Optimal Nonpreemptive Scheduling in a Smart Grid Model

Fu-Hong Liu[1], Hsiang-Hsuan Liu[2], and Prudence W. H. Wong[3]

1 **Department of Computer Science, National Tsing Hua University, Taiwan**
fhliu,hhliu@cs.nthu.edu.tw
2 **Department of Computer Science, National Tsing Hua University, Taiwan; and Department of Computer Science, University of Liverpool, UK**
hhliu,pwong@liverpool.ac.uk
3 **Department of Computer Science, University of Liverpool, UK**
pwong@liverpool.ac.uk

Abstract

We study a scheduling problem arising in demand response management in smart grid. Consumers send in power requests with a flexible feasible time interval during which their requests can be served. The grid controller, upon receiving power requests, schedules each request within the specified interval. The electricity cost is measured by a convex function of the load in each timeslot. The objective is to schedule all requests with the minimum total electricity cost. Previous work has studied cases where jobs have unit power requirement and unit duration. We extend the study to arbitrary power requirement and duration, which has been shown to be NP-hard. We give the first online algorithm for the general problem, and prove that the worst case competitive ratio is asymptotically optimal. We also prove that the problem is fixed parameter tractable. Due to space limit, the missing proofs are presented in the full paper.

1998 ACM Subject Classification F. Theory of Computationm, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Scheduling, Smart Grid, Convex function cost, Fixed parameter tractable, Online algorithms, Non-preemptive

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.53

1 Introduction

We study a scheduling problem arising in "demand response management" in a smart grid [15, 20, 28]. The electrical smart grid is one of the major challenges in the 21st century [25]. The smart grid [22] is a power grid system that makes power generation, distribution and consumption more efficient through information and communication technologies. Peak demand hours happen only for a short duration, yet makes existing electrical grid less efficient. It has been noted in [7] that in the US power grid, 10% of all generation assets and 25% of distribution infrastructure are required for less than 400 hours per year, roughly 5% of the time [25]. *Demand response management*1 attempts to overcome this problem by shifting users' demand to off-peak hours in order to reduce peak load.

It is demonstrated in [20] that demand response is of remarkable advantage to consumers, utilities, and society. Effective demand load management brings down the cost of operating the grid, energy generation and distribution [19]. It is not only advantageous to the supplier but also to the consumers as well. It is common that electricity supplier charges according to the generation cost. Therefore, it is to the consumers' advantage to reduce electricity consumption at high price and hence reduce the electricity bill [24].

The operator and consumers communicate through smart metering devices [22]. A consumer sends in a power request with the power requirement, required duration of service, and the time interval that this request can be served (giving some flexibility). For example, a consumer may want the dishwasher to operate for one hour during the periods from 8am to 11am. The grid operator upon receiving requests has to schedule them in their respective time intervals using the minimum energy cost. The *load* of the grid at each timeslot is the sum of the power requirements of all requests allocated to that timeslot. The *electricity cost* is modeled by a convex function on the load: we consider the cost to be the α-th power of the load, where $\alpha > 1$ is some constant. Typically, α is small, e.g., $\alpha = 2$ [10].

Previous work. Koutsopoulos and Tassiulas [17] have formulated a similar problem to our problem where the cost function is piecewise linear. They show that the problem is NP-hard, and their proof can be adapted to show the NP-hardness of the general problem studied in this paper [6]. Burcea et al. [6] gave polynomial time optimal algorithms for the case of unit height (cf. unit power requirement) and unit width (cf. duration of request). Feng et al. [12] have claimed that a simple greedy algorithm is 2-competitive for the unit case and $\alpha = 2$. However, in our full paper [18], we show a counter example that the greedy algorithm is at least 3-competitive. This implies that it is still an open question to derive online algorithms for the problem. Salinas et al. [24] considered a multi-objective problem to minimize energy consumption cost and maximize some utility. A closely related problem is to manage the load by changing the price of electricity over time [21, 11]. Reviews of smart grid can be found in [15, 20, 28]. The combinatorial problem in this paper has analogy to the load balancing problem [3] and machine minimization problem [8, 9, 23] but the main differences are the objective being maximum load and jobs are unit height [8, 9, 23]. Minimizing maximum load has also been looked at in the context of smart grid [1, 27], some of which further consider allowing reshaping of the jobs [1]. As to be discussed in the full paper, our problem is more difficult than minimizing the maximum load. Our problem also has resemblance to the dynamic speed scaling problem [2, 26, 5] and our algorithm has employed some techniques there.

Our contribution. We propose the first online algorithm for the general Grid problem with competitive ratio polylogarithm in the max-min ratio of the duration of jobs (Section 4); and show that the competitive ratio is asymptotically optimal. The algorithm is based on an $O(1)$-competitive online algorithm for jobs with uniform duration (Section 3). We also propose $O(1)$-competitive online algorithms for jobs with uniform power requirement and agreeable deadlines (Section 5). Table 1 gives a summary. In addition, we show that the Grid problem is fixed parameter tractable by proposing the first fixed parameter exact algorithms for the problem; and derive lower bounds on the running time (Section 6). Interestingly, both our online algorithm and exact algorithms depend on the variation of the job widths but not the variation of the job heights.

Our online algorithms are based on identifying a relationship with the dynamic speed (voltage) scaling (DVS) problem [26]. The main challenge, even when jobs have uniform width or uniform height, is that in time intervals where the "workload" is low, the optimal DVS schedule may have much lower cost than the optimal Grid schedule because jobs in DVS schedules can effectively be stretched as flat as possible while jobs in Grid schedules have rigid duration and cannot be stretched. In such case, it is insufficient to simply compare with the optimal DVS schedule. Therefore, our analysis is divided into two parts: for high workload intervals, we compare with the optimal DVS schedule; and for low workload intervals, we

Table 1 Summary of online algorithms for different input instances.

