Theoretical Computer Science 412 (2011) 1122–1139

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Min-energy scheduling for aligned jobs in accelerate model

Weiwei Wu [a,b,c], Minming Li [b,∗], Enhong Chen [a]

[a] School of Computer Science, University of Science and Technology of China, China
[b] Department of Computer Science, City University of Hong Kong, Hong Kong
[c] USTC-CityU Joint Research Institute, China

ARTICLE INFO

Article history:
Received 8 August 2010
Received in revised form 29 November 2010
Accepted 8 December 2010
Communicated by D.-Z. Du

Keywords:
Scheduling
Optimality
Energy efficiency
Acceleration
Aligned jobs

ABSTRACT

A dynamic voltage scaling technique provides the capability for processors to adjust the speed and control the energy consumption. We study the pessimistic accelerate model where the acceleration rate of the processor speed is at most K and jobs cannot be executed during the speed transition period. The objective is to find a min-energy (optimal) schedule that finishes every job within its deadline. The job set we study in this paper is aligned jobs where earlier released jobs have earlier deadlines. We start by investigating a special case where all jobs have a common arrival time and design an $O(n^2)$ algorithm to compute the optimal schedule based on some nice properties of the optimal schedule. Then, we study the general aligned jobs and obtain an $O(n^2)$ algorithm to compute the optimal schedule by using the algorithm for the common arrival time case as a building block. Because our algorithm relies on the computation of the optimal schedule in the ideal model ($K = \infty$), in order to achieve $O(n^2)$ complexity, we improve the complexity of computing the optimal schedule in the ideal model for aligned jobs from the currently best known $O(n^2 \log n)$ to $O(n^2)$.

1. Introduction

Energy-efficiency has become the first-class design constraint besides the traditional time and space requirements. Portable devices (like laptops and PDAs) equipped with capacity limited batteries are popular in our daily life. Two facts make the energy problem more important. First, the battery capacity is increasing with a rate less than that of the power consumption of the processors. Second, the accumulated heat due to energy consumption will reach a thermal wall and challenge the designers of electronic devices. It is found that, in the CMOS processors, the energy consumption can be saved by executing with a lower speed. Approximately, the speed is a cubic root of the power, which is known as cube-root-rule. The dynamic voltage scaling (DVS) technique is widely adopted by modern processor manufacturers, e.g., Intel, AMD, and IBM. It allows the processor to dynamically adjust its voltage/frequency to control the power consumption. The first theoretical study was initiated decades ago by Yao et al. [20], where they make the standard generalization, a speed to power function $P(s) = s^\alpha$ ($\alpha \geq 1$). Usually, α is 2 or 3 according to the cube-root-rule of the processors. From then on, lots of studies have been triggered in this field. It is usually formulated as a dual objective problem. That is, while conserving the energy, it also needs to satisfy some QoS metric. When all jobs are required to be completed before deadlines, the metric is called *deadline feasibility*. There are also works trying to simultaneously minimize the response time of the jobs, namely, *flow*. A schedule consists of the *speed scaling policy* to determine what speed to run at time t and the *job selection policy* to decide which job to run at that time.

∗ Corresponding author. Tel.: +852 27889538; fax: +852 27888614.
E-mail addresses: wweiwei2@cityu.edu.hk (W. Wu), minmli@cs.cityu.edu.hk (M. Li), cheneh@ustc.edu.cn (E. Chen).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.12.013

If the processor can run at arbitrary speeds, then based on how fast the voltage can be changed, there are two different models.

Ideal model: It is assumed that the voltage/speed of the processor can be changed to any other value without any extra/physical cost or delay. This model provides an ideal fundamental benchmark and has been widely studied.

Accelerate model: It is assumed that the voltage/speed change has some delay. In practice, the processor's acceleration capacity is limited. For example, in the low power ARM microprocessor system (lpARM) [6], the clock frequency transition takes approximately 25 μs (1350 cycles) from 10 to 100 MHz. Equation (EQ1) in [6] pointed out that the delay for transition from one voltage to another voltage is proportional to the difference of these two voltages. Within this scope, there are two variations. In the *optimistic model*, the processor can execute jobs during the speed transition time, while in the *pessimistic model*, the execution of jobs in the transition time is not allowed [21]. In this paper, we consider processors with a DC-DC converter having an efficiency of 1. In other words, we assume that the transition does not consume energy according to Eq. (EQ2) in [6].

1.1. Related works

In recent years, there have been many works on the impact of DVS technology.

For the ideal model, Yao et al. first studied the energy-efficient job scheduling to achieve deadline feasibility in their seminal paper [20]. They proposed an $O(n^3)$ time algorithm YDS to compute the optimal off-line schedule. Later on, the running time is improved to $O(n^2 \log n)$ by Li et al. [17]. Another metric, the response time/flow, was examined by Pruhs et al. in [18] with bounded energy consumption. It is first formulated as a linear single objective (energy + flow) optimization problem by Albers et al. in [1]. This was then specifically studied in [5,14,7,2,3] under different assumptions. Chan et al. [8] investigated the model where the maximum speed is bounded. They proposed an online algorithm which is $O(1)$-competitive in both energy consumption and throughput. More works on the speed bounded model can be found in [4,9,15]. Ishihara and Yasuura [12] initiated the research on discrete DVS problem where a CPU can only run at a set of given speeds. They solved the case when the processor is only allowed to run at two different speeds. Kwon and Kim [13] extended it to the general discrete DVS model where the processor is allowed to run at speeds chosen from a finite speed set. They gave an algorithm for this problem based on the MES algorithm in [20]. Later, [16] improved the computation time to $O(dn \log n)$ where d is the number of supported voltages. A survey on algorithmic problems in power management for DVS by Irani and Pruhs can be found in [11].

For the accelerate model, there are little theoretical studies to the best of our knowledge, except that the single task problem was studied by Hong et al. in [10] and Yuan et al. in [21]. In [10], they showed that the speed function which minimizes the energy is of some restricted shapes even when considering a single task. They also gave some empirical studies based on several real-life applications. In [21], the authors studied both the optimistic model and pessimistic model, but still for the single task problem. They showed that to reduce the energy, the speed function should accelerate as fast as possible.

1.2. Main contributions

This paper is the full version of our previous conference paper [19]. In this paper, we study the pessimistic accelerate model to minimize the energy consumption. The QoS metric is deadline feasibility. The input is an aligned job set $\mathcal{J}$ with n jobs, where jobs with earlier arrival times have earlier deadlines. The processor can execute a job with arbitrary speed but the absolute acceleration rate is at most K, and the processor has no capability to execute jobs during the transition of voltage. The objective is to find a min-energy schedule that finishes all jobs before their deadlines.

We first consider a special case of aligned jobs where all the jobs arrive at time 0. We call this kind of job set *common arrival time instance*. We prove that the optimal schedule should decelerate as fast as possible and the speed curve is non-increasing. Combining with other properties we observe, we construct an $O(n^2)$ time algorithm to compute the optimal schedule.

Then we turn to the general aligned jobs to study the optimal schedule OPT_K. The algorithm for the common arrival time instance is adopted as an elementary procedure to compute OPT_K. Most of the properties for the common arrival time instance can be extended to general aligned jobs. By comparing OPT_K with the optimal schedule OPT_∞ in the ideal model, we first prove that the speed curves of OPT_K and OPT_∞ match during some "peak"s. Then we show that the speed curve of OPT_K between adjacent "peak"s can be computed directly. The whole computation takes $O(n^2)$ time since we improve the computation of OPT_∞ (optimal solution of the ideal model) for aligned jobs from the currently best known $O(n^2 \log n)$ to $O(n^2)$. Our work makes a further step in the theoretical study of the accelerate model and may shed some light on solving the problem for the general job set.

The organization of this paper is as follows. We review the ideal model and the pessimistic accelerate model in Section 2. In Section 3, we study the pessimistic accelerate model and focus on a special but significant case where all jobs are released at the beginning. We then turn to the general aligned jobs that have arbitrary arrival time in Section 4. Finally we conclude the paper in Section 5.

2. Model and notation

In this section, we review the ideal model proposed in [20] and the pessimistic accelerate model.

The input job instance we consider in this paper is an aligned job set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ where each job J_i has an arrival time $r(J_i)$, a deadline $d(J_i)$ (abbreviated as r_i and d_i respectively), and the amount of workload $C(J_i)$. The arrival times and the deadlines follow the same order, i.e., $r_1 \le r_2 \le \cdots \le r_n$ and $d_1 \le d_2 \le \cdots \le d_n$.

In the ideal model, the processor can change its speed to any value instantaneously without any delay. The power function is assumed to be $P(s) = s^\alpha (\alpha \ge 1)$. A schedule S needs to determine what speed and which job to execute at time t. We use $s(t, S)$ to denote the speed took by schedule S at time t and write it as $s(t)$ for short if the context is clear. We use $job(t)$ to represent the index of the job being executed at time t. Jobs are preemptive. The processor has the capability to resume the formerly suspended jobs. We take the deadline feasibility as the QoS metric, i.e., a job is available after its arrival time and need to be completed before its deadline. A feasible schedule must satisfy the timing constraint $\int_{r_i}^{d_i} s(t)\delta(i, job(t))dt = C(J_i)$, where $\delta(i, j) = 1$ if $i = j$ and $\delta(i, j) = 0$ otherwise. The energy consumption is the power integrated over time: $E(S) = \int_t P(s(t, S))dt$. The objective is to minimize the total energy consumption while satisfying the deadline feasibility.

In the pessimistic accelerate model, the processor cannot change the voltage instantaneously. The acceleration rate is at most K, i.e., $|s'(t)| \le K$. Moreover, no jobs can be executed during the transition interval with $s'(t) \ne 0$ and there is always some job being executed when $s'(t) = 0$ and $s(t) > 0$. The energy is the power integrated over the time where $s'(t) = 0$ and $s(t) > 0$, thus $E = \int_{t|s'(t)=0,s(t)>0} P(s(t, S))dt$. With such constraints, a *feasible* schedule is a schedule where all jobs are completed before deadlines and the speed function satisfies $|s'(t)| \le K$. In a feasible schedule S, we denote the maximal interval where the jobs run at the same speed as a *block*. Note that there is an *acceleration interval* (the time used for acceleration) between adjacent blocks because changing the speeds needs some time, during which no workload is executed. The *optimal* schedule is the one with the minimum energy consumption among all feasible schedules.

Let $t_s = \min_i r_i$ and $t_f = \max_i d_i$. The workload executed in interval $[a, b]$ by schedule S is denoted as $C_{[a,b]}(S)$. If a job J has $I(J) = [r(J), d(J)] \subseteq [a, b]$, we say J is *embedded* in interval $[a, b]$. For simplicity, when we say "the first" time (or interval), we mean the earliest time (or interval) on the time axis in left-to-right order. Using a similar definition as [16], we say t_u is a *tight deadline* (or *tight arrival time* respectively) in schedule S if t_u is the deadline (or arrival time respectively) of the job that is executed at $[t_u - \Delta t, t_u]$ (or $[t_u, t_u + \Delta t]$ respectively) in S where $\Delta t \to 0$. Let $w(t_1, t_2)$ denote the workload of the jobs that have an arrival time at least t_1 and have a deadline at most t_2, i.e. $w(t_1, t_2) = \sum_{I(J)\subseteq[t_1,t_2]} C(J)$. Define the *intensity* $Itt(t_1, t_2)$ of the time interval $[t_1, t_2]$ to be $w(t_1, t_2)/(t_2 - t_1)$.

3. Optimal schedules for job set with common arrival time

For the jobs that have a common arrival time, we assume w.l.o.g they are available at the beginning, namely $r_i = 0$ for $1 \le i \le n$. In the following, we will derive a series of properties of the optimal schedule which help us design a polynomial algorithm to compute the optimal schedule. The intuition behind the proofs is shown below. Comparing with the optimal schedule for the ideal model, when the speed decelerates from a faster speed to a slower speed, some time will be lost in the accelerate model. The optimal solution is composed of blocks and the speed is non-increasing (The speed between every two blocks is decreasing). To find all the blocks, we search some "tight" points where the deceleration should begin at this point of time, because at least one job's deadline would be missed if the deceleration was delayed further. There are at most n blocks to be computed which finally gives an $O(n^2)$ time algorithm.

We assume that "optimal schedule" mentioned in a lemma satisfies all the previous lemmas in the same section. We abuse the notation between closed interval $[t_1, t_2]$ and open interval (t_1, t_2) if the context is clear.

Lemma 1. *Given a feasible schedule S with speed function $s(t)$, assume that interval $(t_a, t_a + \Delta t)$ is all for acceleration purposes (i.e. $s'(t) \ne 0$ for $t \in (t_a, t_a + \Delta t)$) and $s(t_a) = s(t_a + \Delta t)$. Then the schedule $\bar{S}$ with speed function $\bar{s}(t)$ defined below, which shifts the workload $C_{[t_a+\Delta t,t_f]}(S)$ left by Δt time, is feasible and consumes the same energy as S.*

$$\bar{s}(t) = \begin{cases} s(t) & t \in [t_s, t_a] \\ s(t + \Delta t) & t \in (t_a, t_f - \Delta t] \\ 0 & t \in (t_f - \Delta t, t_f]. \end{cases}$$

Proof. First, we show that $\bar{S}$ is a feasible schedule. In S, suppose that job J_i is finished after $t_a + \Delta t$ and before deadline d_i. Since J_i is available at the beginning $r_i = 0$, and it is not accelerated in $(t_a, t_a + \Delta t)$, when we shift J_i's execution interval left by Δt time, the resulting schedule is still feasible. Similarly by shifting interval $(t_a + \Delta t, t_f)$ left by Δt, the resulting schedule $\bar{S}$ is feasible. Moreover, after the execution is shifted, there is no workload being executed after $t_f - \Delta t$, thus the energy consumed in $(t_f - \Delta t, t_f)$ is zero. Finally, since the speed profile to execute the workload $C_{[t_a+\Delta t,t_f]}(S)$ does not change, $\bar{S}$ has the same energy consumption as S. The lemma is then proved. □

Lemma 2. *In the optimal schedule, the speed function will accelerate as fast as possible, i.e., either $|s'(t)| = K$ or $|s'(t)| = 0$.*

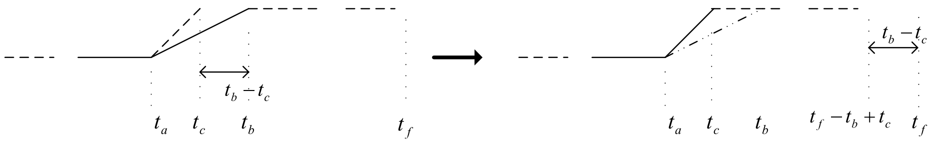

Fig. 1. The transformation when $|s'(t)| < K$.

