_{k,s,g}$, using dynamic programming. From this table we can determine the minimum number of gaps in the (complete) schedule (it 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 assumed all release times to be different and all deadlines to be different, which implies that there is always a feasible schedule (providing that $d_j > r_j$ for all j).

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 $C_{\max}(S) = u$ is maximized. Suppose that in S job k is scheduled at some time $t < u-1$. We show that then, 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, and the total number of gaps in these two schedules equals g. This naturally leads to a recurrence relation for $U_{k,s,g}$.

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

To explain how to compute all values $U_{k,s,g}$, we give the recurrence relation. For the base case $k = 0$ we let $U_{0,s,g} \leftarrow r_s$ for all s and g. For $k \ge 1$, $U_{k,s,g}$ is defined recursively as follows:

$$U_{k,s,g} \leftarrow \max_{l<k, h\le g} \begin{cases} U_{k-1,s,g} \\ U_{k-1,s,g}+1 & \text{if } r_s \le r_k \le U_{k-1,s,g} \;\; \& \; \forall j<k \; r_j \ne U_{k-1,s,g} \\ d_k & \text{if } g>0 \;\; \& \; \forall j<k \; r_j < U_{k-1,s,g-1} \\ U_{k-1,l,g-h} & \text{if } r_k < r_l = U_{k-1,s,h}+1 \end{cases} \tag{2}$$

Note that only the last choice in the maximum depends on h and l. Also, as a careful reader might have noticed, the condition "$\forall j < k \; r_j \neq U_{k-1,s,g}$" in the second option is not necessary (the optimal solution will satisfy it automatically), but we include it to simplify the correctness proof.

In the remainder of this section we justify the correctness of the algorithm and analyze its running time. The first two lemmas establish the feasibility and the optimality of the values $U_{k,s,g}$ computed by Algorithm AlgA.

Lemma 2. *For any choice of indices k, s, g, there is a (k, s)-schedule $S_{k,s,g}$ with $C_{\max}(S_{k,s,g}) = U_{k,s,g}$ and at most g gaps.*

Proof. The proof is by induction on k. For $k = 0$, we take $S_{0,s,g}$ to be the empty (k, s)-schedule, which is trivially feasible and (by our convention) has completion time $r_s = U_{0,s,g}$.

Now fix some $k \geq 1$ and assume that the lemma holds for $k - 1$ and any s' and g', that is, for any s' and g' we have a schedule $S_{k-1,s',g'}$ with completion time $U_{k-1,s',g'}$. The construction of $S_{k,s,g}$ depends on which expression realizes the maximum (2).

If $U_{k,s,g} = U_{k-1,s,g}$, 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 $r_k > U_{k-1,s,g}$. Therefore, directly from the inductive assumption, we get that $S_{k,s,g}$ is a (k, s)-schedule with completion time $U_{k,s,g}$.

If $U_{k,s,g} = U_{k-1,s,g} + 1$, $r_s \leq r_k \leq U_{k-1,s,g}$, and there is no job $j < k$ with $r_j = U_{k-1,s,g}$, let $S_{k,s,g}$ be the schedule obtained from $S_{k-1,s,g}$ by adding to it job k scheduled at time $u = U_{k-1,s,g}$. (Note that we must have $u < d_k$.) Then $S_{k,s,g}$ is a (k, s)-schedule with completion time $u + 1 = U_{k,s,g}$.

Next, suppose that $U_{k,s,g} = d_k$, $g > 0$, and $\max_{j<k} r_j < U_{k-1,s,g-1}$. 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 condition $\max_{j<k} r_j < U_{k-1,s,g-1}$ implies that no jobs $j < k$ are released between $U_{k-1,s,g-1}$ and $d_k - 1$. Therefore $S_{k,s,g}$ is a (k, s)-schedule with completion time $d_k = U_{k,s,g}$ and it has at most g gaps, since adding k can only add one gap to $S_{k-1,s,g-1}$.