Width	Height	Competitive Ratio
Uniform	Arbitrary	$O(1)$-competitive [Section 3]
Arbitrary	Arbitrary	$\Theta(\log^{\alpha}(\frac{w_{\max}}{w_{\min}}))$-competitive [Section 4]
Arbitrary	Uniform	$O(1)$-competitive [input with agreeable deadlines] [Section 5]

directly compare with the optimal Grid schedule via a lower bound on the total workload over these intervals (Lemmas 2 and 11). For jobs with arbitrary width, we adopt the natural approach of classification based on job width. We then align the "feasible interval" of each job in a more uniform way so that we can use the results on uniform width (Lemma 6).

In designing exact algorithms we use interval graphs to represent the jobs and the important notion maximal cliques to partition the time horizon into disjoint windows. Such partition usually leads to optimal substructures; however, non-preemption makes it trickier and requires a smart way to handle jobs spanning multiple windows. We describe how to handle such jobs without adding a lot of overhead. We remark that our approach can solve other problems like minimizing peak load in Grid and the machine minimization problem.

2 Definitions and preliminaries

The input. The time is labeled from 0 to τ and we consider events (release time, deadlines) occurring at integral time. We call the unit time $[t, t+1)$ *timeslot* t. We denote by $\mathcal{J}$ a set of input jobs in which each job J comes with *release time* $r(J)$, *deadline* $d(J)$, *width* $w(J)$ representing the duration required by J, and *height* $h(J)$ representing the power required by J. We assume $r(J)$, $d(J)$, $w(J)$, and $h(J)$ are integers. The *feasible interval*, denoted by $I(J)$, is defined as the interval $[r(J), d(J))$ and we say that J is *available* during $I(J)$.

In Section 4, we consider an algorithm that classifies jobs according to their widths. To ease discussion, we let $w_{\max}$ and $w_{\min}$ be the maximum and minimum width over all jobs, respectively. We further define the max-min ratio of width, denoted by K, to be $K = \frac{w_{\max}}{w_{\min}}$.

Without loss of generality, we assume that $w_{\min} = 1$. We say that a job J is in *class* C_p if and only if $2^{p-1} < w(J) \leq 2^p$ for any $0 \leq p \leq \lceil \log K \rceil$.

Feasible schedule. A *feasible* schedule S assigns for each job J a *start time* $st(S, J) \in \mathbb{Z}$ meaning that J runs during $[st(S, J), et(S, J))$, where the *end time* $et(S, J) = st(S, J) + w(J)$. Note that this means preemption is not allowed. The *load* of S at time t, denoted by $\ell(S, t)$ is the sum of the height (power request) of all jobs running at t, i.e., $\ell(S, t) = \sum_{J:t\in[st(S,J),et(S,J))} h(J)$. We drop S and use $\ell(t)$ when the context is clear. We use $\mathcal{A}(\mathcal{J})$ to denote the schedule of an algorithm $\mathcal{A}$ on $\mathcal{J}$. We denote by $\mathcal{O}$ the optimal algorithm.

The cost of a schedule S is the sum of the α-th power of the load over all time, for a constant $\alpha > 1$, i.e., $\text{cost}(S) = \sum_t (\ell(S, t))^{\alpha}$. For a set of timeslots $\mathcal{I}$ (not necessarily contiguous), we denote by $\text{cost}(S, \mathcal{I}) = \sum_{t\in\mathcal{I}} (\ell(S, t))^{\alpha}$. The objective is to find a feasible schedule with minimum cost. We call this the Grid problem.

Online algorithms. We consider online algorithms, where the job information is only revealed at the time the job is released; the algorithm has to decide which jobs to run at the current time without future information and decisions made cannot be changed later. Let $\mathcal{A}$ be an online algorithm. We say that $\mathcal{A}$ is c-competitive if for all input job sets $\mathcal{J}$, we have

$\text{cost}(\mathcal{A}(\mathcal{J})) \leq c \cdot \text{cost}(\mathcal{O}(\mathcal{J}))$. In particular, we consider non-preemptive algorithms where a job cannot be preempted to resume/restart later.

Special input instances. A job J is said to be unit-width (resp. unit-height) if $w(J) = 1$ (resp. $h(J) = 1$). A job set is said to be uniform-width (resp. uniform-height) if the width (resp. height) of all jobs are the same. A job set is said to have *agreeable deadlines* if for any two jobs J_1 and J_2, $r(J_1) \leq r(J_2)$ implies $d(J_1) \leq d(J_2)$.

Relating to the speed scaling problem. The Grid problem resembles the dynamic speed scaling (DVS) problem [26] and we are going to refer to three DVS algorithms, namely, the optimal $\mathcal{YDS}$ algorithm, the online algorithms called $\mathcal{BKP}$ [4] and $\mathcal{AVR}$ [26], which are $8e^\alpha$- and $(2\alpha)^\alpha/2$-competitive, respectively. In the DVS problem, jobs come with release time $r(J)$, deadline $d(J)$, and a work requirement $p(J)$. A processor can run at speed $s \in [0, \infty)$ and consumes energy in a rate of s^α, for some $\alpha > 1$. The objective is to complete all jobs by their deadlines using the minimum total energy. The main differences of the DVS problem to the Grid problem include (i) jobs in DVS can be preempted while preemption is not allowed in the Grid problem; (ii) as processor speed in DVS can scale, a job can be executed for varying time duration as long as the total work is completed while in Grid a job must be executed for a fixed duration given as input; (iii) the work requirement $p(J)$ of a job J in DVS can be seen as $w(J) \times h(J)$ for the corresponding job in Grid.

Let $\mathcal{O}_D$ and $\mathcal{O}_G$ be the optimal algorithm for the DVS and Grid problems, respectively. Given a job set $\mathcal{J}_G$ for Grid, we can convert it into a job set $\mathcal{J}_D$ for DVS by keeping the release time and deadline for each job and setting the work requirement of a job in $\mathcal{J}_D$ to the product of the width and height of the corresponding job in $\mathcal{J}_G$.

▶ **Observation 1.** *Given a schedule S_G for $\mathcal{J}_G$, we can convert S_G into a feasible schedule S_D for $\mathcal{J}_D$ such that $\text{cost}(S_D(\mathcal{J}_D)) \leq \text{cost}(S_G(\mathcal{J}_G))$; implying $\text{cost}(\mathcal{O}_D(\mathcal{J}_D)) \leq \text{cost}(\mathcal{O}_G(\mathcal{J}_G))$.*

Note that it is not always possible to convert a feasible DVS schedule to a feasible Grid schedule. The observation does not immediately solve the Grid problem but as to be shown it provides a way to analyze algorithms for Grid.