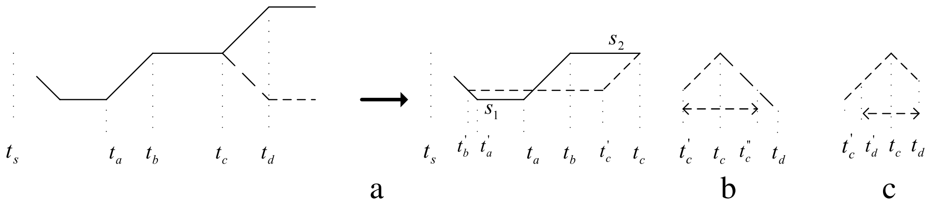

Fig. 2. Removing $s'(t) = K$.

Proof. We only need to remove the possibility of $0 < |s'(t)| < K$. As shown in Fig. 1, when $0 < s'(t) < K$, we prove that another schedule which accelerates as fast as possible will cost less or equal energy. Suppose that (t_a, t_b) is the first acceleration interval where $0 < s'(t) < K$, we can set the acceleration rate as $s'(t) = K$ in interval (t_a, t_c). Since the target speed is not changed, this obviously implies $t_c < t_b$. Then no matter what the speed function in (t_b, t_f) is, it can be shifted left by $t_b - t_c$ time. This transformation will remove the acceleration rate $0 < s'(t) < K$ in (t_a, t_b) and ensure that the resulting schedule incurs no more energy by Lemma 1. Similarly we can handle the case $-K < s'(t) < 0$. By repeatedly applying this transformation, we can ensure $|s'(t)| = K$ or $|s'(t)| = 0$ in (t_s, t_f) for the optimal schedule. □

Lemma 3. *In the optimal schedule, the speed function $s(t)$ is non-increasing.*

Proof. It suffices to remove the possibility $s'(t) = K$. If on the contrary there are some intervals with $s'(t) = K$, suppose that the first interval with $s'(t) = K$ is (t_a, t_b) as shown in Fig. 2. Assume that t_c is the nearest time after t_b where $s'(t) \neq 0$, i.e. $t_c = \arg\min_{t>t_b}(s'(t) \neq 0)$. Let the speed in block (t_b, t_c) be s_2. Note that the speed before t_a is non-increasing. We can further suppose that the speed in t_a's adjacent block (t_a', t_a) is s_1 where $s_1 < s_2$. We will show that there exists another feasible schedule $\bar{S}$ with less energy. The method is to merge $C_{[t_b,t_c]}(S)$ with $C_{[t_a',t_a]}(S)$ and previous adjacent blocks if necessary into a new block (t_b', t_c') and let $\bar{s}'(t) = K$ in (t_c', t_c). The merging process guarantees that the new block executes the same amount of workload as the participating merging blocks. The merging process goes from right to left and stops when the last unmerged block before t_a' has a higher speed than the speed of the new block. Suppose that the new speed in (t_b', t_c') is s_Δ, we first show that the energy in (t_s, t_c) is decreased by discussing two cases $s_\Delta < s_0$ and $s_\Delta \geq s_0$, where s_0 is the speed in the start time t_s. First, if $s_\Delta < s_0$, then the workload $C_{(t_b',t_c)}(S)$ is done in a single block (t_b', t_c') (a unique speed) in $\bar{S}$. Note that such a new schedule is feasible (because jobs are finished earlier in the new schedule) and the speed in (t_s, t_c') is non-increasing. Moreover, according to the convexity of $P(s)$, the new schedule consumes less energy. Second, if $s_\Delta \geq s_0$, then the workload $C_{(t_s,t_c)}(S)$ is done with a unique speed in block (t_s, t_c') in $\bar{S}$ which also leads to a better schedule.

Notice that we have postponed the first interval where $s'(t) = K$ to (t_c', t_c). Then we discuss the following two cases: $s'(t) = K$ for $t_c < t < t_d$ and $s'(t) = -K$ for $t_c < t < t_d$.

If $s'(t) = K$ for $t_c < t < t_d$, the first interval where $s'(t) = K$ in $\bar{S}$ is the acceleration interval (t_c', t_d) and we can apply the analysis again to gradually postpone the first interval $s'(t) = K$ until the time t_f. This finally results in a schedule with less energy where $s'(t) = K$ only exists in some rightmost interval $(\hat{t}, t_f)$. Note that $(\hat{t}, t_f)$ need not accelerate because there is no workload after t_f. Thus we have removed the possibility $s'(t) = K$.

If $s'(t) = -K$ for $t_c < t < t_d$, two subcases should be considered. If $t_d - t_c > t_c - t_c'$, let t_c'' be the symmetric time point of t_c' by vertical line $t = t_c$ (Fig. 2(b)). Note that $\bar{s}(t_c'') = \bar{s}(t_c')$. We can then shift the speed function $s(t)$ in (t_c'', t_f) left by $t_c'' - t_c'$ time. This removes the possibility $s'(t) = K$ in (t_b, t_d). Then we can recursively handle the first interval with $s'(t) = K$ in (t_d, t_f). If $t_d - t_c \leq t_c - t_c'$, let t_d' be the symmetric time point of t_d by vertical line $t = t_c$ (Fig. 2(c)). We then shift the speed function $s(t)$ in (t_d, t_f) left by $t_d - t_d'$ time. Then (t_c', t_d') will be the first interval where $s'(t) = K$. Hence, by applying the analysis again we can gradually postpone the first interval with $s'(t) = K$ to the rightmost interval $(\hat{t}, t_f)$. Since there is no workload after t_f, we can remove the final acceleration interval $(\hat{t}, t_f)$. This finishes the proof. □

Lemma 4. *There exists an optimal schedule where the jobs are completed in EDF (Earliest Deadline First) order.*

Proof. Suppose that J_{i+1} is the first job which violates the EDF order, which means all jobs are finished in the order $\sigma(\mathcal{J}) = (J_1, \ldots, J_i, J_{i+t}, \ldots, J_{i+1}, \ldots)$. Notice that jobs between J_i and J_{i+1} in $\sigma(\mathcal{J})$ have deadlines larger than d_{i+1}. We will show that executing jobs in the order $\sigma'(\mathcal{J}) = (\ldots, J_i, J_{i+1}, \ldots, J_{i+t}, \ldots)$, which is obtained from $\sigma(\mathcal{J})$ by swapping J_{i+t} and J_{i+1}, and using the same speed function is still a feasible schedule. Obviously, jobs $J_1, \ldots, J_i$ can be finished before deadlines. Moreover, since the speed function does not change, the completion time of job J_{i+t} in $\sigma'(\mathcal{J})$ is the same as that of job J_{i+1} in $\sigma(\mathcal{J})$. That is, J_{i+t} is finished at a time not later than d_{i+1} ($d_{i+1} \leq d_{i+t}$). Furthermore, in $\sigma'(\mathcal{J})$, jobs between J_{i+1}

and J_{i+t} are finished before deadline because they are finished before d_{i+1} in $\sigma'(\mathcal{J})$. Thus the new schedule is feasible and the energy remains the same. Then by applying a similar adjustment gradually, we can obtain an optimal schedule with the completion time in EDF order. □

Lemma 5. *In the optimal schedule S, job J_i is executed in one speed* $\min_{0\le t\le d_i, s'(t)=0} s(t)$.

Proof. This lemma is a special case of Lemma 15 and therefore we postpone the proof to Lemma 15. □

Fact 1. *Given n jobs sorted by deadlines, suppose that job i's workload is x_i and it is originally executed for T_i time with $\frac{x_i}{T_i} > \frac{x_{i+1}}{T_{i+1}}$. If in another schedule, job 1 is executed for $T_1 + \Delta$ time and jobs $2, \ldots, n$ are respectively executed for $T_i - \delta_i$ time where $\Delta > \sum_{i=2}^{n} \delta_i$ and the speeds satisfy $\frac{x_1}{T_1+\Delta} > \frac{x_i}{T_i-\delta_i}$, then we have $\sum_{i=1}^{n} \frac{x_i^\alpha}{T_i^{\alpha-1}} > \frac{x_1^\alpha}{(T_1+\Delta)^{\alpha-1}} + \sum_{i=2}^{n} \frac{x_i^\alpha}{(T_i-\delta_i)^{\alpha-1}}$ where $\alpha \ge 1$.*

Proof. We remark that α can be a non-integer. Define $\Delta_1 = \sum_{i=2}^{n} \delta_i < \Delta$. It is sufficient to prove $\sum_{i=1}^{n} \frac{x_i^\alpha}{T_i^{\alpha-1}} > \frac{x_1^\alpha}{(T_1+\Delta_1)^{\alpha-1}} + \sum_{i=2}^{n} \frac{x_i^\alpha}{(T_i-\delta_i)^{\alpha-1}}$. The workload x_1 can be divided into $n-1$ parts $x_2', x_3', \ldots, x_n'$. Let $x_i' = \frac{\delta_i}{\sum_{i=2}^{n}\delta_i} x_1$ where $2 \le i \le n$. Assume that x_i' is executed for T_i' time. Let $T_i' = \frac{\delta_i}{\sum_{i=2}^{n}\delta_i} T_1$. Note that $\frac{x_i'}{T_i'} = \frac{x_1}{T_1}$, $T_1 = \sum_{i=2}^{n} T_i'$ and $x_1 = \sum_{i=2}^{n} x_i'$. We examine the difference $\frac{x_1^\alpha}{T_1^{\alpha-1}} - \frac{x_1^\alpha}{(T_1+\Delta_1)^{\alpha-1}}$.

$$\begin{aligned}
\frac{x_1^\alpha}{T_1^{\alpha-1}} - \frac{x_1^\alpha}{(T_1+\Delta_1)^{\alpha-1}} &= \left(\frac{x_1}{T_1}\right)^\alpha \cdot T_1 - \left(\frac{x_1}{T_1+\Delta_1}\right)^\alpha \cdot (T_1+\Delta_1) \\
&= \left(\frac{x_1}{T_1}\right)^\alpha \cdot \left(\sum_{i=2}^{n} T_i'\right) - \left(\frac{x_1}{T_1+\Delta_1}\right)^\alpha \cdot \left(\sum_{i=2}^{n} T_i' + \sum_{i=2}^{n} \delta_i\right) \\
&= \sum_{i=2}^{n} \left(\frac{x_i'}{T_i'}\right)^\alpha \cdot T_i' - \sum_{i=2}^{n} \left(\frac{x_i'}{T_i'+\delta_i}\right)^\alpha \cdot (T_i'+\delta_i) \\
&= \sum_{i=2}^{n} \frac{x_i'^\alpha}{T_i'^{\alpha-1}} - \sum_{i=2}^{n} \frac{x_i'^\alpha}{(T_i'+\delta_i)^{\alpha-1}}.
\end{aligned}$$

The second equality holds because $T_1 = \sum_{i=2}^{n} T_i'$ and $\Delta_1 = \sum_{i=2}^{n} \delta_i$. The third equality holds because $\frac{x_i'}{T_i'} = \frac{x_1}{T_1}$ and $\frac{x_1}{T_1+\Delta_1} = \frac{x_1}{T_1}\frac{T_1}{T_1+\Delta_1} = \frac{x_i'}{T_i'}\frac{1}{1+\frac{\Delta_1}{T_1}} = \frac{x_i'}{T_i'+\delta_i}$.

Now it suffices to prove $\sum_{i=2}^{n} \frac{x_i'^\alpha}{T_i'^{\alpha-1}} - \sum_{i=2}^{n} \frac{x_i'^\alpha}{(T_i'+\delta_i)^{\alpha-1}} \ge \sum_{i=2}^{n} \frac{x_i^\alpha}{(T_i-\delta_i)^{\alpha-1}} - \sum_{i=2}^{n} \frac{x_i^\alpha}{T_i^{\alpha-1}}$. To show this, we will prove that $\frac{x_i'^\alpha}{T_i'^{\alpha-1}} + \frac{x_i^\alpha}{T_i^{\alpha-1}} \ge \frac{x_i^\alpha}{(T_i-\delta_i)^{\alpha-1}} + \frac{x_i'^\alpha}{(T_i'+\delta_i)^{\alpha-1}}$. Define $f(t) = \frac{x_i'^\alpha}{t^{\alpha-1}} + \frac{x_i^\alpha}{(T_i+T_i'-t)^{\alpha-1}}$. It is sufficient to show $f'(t) \le 0$ when $T_i' \le t \le T_i'+\delta_i$ which will imply $f(T_i') \ge f(T_i'+\delta_i)$.