Finally, suppose that $U_{k,s,g} = U_{k-1,l,g-h}$, for some $1 \leq l < k$, $0 \leq h \leq g$, that satisfy $r_k < r_l = U_{k-1,s,h} + 1$. 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 $U_{k-1,l,g-h} - 1$ using $S_{k-1,l,g-h}$, and scheduling job k at $r_l - 1$. By induction, $S_{k,s,g}$ is a (k, s)-schedule with completion time $U_{k,s,g}$ and at most g gaps.

Lemma 3. *For any choice of indices k, s, g, if Q is a (k, s)-schedule with at most g gaps then $C_{\max}(Q) \leq U_{k,s,g}$.*

Proof. The proof is by induction on k. For $k = 0$, any $(0, s)$-schedule is empty and thus has completion time r_s. For a given $k \geq 1$ assume that the lemma holds for $k - 1$ and any s' and g', that is the values of $U_{k-1,s',g'}$ are indeed optimal. Let Q be a (k, s)-schedule with at most g gaps and maximum completion time u. Withouth loss of generality, we can assume that Q has the earliest-deadline property. The maximality of u implies that no job $j \leq k$ is released at time u, for otherwise we could add j to Q by scheduling it at u and thus increasing the completion time. (This property will be useful in the proof below.) We prove that $u \leq U_{k,s,g}$ by analyzing several cases.

<u>Case 1</u>: 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 $u \leq U_{k-1,s,g} \leq U_{k,s,g}$. In all the remaining cases, we assume that Q schedules k. Obviously, this implies that $r_s \leq r_k < u$.

<u>Case 2</u>: 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 $u' \leq U_{k-1,s,g}$, by induction. If $u' < U_{k-1,s,g}$ then, trivially, $u \leq U_{k-1,s,g} \leq U_{k,s,g}$. Otherwise, assume $u' = U_{k-1,s,g}$. Then, by the earliest deadline property, there is no job $j < k$ with $r_k = u'$. Thus the second condition in the maximum (2) is satisfied, so we have $u = u' + 1 = U_{k-1,s,g} + 1 \leq U_{k,s,g}$.

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 (2) is satisfied, so we have $u = r_s + 1 = U_{s-1,s,g} + 1 = U_{s,s,g}$. Therefore we can assume now that $u > r_s + 1$, which, together with the case condition, implies that $g > 0$.

If $u < d_k$, we can modify Q by rescheduling k at time u, obtaining a $(k, s, u{+}1)$ schedule (by the assumption about Q, no job $j < k$ is released at u) with at most g gaps – contradicting the maximality of u.

By the above paragraph, we can assume that $u = d_k$. Let u' be the smallest time $u' \geq r_s$ such that Q is idle in $[u', d_k - 1]$. Then $\max_{j<k} r_j < u'$ 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 $u' \leq U_{k-1,s,g-1}$. Thus the third option in (2) applies and we get $u = d_k = U_{k,s,g}$.

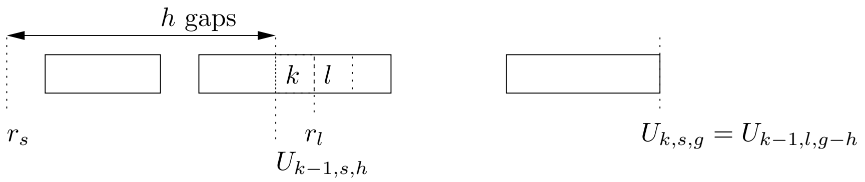


Fig. 1. Case 4 in the proof of Lemma 3.

Case 4: Q schedules k and k is not the last job. Suppose that k is scheduled at time t. Note that Q is not idle at times $t - 1$ and $t + 1$, since otherwise we would have $u < d_k$ and we could reschedule k at u, obtaining a (k, s)-schedule with at most g gaps and completion time $u + 1$, which contradicts the maximality of u. 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 $r_l = t + 1$.