3 Online algorithm for uniform width jobs

To handle jobs of arbitrary width and height, we first study the case when jobs have uniform width (all jobs have the same width $w \geq 1$). The proposed algorithm $\mathcal{UV}$ (Section 3.2) is based on a further restricted case of unit width, i.e., $w = 1$ (Section 3.1).

3.1 Unit width and arbitrary height

We present an online algorithm $\mathcal{V}$ which makes reference to an arbitrary feasible online algorithm for the DVS problem, denoted by $\mathcal{R}$. We require that the speed of $\mathcal{R}$ remains the same during any integral timeslot. When jobs have integral release times and deadlines, many known DVS algorithms satisfy this criteria, including $\mathcal{YDS}$, $\mathcal{BKP}$, and $\mathcal{AVR}$.

Recall in Section 2 how an input for the Grid problem is converted to an input for the DVS problem. We simulate a copy of $\mathcal{R}$ on the converted input and denote the speed used by $\mathcal{R}$ at t as $\ell(\mathcal{R}, t)$. Our algorithm makes reference to $\ell(\mathcal{R}, t)$ but not the jobs run by $\mathcal{R}$ at t.

Algorithm $\mathcal{V}$. For each timeslot t, schedule jobs to start at t until $\ell(\mathcal{V}, t)$ is at least $\ell(\mathcal{R}, t)$ or until all available jobs have been scheduled. Jobs are chosen in an EDF manner.

Analysis. Since $\mathcal{V}$ makes decision at integral time and jobs have unit width, each job is completed before any further scheduling decision is made. In other words, $\mathcal{V}$ is non-preemptive. To analyze the performance of $\mathcal{V}$, we note that $\mathcal{V}$ gives a feasible schedule (Lemma 2 (i)), and then analyze its competitive ratio (Theorem 3).

Let $h_{\max}(\mathcal{V}, t)$ be the maximum height of jobs scheduled at t by $\mathcal{V}$. We first classify each timeslot t into two types: (i) $h_{\max}(\mathcal{V}, t) < \ell(\mathcal{R}, t)$, and (ii) $h_{\max}(\mathcal{V}, t) \geq \ell(\mathcal{R}, t)$. We denote by $\mathcal{I}_1$ and $\mathcal{I}_2$ the union of all timeslots of Type (i) and (ii), respectively. Notice that $\mathcal{I}_1$ and $\mathcal{I}_2$ can be empty and the union of $\mathcal{I}_1$ and $\mathcal{I}_2$ covers the entire time line. Lemma 2 (ii) and (iii) bound the cost of $\mathcal{V}$ in each type of timeslots. By Lemma 2 and Observation 1, we obtain the competitive ratio of $\mathcal{V}$ in Theorem 3.

▶ **Lemma 2.** *(i) $\mathcal{V}$ gives a feasible schedule; (ii) $cost(\mathcal{V}, \mathcal{I}_1) \leq 2^\alpha \cdot cost(\mathcal{R})$; (iii) $cost(\mathcal{V}, \mathcal{I}_2) \leq 2^\alpha \cdot cost(\mathcal{O})$; and (iv) $cost(\mathcal{V}) = cost(\mathcal{V}, \mathcal{I}_1) + cost(\mathcal{V}, \mathcal{I}_2)$.*

▶ **Theorem 3.** *Algorithm $\mathcal{V}$ is $2^\alpha \cdot (R+1)$-competitive, where R is the competitive ratio of the reference DVS algorithm $\mathcal{R}$. $\mathcal{V}$ is $2^\alpha \cdot (8 \cdot e^\alpha + 1)$-competitive and $2^\alpha \cdot 2$-approximate when the algorithm $\mathcal{BKP}$ and $\mathcal{YDS}$ are referenced, respectively.*

3.2 Uniform width and arbitrary height

The idea of handling uniform width jobs is to treat them as if they were unit width, however, this would mean that jobs may have non-integral release times or deadlines. To remedy this, we define a procedure AlignFI to align the feasible intervals (precisely, release times and deadlines) to the new time unit.

Let $\mathcal{J}$ be a set of uniform width jobs each of width w. A job J is said to be *tight* if $|I(J)| \leq 2w$; otherwise, it is *loose*. Let $\mathcal{J}_{\mathrm{T}}$ and $\mathcal{J}_{\mathrm{L}}$ be the disjoint subsets of tight and loose jobs of $\mathcal{J}$, respectively. We design different strategies for tight and loose jobs. We observe that tight jobs can be handled easily by starting them at their release times. For any loose job, we modify it via Procedure AlignFI such that its release time and deadline is an integral multiple of w. With this alternation, we can treat the jobs as unit width and make scheduling decisions at time multiple of w.

Procedure AlignFI**.** Given a loose job set $\mathcal{J}_{\mathrm{L}}$ in which $w(J) = w$ and $|I(J)| > 2 \cdot w$ $\forall J \in \mathcal{J}_{\mathrm{L}}$. We define the procedure AlignFI to transform each loose job $J \in \mathcal{J}_{\mathrm{L}}$ into a job J' with release time and deadline "aligned" as follows: $r(J') \leftarrow \min_{i \geq 0}\{i \cdot w \mid i \cdot w \geq r(J)\}$; and $d(J') \leftarrow \max_{i \geq 0}\{i \cdot w \mid i \cdot w \leq d(J)\}$. We denote the resulting job set by $\mathcal{J}'$.

After AlignFI, the release time and deadline of each loose job are aligned to timeslot $i_1 \cdot w$ and $i_2 \cdot w$ for some integers $i_1 < i_2$. Hence, the job set $\mathcal{J}'$ can be treated as job set with unit width, where each unit has duration w instead of 1. We further observe that a feasible schedule of $\mathcal{J}'$ is also a feasible schedule of $\mathcal{J}_{\mathrm{L}}$.

