Algorithmica manuscript No.
(will be inserted by the editor)

Non-preemptive Scheduling in a Smart Grid Model and its Implications on Machine Minimization

Fu-Hong Liu · Hsiang-Hsuan Liu · Prudence W.H. Wong

Received: date / Accepted: date

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 problem is fixed parameter tractable. We also show that the online algorithm is the best-possible in an asymptotically sense when the objective is to minimize the peak load. In addition, we observe that the classical non-preemptive machine minimization problem is a special case of the smart grid problem with min-peak objective and show that we can achieve the best-possible competitive ratio in an asymptotically sense when solving the non-preemptive machine minimization problem.

A preliminary version of this paper titled "Optimal Nonpreemptive Scheduling in a Smart Grid Model" appeared in Proceedings of the 27th International Symposium on Algorithms and Computation, ISAAC 2016 [30] and some results are improved in this version.

Fu-Hong Liu
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
E-mail: fhliu@cs.nthu.edu.tw

Hsiang-Hsuan Liu (partially supported by a studentship from the University of Liverpool-National Tsing-Hua University Dual PhD programme.)
Department of Information and computing sciences, Utrecht University, Netherlands
E-mail: h.h.liu@uu.nl

Prudence W.H. Wong (partially supported by the Networks Sciences & Technologies (NeST) initiative of University of Liverpool.)
Department of Computer Science, University of Liverpool, Liverpool, UK
E-mail: pwong@liverpool.ac.uk

Keywords Scheduling · Non-preemptive · Convex cost function · Online algorithms · Fixed parameter tractable

1 Introduction

The smart grid [14, 46, 47] is a power grid system [17, 37] that makes power generation, distribution and consumption more efficient through information and communication technologies compared to the traditional power system. Peak demand hours happen only for a short duration, which makes existing electrical grid less efficient. It has been noted in [9] 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 [47]. *Demand response management* [16, 21, 22, 34, 51] attempts to overcome this problem by shifting users' demand to off-peak hours in order to reduce peak load [7, 26, 33, 36, 39, 42]. Research initiatives in the area include [24, 32, 40, 45].

We study a scheduling problem arising in demand response management. The electricity grids supports demand response mechanisms and obtains energy efficiency by organizing customer consumption of electricity in response to supply conditions. It is demonstrated in [34] that demand response is of remarkable advantage to consumers, utilities and society. Effective demand load management brings down the cost of operating the grid, as well as energy generation and distribution [33]. Demand response management 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, i.e., the higher the generation cost the higher the electricity price. Therefore, it is to the consumers' advantage to reduce electricity consumption at high price and hence reduce the electricity bill [42].

The smart grid operator and consumers communicate through smart metering devices [27, 37]. A consumer sends in a power request with the power requirement (cf.*height* of request), required non-preemptable duration of service (cf.*width* of request) 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, in particular 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., $\alpha = 2$ [13, 43].

Previous work. Koutsopoulos and Tassiulas [26] has 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. unit duration). Feng et al. [18]

have claimed that a simple greedy algorithm is 2-competitive for the unit case and $\alpha = 2$. However, as to be described below in Lemma 1, there is indeed a counter example that the greedy algorithm is at least 3-competitive. Hence it is still an open question to derive online algorithms for the problem. Salinas et al. [42] considered a multi-objective problem to minimize energy consumption cost and maximize utility. A closely related problem is to manage the load by changing the price of electricity over time [7, 15, 36, 38]. Heuristics have also been developed for demand side management [33]. Other aspects of smart grid have also been considered, e.g., communication [9, 28, 29, 31], security [31, 35]. Reviews of the smart grid can be found in [16, 21, 22, 34, 51].

The main combinatorial problem we defined in this paper has an analogy to the traditional load balancing problem [3] and machine minimization problem [10–12, 41] but the main differences are: 1) the job in our problem is non-preemptive and 2) the objective being maximum load and jobs are unit height [10–12, 41]. Minimizing maximum load has also been looked at in the context of smart grid [1, 25, 44, 49, 50], some of which further consider allowing reshaping of the jobs [1, 25]. As to be discussed in Section 2, our problem is more difficult than minimizing the maximum load. Our problem also has resemblance to the dynamic speed scaling problem [2, 5, 48] and our algorithm has employed some techniques there.