Let Q_1 be the segment of Q in the interval $[r_s, t]$. Clearly, Q_1 is a $(k-1, s)$-schedule with completion time t. Denote by h the number of gaps in Q_1. We claim that Q_1 is in fact optimal, that is:

Claim 1: $t = U_{k-1,s,g}$.

Suppose for now that Claim 1 is true (see the proof below). Then the conditions of the last option in (2) are met: $l < k$, $h \leq g$, and $r_k < r_l = U_{k-1,s,h} + 1$. Let Q_2 be the segment of Q in $[r_l, u]$. Then Q_2 is a $(k - 1, l)$-schedule with completion time u and at most $g - h$ gaps, so by induction we get $u \leq U_{k-1,l,g-h}$, completing the argument for Case 4.

To complete the proof it only remains now to prove Claim 1. Denote $v = U_{k-1,s,g}$. By induction, v is the maximum completion time of a $(k - 1, s)$-schedule with at most g gaps. Clearly, as Q_1 is a $(k - 1, s)$-schedule with g gaps, we have $v \geq t$, and thus it suffices to show that $v \leq t$. Towards contradiction, suppose that $v > t$ and let R be a $(k - 1, s)$-schedule with completion time v and at most h gaps. We consider two cases.

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

Case (b): R schedules some job $j < k$ with $r_s \leq r_j \leq t$ at or after t. In this case, Lemma 1 implies that there is a $(k-1, s)$-schedule R' with at most h gaps and completion time $t+1$. Replace the segment $[r_s, t+1]$ of Q by the same segment of R' and reschedule k at u. The resulting schedule Q' is a (k, s)-schedule and, since Q executes job l at time $t+1 = r_l$, Q' has at most g gaps. We thus again obtain a contradiction with the choice of Q, because $C_{\max}(Q') = u+1 > C_{\max}(Q)$.

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 2 and Lemma 3, so it is sufficient to give the running time analysis. There are $O(n^3)$ values $U_{k,s,g}$ to be computed. For fixed k, s, g, the first two choices in the maximum (2) can be computed in time $O(1)$ and the third choice in time $O(n)$. In the last choice we maximize over pairs (l, h) that satisfy the condition $r_l = U_{k-1,s,h} + 1$, and thus we only have $O(n)$ such pairs. Since the values of $U_{k-1,s,h}$ 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 $U_{k-1,s,h}+1$, for $h = 0, 1, ..., n$. Thus each value $U_{k,s,g}$ can be computed in time $O(n)$, and the overall running time 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 $1|r_j; \text{pmtn}; L = 1|E$.

We first extend the definition of $U_{k,s,g}$ as follows. Let $0 \leq k \leq n$, $1 \leq s \leq n$, and $0 \leq g \leq n-1$. For any $p = 0, \ldots, p_k$, define $U_{k,s,g}(p)$ as the value of $U_{k,s,g}$ — the maximum completion time of a (k, s)-schedule with at most g gaps — for the modified instance where $p_k \leftarrow p$.

The following "expansion lemma" will be useful in the correctness proof. The proof of the lemma will appear in the final version.

Lemma 4. *Fix any* k*,* s*,* g *and* p*. Then*

(a) *If* $U_{k,s,g}(p) < d_k$*, then in the schedule realizing* $U_{k,s,g}(p)$ *the last block has at least one job other than* k*.*

(b) *If* $p < p_k$ *and* $U_{k,s,g}(p) < d_k$*, then* $U_{k,s,g}(p+1) > U_{k,s,g}(p)$*.*

(c) *If* $p < p_k$ *and* $U_{k,s,g}(p) = d_k$ *then* $U_{k,s,g}(p+1) = d_k$ *as well.*