Online algorithm $\mathcal{UV}$. The algorithm takes a job set $\mathcal{J}$ with uniform width w as input and schedules the jobs in $\mathcal{J}$ as follows. Let $\mathcal{J}_{\mathrm{T}}$ be the set of tight jobs in $\mathcal{J}$ and $\mathcal{J}_{\mathrm{L}}$ be the set of loose jobs in $\mathcal{J}$. Note that the decisions of $\mathcal{UV}$ can be made online.

1. For any tight job $J \in \mathcal{J}_{\mathrm{T}}$, schedule J to start at $r(J)$.
2. Loose jobs in $\mathcal{J}_{\mathrm{L}}$ are converted to $\mathcal{J}'$ by AlignFI. For $\mathcal{J}'$, we run Algorithm $\mathcal{V}$ in Section 3.1 with $\mathcal{BKP}$. Jobs are chosen in an earliest deadline first (EDF) manner.

The "inflexibility" of tight jobs guarantees that the simple strategy (Step 1 of $\mathcal{UV}$) gives good enough ratio. That is, since tight jobs have short feasible intervals, even the optimal

schedule has to have high cost if our strategy has high load. On the other hand, AlignFI only increases the competitive factor of loose jobs by a constant factor because the feasible interval duration is only decreased by at most two thirds. The overall performance is:

▶ **Theorem 4.** *cost*$(\mathcal{UV}(\mathcal{J})) \leq 12^{\alpha} \cdot (8e^{\alpha}+1) \cdot$ *cost*$(\mathcal{O}(\mathcal{J}))$.

4 Online algorithm for the general case

In this section, we present an algorithm $\mathcal{G}$ for jobs with arbitrary width and height. We first transform job set $\mathcal{J}$ to a "nice" job set $\mathcal{J}^*$ (to be defined) and show that such a transformation only increases the cost modestly. Furthermore, we show that for any nice job set $\mathcal{J}^*$, we can bound $\text{cost}(\mathcal{G}(\mathcal{J}^*))$ by $\text{cost}(\mathcal{O}(\mathcal{J}^*))$ and in turn by $\text{cost}(\mathcal{O}(\mathcal{J}))$. Then we can establish the competitive ratio of $\mathcal{G}$.

4.1 Upper bound

A job J is said to be a *nice job* if $w(J) = 2^p$, for some non-negative integer p and a job set $\mathcal{J}^*$ is said to be a *nice job set* if all its jobs are nice. Note that the nice job J is in class C_p.

Procedure Convert**.** Given a job set $\mathcal{J}$, we define the procedure Convert to transform each job $J \in \mathcal{J}$ into a nice job J^*. We denote the resulting nice job set by $\mathcal{J}^*$ and the subset in C_p by $\mathcal{J}_p^*$. Suppose J is in class C_p. We modify it as follows: $w(J^*) \leftarrow 2^p$; $r(J^*) \leftarrow r(J)$; and $d(J^*) \leftarrow r(J^*) + \max\{d(J) - r(J), 2^p\}$.

We then define two procedures that transform schedules related to nice job sets.

Transformation RelaxSch**.** RelaxSch transforms a schedule S for a job set $\mathcal{J}$ into a schedule S^* for the corresponding nice job set $\mathcal{J}^*$ by moving the start and end time of every job J such that $st(S^*, J^*) = \min\{d(J^*) - w(J^*), st(S, J)\}$; and $et(S^*, J^*) = st(S^*, J^*) + w(J^*)$.

▶ **Observation 5.** *Consider any schedule S for $\mathcal{J}$ and the schedule S^* constructed by* RelaxSch *for the corresponding $\mathcal{J}^*$. We have $[st(S^*, J^*), et(S^*, J^*)] \subseteq [r(J^*), d(J^*)]$; in other words, S^* is a feasible schedule for $\mathcal{J}^*$.*

Transformation ShrinkSch**.** ShrinkSch converts a schedule S^* for a nice job set $\mathcal{J}^*$ to a schedule S for the corresponding $\mathcal{J}$. We set $st(S, J) \leftarrow st(S^*, J^*)$; and $et(S, J) \leftarrow st(S, J) + w(J)$, therefore, $et(S, J) \leq et(S^*, J^*)$. Let S_p^* be the partial schedule for class C_p.

Online algorithm $\mathcal{G}$. When a job J is released, it is converted to J^* by Convert and classified into one of the classes C_p. Jobs in the same class after Convert (being a uniform-width job set) are scheduled by $\mathcal{UV}$ independently of other classes. We then modify the execution time of J^* in $\mathcal{UV}$ to the execution time of J in $\mathcal{G}$ by ShrinkSch. Note that all these procedures can be done in an online fashion.

Using the results in Sections 3 and 4.1, we can compare the cost of $\mathcal{G}(\mathcal{J}_p)$ with $\mathcal{O}(\mathcal{J}_p^*)$ and $\mathcal{O}(\mathcal{J}_p^*)$ with $\mathcal{O}(\mathcal{J})$ for each class C_p. The most tricky part is in Lemma 6 (i). Intuitively, one can show that for each class, the load of $\mathcal{O}(\mathcal{J}_p^*)$ at any time is bounded above by that of $\mathcal{O}(\mathcal{J}_p)$ at the current time and $2^{p-1}-1$ timeslots before and after the current time. This allows us to bound $\text{cost}(\mathcal{O}(\mathcal{J}_p^*))$ by 3^{α} times of $\text{cost}(\mathcal{O}(\mathcal{J}_p))$ (Lemma 6 (ii)).

▶ **Lemma 6.** *Consider any class C_p. (i) At any time t, $\ell(S_p^*, t) \leq \ell(S_p, t) + \ell(S_p, t-(2^{p-1}-1)) + \ell(S_p, t+(2^{p-1}-1))$. (ii) cost$(\mathcal{O}(\mathcal{J}_p^*)) \leq 3^{\alpha} \cdot$ cost$(\mathcal{O}(\mathcal{J}_p))$; (iii) cost$(\mathcal{O}(\mathcal{J}_p)) \leq$ cost$(\mathcal{O}(\mathcal{J}))$.*

▶ **Theorem 7.** *For any job set $\mathcal{J}$, cost$(\mathcal{G}(\mathcal{J})) \leq (36\lceil \log \frac{w_{\max}}{w_{\min}} \rceil)^{\alpha} \cdot (8e^{\alpha}+1) \cdot$ cost$(\mathcal{O}(\mathcal{J}))$.*

4.2 Lower bound

We show a lower bound of competitive ratio for Grid problem with unit height and arbitrary width by designing an adversary for the problem. This lower bound is immediately a lower bound for the general case of Grid problem. Note that the lower bound in [23] may look similar but it does not work in our case since that adversary only guarantees a high peak load at a particular timeslot which is not sufficient for the total cost measurement.