As to be discussed, our problem is closely related to the non-preemptive machine minimization problem [11,12], which was solved optimally in asymptotically sense for the online setting [41]. We provide an alternative asymptotically best-possible competitive algorithm for the non-preemptive machine minimization problem. More precisely, we show that our algorithm for the smart grid problem can also solve the non-preemptive machine minimization problem with asymptotically best-possible competitive ratio. A more detailed discussion is in Section 7.

Our contribution. In this paper, we consider a demand response optimization problem, which we call GRID, minimizing the total electricity cost and study its relation with other scheduling problems. We propose the first online algorithm for the general problem with worst case competitive ratio, which is polylogarithmic in the max-min ratio of the duration of jobs (Theorem 11 in Section 4); and give a lower bound for any online algorithm. Interestingly, the ratio depends on the max-min width ratio but not the max-min height ratio. 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 some special cases (Section 5). In addition, we show that the problem is fixed parameter tractable by proposing the first fixed parameter exact algorithms for the problem; and derive lower bounds for the running time (Section 6). Table 1 gives a summary of our results. Interestingly, our online algorithm and exact algorithms depend on the variation of the job widths but not the variation of the job heights.

We further show that our online algorithms and exact algorithms can be adapted to the objective of minimizing the peak electricity cost, as well as the related problem of non-preemptive machine minimization. Our online

Width	Height	Ratio
Unit	Arbitrary	$2^\alpha \cdot (8(e+e^2)^\alpha + 1)$-competitive
		$2^{\alpha+1}$-approximate
Uniform	Arbitrary	$12^\alpha \cdot (8(e+e^2)^\alpha + 1)$-competitive
Arbitrary	Arbitrary	$\Theta(\log^\alpha(\frac{w_{\max}}{w_{\min}}))$-competitive
Unit	Uniform	$\min((4\alpha)^\alpha/2+1, 2^\alpha \cdot (8(e+e^2)^\alpha + 1))$-competitive
Arbitrary	Uniform	$((8\alpha)^\alpha/2 + 2^\alpha)$-competitive agreeable deadline

Table 1 Summary of our online results or total electricity cost.

algorithms are asymptotically best-possible for both problems (Subsection 7.1), with competitive ratio being logarithmic in the max-min ratio of the job duration. In addition, we show that both problems are fixed-parameter tractable (Subsection 7.2).

Technically speaking, our online algorithms are based on identifying a relationship with the dynamic speed (voltage) scaling (DVS) problem [48]. 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 a 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 directly compare with the optimal GRID schedule via a lower bound for the total workload over these intervals (Lemmas 3 and 13). 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 8).

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; nevertheless, non-preemptiveness 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.

Organization of the paper. We define the problem and provide some basic observations in Section 2. The online algorithms for uniform time duration and arbitrary power requirement are developed in Section 3 and are extended for solving the general case in Section 4. The lower bound for online algorithms is provided in Section 4.3. Several special cases regarding uniform power requirement are discussed in Section 5. We design fixed-parameter exact algorithms in Section 6 and derive a lower bound for the running time in Subsection 6.3. In Section 7, we extend our online and exact algorithms to the objective of maximum load and the related non-preemptive machine minimization problem. We conclude the paper in Section 8.