We have $f'(t) = (1-\alpha)\frac{x_i'^\alpha}{t^\alpha} - (1-\alpha)\frac{x_i^\alpha}{(T_i+T_i'-t)^\alpha} = (1-\alpha)\cdot((\frac{x_i'}{t})^\alpha - (\frac{x_i}{T_i+T_i'-t})^\alpha)$ because $(x^\alpha)' = \alpha \cdot x^{\alpha-1}$ for all real $\alpha \ge 1$. Since $f'(t)$ increases as t increases when $\alpha \ge 1$, we only need to show $f'(T_i'+\delta_i) \le 0$. By $\frac{x_1}{T_1+\Delta} > \frac{x_i}{T_i-\delta_i}$, we have $f'(T_i'+\delta_i) = (1-\alpha)\cdot((\frac{x_i'}{T_i'+\delta_i})^\alpha - (\frac{x_i}{T_i-\delta_i})^\alpha) \le 0$ because $\frac{x_i'}{T_i'+\delta_i} = \frac{x_1}{T_1+\Delta_1} > \frac{x_i}{T_i-\delta_i}$. This finishes the proof. □

Lemma 6. *There exists an optimal schedule S, where the finishing time $\hat{t}$ of each block (where $\lim_{t\to\hat{t}^-} s'(t) = 0 \wedge \lim_{t\to\hat{t}^+} s'(t) = -K$) is a tight deadline.*

Proof. Suppose on the contrary that *block*$_p$'s completion time $\hat{t_1}$ is the first such time point which is not a tight deadline. Then for all the blocks that are before *block*$_p$, their completion times are tight deadlines. We assume that $\hat{t_0}$ is the completion time of the nearest block before *block*$_p$. If such a time point does not exist, we set $\hat{t_0} = 0$. We prove that the jobs in $(\hat{t}_0, \hat{t}_1)$ can be done with lower speed and longer length which leads to less energy. We start with the simplest case. As shown in Fig. 3(a), in the new schedule $\bar{S}$ with speed function $\bar{s}(t)$ that is drawn in a dashed line, the completion time of J_i is postponed to its deadline d_i (postponed by $d_i - \hat{t}_1$ time). Meanwhile, we ensure that the blocks after d_i will keep their completion time unchanged. To achieve this, jobs $J_{i+1}, \ldots, J_n$ will be done at a slightly higher speed compared with that in $s(t)$. We first consider the case that $\bar{S}$ is feasible, i.e. no jobs miss deadlines in $\bar{S}$. Notice that with such an assumption, the speed allocation in (d_i, t_f) is unique after J_i's completion time is postponed to its deadline (because the speed $\bar{s}(d_i)$ can be determined). We will prove that such a schedule consumes less energy by using Fact 1. Suppose that the blocks after d_i have execution time $t_2, \ldots, t_m$ in S and the executed workloads are $x_2, \ldots, x_m$ respectively. While workload x_j is executed by $t_j - \delta_j$ $(2 \le j \le m)$ time in $\bar{S}$. Then the conditions $\frac{x_j}{t_j} > \frac{x_{j+1}}{t_{j+1}}$ and $\frac{x_1}{t_1+\Delta} > \frac{x_j}{t_j-\delta_j}$ obviously hold where x_1, t_1 and Δ denote the workload $C_{[\hat{t}_0,\hat{t}_1]}(S)$,

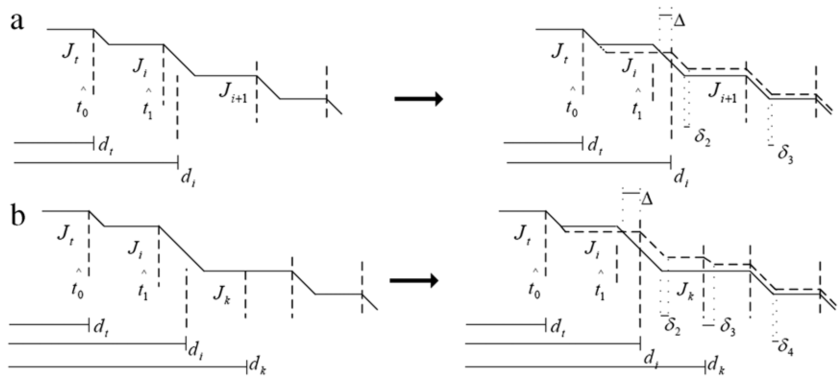

Fig. 3. Postpone procedure.

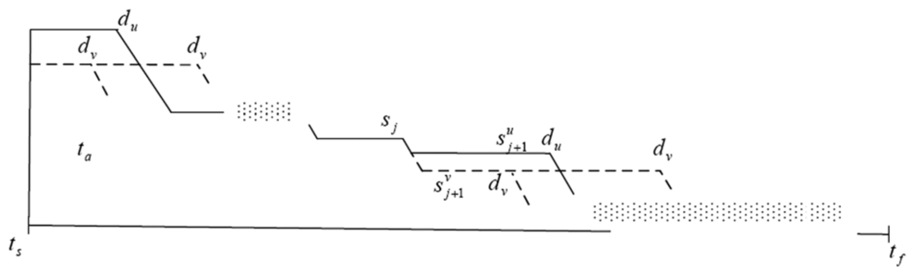

Fig. 4. An example that shows the choice of blocks.

the length of time executing x_1 in S, and the difference of the length of executing x_1 by S and $\bar{S}$, respectively. The energy used by $\bar{S}$ is less than S by $\sum_{i=1}^{n}\frac{x_i^\alpha}{t_i^{\alpha-1}}-\frac{x_1^\alpha}{(t_1+\Delta)^{\alpha-1}}-\sum_{i=2}^{n}\frac{x_i^\alpha}{(t_i-\delta_i)^{\alpha-1}}$. Thus it remains to prove that $\Delta>\sum_{j=2}^{m}\delta_j$.

To see this, consider the time $u>\hat{t}_1$, which is the first intersection between $s(t)$ and $\bar{s}(t)$. We have $s(u)=\bar{s}(u)$. Note that at the final time t_f, the speed $\bar{s}(t_f)>s(t_f)$. In the interval $[u,t_f]$, $s(t)$ uses more time for speed transition than $\bar{s}(t)$ as $\frac{s(u)-s(t_f)}{K}>\frac{\bar{s}(u)-\bar{s}(t_f)}{K}$. Thus the length for executing jobs (the length of intervals with $s'(t)=0$ in $[u,t_f]$) in $\bar{S}$ is larger than that of S. Furthermore, the difference between these two lengths is exactly $\Delta-\delta_2-\cdots-\delta_m$. Hence we have $\Delta>\sum_{j=2}^{m}\delta_j$.

Now we consider the general case, where some jobs miss deadlines if J_i is postponed to its deadline as we do above. Notice that our transformation in Fig. 3(a) also makes sense when we only postpone the execution of J_i by any small amount of time less than $d_i-\hat{t}_1$. Thus we first postpone the execution of J_i by a time less than $d_i-\hat{t}_1$ which makes a job J_k finish exactly at its deadline and guarantees all the other jobs are finished by their deadlines. If $k<i$, we fix the schedule before d_k, let $\hat{t}_0=d_k$, and keep on dealing with the speed curve in $[\hat{t}_0,d_i]$. If $k>i$, we take another transformation as shown in Fig. 3(b). Note that we assign an acceleration interval immediately after time d_k. Such a transformation also ensures a better schedule where the proof is quite similar to the simplest case. Then we first deal with the curve in $[\hat{t}_0,d_k]$ to make it satisfy the lemma (also arriving at a fixed speed $s'(d_k)$ at d_k), and then deal with the curve in $[d_k,t_f]$ with starting speed $s'(d_k)$ to make it satisfy the lemma.

Thus step by step, we can ensure that $\hat{t}$ is exactly one job's deadline and completion time, namely tight deadline, in the optimal schedule. □

Theorem 1. *In the optimal schedule,*

(1) *The first block is the interval* $(0,d_t)$ *which maximizes* $\frac{\sum_{J\in\mathcal{J}_t}C(J)}{d_t}$ *where* $\mathcal{J}_t=\{J_j|d_j\le d_t]\}$ *and* $t\in\{1,\ldots,n\}$, *i.e. the maximum speed in the optimal schedule is* $s_1=\max_i\frac{\sum_{J\in\mathcal{J}_i}C(J)}{d_i}$.

(2) *Suppose that block* j *has speed* s_j *and finishes at* J_{t_j}*'s deadline, then the speed in block* $j+1$ *is* $s_{j+1}=\max_t\frac{s_j-K(d_t-d_{t_j})+\sqrt{(K(d_t-d_{t_j})-s_j)^2+4K\sum_{i=t_j+1}^{t}C(J_i)}}{2}$ *where* $t\in\{t_j+1,\ldots,n\}$.

Proof. Fig. 4 shows an example. We first prove (1). According to Lemma 6, the finish time of the optimal schedule's block is one job's deadline. Thus for the first block in the optimal schedule, if the finish time is J_u's deadline, then the speed of this block is $\frac{\sum_{J\in\mathcal{J}_u}C(J)}{d_u}$ where $\mathcal{J}_u=\{J_j|d_j\le d_u\}$ according to the EDF schedule in Lemma 4. We prove that the first block of the optimal schedule achieves the maximum possible value $\frac{\sum_{J\in\mathcal{J}_t}C(J)}{d_t}$ where $t\in\{1,\ldots,n\}$. Let $u=\arg\max_t\frac{\sum_{J\in\mathcal{J}_t}C(J)}{d_t}$. We suppose on the contrary that the first block finishes at J_v's deadline where $v\neq u$. Note that $\frac{\sum_{J\in\mathcal{J}_u}C(J)}{d_u}>\frac{\sum_{J\in\mathcal{J}_v}C(J)}{d_v}$. If

$v < u$, since the speed curve of the optimal schedule is non-increasing by Lemma 3, the total workload finished before d_u will be at most $d_u \cdot \frac{\sum_{J \in \mathcal{J}_v} C(J)}{d_v}$ and hence less than $d_u \cdot \frac{\sum_{J \in \mathcal{J}_u} C(J)}{d_u}$. Therefore some jobs in $\mathcal{J}_u$ will miss deadlines because all jobs in $\mathcal{J}_u = \{J_k | d_k \in [0, d_u]\}$ with a total workload $d_u \cdot \frac{\sum_{J \in \mathcal{J}_u} C(J)}{d_u}$ should be finished before d_u. This is a contradiction to the feasibility of the optimal schedule. If $v > u$, the total workload finished before d_u is also less than $d_u \cdot \frac{\sum_{J \in \mathcal{J}_u} C(J)}{d_u}$, again a contradiction.

We prove (2) by induction. Assume as the induction hypothesis that the jth block of the optimal schedule has speed s_j and finishes at job J_{t_j}'s deadline where $1 \le t_j < n$. We will prove that the speed of the $j+1$th block in the optimal schedule achieves the maximum value $\frac{s_j - K(d_t - d_{t_j}) + \sqrt{(K(d_t - d_{t_j}) - s_j)^2 + 4K \sum_{i=t_j+1}^{t} C(J_i)}}{2}$ where $t \in \{t_j + 1, t_j + 2 \dots, n\}$. Since there is a transition interval with $s'(t) = -K$ after block j's finish time d_{t_j}, if block $j+1$ finishes at J_t's deadline, then the speed of block $j+1$ satisfies $s_{j+1} = \frac{\sum_{J \in \mathcal{J}_{(t_j, t]}} C(J)}{d_t - d_{t_j} - (s_j - s_{j+1})/K}$ where $\mathcal{J}_{(t_j,t]} = \{J_k | d_k \in (d_{t_j}, d_t]\}$ by Lemma 6. Assume that $u = \arg\max_t \frac{s_j - K(d_t - d_{t_j}) + \sqrt{(K(d_t - d_{t_j}) - s_j)^2 + 4K \sum_{i=t_j+1}^{t} C(J_i)}}{2}$. Suppose on the contrary that the $(j+1)$th block of the optimal schedule finishes at J_v's deadline where $v \neq u$. Write $s_{j+1}^u = \frac{s_j - K(d_u - d_{t_j}) + \sqrt{(K(d_u - d_{t_j}) - s_j)^2 + 4K \sum_{i=t_j+1}^{u} C(J_i)}}{2}$ and $s_{j+1}^v = \frac{s_j - K(d_v - d_{t_j}) + \sqrt{(K(d_v - d_{t_j}) - s_j)^2 + 4K \sum_{i=t_j+1}^{v} C(J_i)}}{2}$. We have $s_{j+1}^u > s_{j+1}^v$. If $v < u$, since the speed curve of the optimal schedule is non-increasing (Lemma 3), the total workload finished between $[d_{t_j}, d_u]$ will be at most $(d_u - d_{t_j} - \frac{s_j - s_{j+1}^v}{K}) \cdot s_{j+1}^v$ which is less than $(d_u - d_{t_j} - \frac{s_j - s_{j+1}^u}{K}) \cdot s_{j+1}^u$. Thus some jobs in $\mathcal{J}_{(t_j,u]}$ will miss deadlines by a similar analysis with the proof of 1. Therefore, it contradicts the feasibility of the optimal schedule. If $v > u$, the total workload finished between $[d_{t_j}, d_u]$ is also less than $(d_u - d_{t_j} - \frac{s_j - s_{j+1}^u}{K}) \cdot s_{j+1}^u$ which again leads to a contradiction. Therefore, the $(j+1)$th block must finish at d_u. This finishes the proof. □

Theorem 2. *The optimal schedule can be computed by Algorithm 1 in $O(n^2)$.*

Proof. Algorithm 1 is a direct implementation of Theorem 1. Steps 2–4 compute the first block. The two loops in Steps 6–10 compute the remaining blocks. By keeping the information of the summation on the computed jobs, the optimal schedule can be computed in $O(n^2)$ time. □

Algorithm 1 *CRT_schedule*

1. $t = 0$;
2. $s_1 = \max_i \frac{\sum_{j=1}^{i} C(J_j)}{d_i}$;
3. $t = \arg\max_i s_1$;
4. Let the block with speed s_1 be $[0, d_t]$;
5. $m = 1$;

while $t < n$ **do**

6. $s_{m+1} = \max_{t+1 \le i \le n} \frac{s_m - K(d_i - d_t) + \sqrt{(K(d_i - d_t) - s_m)^2 + 4K \sum_{j=t+1}^{i} C(J_j)}}{2}$;
7. $t' = \arg\max_i s_{m+1}$;
8. Let the block with speed s_{m+1} be $[d_t + (s_m - s_{m+1})/K, d_{t'}]$;
9. $m = m + 1$;
10. $t = t'$;

end while

4. Optimal schedules for aligned jobs

4.1. Ideal model

In the ideal model, the acceleration rate is infinity $K = \infty$. We review the Algorithm YDS in [20] to compute OPT_∞ as shown in Algorithm 2. The algorithm tries every possible pair of arrival times and deadlines to find an interval with largest intensity (called *critical interval*), schedule the jobs embedded in the critical interval and then repeatedly deal with the remaining jobs. Their algorithm has a complexity $O(n^3)$, which was then proved to $O(n^2 \log n)$ in [17].

We first show that the optimal schedule for aligned jobs in the ideal model can be computed in $O(n^2)$ time. The proof is based on two key observations. First, the faster search for a critical special time (called descending-time/acending-time in this paper). Second, some intervals/jobs can be independently picked out and then iteratively handled. In the proof, we use OPT to denoted the solution returned by Algorithm 2.

Algorithm 2 YDS Schedule for the Ideal Model

The algorithm repeats the following steps until all jobs are scheduled:
(1) Let $[t_1, t_2]$ be the maximum intensity time interval.
(2) The processor will run at speed $Itt(t_1, t_2)$ during $[t_1, t_2]$ and schedule all the jobs with $I(J) \subseteq [t_1, t_2]$ in EDF order.
(3) Then the instance is modified as if the time in $[t_1, t_2]$ did not exist. That is, all deadlines $d_i > t_1$ are changed to $\max(t_1, d_i - (t_2 - t_1))$, and all arrival times $r_i > t_1$ are changed to $\max(t_1, r_i - (t_2 - t_1))$.