Adversary Λ and job instance $\mathcal{J}$. Given an online algorithm $\mathcal{A}$, a constant $\alpha > 1$ and a large number x, adversary Λ outputs a set of jobs $\mathcal{J}$ with $\lfloor\alpha\rfloor + 1$ jobs. Let J_i be the ith job of $\mathcal{J}$. The adversary first computes a width for each job before running algorithm $\mathcal{A}$. It sets $w(J_{\lfloor\alpha\rfloor}) = x$, $w(J_{\lfloor\alpha\rfloor+1}) = x - 1$, and $w(J_i) = 3w(J_{i+1}) + 1$ for $1 \le i \le \lfloor\alpha\rfloor - 1$. Then adversary Λ computes a release time and deadline for each job through a interaction with algorithm $\mathcal{A}$. For the first job J_1, adversary Λ chooses any release time and deadline such that $d(J_1) - r(J_1) \ge 3w(J_1)$. For the ith job $J_i \in \mathcal{J}$ for $2 \le i \le \lfloor\alpha\rfloor + 1$, adversary Λ sets $r(J_i) = st(\mathcal{A}, J_{i-1}) + 1$ and $d(J_i) = et(\mathcal{A}, J_{i-1})$. This limits algorithm $\mathcal{A}$ to fewer choices of start times for scheduling a new job. A job can only be scheduled in the execution interval of the previous job by algorithm $\mathcal{A}$. On the other hand, no two jobs have the same release time. Algorithm $\mathcal{A}$ shall schedule the jobs accordingly from J_1 to $J_{\lfloor\alpha\rfloor+1}$ and one job at a time.

▶ **Lemma 8.** *$cost(\mathcal{O}(\mathcal{J})) \le x \cdot 3^{\lfloor\alpha\rfloor}$.*

▶ **Theorem 9.** *For any deterministic online algorithm $\mathcal{A}$ for Grid problem with unit height and arbitrary width, adversary Λ constructs an instance $\mathcal{J}$ such that $\frac{cost(\mathcal{A}(\mathcal{J}))}{cost(\mathcal{O}(\mathcal{J}))} \ge \left(\frac{1}{3}\log\frac{w_{\max}}{w_{\min}}\right)^{\alpha}$.*

5 Online algorithm for uniform height jobs

We consider (i) jobs with uniform-height h and unit-width and (ii) jobs with uniform-height h, arbitrary width and agreeable deadlines. To ease the discussion, we define the *density* of J, denoted by $\mathrm{den}(J)$, to be $\frac{h(J)*w(J)}{d(J)-r(J)}$. Roughly speaking, the density signifies the average load required by the job over its feasible interval. We then define the "average" load at any time t as $\mathrm{avg}(t) = \sum_{J:t\in I(J)} \mathrm{den}(J)$.

Basically, at any time t, $\mathcal{AVR}$ runs the processor at a speed which is the sum of the densities of jobs that are available at t. By Observation 1, we have the following corollary.

▶ **Corollary 10.** *For any input $\mathcal{J}_G$ and the corresponding input $\mathcal{J}_D$, $cost(\mathcal{AVR}(\mathcal{J}_D)) \le \frac{(2\alpha)^{\alpha}}{2} \cdot cost(\mathcal{O}_G)$.*

Main Ideas

The main idea is to make reference to the online algorithm $\mathcal{AVR}$ and consider two types of intervals, $\mathcal{I}_{>h}$ where the average load is higher than h and $\mathcal{I}_{\le h}$ where the average load is at most h. For the former, we show that we can base on the competitive ratio of $\mathcal{AVR}$ directly; for the latter, our load could be much higher than that of $\mathcal{AVR}$ and in such case, we compare directly to the optimal algorithm. Combining the two cases, we have Lemma 11, which holds for any job set. We show how we can use this lemma to obtain algorithms for the special cases. Notice that the number $\lceil\frac{\mathrm{avg}(t)}{h}\rceil$ is the minimum number of jobs needed to make the load at t at least $\mathrm{avg}(t)$.

▶ **Lemma 11.** *Suppose we have an algorithm $\mathcal{A}$ for any job set $\mathcal{J}$ such that (i) $\ell(\mathcal{A}, t) \le c \cdot \lceil avg(t)\rceil$ for all $t \in \mathcal{I}_{>h}$, and (ii) $\ell(\mathcal{A}, t) \le c'$ for all $t \in \mathcal{I}_{\le h}$. Then we have $cost(\mathcal{A}(\mathcal{J})) \le (\frac{(4c\alpha)^{\alpha}}{2} + c'^{\alpha}) \cdot cost(\mathcal{O}(\mathcal{J}))$.*

Uniform-height and unit-width

We consider jobs with uniform-height and unit-width, i.e., $w(J) = 1$ and $h(J) = h\ \forall J$. Note that such case is a subcase discussed in Section 3.1. Here we illustrate a different approach using the ideas above and describe the algorithm $\mathcal{UU}$ for this case. The competitive ratio of $\mathcal{UU}$ is better than that of Algorithm $\mathcal{V}$ in Section 3.1 when $\alpha < 3.22$.

Algorithm $\mathcal{UU}$. At any time t, choose $\lceil \frac{\text{avg}(t)}{h} \rceil$ jobs according to the EDF rule and schedule them to start at t. If there are fewer jobs available, schedule all available jobs.

▶ **Theorem 12.** *Algorithm $\mathcal{UU}$ gives feasible schedules and it is $(\frac{(4\alpha)^\alpha}{2} + 1)$-competitive.*

Uniform-height, arbitrary width and agreeable deadlines

We then consider jobs with agreeable deadlines. We first note that simply scheduling $\lceil \frac{\text{avg}(t)}{h} \rceil$ number of jobs may not return a feasible schedule.