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 integer parameters *release time* $r(J) \geq 0$, *deadline* $d(J) > 0$, *width* $w(J) > 0$ representing the duration required by J and *height* $h(J) > 0$ 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 $I(J)$. We denote by $|I|$ the length of an interval I, i.e., $|I| = t_2 - t_1$ where $I = [t_1, t_2)$. We define the *density* of J, denoted by $\text{den}(J)$, to be $\frac{h(J) \cdot w(J)}{|I(J)|}$. The density signifies the average processing work required by the job over its feasible interval. We then define the "average" load at any time t as $\text{avg}(t) = \sum_{J:t \in I(J)} \text{den}(J)$. In our analysis, we have to distinguish timeslots with high and low average load. Therefore, for any $h > 0$, we define $\mathcal{I}_{>h}$ and $\mathcal{I}_{\leq h}$ to be set of timeslots where the average load $\text{avg}(t)$ is larger than h and at most h, respectively. Note that $\mathcal{I}_{>h}$ and $\mathcal{I}_{\leq h}$ do not need to be contiguous.

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 $w_{\min} = 1$ making $K = w_{\max}$. 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 has to assign 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* $et(S, J) = st(S, J) + w(J)$ and $[st(S, J), et(S, J)) \subseteq I(J)$. 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. For any algorithm $\mathcal{A}$, we use $\mathcal{A}(\mathcal{J})$ to denote the schedule of $\mathcal{A}$ on $\mathcal{J}$. We denote by $\mathcal{O}$ the optimal schedule.

The cost of a schedule S is the sum of the α-th power of the load over all time steps, 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 $\text{cost}(S, \mathcal{I}) = \sum_{t \in \mathcal{I}} (\ell(S, t))^\alpha$. Our goal is to find a feasible schedule S such that $\text{cost}(S)$ is minimized. We call this the GRID problem.

Online algorithms. In this paper, 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. We consider various 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 [48] and we are going to refer to three algorithms for the DVS problem, namely, the $\mathcal{YDS}$ algorithm which is optimal for the DVS problem, the online algorithms called $\mathcal{BKP}$ and $\mathcal{AVR}$. We first recap the DVS problem and the associated algorithms. In the DVS problem, jobs come with release time $r(J)$, deadline $d(J)$ and a work requirement $p(J)$. The processor can run at speed $s \in [0, \infty)$ and consumes energy in a rate of s^α, for some $\alpha > 1$. The processor in DVS resembles the grid in GRID. The objective is to complete all jobs by their deadlines using the minimum total energy. The main differences of DVS problem to the GRID problem include (i) jobs in DVS can be preempted while preemption is not allowed in our 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 our problem 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.

With the resemblance of the two problems, we make an observation about their optimal solutions. Let $\mathcal{O}_D$ and $\mathcal{O}_G$ be the optimal schedule for the DVS and GRID problem, respectively. Given a job set $\mathcal{J}_G$ for the GRID problem, 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 $\mathcal{J}_G$. Then we have the following observation.

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

Proof Consider any feasible schedule S_G. At timeslot t, suppose there are k jobs scheduled and their sum of heights is H. The schedule for S_D during timeslot t can be obtained by running the processor at speed H and the jobs time-share the processor in proportion to their height. This results in a feasible schedule with the same cost and the observation follows. □

It is known that the online algorithm $\mathcal{AVR}$ for the DVS problem is $\frac{(2\alpha)^\alpha}{2}$-competitive [48]. 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. Note that it is not always possible to convert a feasible schedule for the DVS problem to a feasible schedule for the GRID problem easily. Therefore, the corollary does not immediately solve the GRID problem but as to be shown it provides a way to analyze algorithms for GRID.

Corollary 1 *For any input $\mathcal{J}_G$ and the corresponding input $\mathcal{J}_D$, cost($\mathcal{AVR}(\mathcal{J}_D)$) $\leq \frac{(2\alpha)^\alpha}{2} \cdot cost(\mathcal{O}_G)$.*

The online algorithm $\mathcal{BKP}$ proposed by Bansal et al. [4] for the DVS problem is $8e^\alpha$-competitive with respect to total cost. The $\mathcal{BKP}$ algorithm can be seen as an online version of $\mathcal{YDS}$, as it estimates the possible maximum work load at every time steps. At time t, $\mathcal{BKP}$ concentrates on the set of special intervals $[t_1, t_2)$ where $t_1 \leq t \leq t_2$ and $(t_2 - t_1) : (t_2 - t) = e : 1$. Among these intervals, $\mathcal{BKP}$ choose the one with maximum released average work and treat the average work as an estimation of the speed required in the offline setting. The speed at t in the $\mathcal{BKP}$ schedule is then chosen as e times of the maximum released average work. By Observation 1 we have the following corollary:

Corollary 2 *For any input $\mathcal{J}_G$ and the corresponding input $\mathcal{J}_D$, cost($\mathcal{BKP}(\mathcal{J}_D)$) $\leq 8e^\alpha \cdot cost(\mathcal{O}(\mathcal{J}_D)) \leq 8e^\alpha \cdot cost(\mathcal{O}(\mathcal{J}_G))$.*

Remark: One may consider the non-preemptive DVS problem as the reference of the GRID problem. However, given a job set $\mathcal{J}_G$ and the corresponding $\mathcal{J}_D$, cost($\mathcal{O}_D(\mathcal{J}_D)$) may not necessarily lower than cost($\mathcal{O}_G(\mathcal{J}_G)$), where $\mathcal{O}_D$ here is the optimal schedule for non-preemptive DVS. There is an instance where the optimal cost of GRID is smaller. The instance contains two jobs. One has release time 0, deadline 3, width 3 and height 1. The other has release time 1, deadline 2, width 1 and height 1. Both jobs can only schedule at their release time in GRID since their widths are the same as the lengths of their feasible intervals. The optimal cost of GRID is $1^\alpha + 2^\alpha + 1^\alpha = 2^\alpha + 2$, whereas the optimal cost of non-preemptive DVS is $2^\alpha + 2^\alpha = 2 \cdot 2^\alpha$. This is because the schedule uses speed 2 and runs the longer job with 1.5 time units and the shorter job with 0.5 time units. The optimal cost of GRID is lower when $\alpha > 1$. Therefore, it is unclear how we may use the results on non-preemptive DVS problem and so we would stick with the preemptive DVS algorithms.

Relating to minimizing maximum cost. The problem of minimizing maximum cost over time (min-max) has been studied before [49]. We note that there is a polynomial time reduction of the decision version of the min-max problem to that of the min-sum problem (the GRID problem we study in this paper) for a large enough α. In particular, one can show that with $\alpha > (\tau - 1)(2\sum_{J \in \mathcal{J}} h(J) + 1)$, the maximum load would dominate the load in other timeslots and we would be able to solve the min-max problem if we have an optimal solution for the min-sum problem on α.

On the other hand, minimizing the maximum cost does not necessarily minimize the total cost. To minimize the total cost, the optimal solution might give rise to higher peak cost. For example, consider an input of three jobs J_1, J_2 and J_3 where $I(J_1) = [0, 2^\alpha)$, $h(J_1) = 1$, $w(J_1) = 2^\alpha$; $I(J_2) = [2^\alpha, 2^\alpha + 1)$, $h(J_2) = 3$, $w(J_2) = 1$; and $I(J_3) = [0, 2^{\alpha+1})$, $h(J_3) = 1$, $w(J_3) = 2^\alpha$. Note that only J_3 has flexibility where it can be scheduled. To minimize the maximum cost over time, we would schedule J_3 to start at time 0 and achieve a maximum load of 3. This gives a total cost of $2^\alpha \cdot 2^\alpha + 3^\alpha = 4^\alpha + 3^\alpha$. However, to minimize the total cost, we would schedule J_3 to start at time 2^α giving a total cost of $2^\alpha + 4^\alpha + (2^\alpha - 1) = 4^\alpha + 2^{\alpha+1} - 1$, which is smaller than $4^\alpha + 3^\alpha$ when $\alpha > 1$.