We now define another table $P_{k,s,l,g}$. For any $k, s, l = 1, \ldots, n$, $g = 0, \ldots, n-1$, if $l = s$, then $P_{k,s,l,g} = 0$, otherwise

$$P_{k,s,l,g} = \min_{p} \{p + r_l - U_{k,s,g}(p)\},$$

where the minimum is taken over $0 \leq p \leq p_k$ such that l is the next job to be released after $U_{k,s,g}(p)$, that is $r_l = \min_{j<k} \{r_j : r_j > U_{k,s,g}(p)\}$. If there is no such p, we let $P_{k,s,l,g} = +\infty$. The intuition is that $P_{k,s,l,g}$ is the minimum amount of job k such that there is a (k, s)-schedule S with completion time r_l and at most

g gaps. To be more precise, we also require that (for $P_{k,s,l,g} > 0$) S executes k at time $r_l - 1$ and that has maximal completion time among all schedules over the same set of jobs than S.

Our algorithm computes both tables $U_{k,s,g}$ and $P_{k,s,l,g}$. The intuition is this. Let S be a (k,s)-schedule with g gaps and maximum possible completion time u for the given values of k, s, g. Assume that S schedules job k and $u < d_k$. Moreover assume that k is scheduled in more than one interval, and let t be the end of the second last interval of k. Then S schedules at t some job $l < k$, for otherwise we could move some portion of k to the end, contradicting maximality of u. Furthermore, $r_l = t$ by the earliest deadline policy. Now the part of S up to r_l has some number of gaps, say h. The key idea is that the amount of job k in this part is minimal among all (k,s)-schedules with completion time r_k and at most h gaps, so this amount is equal to $P_{k,s,l,h}$.

Algorithm AlgB**.** For any $k = 0, ..., n$, $s = 1, ..., n$ and $g = 0, ..., n-1$, the algorithm computes $U_{k,s,g}$, and $P_{k,s,l,g}$ for all $l = 1, ..., n$. These values are computed in order of increasing values of k, with all $P_{k,s,l,g}$ computed before all $U_{k,s,g}$, using the following recurrence relations.

Computing $U_{k,s,g}$. For the base case $k = 0$ we let $U_{0,s,g} \leftarrow r_s$ for all s and g. For $k \geq 1$, $U_{k,s,g}$ is defined recursively as follows:

$$U_{k,s,g} \leftarrow \max_{l<k, h\leq g} \begin{cases} U_{k-1,s,g} & \text{if } r_k < r_s \text{ or } r_k \geq U_{k-1,s,g} \\ d_k & \text{if } P_{k,s,l,h} < p_k,\ \max_{j<k} r_j < U_{k-1,l,g-h-1} \\ & \text{and } d_k - U_{k-1,l,g-h-1} > p_k - P_{k,s,l,h} \\ d_k & \text{if } P_{k,s,l,h} < p_k,\ \max_{j<k} r_j < U_{k-1,l,g-h} \\ & \text{and } d_k - U_{k-1,l,g-h} \leq p_k - P_{k,s,l,h} \\ U_{k-1,l,g-h} + p_k - P_{k,s,l,h} & \text{if } P_{k,s,l,h} \leq p_k \text{ and} \\ & \nexists j < k : 0 \leq r_j - U_{k-1,l,g-h} < p_k - P_{k,s,l,g-h} \end{cases} \tag{3}$$

Computing $P_{k,s,l,g}$. If $r_s = r_l$, let $P_{k,s,s,g} \leftarrow 0$ for $r_k \leq r_s < d_k$ and $P_{k,s,s,g} = +\infty$ otherwise. Suppose now that $r_s < r_l$. If $r_k < r_s$ or $r_k \geq r_l$, let $P_{k,s,l,g} = +\infty$. For $r_s \leq r_k < r_l$, we compute $P_{k,s,l,g}$ recursively as follows:

$$P_{k,s,l,g} \leftarrow \min_{0\leq h\leq k, j<k} \begin{cases} r_j - U_{k-1,s,h} + P_{k,j,l,g-h} & \text{if } r_k \leq U_{k-1,s,h},\ U_{k-1,s,h} < r_j \leq r_l, \text{ and} \\ & \nexists i < k : U_{k-1,s,h} \leq r_i < r_j \end{cases} \tag{4}$$

As usual, if the conditions in the minimum are not satisfied by any h, j, then $P_{k,s,l,g}$ is assumed to be $+\infty$. The cases considered in the algorithm are illustrated in Figure 2.