To schedule these jobs, we first observe that for a set of jobs with total densities at most h, it is feasible to schedule them such that the load at any time is at most h. Roughly speaking, we consider jobs in the order of release, and hence in EDF manner since the jobs have agreeable deadlines. We keep the current ending time of all jobs that have been considered. As a new job is released, if its release time is earlier than the current ending time, we set its start time to the current ending time (and increase the current ending time by the width of the new job); otherwise, we set its start time to be its release time.

Using this observation, we then partition the jobs into "queues" each of which has sum of densities at most h. Each queue $\mathcal{Q}_i$ is scheduled independently and the resulting schedule is to "stack up" all these schedules. The queues are formed in a Next-Fit manner: (i) the current queue $\mathcal{Q}_q$ is kept "open" and a newly arrived job is added to the current queue if including it makes the total densities stays at most 1; (ii) otherwise, the current queue is "closed" and a new queue $\mathcal{Q}_{q+1}$ is created as open.

Algorithm $\mathcal{AD}$. The algorithm consists of the following components: InsertQueue, SetStartTime and ScheduleQueue.

InsertQueue: We keep a counter q for the number of queues created. When a job J arrives, if $\text{den}(J) + \sum_{J' \in \mathcal{Q}_q} \text{den}(J') \le h$, then job J is added to $\mathcal{Q}_q$; otherwise, job J is added to a new queue $\mathcal{Q}_{q+1}$ and we set $q \leftarrow q + 1$.

SetStartTime: For the current queue, we keep a current ending time E, initially set to 0. When a new job J is added to the queue, if $r(J) \le E$, we set $st(J) \leftarrow E$; otherwise, we set $st(J) \leftarrow r(J)$. We then update E to $st(J) + w(J)$.

ScheduleQueue: At any time t, schedule all jobs in all queues with start time set at t.

▶ **Lemma 13.** *(i) $\ell(\mathcal{AD}, t) \le 3 \cdot h \cdot \lceil \frac{avg(t)}{h} \rceil$ for $t \in \mathcal{I}_{>h}$; (ii) $\ell(\mathcal{AD}, t) \le h$ for $t \in \mathcal{I}_{\le h}$.*

By Lemmas 11 and 13, we have Theorem 14 by setting $c = 3$ and $c' = 1$.

▶ **Theorem 14.** *For jobs with uniform height, arbitrary width and agreeable deadlines, $\mathcal{AD}$ is $(\frac{(12\alpha)^\alpha}{2} + 1)$-competitive. For jobs with uniform height, arbitrary width and same release time or same deadline, the competitive ratio can be improved to $(\frac{(8\alpha)^\alpha}{2} + 1)$ by using first-fit instead of next-fit for InsertQueue.*

6 Exact Algorithms

We first present some key notions. An algorithm with parameters $p_1, p_2, \ldots$ is said to be a *fixed parameter algorithm* if it runs in $f(p_1, p_2, \ldots) \cdot O(g(N))$ time for any function f and any polynomial function g, where N is the size of input. A problem is *fixed-parameter tractable (FPT)* if it can be solved by a fixed parameter algorithm. In this section, we show that the general case of the Grid problem is FPT with respect to a few parameters, and derive lower bounds of it.

We design two fixed parameter algorithms that are based on dynamic programming. Roughly speaking, we divide the timeline into k contiguous windows in a specific way, where each window W_i represents a time interval $[b_i, b_{i+1})$ for $1 \leq i \leq k$. The algorithm visits all windows accordingly from left to right and maintains a candidate set of schedules for the visited windows that no optimal solution is deleted from the set. The parameters of the algorithms emerge if we interpret the input as an "interval graph".

Interval graph. A graph $G = (V, E)$ is an *interval graph* if it captures the intersection relation for some set of intervals on the real line. Formally, for each $v \in V$, we can associate v to an interval I_v such that (u, v) is in E if and only if $I_u \cap I_v \neq \emptyset$. It has been shown in [13, 14] that an interval graph has a "consecutive clique arrangement", i.e., its maximal cliques can be linearly ordered in a way that for every vertex v in the graph, the maximal cliques containing v occur consecutively in the linear order. For any instance of the Grid problem, we can transform it into an interval graph $G = (V, E)$: For each job J with interval $I(J)$, we create a vertex $v(J) \in V$ and an edge is added between $v(J)$ and $v(J')$ if and only if $I(J)$ intersects $I(J')$. We can then obtain a set of maximal cliques in linear order, C_1, C_2, $\cdots$, C_k, by sweeping a vertical line from left to right, where k denotes the number of maximal cliques thus obtained. Our parameter, the maximum number of overlapped feasible intervals, is the maximum size of these maximal cliques.

Boundaries and windows. Based on the maximal cliques described above, we define some "windows" W_1, W_2, $\cdots$, W_k with "boundaries" b_1, b_2, $\cdots$, b_{k+1} as follows. We first give the definition of boundaries for the first algorithm. This definition will be generalized in Section 6.1 for the second algorithm. For $1 \leq i \leq k$, the i-th *boundary* b_i is defined as the earliest release time of jobs in clique C_i but not in cliques before C_i, precisely, $b_i = \min\{t \mid t = r(J) \text{ and } J \in C_i \setminus (\cup_{s=1}^{i-1} C_s)\}$. The rightmost boundary b_{k+1} is defined as the latest deadline among all jobs. With the boundaries, we partition the timeslots into contiguous intervals called *windows*. Figure 1 is an example of a set of jobs, its corresponding interval graph and maximal cliques.