Lower bound on Greedy on the online-list model. On the online-list model, requests with feasible intervals come in arbitrary order. On the other hand, in the online-time model, requests with feasible intervals come at the first time they are available. A lower bound in the online-time model is also a lower bound in the online-list model, while an upper bound in the online-list model is also an upper bound in the online-time model. The only lower bound of online algorithms was investigated on online-list model. In [18], the greedy algorithm that assigns a job to a timeslot with the minimum load is considered. It is claimed in the paper that the greedy algorithm is 2-competitive on the online-list model and for the case where the load of a timeslot t is $\ell(t)^2$, jobs are of unit length and height and the feasible timeslots of a job is a set of (non-contiguous) timeslots that the job can be assigned to. We show a counter-example to this claim by showing that Greedy is at least 3-competitive. This implies that it is still an open question to derive competitive online algorithms for the GRID problem.

Lemma 1 *Greedy is no better than* 3*-competitive for the online-list model when* $\alpha = 2$*.*

Proof Let k be an arbitrarily large integer. The adversary works in k rounds and all the jobs released are of width and height 1. In the i-th round, where $1 \leq i < k$, the adversary releases 2^{k-i} jobs; and in the k-th round (the final one), the adversary releases two jobs. In the first round, the feasible timeslots of each job released are $[1, 2^k]$. In the i-th round, where $2 \leq i \leq k$, the feasible timeslots of each job released are all the timeslots that Greedy has assigned jobs in the $(i-1)$-th round. We claim that the total cost of Greedy is $3 \cdot 2^k - 4$ and the total cost of the optimal schedule is 2^k. Therefore, the competitive ratio of Greedy is arbitrarily close to 3 with an arbitrarily large integer k.

We first analyze Greedy. Since Greedy always assigns to a timeslot with the minimum load, in the first round, Greedy assigns jobs to 2^{k-1} timeslots with each job to a different timeslot. These 2^{k-1} timeslots will be the feasible timeslots for the 2^{k-2} jobs in the second round. Using a similar argument, we can see that in each round, the number of feasible timeslots is twice the number of jobs released in that round. In addition, before the i-th round, the load of each feasible timeslot is $i-1$ and Greedy adds a load of 1 to each timeslot that it assigns a job, making the load become i. Therefore, the total cost of Greedy is $\sum_{i=1}^{k-2}(i^2 \cdot 2^{k-i-1}) + k^2 \cdot 2 = 3 \cdot 2^k - 4$. On the other hand, we can assign jobs released in a round to the timeslots that are not feasible timeslots for later rounds since in the i-th round, the number of feasible timeslots is 2^{k-i+1} and the number of jobs released is 2^{k-i}. Therefore, in the optimal schedule, the load of each timeslot is exactly 1 and the total cost is 2^k. □

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 $\mathcal{UV}$ (Subsection 3.2) is based on a further restricted case of unit width, i.e., $w = 1$ (Subsection 3.1).

3.1 Unit width and arbitrary height

In this section, we consider jobs with 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 $\mathcal{R}$. In particular, we require that the speed of $\mathcal{R}$ remains the same during any integral timeslot, i.e., in $[t, t+1)$ for all integers t. Note that when jobs have integral release times and deadlines, many known DVS algorithms satisfy this criteria, including $\mathcal{YDS}$ and $\mathcal{AVR}$. After modification, $\mathcal{BKP}$ also satisfies the criteria and can be used as the reference (Lemma 4).

Recall in Section 2 how a job set for the GRID problem is converted to a job set for the DVS problem. We simulate a copy of $\mathcal{R}$ on the converted job set 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 job run by $\mathcal{R}$ at t.

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

Analysis. We note that 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 first note that $\mathcal{V}$ gives a feasible schedule (Lemma 2) and then analyze its competitive ratio (Theorem 2).

Lemma 2 *$\mathcal{V}$ gives a feasible schedule.*

Proof Let $\ell(S, \mathcal{I})$ denote the total work done by schedule S in $\mathcal{I}$. That is, $\ell(S, \mathcal{I}) = \sum_{t \in \mathcal{I}} \ell(S, t)$. According to the algorithm, for all $\mathcal{I}_t = [0, t)$, $\ell(\mathcal{V}, \mathcal{I}_t) \geq \ell(\mathcal{R}, \mathcal{I}_t)$.