Theorem 3. *The optimal schedule for aligned jobs in the ideal model can be computed in $O(n^2)$ time.*

Given an interval $[t_L, t_R]$, all the fully embedded jobs' workload over $[t_L, t_R]$ is called *average density* of $[t_L, t_R]$, which is denoted as $a\_den([t_L, t_R]) = \frac{\sum_{J|I(J)\subseteq[t_L,t_R]} C(J)}{t_R - t_L}$.

Let $r_{\min} = \min_{J\in\mathcal{J}} r(J)$ and $d_{\max} = \max_{J\in\mathcal{J}} d(J)$. We define a special kind of time as follows.

Definition 1. A time t is called down-point if $a\_den([r_{\min}, t]) > a\_den([r_{\min}, t + \Delta t])$ where $\Delta t \to 0$. Further, if $\hat{t}$ is the latest time at which $[r_{\min}, \hat{t}]$ has the highest average density among all other intervals, then $\hat{t}$ is the *global-down-point* (shorted as GDP) of $[r_{\min}, d_{\max}]$, i.e. $\hat{t} = \max\{\arg\max_{r_{\min}\le T\le d_{\max}} a\_den([r_{\min}, T])\}$.

We define d to be a *descending-time* in OPT if $\lim_{t\to d^-} s(t, \mathrm{OPT}) > \lim_{t\to d^+} s(t, \mathrm{OPT})$ (and correspondingly *ascending-time* if $\lim_{t\to d^-} s(t, \mathrm{OPT}) < \lim_{t\to d^+} s(t, \mathrm{OPT})$). The following is a basic property.

Lemma 7. *If d is a descending-time in* OPT*, then OPT never executes jobs with $I(J) \cap (d, d_{\max}] \neq \phi$ in $[r_{\min}, d]$.*

Proof. It is proved in [16] that every descending-time in OPT is a tight deadline (suppose that the job finished at this time is J_i) if the schedule follows EDF order. On the other hand EDF schedule for aligned jobs generates no preemption. Therefore, jobs with $I(J) \cap (d, d_{\max}] \neq \phi$ cannot be executed before d since their deadlines are larger than $d(J_i)$. □

We first derive some properties of GDP, comparing with the speed in OPT over $\mathcal{J}$. Fig. 5 shows an example. The height of the dashed line is the value $a\_den(\cdot)$ over the interval that is covered by the line.

Lemma 8. *If time g is the GDP of $I = [r_{\min}, d_{\max}]$, then $a\_den([r_{\min}, g]) \le \lim_{t\to g^-} s(t, \mathrm{OPT})$ and $a\_den([r_{\min}, g]) > \lim_{t\to g^+} s(t, \mathrm{OPT})$.*

Proof. First, if on the contrary $a\_den([r_{\min}, g]) \le \lim_{t\to g^+} s(t, \mathrm{OPT})$, then we assume time d is the nearest descending-time after g, which implies $s(t, \mathrm{OPT}) \ge a\_den([r_{\min}, g])$ when $g \le t \le d$. We have $C_{[g,d]}(\mathrm{OPT}) \ge a\_den([r_{\min}, g]) \cdot (d - g)$. Since d is a descending-time, all the workload $C_{[g,d]}(\mathrm{OPT})$ belongs to jobs with $g < d(J) \le d$ by Lemma 7. Thus $\sum_{J|g<d(J)\le d} C(J) \ge C_{[g,d]}(\mathrm{OPT}) \ge a\_den([r_{\min}, g]) \cdot (d - g)$. We have

$$\begin{aligned} a\_den([r_{\min}, d]) = \frac{\sum\limits_{J|I(J)\in[r_{\min},d]} C(J)}{d - r_{\min}} &= \frac{\sum\limits_{J|I(J)\in[r_{\min},g]} C(J) + \sum\limits_{J|g<d(J)\le d} C(J)}{d - r_{\min}} \\ &\ge \frac{(g - r_{\min}) \cdot a\_den([r_{\min}, g]) + (d - g) \cdot a\_den([r_{\min}, g])}{d - r_{\min}} \\ &\ge a\_den([r_{\min}, g]). \end{aligned}$$

Thus d has an average density at least that of g and d is later than g, this contradicts the definition of g.

Second, if on the contrary $a\_den([r_{\min}, g]) > \lim_{t\to g^-} s(t, \mathrm{OPT})$, we assume $[c, d]$ is the nearest block before g which has speed larger than $a\_den([r_{\min}, g])$ in OPT. Such a block exists because otherwise $C_{[r_{\min},g]}(\mathrm{OPT}) < a\_den([r_{\min}, g]) \cdot (g - r_{\min})$ which implies $a\_den([r_{\min}, g]) = \frac{\sum_{J|I(J)\subseteq[r_{\min},g]} C(J)}{g - r_{\min}} \le \frac{C_{[r_{\min},g]}(\mathrm{OPT})}{g - r_{\min}} < a\_den([r_{\min}, g])$. According to the choice of d, we have $C_{[d,g]}(\mathrm{OPT}) < a\_den([r_{\min}, g]) \cdot (g - d)$. All jobs with $d < d(J) \le g$ should be executed in $[d, g]$ in OPT by Lemma 7. Thus $\sum_{d<d(J)\le g} C(J) \le C_{[d,g]}(\mathrm{OPT})$. By the definition of GDP, every time before g has average density at most $a\_den([r_{\min}, g])$. Thus $\sum_{J|I(J)\subseteq[r_{\min},d]} C(J) = (d - r_{\min}) \cdot a\_den([r_{\min}, d]) \le (d - r_{\min}) \cdot a\_den([r_{\min}, g])$. We have

$$\begin{aligned} a\_den([r_{\min}, g]) = \frac{\sum\limits_{J|I(J)\subseteq[r_{\min},g]} C(J)}{g - r_{\min}} &= \frac{\sum\limits_{J|I(J)\subseteq[r_{\min},d]} C(J) + \sum\limits_{J|d<d(J)\le g} C(J)}{g - r_{\min}} \\ &\le \frac{(d - r_{\min}) \cdot a\_den([r_{\min}, g]) + C_{[d,g]}(\mathrm{OPT})}{g - r_{\min}} \\ &< \frac{(d - r_{\min}) \cdot a\_den([r_{\min}, g]) + a\_den([r_{\min}, g]) \cdot (g - d)}{g - r_{\min}} \\ &= a\_den([r_{\min}, g]). \end{aligned}$$

Again a contradiction. Hence, the lemma is then proved. □

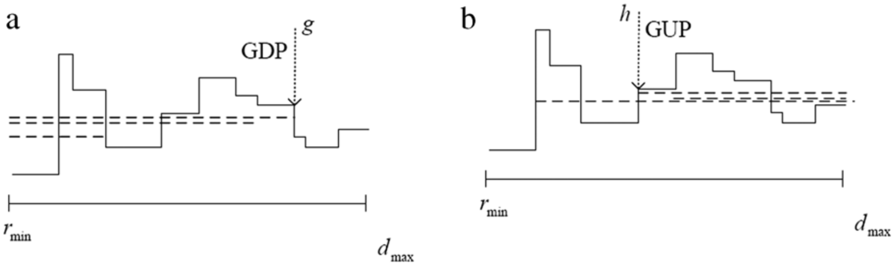

Fig. 5. An example that shows the GDP g and GUP h for job set $\mathcal{J}$.

Lemma 8 implies that GDP g is a descending-time in OPT. We remark that a normal down-point is not necessarily a descending-time in OPT. Since each descending-time is one job's deadline by [16], it is sufficient to compute the average density in interval $[r_{\min}, d_i]$ for all deadlines d_i in order to find the GDP. Among all the $|\mathcal{J}|$ values, the GDP g equals the latest deadline that achieves the maximum density, $g = \max\{\arg\max_{d_i} a\_den([r_{\min}, d_i])\}$.

On the other hand, if we symmetrically compute the average density over $[t, d_{\max}]$ in right-to-left order, we could define the *up-point* t which has $a\_den([t - \Delta t, d_{\max}]) < a\_den([t, d_{\max}])$ where $\Delta t \to 0$. Similar to Lemma 8, we have a *global-up-point* (GUP) $h \in [r_{\min}, d_{\max}]$ of I which corresponds to an ascending-time in OPT. Moreover, $h = \min\{\arg\max_{r_i} a\_den([r_i, d_{\max}])\}$. The following property is useful for our algorithm to iteratively compute OPT.

Lemma 9. *In the optimal schedule, jobs that are executed in* $[r_{\min}, g]$ *have a deadline at most* g*. Moreover, for jobs* J *where* $r(J) < g$ *and* $d(J) > g$*, if we re-scale the arrival time* $r(J)$ *to be* g*, then OPT for the modified job set is the same as OPT for the original job set.*

Proof. It is sufficient to prove that the optimal schedule OPT never executes jobs with $d(J) > g$ in $[r_{\min}, g]$. This is true by Lemma 7 since g is a descending-time. □

Note that this property holds symmetrically for a GUP h.

Lemma 10. *In the optimal schedule, jobs that are executed in* $[h, d_{\max}]$ *have arrival time at least* h*. Moreover, for jobs* J *where* $r(J) < h$ *and* $d(J) > h$*, if we re-scale the deadline* $d(J)$ *to be* h*, then OPT for the modified job set is the same as OPT for the original job set.*

By these observations, Algorithm 3 will pick out some sub-intervals in which OPT only executes jobs embedded in these sub-intervals. In each iteration, by computing a pair of GDP/GUP times, the original interval/job-set is partitioned into at most three intervals/job-sets. Then it iteratively computes the optimal schedule for the three subsets. The algorithm terminates when the re-scaled job-set has $g = d_{\max}$ and $h = r_{\min}$. We then prove the following lemma which implies that if $g = d_{\max}$ and $h = r_{\min}$, then it is equivalent to finding a block in OPT.

Lemma 11. *For aligned job set* $\mathcal{J}$*, if* $g = d_{\max}$ *and* $h = r_{\min}$*, then OPT executes all jobs with speed* $a\_den([r_{\min}, d_{\max}])$*.*

Proof. It suffices to prove that there is no descending-time/ascending-time in interval $[r_{\min}, d_{\max}]$ when $h = r_{\min}$ and $g = d_{\max}$. We prove it by contradiction. Suppose on the contrary that such a time exists, then we assume block $[t_a, t_b]$ is the first (earliest) block that has speed $s > a\_den([r_{\min}, d_{\max}])$ in OPT. Let $[t_c, t_d]$ be the nearest peak after t_a ($[t_c, t_d]$ can possibly be $[t_a, t_b]$ itself). First, all blocks between $[r_{\min}, t_a]$ have a speed at most $a\_den([r_{\min}, d_{\max}])$ according to the choice of $[t_a, t_b]$. Second, we will prove that OPT has a speed exactly $a\_den([r_{\min}, d_{\max}])$ in the interval $[r_{\min}, t_a]$. We suppose on the contrary that there exists at least one block between $[r_{\min}, t_a]$ with a speed less than $a\_den([r_{\min}, d_{\max}])$. Then the workload $C_{[r_{\min}, t_a]}(\mathrm{OPT}) < a\_den([r_{\min}, d_{\max}]) \cdot (t_a - r_{\min})$. By symmetrically applying Lemma 7, all workload $C_{[r_{\min}, t_a]}(\mathrm{OPT})$ belongs to jobs with $I(J) \cap [r_{\min}, t_a] \neq \phi$. We have

$$
\begin{aligned}
a\_den([t_a, d_{\max}]) &= \frac{\sum_{J|I(J)\subseteq[t_a, d_{\max}]} C(J)}{d_{\max} - t_a} \\
&= \frac{\sum_{J|I(J)\subseteq[r_{\min}, d_{\max}]} C(J) - \sum_{J|I(J)\cap[r_{\min}, t_a]\neq\phi} C(J)}{d_{\max} - t_a} \\
&> \frac{a\_den([r_{\min}, d_{\max}] \cdot (d_{\max} - r_{\min} - (t_a - r_{\min})))}{d_{\max} - t_a} = a\_den([r_{\min}, d_{\max}])
\end{aligned}
$$

which implies $h > r_{\min}$, a contradiction.

Thus the interval $[r_{\min}, t_a]$ is exactly a block with speed $a\_den([r_{\min}, d_{\max}])$ in OPT. Then we have $C_{[r_{\min}, t_d]}(\mathrm{OPT}) > (t_d - r_{\min}) \cdot a\_den([r_{\min}, d_{\max}])$. By Lemma 7, all workload in $C_{[r_{\min}, t_d]}(\mathrm{OPT})$ belongs to jobs with $d(J) \leq t_d$. We have $a\_den([r_{\min}, t_d]) \geq \frac{C_{[r_{\min}, t_d]}(\mathrm{OPT})}{t_d - r_{\min}} > a\_den([r_{\min}, d_{\max}])$ which indicates $g < d_{\max}$ by the definition of GDP. In this way, we successfully obtain a contradiction and arrive at the conclusion that $[r_{\min}, d_{\max}]$ should be one block in OPT when $g = d_{\max}$ and $h = r_{\min}$. □

Now we can show that Algorithm 3 computes the optimal schedule for aligned jobs in $O(n^2)$ time. Steps 4–9 show the re-scaling procedure. In both cases $h < g$ (e.g. Fig. 5) and $h > g$, the job set $\mathcal{J}$ is divided into three subsets $\mathcal{J}_{\mathcal{L}}, \mathcal{J}_{\mathcal{R}}, \mathcal{J}_{\mathcal{M}}$ with adjusted arrival times and deadlines. Notice that $h = g$ is not possible because a time cannot be both a descending-time and an ascending-time. By Lemmas 9 and 10, the optimal schedule computed for the re-scaled job sets can directly combine into the optimal schedule for the original job set. Note that for the re-scaled jobs, the algorithm terminates when $g = d_{\max}$ and $h = r_{\min}$. Because this corresponds to the discovery of a block in OPT by Lemma 11.

Now we analyze the running time of Algorithm 3. Looking for GDP and GUP (Step 2) in one job set needs at most $O(n)$ time because there are at most $2n$ ascending/descending times. We can organize all the job sets we deal with by a tree reflecting the subset relation between job sets. In this way, a leaf in the tree represents a block (the number of blocks is at most n) in OPT because no further partition is done on the leaf due to the reason $g = d_{\max}$ and $h = r_{\min}$. While the number of nodes in the tree is the total number of different job sets for which we need to find GDP and GUP. We know that the number of nodes in a tree is twice the number of leaves in the tree minus 1. Furthermore, by Lemma 11, every job set only needs to be dealt with once. Therefore, the running time of Algorithm 3 is $O(n^2)$.