6.1 Fixed parameter algorithms

Framework of the algorithms

We propose two exact algorithms, both run in k stages one for each of the k windows. We maintain a table T_{left} storing all "valid configurations" of jobs in all windows that have been considered so far. A row in the table consists of the configurations of all jobs. In addition, for each window W_i, we compute a table T_{right_i} to store all possible configurations of start and end time of jobs available in W_i. The configurations in T_{right_i} is then "concatenated" to some configurations in T_{left} that are "compatible" with each other. These merged configurations will be filtered to remove non-optimal ones. The remaining configurations become the new

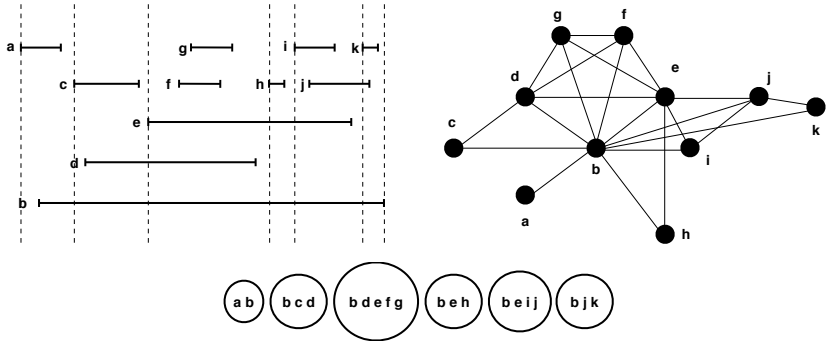

Figure 1 (Left) A set of jobs, where the horizontal line segments are the feasible time intervals of jobs and the vertical lines are boundaries of windows. (Right) An interval graph of the corresponding job set. (Bottom) A set of all the maximal cliques in the interval graph.

T_{left} for the next window. We denote by W_{left} the union of the windows corresponding to T_{left}. We drop the subscript i in T_{right_i} when the context is clear.

Configurations, validity, compatibility, concatenation. A *configuration* of job J in window W_i is an execution segment $[st_i(J), et_i(J))$. It is *valid* if $[st_i(J), et_i(J)) \subseteq [r(J), d(J))$. The cost of a window W_i with respect to some configurations is $\sum_{t \in W_i} (\sum_{J: t \in [st_i(J), et_i(J))} h(J))^\alpha$. Two configurations of the same job J are *compatible* if the union of the two configurations is a valid and contiguous execution segment with length exactly $w(J)$. To *concatenate* two configurations from two different windows, we make a union of the two configurations, and the corresponding cost is to add the costs of the two windows.

An algorithm with three parameters

Algorithm $\mathcal{E}$. We first transform the input job set $\mathcal{J}$ to an interval graph, and obtain the maximal cliques C_i for $1 \le i \le k$ and the corresponding windows W_i. We start with T_{left} containing the only configuration, which sets all the jobs to be not yet executed. Then we visit the windows from left to right with three procedures: ListConfigurations, ConcatenateTables and FilterTable.

ListConfigurations: For window W_i and jobs in C_i, we construct T_{right} storing all configurations of $J \in C_i$ with their cost in a brute force manner. We also delete the invalid configurations.

ConcatenateTables: We then concatenate compatible configurations in T_{left} and T_{right}. The resulting table is the new T_{left}. We also delete the invalid configurations in the new T_{left}.

FilterTable: After concatenation, we filter non-optimal configurations. We leave the configuration with the lowest cost among the configurations with the same execution segment.

After processing all the windows, the configuration with the lowest cost in the final T_{left} is returned as the solution. Note that a configuration is deleted only when it is invalid or its cost is higher than another configuration with the same execution segment for each job. Thus the algorithm outputs an optimal solution.

Let n to be the number of jobs, $w_{\max}$ to be the maximum width of jobs, m to be the maximum size of cliques, and $W_{\max}$ to be the maximum length of windows. Through a detail analysis, the running time of the three procedures depend mainly on $w_{\max}$, m and $W_{\max}$. So we have:

▶ **Theorem 15.** *Algorithm $\mathcal{E}$ computes an optimal solution in $f(w_{\max}, m, W_{\max}) \cdot O(n^2)$ time for some function f, i.e., the problem is FPT with respect to $w_{\max}$, m, and $W_{\max}$.*

An algorithm with two parameters

Algorithm $\mathcal{E}^+$. This algorithm is similar to algorithm $\mathcal{E}$ except the definitions of boundaries and ListConfigurations. Given a set of jobs $\mathcal{J}$, the algorithm uses the set of boundaries $\{r(J) \mid J \in \mathcal{J}\} \cup \{d(J) \mid J \in \mathcal{J}\}$ to construct the windows and obtain the corresponding cliques. For ListConfigurations, the number of timeslots we are considering for each window is at most a constant times the total width of all jobs in the window, i.e., when the window length is relatively larger than the total width of all jobs in the window, we consider much fewer than the window length. Let n to be the number of jobs, $w_{\max}$ to be the maximum width of jobs, and m to be the maximum size of cliques. We observed that the running time of the three procedures depend only on $w_{\max}$ and m. So we have:

▶ **Theorem 16.** *Algorithm $\mathcal{E}^+$ computes an optimal solution in $f(w_{\max}, m) \cdot O(n^2)$ time for some function f. Hence, the Grid problem is FPT with respect to $w_{\max}$ and m.*

6.2 Exact algorithm without parameter

For the case with unit width and arbitrary height of Grid problem, we can use Algorithm $\mathcal{E}$ to design an exact algorithm that its time complexity is only measured in the size of the input. In the case with unit width and arbitrary height, one may observe that the functionalities of the components of Algorithm $\mathcal{E}$ are not affected by the length of the windows. Without loss of generality, we assume that the number of timeslots τ is even. And we enforce all windows to have length 2. By this setting, the new algorithm runs in $O((\tau/2) \cdot 4^{2n} \cdot n)$ time where n is the number of jobs. Note that the input size N of the problem is $3n \log \tau + n \log h_{\max}$ where $h_{\max}$ is the maximum height over all jobs. Since $\log \tau = O(N)$, the running time becomes $2^{O(N)}$. Thus we have the following theorem.