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* $O(n^5)$*.*

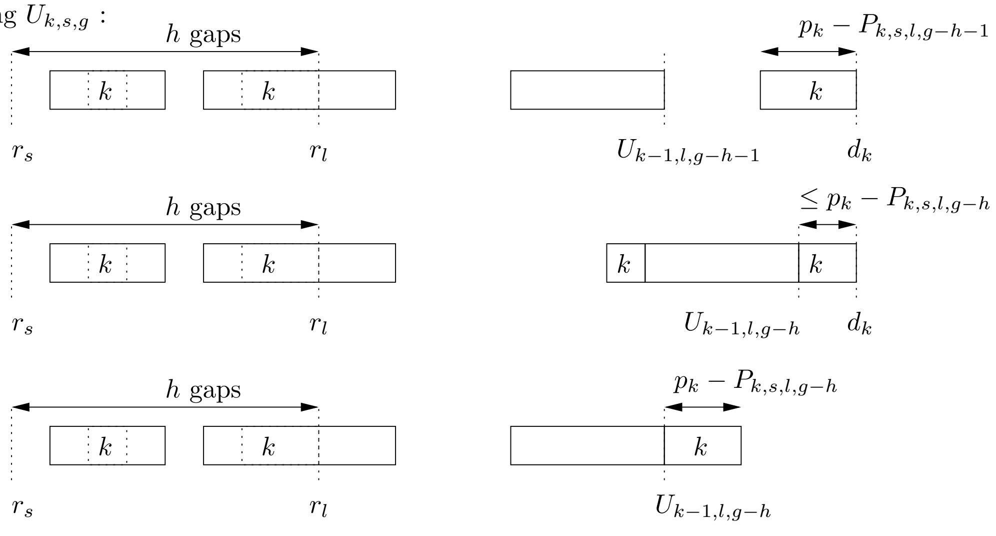
The proof of this theorem will appear in the full version of the paper.

5 Minimizing the Energy

We now show how minimize the energy for an arbitrary L. This new algorithm consists of computing the table $U_{k,s,g}$ (using either Algorithm AlgA or AlgB) and an $O(n^2)$-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. The idea of the algorithm is this: We

Computing $U_{k,s,g}$:



Computing $P_{k,s,l,g}$:

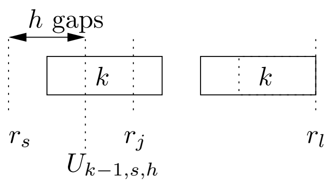


Fig. 2. Illustration of the cases in Algorithm AlgB.

show first that there is an optimal schedule where the short gaps divide the instance into disjoint sub-instances. 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 at time t S schedules a job 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 5. *There is an optimal schedule S with the following property: For any small gap $[u, v]$ of S, if a job j is scheduled at or after v then $r_j \geq v$.*

Proof. Among all optimal schedules, choose S to be the one not dominated by another optimal schedule, and let $[u, v]$ be a small gap in S. If there is a job j with $r_j < v$ and scheduled at some time unit $t \geq v$, then we can move this execution unit to the time unit $v-1$. This will not increase the overall cost, since the cost in 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.