Algorithm 3 *prec_Ideal*($\mathcal{J}$)

1. $r_{\min} = \min_{J\in\mathcal{J}} r(J)$; $d_{\max} = \max_{J\in\mathcal{J}} d(J)$;
2. Find a GDP $g \in [r_{\min}, d_{\max}]$ and a GUP $h \in [r_{\min}, d_{\max}]$.

/* Compute the schedule for the minimal interval*/
if $g = d_{\max}$ and $h = r_{\min}$ **then**
  3. Execute all jobs with speed $s = a\_den([r_{\min}, d_{\max}])$ in EDF order.
else
  /* Otherwise, re-scale the jobs into three subsets*/
  if $h < g$ **then**
    4. $\mathcal{J}_{\mathcal{L}} = \{J | I(J) \cap [r_{\min}, h] \neq \phi$ where we adjust $d(J) = \min\{d(J), h\}$;
    5. $\mathcal{J}_{\mathcal{R}} = \{J | I(J) \cap [g, d_{\max}] \neq \phi$ where we adjust $r(J) = \max\{r(J), g\}$;
    6. $\mathcal{J}_{\mathcal{M}} = \{J | I(J) \subseteq [h, g]\}$;
  end if
  if $h > g$ **then**
    7. $\mathcal{J}_{\mathcal{L}} = \{J | I(J) \subseteq [r_{\min}, g]\}$;
    8. $\mathcal{J}_{\mathcal{R}} = \{J | I(J) \subseteq [h, d_{\max}]\}$;
    9. $\mathcal{J}_{\mathcal{M}} = \mathcal{J} \backslash \mathcal{J}_{\mathcal{L}} \backslash \mathcal{J}_{\mathcal{R}}$ where we adjust $d(J) = \min\{d(J), h\}$ and $r(J) = \max\{r(J), g\}$;
  end if
end if
/* Iteratively compute over the subset $\mathcal{J}_{\mathcal{L}}, \mathcal{J}_{\mathcal{R}}, \mathcal{J}_{\mathcal{M}}$ if they are not empty*/
10. *prec_Ideal*($\mathcal{J}_{\mathcal{L}}$); *prec_Ideal*($\mathcal{J}_{\mathcal{R}}$);*prec_Ideal*($\mathcal{J}_{\mathcal{M}}$);

4.2. Accelerate model

In this section, we study the optimal schedule for the general aligned jobs. Note that jobs with a common arrival time is a special case of aligned jobs. We first extend some of its basic properties in Section 4.2.1. We will compute the optimal schedule for aligned jobs by adopting Algorithm 1 as a building block. We use OPT_K to denote the optimal schedule where K is the maximum acceleration rate.

Given a block $block_p$, we denote the corresponding interval as $[L(block_p), R(block_p)]$. We define *virtual canyon* to be a block with length 0. Next, we derive some properties of OPT_K.

4.2.1. Basic properties

Lemma 12. *There is an optimal schedule where jobs are executed in EDF order.*

The proof for the extension is similar as Lemma 4.

Lemma 13. *In the optimal schedule* OPT_K*, the speed function will accelerate as fast as possible, i.e., either* $|s'(t)| = K$ *or* $s'(t) = 0$.

Proof. If there exists an acceleration interval $[a, b]$ with $|s'(t)| < K$ as shown in Fig. 6, then we can construct a virtual canyon $block_p$ between $[a, b]$ and let the acceleration rate in $[a, L(block_p)]$ and $[R(block_p), b]$ be $-K$ and K respectively. If $s(L(block_p)) < 0$ due to this transformation, then we replace the curve below $s = 0$ by a segment with speed 0. This can remove the possibility $0 < |s'(t)| < K$, and ensure that the schedule is feasible and the energy does not increase. □

Unlike Lemma 3, $s'(t) = K$ cannot be eliminated in OPT_K.

Among all the blocks, we define the block $[t_a, t_b]$ where $\lim_{t\to t_a^-} s'(t) = K \wedge \lim_{t\to t_a^+} s'(t) = 0$ and $\lim_{t\to t_b^-} s'(t) = 0 \wedge \lim_{t\to t_b^+} s'(t) = -K$ to be a *peak*. Reversely, the block where $\lim_{t\to t_a^-} s'(t) = -K \wedge \lim_{t\to t_a^+} s'(t) = 0$ and $\lim_{t\to t_b^-} s'(t) = 0 \wedge \lim_{t\to t_b^+} s'(t) = K$ is called a *canyon*.

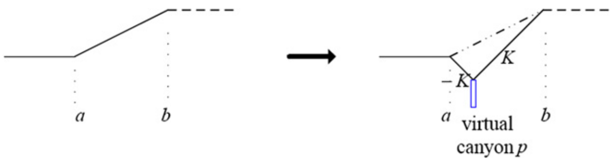

Fig. 6. Virtual canyon.

We say $\hat{t}$ is *down-edge-time* if $\lim_{t\to\hat{t}^-} s'(t) = 0 \wedge \lim_{t\to\hat{t}^+} s'(t) = -K$ or $\lim_{t\to\hat{t}^-} s'(t) = K \wedge \lim_{t\to\hat{t}^+} s'(t) = 0$. For example, both the start time and finish time of a peak are down-edge-times. Note that for jobs with a common arrival time, $s(t)$ is non-increasing, thus the finish time of a block is a down-edge-time. The following lemma extends Lemma 6 to consider the aligned jobs.

Lemma 14. *In the optimal schedule* OPT_K*, every down-edge-time is either a tight deadline or a tight arrival time.*

Proof. We first show that down-edge-time $\hat{t}$ is a tight deadline when $\lim_{t\to\hat{t}^-} s'(t) = 0 \wedge \lim_{t\to\hat{t}^+} s'(t) = -K$. In the optimal schedule, assume block $[a, \hat{t}]$ has a down-edge-time $\hat{t}$. Suppose on the contrary that the job executed at time $\hat{t}$ (let the job be J) has a deadline $d(J) > \hat{t}$. We will prove that the energy can be reduced which contradicts the optimality. Note that the definition of down-edge-time implies the existence of a canyon which has a finish time (assume to be d) larger than $\hat{t}$. Then the speed function $s(t)$ in interval $[a, d]$ is non-increasing. This allows us to apply the similar transformation as in the proof of Lemma 6. The method is also to gradually postpone the completion time of J, which will finally ensure all down-edge-times in $[a, d]$ are tight deadlines (or tight arrival times symmetrically). Since all the jobs that are executed originally after time $\hat{t}$ are now executed with a higher speed and executed later after the postpone procedure, this will not violate the timing constraint (both for arrival time and deadline). Thus in this case we can ensure that $\hat{t}$ is a tight deadline. For the other type of down-edge-time, we can similarly show that it is a tight arrival time. This finishes the proof. □

Lemma 15. *In the optimal schedule* OPT_K*, each job J is executed only in one block, and this block is the lowest one in interval* $[r(J), d(J)]$.

Proof. If a job J is executed in several blocks, we can see that these executions must form a continuous interval if we remove all the acceleration intervals, because for aligned jobs no preemption will happen since the schedule follows EDF order by Lemma 12.

W.l.o.g. we assume that OPT_K executes J in two adjacent blocks $block_{p-1}$,$block_p$, and $R(block_{p-1})$ is a down-edge-time (in this case $\lim_{t\to R(block_{p-1})^-} s'(t, \mathrm{OPT}_K) = 0 \wedge \lim_{t\to R(block_{p-1})^+} s'(t, \mathrm{OPT}_K) = -K$). This contradicts Lemma 14 because at time $R(block_{p-1})$, OPT_K executes job J with $d(J) > R(block_{p-1})$. Thus every job J can only be executed in one block in OPT_K. Moreover, assume that on the contrary J is not executed in the lowest block overlapping $[r(J), d(J)]$. Without loss of generality, we can assume that J is executed in $block_p$ with speed s and there is another block $block_q$ on the right of $block_p$ with speed less than s and $d(J) > L(block_q)$. Then there exists a down-edge-time t in the interval $[R(block_p), L(block_q)]$ which is a tight deadline or equivalently a job's deadline and completion time (Let this job be J'). If $J = J'$, then it is a contradiction because $d(J) > L(block_q)$; if $J \neq J'$, then J' is executed after J but has a deadline before $d(J)$. According to Lemma 12, J' should be executed before J since the input job set is an aligned job set, a contradiction. □

4.2.2. *$O(n^2)$ time algorithm to compute* OPT_K

To find the optimal schedule, our method is to identify some special blocks belonging to OPT_K. After enough blocks are selected, the remaining interval of OPT_K can be easily computed. To be more specific, we compare OPT_K with schedule OPT_∞, which is the optimal schedule for the special case $K = \infty$, namely the ideal model. We observe that the block with the highest speed (we call it *global-peak*) of OPT_K can be computed first.

Lemma 16. OPT_K *executes the same as* OPT_∞ *in the first critical interval.*

Proof. Suppose that $[a, b]$ is the first critical interval computed by Algorithm 2. Then in OPT_∞, all jobs with $I(J) \subseteq [a, b]$ are executed at a speed $Itt(a, b)$. We prove this lemma by investigating two properties of OPT_K.

The first property is: in OPT_K, no jobs need to be executed with a speed higher than $Itt(a, b)$. For any job, OPT_K runs it in a unique block (speed) according to Lemma 15. Let J be the job with the highest speed s in OPT_K. We suppose on the contrary that $s > Itt(a, b)$ and it belongs to block p. Because the two down-edge-times of $block_p$ are exactly a tight deadline and a tight arrival time, the interval of $block_p$ will have a larger intensity than $[a, b]$ because the job set under investigation is an aligned job set, a contradiction. Thus the first property is true.

The second property is: for any job with $I(J) \subseteq [a, b]$, OPT_K cannot run it with speed $s < Itt(a, b)$. According to Algorithm 2, the first critical interval $[a, b]$ execute all (and only) jobs with $I(J) \subseteq [a, b]$. Note that when all the jobs with $I(J) \subseteq [a, b]$ run in EDF order and with speed $Itt(a, b)$, the workload $w(a, b)$ are finished exactly at b. If any one of these jobs runs with a lower speed than $Itt(a, b)$ in OPT_K, then to ensure that the remaining workload satisfies the timing constraint, some jobs must have a speed $s > Itt(a, b)$. This contradicts the first property above.

The combination of the two properties indicates that OPT_K runs all/only jobs with $I(J) \subseteq [a, b]$ at a speed of exactly $Itt(a, b)$ in the interval $[a, b]$. This is the same as that of OPT_∞. Therefore, the lemma is true. □

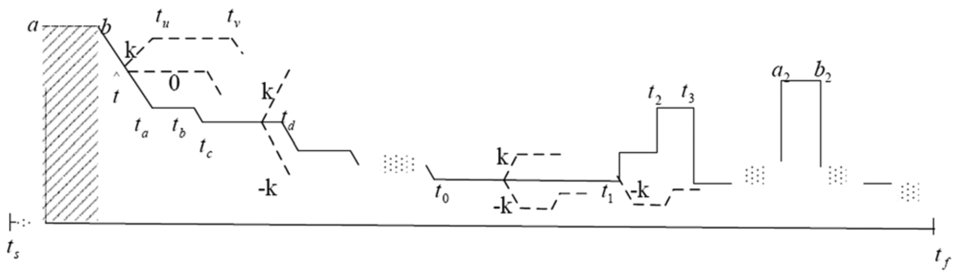

Fig. 7. Possible cases of the separation-time.

After we have fixed the first block (global-peak) of OPT_K, a natural question is whether we can apply the same proof of Lemma 16 to select other blocks. For example, in the remaining interval of OPT_∞, does the block with maximum intensity have the same schedule as that of OPT_K? Although this is not true, we will show that some other blocks in OPT_∞ can be proved to be the same as OPT_K. The key observation is that by appropriately dividing the whole interval into two sub-intervals, the block with the maximum intensity inside one of the sub-intervals in OPT_∞ can be proved to be the same as OPT_K. Our partition of intervals is based on a *monotone-interval* defined below.

Definition 2. Given a schedule, we define the sub-interval where the speed function/curve is strictly non-increasing or non-decreasing to be a *monotone-interval*.

Here and in the following, by "strictly non-increasing" we mean non-increasing but not constant; similarly by "strictly non-decreasing" we mean non-decreasing but not constant. Since the speed in OPT_K outside the global-peak $[a, b]$ is at most $Itt(a, b)$, there exists a monotone-interval immediately after time b (non-increasing curve) and symmetrically before time a (non-decreasing curve). At time b and a, the speeds are respectively $s_b = Itt(a, b)$ and $s_a = Itt(a, b)$.

In the following, we will study a schedule $S_{[b,t_1]}$ (only specifying speeds in interval $[b, t_1]$) with monotone-interval $[b, t_1]$ (non-increasing speed with $s(b, S_{[b,t_1]}) = s(b, \mathrm{OPT}_\infty)$). Suppose that t_1 is the first (earliest) intersection of the two curves $s(t, S_{[b,t_1]})$ and $s(t, \mathrm{OPT}_\infty)$ with $\lim_{t\to t_1^+} s(t, \mathrm{OPT}_\infty) > 0$. Fig. 8 shows an example. We will compare the speed curve of OPT_K with that of $S_{[b,t_1]}$ based on the definition below.

Definition 3. In interval $[b, t_1]$, we say that t is a separation-time of OPT_K w.r.t $S_{[b,t_1]}$ if their speed curves totally overlap in interval $[b, t]$ and separate at $t + \Delta t$ where $\Delta t \to 0$.