▶ **Theorem 17.** *There is an exact algorithm running in $2^{O(N)}$ time for the Grid problem with unit width and arbitrary height where N is the length of the input.*

Jansen et al. [16] derived several lower bounds for scheduling and packing problems which can be used to develop lower bounds for our problem. Their lower bounds assume *Exponential Time Hypothesis* (ETH) holds, which conjectures that there is a positive real ϵ such that 3-Sat cannot be decided in time $2^{\epsilon n} N^{O(1)}$ where n is the number of variables in the formula and N is the length of the input. A lower bound for other problems can be shown by making use of strong reductions, i.e. reductions that increase the parameter at most linearly. Through a sequence of strong reductions, they obtain two lower bounds for Partition, $2^{o(n)} N^{O(1)}$ and $2^{o(\sqrt{N})}$ where n is the cardinality of the given set and N is the length of the input. We design a strong reduction from Partition to the decision version of Grid problem with unit width and arbitrary height. For each integer s in an integer set S, we convert it to a job J with $r(J) = 0$, $d(J) = 2$, $w(J) = 1$ and $h(J) = 2s$. We claim that S is a partition if and only if the set of jobs can be scheduled with cost at most $2(\sum_{s \in S} s)^{\alpha}$. By setting the length of the input or the number of jobs as the parameter, we have:

▶ **Theorem 18.** *There is a lower bound of $2^{o(\sqrt{N})}$ and a lower bound of $2^{o(n)} N^{O(1)}$ on the running time for the Grid problem unless ETH fails, where n is the number of jobs and N is the length of the input.*

7 Conclusion

We develop the first online algorithm with polylogarithm-competitive ratio and the first FPT algorithms for non-preemptive smart grid scheduling problem for the general case. Our algorithm in Section 4 relies on a classification into powers of 2. In the full paper, we show that one can classify into powers of $1 + \lambda$, for $\lambda > 0$, and the competitive ratio only changes by a constant factor. We also discuss in the full paper why we base on the preemptive instead of non-preemptive DVS problem.

There are many future directions: different cost functions to capture varying electricity cost over time or measuring the maximum cost instead of the total [27]; jobs with varying power requests during its execution (it is a constant value in this paper); other objectives like response time. A preliminary result is that we can extend our online algorithm to the case where a job may have varying power requests during its execution, in other words, a job can be viewed as having rectilinear shape instead of being rectangular. In such case, the competitive ratio is increased by a factor which measures the maximum height to the minimum height ratio of a job (see the full paper for more details).

References

1 Soroush Alamdari, Therese Biedl, Timothy Chan, Elyot Grant, Krishnam Jampani, Srinivasan Keshav, Anna Lubiw, and Vinayak Pathak. Smart-grid electricity allocation via strip packing with slicing. In *WADS*, pages 25–36, 2013.

2 Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

3 Yossi Azar. On-line load balancing. In *Online Algorithms*, pages 178–195, 1998.

4 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

5 Paul C. Bell and Prudence W. H. Wong. Multiprocessor speed scaling for jobs with arbitrary sizes and deadlines. *J. Comb. Optim.*, 29(4):739–749, 2015.

6 Mihai Burcea, Wing-Kai Hon, Hsiang-Hsuan Liu, Prudence W. H. Wong, and David K. Y. Yau. Scheduling for electricity cost in smart grid. In *COCOA*, pages 306–317, 2013.

7 Chen Chen, K. G. Nagananda, Gang Xiong, Shalinee Kishore, and Lawrence V. Snyder. A communication-based appliance scheduling scheme for consumer-premise energy management systems. *IEEE Trans. Smart Grid*, 4(1):56–65, 2013.

8 Lin Chen, Nicole Megow, and Kevin Schewior. An $O(\log m)$-competitive algorithm for online machine minimization. In *SODA*, pages 155–163, 2016.

9 Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *IFIP TCS*, pages 209–222, 2004.

10 Marijana Živić Djurović, Aleksandar Milačić, and Marko Kršulja. A simplified model of quadratic cost function for thermal generators. In *DAAAM*, pages 25–28, 2012.

11 Kan Fang, Nelson A. Uhan, Fu Zhao, and John W. Sutherland. Scheduling on a single machine under time-of-use electricity tariffs. *Annals OR*, 238(1-2):199–227, 2016.

12 Xin Feng, Yinfeng Xu, and Feifeng Zheng. Online scheduling for electricity cost in smart grid. In *COCOA*, pages 783–793, 2015.

13 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

14 Rudolf Halin. Some remarks on interval graphs. *Combinatorica*, 2(3):297–304, 1982.

15 Katherine Hamilton and Neel Gulhar. Taking demand response to the next level. *IEEE Power and Energy Mag.*, 8(3):60–65, 2010.

16 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. In *WADS*, pages 439–450, 2013.

17 Iordanis Koutsopoulos and Leandros Tassiulas. Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In *Proc. e-Energy*, pages 41–50, 2011.

18 Fu-Hong Liu, Hsiang-Hsuan Liu, and Prudence W. H. Wong. Optimal nonpreemptive scheduling in a smart grid model. *CoRR*, abs/1602.06659, 2016.

19 Thillainathan Logenthiran, Dipti Srinivasan, and Tan Shun. Demand side management in smart grid using heuristic optimization. *IEEE Trans. Smart Grid*, 3(3):1244–1252, 2012.

20 T. Lui, W. Stirling, and H. Marcy. Get smart. *IEEE Power & Energy Mag.*, 8(3):66–78, 2010.

21 Sabita Maharjan, Quanyan Zhu, Yan Zhang, Stein Gjessing, and Tamer Basar. Dependable demand response management in the smart grid: A stackelberg game approach. *IEEE Trans. Smart Grid*, 4(1):120–132, 2013.

22 Gilbert Masters. *Renewable and efficient electric power systems*. John Wiley & Sons, 2013.

23 Barna Saha. Renting a cloud. In *FSTTCS*, pages 437–448, 2013.

24 Sergio Salinas, Ming Li, and Pan Li. Multi-objective optimal energy consumption scheduling in smart grids. *IEEE Trans. Smart Grid*, 4(1):341–348, 2013.

25 US Department of Energy. The Smart Grid: An Introduction. `http://www.oe.energy.gov/SmartGridIntroduction.htm`, 2009.

26 F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

27 Sean Yaw and Brendan Mumey. An exact algorithm for non-preemptive peak demand job scheduling. In *COCOA*, pages 3–12, 2014.

28 Zpryme Research & Consulting. Power systems of the future: The case for energy storage, distributed generation, and microgrids, 2012.