Algorithm AlgC. The algorithm first computes the table $U_{k,s,g}$, for all $k = 0, ..., n$, $s = 1, ..., n$, and $g = 0, 1, ..., n$, using either Algorithm AlgA or AlgB, whichever applies. Then we use dynamic programming to compute all values E_s, in order of decreasing release times r_s:

$$E_s \leftarrow \min_{0 \le g \le n} \begin{cases} Lg & \text{if } U_{n,s,g} > \max_j r_j \\ Lg + r_l - u + E_l & \text{otherwise, where } u = U_{n,s,g}, r_l = \min \{r_j : r_j > u\} \end{cases} \tag{5}$$

The minimum energy of the whole instance is then E_1, 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.)

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

Lemma 6. *For each job $s = 1, 2, ..., n$ there is an s-schedule S_s of cost at most E_s.*

Proof. 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 S_s that executes s at r_s. The cost of S_s is 0. Also, since $U_{n,s,0} > r_s$ we have $E_s = 0$, so the lemma holds.

Suppose now that for any $s' > s$ we have already constructed an s'-schedule $S_{s'}$ of cost at most $E_{s'}$. Let g be the value that realizes the minimum in (5).

If $U_{n,s,g} > \max_j r_j$ then, by Theorem 2, 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.

So now we can assume that $U_{n,s,g} \le \max_j r_j$. By the maximality of $U_{n,s,g}$, this inequality is strict. As in the algorithm, let l be the first job released after $U_{n,s,g}$. Choose a schedule S' realizing $U_{n,s,g}$. By induction, there exists an l-schedule S_l of cost at most E_l. We then define S_s as the disjoint union of S' and S_l. The cost of S' is at most Lg. Denote $u = U_{n,s,g}$. If $v \ge r_l$ is the first start time of a job in S_l, write E_l as $E_l = \max\{v - r_l, L\} + E'$. In other words, E' is the cost of the gaps in E_l excluding the gap before v (if any). Then the cost of S_s is at most $Lg + \max\{v - u, L\} + E' \le Lg + (r_l - u) + \max\{v - r_l, L\} + E' = Lg + r_l - u + E_l = E_s$.

Lemma 7. *For each job $s = 1, 2, ..., n$ there is an s-schedule S_s of cost at most E_s.*

Proof. For any job s, we now prove that any s-schedule Q has cost at least 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 $E_s = 0$, so the claim is true.

Suppose now that s is not the last job and let Q be an optimal s-schedule. By Lemma 5, we can assume that Q is not dominated by any other s-schedule with optimal cost. If Q does not have any small gaps then, denoting by g the number of gaps in Q, the cost of Q is $Lg \ge E_s$.

Otherwise, let $[u, v]$ be the first small gap in Q. Denote by Q' the segment of Q in $[r_s, u]$ and by Q'' the segment of Q in $[v, C_{\max}(S)]$. By Lemma 5, Q'' contains only jobs j with $r_j \ge v$. In particular the job l to be scheduled at v is released at $r_l = v$. By induction, the cost of Q'' is at least E_l.

Let g be the number of gaps in Q' and let R be the schedule realizing $U_{n,s,g}$. By the optimality of $U_{n,s,g}$, we have $C_{\max}(R) \ge u$. If $C_{\max}(R) = u$, then, by (5), the cost of Q is $Lg + r_l - u + E_l \ge E_s$, and we are done.

The remaining case is when $C_{\max}(R) > u$. By Lemma 1, this implies that there is a (n, s)-schedule R' with at most g gaps and $C_{\max}(R') \le v$. But then we could replace Q' in Q by R', getting a schedule of cost strictly smaller than that of Q, contradicting the optimality of Q.

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 Algorithm AlgC follows from Lemma 6 and Lemma 7, 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 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 $1|r_j; \mathrm{pmtn}|E$, 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.

To our knowledge, no work has been done on the multiprocessor case. Can our results be extended to more processors? Another generalization is to allow multiple power-down states [8, 7]. Can this problem be solved in polynomial-time? In fact, the SS-PD problem discussed by Irani and Pruhs [8] 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. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H.Freeman and Co., 1979.
5. 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.
6. 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.
7. 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.
8. Sandy Irani and Kirk R. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.