Fact 2. *If there exists a schedule $S_{[b,t_1]}$ (let the lowest/latest block and the second lowest block in $S_{[b,t_1]}$ be $[t_0, t_1]$ and $block_{\tilde{p}}$) satisfying the following properties,*

- (1) $s(t_1, S_{[b,t_1]}) < \lim_{t\to t_1^+} s(t, \mathrm{OPT}_\infty)$.
- (2) *Block $[t_0, t_1]$ executes all jobs with $I(J) \cap [t_0, t_1] \neq \phi$.*
- (3) $S_{[b,t_1]}$ *restricted to $[b, R(block_{\tilde{p}})]$ is feasible for all jobs with $d(J) \in (b, R(block_{\tilde{p}})]$ and only executes these jobs. The down-edge-times in $S_{[b,R(block_{\tilde{p}})]}$ are tight deadlines.*

Then, let $\hat{t}$ be the separation-time of OPT_K w.r.t $S_{[b,t_1]}$, we have

- (a) $b < \hat{t} < t_1$ *and* $\lim_{t\to\hat{t}^+} s'(t, \mathrm{OPT}_K) \neq -K$.
- (b) *the speed curve of OPT_K in interval $[\hat{t}, t_1]$ is strictly non-decreasing.*
- (c) OPT_K *executes all jobs with $I(J) \cap [t_0, t_1] \neq \phi$ before time t_1 and time t_1 is a down-edge-time with* $\lim_{t\to t_1^-} s'(t, \mathrm{OPT}_K) = K \wedge \lim_{t_1\to t_1^+} s'(t, \mathrm{OPT}_K) = 0$. *Furthermore, OPT_K executes a job J with $r(J) = t_1$ at time t_1.*

Proof. As shown in Fig. 7, suppose that $S_{[b,t_1]}$ has blocks $[t_a, t_b], [t_c, t_d], \ldots, [t_0, t_1]$ in left-to-right order. We first remove the possibility that $t_1 \leq \hat{t}$. Let $[t_2, t_3]$ be the nearest peak after t_1 in OPT_∞. We can see that if $t_1 \leq \hat{t}$, then OPT_K cannot have a speed curve strictly non-increasing in $[t_1, t_3]$ because otherwise some job will miss the deadline. Therefore, there exists a canyon (or virtual canyon) after t_1 in OPT_K, and we assume that $[t_u, t_v]$ is the first block after this canyon. We have $t_u < t_3$. In this case, OPT_K executes the workload $C_{[t_u,t_v]}(\mathrm{OPT}_K)$ later than that in OPT_∞ which contradicts "t_u is a tight arrival time in OPT_K". Until now, we have removed the possibility $t_1 \leq \hat{t}$.

Now we prove the second part of property (a). First, OPT_K has a monotone-interval (non-increasing) after time b, from the analysis above, we know that the separation-time of OPT_K w.r.t $S_{[b,t_1]}$ satisfies $b < \hat{t} < t_1$. We then discuss case by case. If $\lim_{t\to\hat{t}^+} s'(t, S_{[b,t_1]}) = -K$, we have $\lim_{t\to\hat{t}^+} s'(t, \mathrm{OPT}_K) \neq -K$ by the definition of separation-time. If $s(\hat{t}, S_{[b,t_1]}) = 0$, obviously $\lim_{t\to\hat{t}^+} s'(t, \mathrm{OPT}_K) \neq -K$. For the remaining case that $\lim_{t\to\hat{t}^+} s'(t, S_{[b,t_1]}) = 0 \wedge s(\hat{t}, S_{[b,t_1]}) \neq 0$, we suppose on the contrary that $\lim_{t\to\hat{t}^+} s'(t, \mathrm{OPT}_K) = -K$. We assume that the block containing $\hat{t}$ is $[t_u, t_v]$. Note that the workload $C_{[t_u,t_v]}(S_{[b,t_1]})$ needs to be finished before time t_v in OPT_K, because t_v is a tight deadline according to condition (3). This implies that OPT_K cannot be strictly non-increasing in $(\hat{t}, t_v]$ because otherwise some jobs will miss deadlines. Thus we assume $block_p$ to be the first canyon after $\hat{t}$ and block $[t'_u, t'_v]$ to be the nearest block after $block_p$. Note that t'_u should be a down-edge-time and a tight arrival time. However, OPT_K executes $C_{[t'_u,t'_v]}(\mathrm{OPT}_K)$ later than that in $S_{[b,t_1]}$ restricted to $[\hat{t}, t_1]$. Thus t_u cannot be a tight arrival time due to condition (3), a contradiction.

For property (b), since $\hat{t}$ is a separation-time, we have $\lim_{t\to\hat{t}^+} s'(t, \mathrm{OPT}_K) = K$ or $\lim_{t\to\hat{t}^+} s'(t, \mathrm{OPT}_K) = 0$ by property (a). Suppose on the contrary that there exists a block (let it be $[t_u, t_v]$ with speed s and $t_v < t_1$) such that OPT_K is strictly non-decreasing in $[\hat{t}, t_v]$ and $\lim_{t\to\hat{t}_v^+} s'(t, \mathrm{OPT}_K) = -K$. Two subcases should be discussed. If $t_u = \hat{t}$, then t_v is a down-edge-time and hence also a tight deadline according to Lemma 14. However, we note that the speed of the block $[t_u, t_v]$ in OPT_K is higher than the speed of all the blocks between $[\hat{t}, t_1]$ in $S_{[b,t_1]}$. Therefore, $C_{[t_u,t_v]}(\mathrm{OPT}_K)$ is finished earlier in OPT_K than the corresponding workload executed in $S_{[b,t_1]}$. By condition (3), time t_v cannot be a tight deadline which contradicts Lemma 14. If $t_u > \hat{t}$, then t_u is a tight arrival time and t_v is a tight deadline by Lemma 14. $C_{[t_u,t_v]}(\mathrm{OPT}_K)$ belongs to the jobs with $I(J) \subseteq [t_u, t_v]$. However, $S_{[b,t_1]}$ has a speed lower than that in OPT_K in $[t_u, t_v]$. Thus some jobs will violate the timing constraints in $S_{[b,t_1]}$ which contradicts condition (3).

The above analysis shows that OPT_K must have a speed curve that coincides with $S_{[b,t_1]}$ in interval $[b, \hat{t}]$ where $b < \hat{t} < t_1$. Furthermore, OPT_K is strictly non-decreasing in $[\hat{t}, t_1]$. We now prove property (c). The above analysis shows that the nearest canyon after b in OPT_K contains separation-time $\hat{t}$. Assume that $[t_l, t_r]$ is the canyon with speed s'. Suppose that $S_{[b,t_1]}$ has speed s_0 at block $[t_0, t_1]$. Note that $s' \geq s_0$ since $\hat{t} < t_1$. We can further remove the possibility $s' = s_0$. Because otherwise $t_0 \leq \hat{t} < t_1$ and $\lim_{t\to\hat{t}^+} s(t, \mathrm{OPT}_K) = K$. There exists a tight arrival time (assume to be t_u) immediately after $\hat{t}$. The workload $C_{[\hat{t},t_1]}(S_{[b,t_1]})$ is executed later (starting from time t_u) in OPT_K and this contradicts that t_u is a tight arrival time. We will discuss the two cases that $\lim_{t\to t_1^-} s(t, \mathrm{OPT}_K) = K$ and $\lim_{t\to t_1^-} s(t, \mathrm{OPT}_K) = 0$. First, if $\lim_{t\to t_1^-} s(t, \mathrm{OPT}_K) = 0$, then there exists a block $[t_u, t_v]$ with $t_u < t_1 \leq t_v$ in OPT_K and $[t_u, t_v]$ is after the canyon $[t_l, t_r]$ (otherwise $[t_u, t_v] = [t_l, t_r]$, then OPT_K is doing the workload faster than $S_{[b,t_1]}$ in $[t_l, t_1]$ but they are doing the same amount of workload, a contradiction). Thus t_u is a down-edge-time. Let the speed in $[t_u, t_v]$ be s''. We have $s'' > s' \geq s_0$. Interval $[t_u, t_1]$ cannot execute the jobs with $I(J) \subseteq [t_1, t_f]$. However, the workload $C_{[t_u,t_1]}(\mathrm{OPT}_K)$ is executed later and with a higher speed in OPT_K than that in $S_{[b,t_1]}$. By the feasibility of $S_{[b,t_1]}$ (condition (3)), time t_u cannot be a tight arrival time, which is a contradiction. Second, if $\lim_{t\to t_1^-} s(t, \mathrm{OPT}_K) = K$, we can remove the possibility of $\lim_{t\to t_1^+} s(t, \mathrm{OPT}_K) = K$. Because otherwise there exists a block $[t_u, t_v]$ immediately after t_1 where t_u is a tight arrival time. However, this is impossible since all jobs with $I(J) \subseteq [t_1, t_f]$ is executed after t_1 and thus one of the jobs with $r(J) = t_1$ or $I(J) \cap [t_0, t_1] \neq \phi$ is the first job executed in t_u in OPT_K (ties can be arbitrarily broken). This contradicts the fact that t_u is a tight arrival time. Thus the only case that remains is $\lim_{t\to t_1^-} s(t, \mathrm{OPT}_K) = K \wedge \lim_{t\to t_1^+} s(t, \mathrm{OPT}_K) = 0$. Note that t_1 is a down-edge-time in this case and should be a tight arrival time. Thus jobs with $r(J) < t_1$ cannot be executed at t_1 in OPT_K. In other words, OPT_K can only execute the job with $r(J) = t_1$ at time t_1. Therefore, all jobs with $I(J) \subseteq [t_0, t_1] \neq \phi$ are executed before t_1 because OPT_K executes the jobs in EDF order. This finishes the proof of property (c). □

Fact 3. *If there exists a schedule $S_{[b,t_1]}$ satisfying the three conditions in Fact 2, let $[a_2, b_2]$ be the maximum intensity block in OPT_∞ among the remaining interval $[t_1, t_f]$, then OPT_K has the same schedule as OPT_∞ in interval $[a_2, b_2]$.*

Proof. $S_{[b,t_1]}$ has divided the interval $[t_s, t_f]$ into two sub-intervals $[t_s, t_1]$ and $[t_1, t_f]$. By properties (b) and (c) in Fact 2, the speed curve of OPT_K in interval $[\hat{t}, t_1 + \Delta t]$ ($\Delta t \to 0$) is strictly non-decreasing, where $\hat{t}$ is the separation-time of OPT_K w.r.t $S_{[b,t_1]}$. Moreover, t_1 is a down-edge-time and OPT_K executes the job with $r(J) = t_1$ in $[t_1, t_1 + \Delta t]$ and all jobs with $I(J) \cap [t_0, t_1] \neq \phi$ in $[t_s, t_1]$. Since $\lim_{t\to t_1^-} s'(t, \mathrm{OPT}_K) = K \wedge \lim_{t\to t_1^+} s'(t, \mathrm{OPT}_K) = 0$, there exists at least one peak among the interval $[t_1, t_f]$ in OPT_K. W.l.o.g we assume the peak $[t_u, t_v] \subseteq [t_1, t_f]$ to be the interval with maximum speed s among $[t_1, t_f]$ in OPT_K. We will examine the schedule OPT_K and OPT_∞ in the interval $[t_1, t_f]$.

We know that t_u is a tight arrival time and t_v is a tight deadline by Lemma 14. If $s > Itt(a_2, b_2)$, since OPT_K executes all/only jobs with $I(J) \subseteq [t_u, t_v]$ in $[t_u, t_v]$ (otherwise violating Lemma 14), this implies that OPT_∞ has an intensity in $[t_u, t_v]$ larger than that in $[a_2, b_2]$. This contradicts the condition that "$[a_2, b_2]$ is the maximum intensity block in $[t_1, t_f]$". If $s < Itt[a_2, b_2]$, then since the total workload of jobs with $I(J) \subseteq [a_2, b_2]$ is exactly $(b_2 - a_2) \cdot Itt(a_2, b_2)$, OPT_K cannot complete all these jobs with a speed less than $Itt(a_2, b_2)$. Therefore $s = Itt(a_2, b_2)$. Now we look at OPT_K restricted to the interval $[a_2, b_2]$, if OPT_K uses a speed less than s in part of $[a_2, b_2]$, then in order to finish all the workload of jobs with $I(J) \subseteq [a_2, b_2]$, OPT_K must use a speed higher than s in some other part of $[a_2, b_2]$, which contradicts the definition of s. Hence, OPT_K will execute all/only jobs with $I(J) \subseteq [a_2, b_2]$ in interval $[a_2, b_2]$ and the speed is exactly $Itt(a_2, b_2)$. This is the same as the schedule OPT_∞ in $[a_2, b_2]$. This ends the proof. □

Next we present Algorithm 4 which can compute a schedule $S_{[b,t_1]}$ satisfying the three conditions in Fact 2. This algorithm repeatedly calls Algorithm 1 to handle several blocks in OPT_∞. Blocks in OPT_∞ starting from the global-peak are indexed as 0,1,2,.... We use $s(block_p)$ to denote the speed of $block_p$ in OPT_∞. It outputs a monotone-interval starting at time b (which equals $R(block_0)$), where the speed at b should be $s(b) = Itt(a, b)$ according to Lemma 16. Let $t_1 = R(block_r)$ where $block_r$ is the current block being handled. The algorithm terminates when the lowest speed in $[b, t_1]$ is less than $s(L(block_{r+1}), \mathrm{OPT}_\infty)$. Fig. 8 shows an example, where $S_{[b,t_1]}$ is the schedule with monotone-interval computed by Algorithm 4 and $[t_1, t_f]$ is the un-handled interval in OPT_∞.

Lemma 17. *Algorithm 4 computes a schedule $S_{[b,t_1]}$ with a non-increasing speed curve in $O(n^2)$ time.*

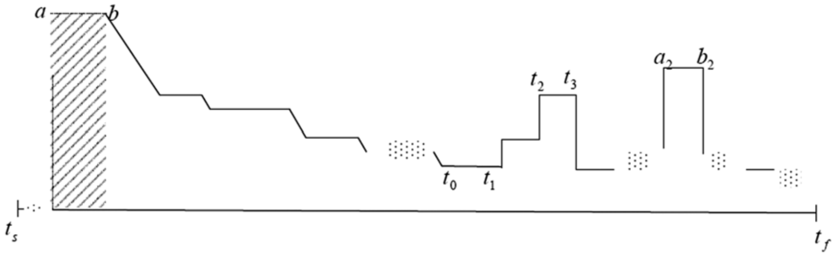

Fig. 8. An example that shows schedule $S_{[b,t_1]}$ in monotone-interval $[b, t_1]$.

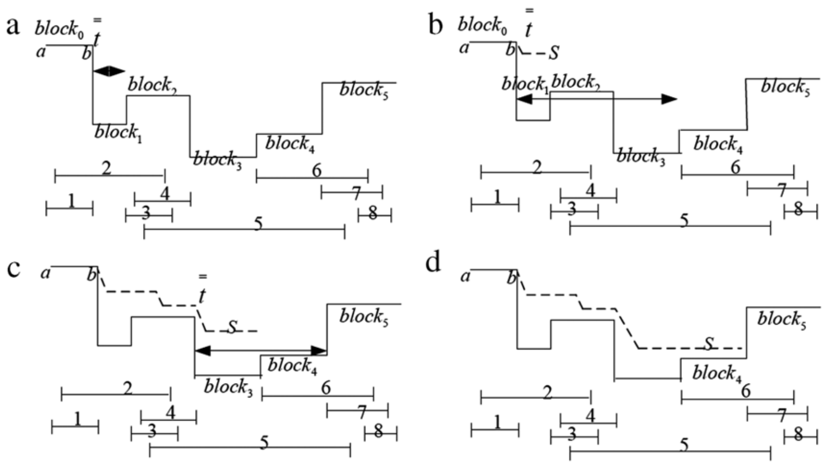

Fig. 9. An example for Algorithm 4: In the first iteration, we have $\bar{\bar{t}} = b$ and $s_{last} = s_b$. $block_1$ is the selected $block_i$. Thus interval $[t_L, t_R] = [b, R(block_1)]$ will be handled with $p = 1$ and job 2 adjusted. Since there is only one block in the computed schedule S, the finish time of the second lowest block is still $\bar{\bar{t}} = b$.
In the second iteration, s_{last} is still s_b. Job 2 is recovered. $block_3$ is the selected $block_i$. Thus interval $[t_L, t_R] = [b, R(block_3)]$ will be handled with $p = 3$ and jobs 2, 5 adjusted. For the computed schedule S, the finish time of the second lowest block is $\bar{\bar{t}}$ as shown in (c).
In the third iteration, job 5 is recovered. s_{last} is set to be $s(\bar{\bar{t}}, S)$. $block_4$ is the selected $block_i$. Thus $[t_L, t_R] = [\bar{\bar{t}}, R(block_4)]$ will be handled with jobs 5, 6 adjusted. The new computed schedule S has lowest speed less than $s(block_5)$ as shown in (d). Therefore, Algorithm 4 terminates here.

Proof. The concept of Algorithm 4 is to utilize Algorithm 1 to handle the blocks (common arrival time job instance by appropriate re-scaling) in OPT_∞. Fig. 9 shows an example. Let $[t_L, t_R]$ be the current interval that is handled and $S_{[t_L,t_R]}$ be the computed schedule in the current iteration where $s(t, S_{[t_L,t_R]})$ is non-increasing. Let $\bar{\bar{t}}$ be the finish time of the second lowest block in $S_{[t_L,t_R]}$. In the next iteration, if $s(t_R, S_{[t_L,t_R]}) > s(block_{p+1})$ where $block_{p+1}$ is the first un-handled block in OPT_∞, the algorithm will set $s_{last} = s(\bar{\bar{t}}, S_{[t_L,t_R]}) > s(t_R, S_{[t_L,t_R]}) > s(block_{p+1})$ in Step 4. The schedule computed in $[b, \bar{\bar{t}}]$ is fixed as part of $S_{[b,t_1]}$. We would like to further compute a monotone-interval after $\bar{\bar{t}}$ (it is non-increasing in this case) in the next iteration, which has starting speed s_{last} and satisfies that every down-edge-time is a tight deadline (property 3) of Fact 2. Remember that Algorithm 1 outputs a non-increasing speed curve and every down-edge-time is a tight deadline. Thus we utilize Algorithm 1 to generate such a schedule. Two properties should be guaranteed. The computed schedule by Step 10 should be not only non-increasing but also feasible for the jobs' timing constraints. To ensure that Algorithm 1 generates a non-increasing speed curve, we choose the $block_i$ after $\bar{\bar{t}}$ (Step 5) and hence Algorithm 1 will handle interval $[\bar{\bar{t}}, R(block_i)]$ in the next iteration. After the adjust procedure Steps 7–9, all jobs have $I(J) \subseteq [\bar{\bar{t}}, R(block_i)]$. All these jobs will be the input for Algorithm 1 with starting speed s_{last} at time $\bar{\bar{t}}$. Their arrival time will be adjusted to be the same (at time $\bar{\bar{t}}$) while computing. For Algorithm 1 with adjusted jobs, we note that only when there exists a time $t \in [\bar{\bar{t}}, R(block_i)]$ with $\sum_{J|I(J)\subseteq[\bar{\bar{t}},t]} C(J) > (t - \bar{\bar{t}}) \cdot s_{last}$ should it output a schedule violating the "non-increasing" requirement. According to the choice of $block_i$, OPT_∞ has speed $s(block_{p+1}) > s(block_{p+2}) > \cdots > s(block_i)$ and $s(block_i) < s(block_{i+1})$. Note that in $S_{[t_L,t_R]}$ from the last iteration, the lowest block (let it be $block_q$) may execute jobs with original deadline $d(J) > t_R$. These jobs will be recovered in Step 3. In Step 8, these jobs will be adjusted and there may be more jobs with deadline $d(J) > R(block_i)$ having their deadlines adjusted to $R(block_i)$. If originally all these jobs have $d(J) \le R(block_i)$, they are obviously executed before time $R(block_i)$ in OPT_∞ and thus $\sum_{J|I(J)\subseteq[\bar{\bar{t}},t]} C(J) \le C_{[L(block_q),R(block_q)]}(S_{[t_L,t_R]}) + C_{[L(block_{p+1}),t]}(\mathrm{OPT}_\infty) < (t - \bar{\bar{t}}) \cdot s_{last}$ for $t \in (R(block_q), R(block_i)]$ and $\sum_{J|I(J)\subseteq[\bar{\bar{t}},t]} C(J) \le C_{[L(block_q),t]}(S_{[t_L,t_R]}) < (t - \bar{\bar{t}}) \cdot s_{last}$ for $t \in [\bar{\bar{t}}, R(block_q)]$. If some adjusted jobs originally have $d(J) > R(block_i)$, then $C_{[L(block_i),R(block_i)]}(\mathrm{OPT}_\infty)$ is at least the total workload of these jobs, because jobs with $d(J) > R(block_i)$ are not executed after time $R(block_i)$ in OPT_∞ by applying Lemma 15 for $K = \infty$. Thus the adjusted

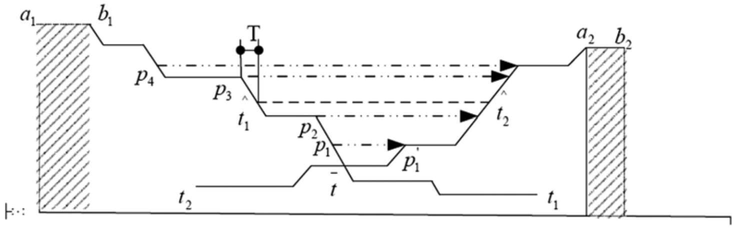

Fig. 10. The schedule OPT_K restricted in the interval between two (local-)peaks.

jobs also satisfy $\sum_{J|I(J)\subseteq[\bar{\bar{t}},t]} C(J)/(t-\bar{\bar{t}}) < s_{last}$ by similar reasons, which implies that the computed schedule in $[\bar{\bar{t}}, R(block_i)]$ is non-increasing.

Since Algorithm 1 handles the special case where every job has a common arrival time, we have to modify the arrival time of the input jobs with $I(J) \cap [t_L, t_R] \neq \phi$ and $r(J) > t_L$ to be t_L (common arrival time, Step 7). The computed schedule has a non-increasing speed curve in $[t_L, t_R]$ and each down-edge-time is a tight deadline. We need to show that the schedule is still feasible for the input before modification (where $I(J) \cap [t_L, t_R] \neq \phi$ and $r(J) > t_L$). This can be verified since the computed schedule can be considered as a process to postpone the jobs' execution time until each down-edge-time is a tight deadline as shown in Lemma 6. With the starting speed $s_{last} > s(block_{p+1})$, jobs in the computed schedule with modified input is executed with a higher speed than that of OPT_∞ in $[t_L, t_R]$. Thus jobs in the computed schedule with modified input begin to execute later than that of OPT_∞ and hence the computed schedule is still feasible for the non-modified input.

We finally examine the running time for this algorithm. Suppose that in iteration i, n_i jobs are involved and the resulting schedule computed in the current block consists of k_i blocks, then the time needed to do this computation is $O(k_i * n_i)$; furthermore, at least $k_i - 1$ jobs will have their schedule fixed and no longer be involved in the future computation. Since every job can only fix their schedule once, we conclude that the total time needed will be $O(\sum k_i * n_i)$ where $\sum(k_i - 1) \leq n$. This implies the $O(n^2)$ running time. □

Among the un-handled intervals (e.g. $[t_1, t_f]$), we define the *local-peak* to be the peak which has the local maximal intensity in OPT_∞. For example, in $[t_1, t_f]$, $[a_2, b_2]$ is the local-peak (Fig. 8). The following lemma shows that the schedules OPT_K and OPT_∞ are the same in local-peaks.

Lemma 18. *The schedule of local-peaks in* OPT_K *is the same as* OPT_∞*.*

Proof. Algorithm 4 (Fig. 9 shows an example) results in a schedule $S_{[b,t_1]}$ in monotone-interval $[b, t_1]$ by Lemma 17. It suffices to prove that the computed schedule satisfies Fact 3. Thus we only need to verify the three conditions in Fact 2.

Condition (1) holds since the algorithm terminates when $s_{last} < s(L(block_{p+1}), \mathrm{OPT}_\infty)$. It remains to show conditions (2) and (3). In a single iteration handling the block $[t_L, t_R]$, let the computed schedule be $S_{[t_L,t_R]}$. We suppose that $block_q$ is the lowest block in $S_{[t_L,t_R]}$. The way of adjusting jobs implies that all jobs with $I(J) \cap [L(block_q), R(block_q)] \neq \phi$ are executed in the lowest block $block_q$. Furthermore, the recover procedure (Steps 3–5) ensures that at each iteration (and hence the final iteration), this property holds. Therefore, condition (2) is true. For condition (3), we let $[t_0, t_1]$ be the lowest block after the final iteration and let $block_{\bar{p}}$ be the second lowest block. No jobs with $I(J) \cap [t_0, t_1] \neq \phi$ will be executed in $[b, R(block_{\bar{p}})]$ because $R(block_{\bar{p}})$ is a tight deadline and all such jobs are executed in $[t_0, t_1]$ due to the adjust procedure. On the other hand, since all involved jobs with $d(J) \subseteq (b, t_0]$ are recovered before the schedule for them in $S_{[b,t_1]}$ is computed, the final schedule $S_{[b,t_1]}$ must be feasible for them as the original input. And each down-edge-time is a tight deadline by the property of Algorithm 1. Hence, condition (3) is also true, which implies the correctness of the lemma. □

Note that there is a monotone-interval respectively before and after the computed global-peak or local-peaks. We can repeatedly call Algorithm 4 (a symmetric version of Algorithm 4 can be used to compute a monotone-interval before a "peak") until no such peak exists in the un-handled intervals. Then the schedule of the remaining intervals (all intervals between the adjacent peaks computed in Algorithm 6) can be uniquely computed as shown in Lemma 19.

Lemma 19. *The schedule of* OPT_K *in intervals between two (local-)peaks found by Algorithm* 6 *can be computed by Algorithm* 5*. Notice that in Algorithm* 5*, "down-edge-time" means the corresponding point on the speed curve at the down-edge-time.*

Proof. Fig. 10 shows an example. We need to compute the optimal schedule in the interval between two adjacent peaks $[a_1, b_1]$, $[a_2, b_2]$ that are computed in Algorithm 6. W.l.o.g assume Algorithm 6 finds $[a_1, b_1]$ first and then computes $S_{[b_1,t_1]}$ in monotone-interval $[b_1, t_1]$. After $[a_2, b_2]$ is found, Algorithm 6 will compute a monotone-interval $[t_2, a_2]$ and schedule $S_{[t_2,a_2]}$ by calling Algorithm 1 symmetrically handling the blocks in $[t_2, a_2]$. We first prove that these two curves intersect each other (notice that there are no un-handled intervals now). Otherwise without loss of generality, we assume that the speed curve of $S_{[t_2,a_2]}$ is above that of $S_{[b_1,t_1]}$. Then, we have $s(t_2, \mathrm{OPT}_\infty) > s(t_2, S_{[b_1,t_1]})$. Therefore, since t_1 is the first intersection after b_1 between $S_{[b_1,t_1]}$ and OPT_∞, we must have $t_1 < t_2$. This implies an un-handled interval $[t_1, t_2]$, a contradiction. Assume that the two speed curves intersect at time $\bar{t}$ (if they intersect at a line segment, we can pick any point on the line segment as $\bar{t}$). Note that the speed curves in $[b_1, \bar{t}]$ and $[\bar{t}, a_2]$ are respectively non-increasing and non-decreasing. The down-edge-times

Algorithm 4 Computing a Monotone-interval

Input: OPT_∞, Schedule computed by YDS.
$[a, b]$, computed peak (it can be the global-peak or local-peak)
s_b, Starting speed at time b.
Output: $S_{[b,t_1]}$, a monotone-interval starting from b and its corresponding schedule.
/* Let $[t_L, t_R]$ be the current interval being handled. Let S be the the computed schedule for current interval $[t_L, t_R]$. s_{last} denotes the lowest speed in the computed S. $block_{p+1}$ is the first un-handled block in OPT_∞.*/
1. $s_{last} = s_b$; $t_L = t_R = b$; $S_{[b,t_1]} = \phi$; $S = \phi$; $p = 0$; $\bar{\bar{t}} = b$.
2. In OPT_∞, index the blocks from the peak $[a, b]$ as $block_0, block_1, block_2, \ldots$ in the left to right order.
while $s_{last} \ge s(block_{p+1})$ **do**
  /*recover procedure*/
  if $S \neq \emptyset$ **then**
    3. For jobs that are executed in the lowest block of S, recover their arrival time/deadline to the original value.
    4. Reset s_{last} to be the speed of S in time $\bar{\bar{t}}$;
  end if
  5. Select $block_i$ to be the block after t_R in OPT_∞ with i.e. $s(block_{p+1}) > \cdots > s(block_i)$ and $s(block_i) < s(block_{i+1})$; if such a block does not exist, then let $i = p + 1$; Reset $t_R = R(block_i)$.
  6. Set $p = i$;
  /*adjust procedure*/
  for every job with $I(J) \cap [t_L, t_R] \neq \phi$ **do**
    7. Adjust $r(J)$ to be $\max\{r(J), t_L\}$;
    8. Adjust $d(J)$ to be $\min\{d(J), t_R\}$;
    9. Backup the original value of $r(J)$ and $d(J)$;
  end for
  /*handle interval $[t_L, t_R]$ in OPT_∞*/
  10. Call **Algorithm 1** to compute a schedule S for jobs involved in Steps 7–9 according to common arrival time t_L with starting speed s_{last}.
  11. If the $block_i$ found in Step 6 has speed 0, then we make S accelerate with rate $-K$ after the last time with positive speed and insert a virtual canyon at time t_R.
  12. Reset s_{last} to be the lowest positive speed in the computed S.
  if $s_{last} < s(block_{p+1})$ **then**
    13. $S_{[b,t_1]} = S_{[b,t_1]} \cup S$; Return $S_{[b,t_1]}$.
  else
    14. Let $\bar{\bar{t}}$ be the finish time of the second lowest (including the virtual canyon inserted in Step 11) block in S.
    15. $S_{[b,t_1]} = S_{[b,t_1]} \cup (S$ restricted in interval $[t_L, \bar{\bar{t}}])$.
    16. Reset $t_L = \bar{\bar{t}}$.
  end if
end while

in $S_{[b_1,\bar{t}]}$ (or $S_{[\bar{t},a_2]}$) are tight deadlines (or tight arrival times symmetrically) according to property (3) of Fact 2. Suppose that OPT_K has a separation-time $\hat{t}_1$ w.r.t $S_{[b_1,t_1]}$ and symmetrically a separation-time $\hat{t}_2$ w.r.t $S_{[t_2,b_2]}$. We have $\hat{t}_1 \le \bar{t} \le \hat{t}_2$. (E.g. if otherwise $\hat{t}_2 < \bar{t}$, then OPT_K has part of speed curve between interval $[\hat{t}_1, \hat{t}_2]$ that is below the speed curve of $s(t, S_{[b_1,t_1]})$. This implies that there exists at least a time with $s'(t, S_{[b_1,t_1]}) = -K$ where $\hat{t}_1 < t < \hat{t}_2 \le t_1$, contradicting the property (b) in Fact 2.) It is also easy to see that $t_2 \le \hat{t}_1 < \hat{t}_2 \le t_1$ (E.g. if otherwise $\hat{t}_1 < t_2$, then after separation the speed function of OPT_K must go down at t_2 by property (c) of Fact 2. However this contradicts property (b) of Fact 2.) By property (b) of Fact 2, we know that the speed curve in OPT_K should be non-decreasing in $[\hat{t}_1, t_1]$ and non-increasing in $[t_2, \hat{t}_2]$. Therefore, it should be a line with constant speed in interval $[\hat{t}_1, \hat{t}_2]$. Thus in OPT_K, interval $[\hat{t}_1, \hat{t}_2]$ with a constant speed is the lowest block between the two peaks. This block will execute all jobs with $I(J) \cap [\hat{t}_1, \hat{t}_2] \neq \phi$ by Lemma 15. The speed for this lowest block is unique. Because otherwise if OPT_K has two possible speeds s and s' with $s < s'$ for this block and the whole speed curves in $[t_s, t_f]$ both complete the workload $\sum_{1 \le i \le n} C(J_i)$, then the one with speed s will complete less workload in total than that with s', a contradiction.

We note that once the speed s in this lowest block is determined, the separation-time $\hat{t}_1$ and $\hat{t}_2$ are hence known. We can compute this speed by dividing the speed into several ranges so that speeds in the same range need to finish the same set of jobs. Then we search from the lowest speed range to the highest speed range. If the maximum speed in a range (the execution time is also longest in the range) cannot finish the jobs that should be executed by this range, then we move on to the next range until this condition does not hold. Then there is no need to move the speed into higher regions because the ability to execute jobs grows more than the the workload of the new jobs added into the region. According to the existence and uniqueness of the desired speed, we can just calculate it in the current region by solving some equation to achieve exact feasibility. The details are shown in Algorithm 5. □

Algorithm 5 Computing the Optimal Schedule between Two Adjacent Peaks

Input:
$[a_1, b_1]$, $[a_2, b_2]$, the two adjacent peaks found in Algorithm 6.
$S_{[b_1,t_1]}$, $S_{[t_2,a_2]}$, the two schedules computed by Algorithm 4. Choose one of the intersection points as $\bar{t}$.
Output: schedule of OPT_K in interval $[b_1, a_2]$.
1. For each down-edge-time p on $s(t, S_{[b_1,t_1]})$ in $[b_1, \bar{t})$ or on $s(t, S_{[t_2,a_2]})$ in $(\bar{t}, a_2]$, let the point with the same speed on the other curve be p'. If there are more than one such point, let p' be the one minimizing $|pp'|$; if there is no such point, we do not consider line segment originating from p.
2. Sort all the segments pp' by increasing order of their speed (denoted by $Speed(p)$) into $p_1p'_1, p_2p'_2, \dots, p_mp'_m$ (duplicate segments are treated as one). The end points are relabeled so that p_i is always on $s(t, S_{[b_1,t_1]})$ and p'_i is always on $s(t, S_{[t_2,a_2]})$.
3. Find augment segment for each segment $p_ip'_i$ as follows. If p_i and p'_i are both down-edge-time, then the augment segment is $p_ip'_i$ itself; if p_i is a down-edge-time and p'_i is not, then the augment segment is p_ip' where p' is the closest down-edge-time on $s(t, S_{[t_2,a_2]})$ to the right of p'_i; the symmetric case is similarly defined. We use $q_iq'_i$ to represent the augment segment of $p_ip'_i$.
for $i = 1$ to m **do**
  4. Let $C = \sum_{I(J)\cap[q_i,q'_i]\neq\emptyset} C(J)$.
  if $(\frac{C}{|p_ip'_i|} < Speed(p_i))$ **then**
    5. Let $S_{[\hat{t}_1,\hat{t}_2]}$ be the schedule that executes all jobs with $I(J) \cap [p_i, p'_i] \neq \phi$ with speed s in interval $[\hat{t}_1, \hat{t}_2]$. (The parameters can be calculated as $\hat{t}_1 = p_i + T$; $\hat{t}_2 = p'_i - T$; $s = Speed(p_i) - 2KT$; $T = \frac{Speed(p_i)+K|p_ip'_i|-\sqrt{(Speed(p_i)-K|p_ip'_i|)^2+4KC}}{4K}$)
    6. **break**;
  end if
end for
7. The optimal schedule in interval $[b_1, a_2]$ is ($S_{[b_1,\hat{t}_1]}$ restricted to $[b_1, \hat{t}_1]$) $\cup\, S_{[\hat{t}_1,\hat{t}_2]} \cup$ ($S_{[t_2,a_2]}$ restricted to $[\hat{t}_2, a_2]$).

Theorem 4. *Algorithm 6 computes* OPT_K *for aligned jobs in* $O(n^2)$ *time.*

Proof. The algorithm tries to find the global-peak and local-peak gradually, until none of them exists. The global-peak is easy to compute. Note that we need to compute OPT_∞ for searching. To find the local-peak iteratively, we compute the monotone-interval (Algorithm 4) adjacent to the peak that is found. For the sub-intervals excluding the computed monotone-interval, the maximum peak in OPT_∞ of those sub-intervals are the local-peaks. After all such peaks are found, the computed monotone-intervals from adjacent computed peaks will intersect each other and therefore Algorithm 5 can be used.

The correctness of the algorithm follows naturally from the analysis in this section. Now we focus on the running time of the algorithm. By using Theorem 3, OPT_∞ can be computed in $O(n^2)$ time. Suppose that there are n_i jobs to be handled by one call of Algorithm 4. The time for this call will be $O(n_i^2)$ as shown in Lemma 17. On the other hand, every job will only be involved in two such calls (backward and forward). Therefore, the total running time of executing Algorithm 4 in Algorithm 6 is $O(n^2)$ because $\sum n_i \leq 2n$. Next, we analyze the execution of Algorithm 5 in Algorithm 6. Notice that the jobs and down-edge-times involved in the calls of Algorithm 5 are disjoint. So we can first partition the jobs in $O(n)$ time into different groups according to which call it is involved in. Suppose that m_i down-edge-times and k_i jobs are involved in a certain call of Algorithm 5. Then the running time of this call will be $O(m_i \log m_i + k_i * m_i)$. Since $\sum m_i < 2n$ and $\sum k_i < n$, we can see that the total time for executing Algorithm 5 is also $O(n^2)$. Thus we can compute OPT_K in $O(n^2)$ time. □

Algorithm 6 Computing the Optimal Schedule for Aligned Jobs

Input: Aligned job set $\mathcal{J}$
Output: OPT_K
1. Compute OPT_∞.
2. Let the maximum intensity block in OPT_∞ be the global-peak in OPT_K .
3. Index the global-peak as an un-handled peak.
while there is a peak [L,R] un-handled **do**
  4. Let OPT_K execute jobs the same way as OPT_∞ in $[L, R]$.
  5. Call **Algorithm** 4 to compute the monotone-interval starting from R (and also symmetrically a monotone-interval ending at L).
  6. If there are local-peaks in OPT_∞ in the un-handled interval on either side of the monotone-intervals, then index the local-peaks as un-handled peaks.
end while
7. Compute the OPT_K for all the intervals between adjacent peaks found in the previous while loop using **Algorithm** 5.

5. Conclusion

In this paper, we study the energy-efficient dynamic voltage scaling problem and mainly focus on the pessimistic accelerate model and aligned jobs. All jobs are required to be completed before deadlines and the objective is to minimize the energy. We start by examining the properties for the special case where jobs are released at the same time. We show that the optimal schedule can be computed in $O(n^2)$. Based on this result, we study the general aligned jobs. The algorithm for jobs with a common arrival time is adopted as an elementary procedure to compute the optimal schedule for general aligned jobs. By repeatedly computing heuristic schedules that are non-increasing, we fix some peaks of the optimal schedule first. This makes the optimal schedule in the remaining interval easier to compute. The complexity of the algorithm is $O(n^2)$ since we improve the computation of the optimal schedule for aligned jobs in the ideal model to $O(n^2)$. The computation of optimal schedules for general job sets under the pessimistic accelerate model remains as an open problem.

Acknowledgements

This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117408], National Natural Science Foundation of China (grant no. 60775037, 61073110) and the Research Fund for the Doctoral Program of Higher Eduction of China (20093402110017).

References

[1] S. Albers, H. Fujiwara, Energy-efficient algorithms for flow time minimization, in: Proceeding of Symposium on Theoretical Aspects of Computer Science, STACS, vol. 3884, 2006, pp. 621–633.
[2] N. Bansal, H.L. Chan, T.W. Lam, L.K. Lee, Scheduling for bounded speed processors, in: International Colloquium on Automata, Languages and Programming, ICALP, 2008, pp. 409–420.
[3] N. Bansal, H.L. Chan, K. Pruhs, Speed scaling with an arbitrary power function, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete algorithms, SODA, 2009, pp. 693–701.
[4] N. Bansal, H.L. Chan, T.W. Lam, L.K. Lee, Scheduling for speed bounded processors, in: Proceedings of the 35th International Symposium on Automata, Languages and Programming, 2008, pp. 409–420.
[5] N. Bansal, K. Pruhs, C. Stein, Speed scaling for weighted flow time, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, 2007, pp. 805–813.
[6] T.D. Burd, T.A. Pering, R.W. Brodersen, Design issues for dynamic voltage scaling, in: Proceedings of the 2000 International Symposium on Low Power Electronics and Design, pp. 9–14.
[7] H.L. Chan, J. Edmonds, T.W. Lam, L.K. Lee, A. Marchetti-Spaccamela, K. Pruhs, Nonclairvoyant speed scaling for fow and energy, in: STACS, 2009, pp. 409–420.
[8] H.L. Chan, W.T. Chan, T.W. Lam, L.K. Lee, K.S. Mak, P.W.H. Wong, Energy efficient online deadline scheduling, in: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 795–804.
[9] W.T. Chan, T.W. Lam, K.S. Mak, P.W.H. Wong, Online deadline scheduling with bounded energy efficiency, in: Proceedings of the 4th Annual Conference on Theory and Applications of Models of Computation, 2007, pp. 416–427.
[10] I. Hong, G. Qu, M. Potkonjak, M.B. Srivastava, Synthesis techniques for low-power hard real-time systems on variable voltage processors, in: Proceedings of the IEEE Real-Time Systems Symposium, RTSS, December 02–04, 1998, pp. 178–187.
[11] S. Irani, K. Pruhs, Algorithmic problems in power management, SIGACT News 36 (2) (2005) 63–76.
[12] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processors, in: Proceedings of International Symposium on Low Power Electronics and Design, 1998, pp. 197–202.
[13] W. Kwon, T. Kim, Optimal voltage allocation techniques for dynamically variable voltage processors, in: Proceedings of the 40th Conference on Design Automation, 2003, pp. 125–130.
[14] T.W. Lam, L.K. Lee, I.K.K. To, P.W.H. Wong, Speed scaling functions for fow time scheduling based on active job count, in: ESA, Karlsruhe, Germany, 2008, pp. 647–659.
[15] T.W. Lam, L.K. Lee, I.K.K. To, P.W.H. Wong, Energy efficient deadline scheduling in two processor systems, in: Proceedings of the 18th International Symposium on Algorithm and Computation, 2007, pp. 476–487.
[16] M. Li, F.F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, SIAM Journal on Computing 35 (2005) 658–671.
[17] M. Li, A.C. Yao, F.F. Yao, Discrete and continuous min-energy schedules for variable voltage processors, Proceedings of the National Academy of Sciences of the United States of America 103 (2006) 3983–3987.
[18] K. Pruhs, P. Uthaisombut, G. Woeginger, Getting the best response for your erg, in: Scandanavian Workshop on Algorithms and Theory, 2004, pp. 14–25.
[19] W. Wu, M. Li, E. Chen, Min-Energy Scheduling for Aligned Jobs in Accelerate Model, in: ISAAC, 2009, pp. 462–472.
[20] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: Proc. IEEE Symp. Foundations of Computer Science, FOCS, 1995, pp. 374–382.
[21] L. Yuan, G. Qu, Analysis of energy reduction on dynamic voltage scaling-enabled systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (12) (2005) 1827–1837.