Suppose on the contrary that $\mathcal{V}$ has a job J_m missing its deadline at t. That is, $d(J_m) = t$ but J_m is not finished before t. By the algorithm, for all $t' \in [0, t)$, $\ell(\mathcal{V}, t') \geq \ell(\mathcal{R}, t')$ unless there are less than $\ell(\mathcal{R}, t')$ available jobs at t' for $\mathcal{V}$. Let t_0 be the last timeslot in $[0, t)$ such that $\ell(\mathcal{V}, t_0) < \ell(\mathcal{R}, t_0)$ (if there is no such timeslot, set t_0 as -1), $r(J_m) > t_0$ since all jobs released at or before t_0 have been assigned. For all $t' \in [t_0 + 1, t)$, $\ell(\mathcal{V}, t') \geq \ell(\mathcal{R}, t')$. Also, all jobs J with $r(J) \leq t_0$ are finished by $t_0 + 1$ and jobs executed in $[t_0+1, t)$ are those released after t_0. Consider set $\mathcal{J}_t$ of jobs with feasible interval completely inside $\mathcal{I} = [t_0 + 1, t)$ (note that $J_m \in \mathcal{J}_t$), $\ell(S, \mathcal{I}) \geq \sum_{J \in \mathcal{J}_t} h(J)$ for any feasible schedule S. Since $\mathcal{V}$ assigns jobs in EDF manner and is not feasible, $\ell(\mathcal{V}, \mathcal{I}) < \sum_{J \in \mathcal{J}_t} h(J)$. It follows that $\sum_{J \in \mathcal{J}_t} h(J) > \ell(\mathcal{V}, \mathcal{I}) \geq \ell(\mathcal{R}, \mathcal{I})$. It contradicts to the fact that $\mathcal{R}$ is feasible. Hence, $\mathcal{V}$ finishes all jobs before their deadlines. □

Now we analyze the performance of $\mathcal{V}$. Recall that the intervals start and end at integral points. Let $h_{\max}(\mathcal{V}, t)$ be the maximum height of jobs scheduled at t by $\mathcal{V}$; we set $h_{\max}(\mathcal{V}, t) = 0$ if $\mathcal{V}$ assigns no job at t. 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. The following lemma bounds the cost of $\mathcal{V}$ in each type of timeslots. Recall that $\text{cost}(S, \mathcal{I})$ denotes the cost of the schedule S over the interval $\mathcal{I}$ and $\text{cost}(S)$ denotes the cost of the entire schedule.

Lemma 3 *The cost of $\mathcal{V}$ satisfies the following properties.* (i) *cost*$(\mathcal{V}, \mathcal{I}_1) \leq 2^\alpha \cdot$ *cost*$(\mathcal{R})$*; and* (ii) *cost*$(\mathcal{V}, \mathcal{I}_2) \leq 2^\alpha \cdot$ *cost*$(\mathcal{O})$.

Proof (i) By the algorithm, $\ell(\mathcal{V}, t) < \ell(\mathcal{R}, t) + h_{\max}(\mathcal{V}, t) \leq 2 \cdot \ell(\mathcal{R}, t)$ for $t \in \mathcal{I}_1$. It follows that $\text{cost}(\mathcal{V}, \mathcal{I}_1) \leq 2^\alpha \cdot \sum_{t \in \mathcal{I}_1} \ell(\mathcal{R}, t)^\alpha = 2^\alpha \cdot \text{cost}(\mathcal{R}, \mathcal{I}_1) \leq 2^\alpha \cdot \text{cost}(\mathcal{R})$.

(ii) By convexity, $\text{cost}(\mathcal{O}) \geq \sum_J h(J)^\alpha$. And we can see that $\text{cost}(\mathcal{O}) \geq \sum_{t \in \mathcal{I}_2} h_{\max}(\mathcal{V}, t)^\alpha$. According to the algorithm, $\ell(\mathcal{V}, t) < \ell(\mathcal{R}, t) + h_{\max}(\mathcal{V}, t) \leq 2 \cdot h_{\max}(\mathcal{V}, t)$ for $t \in \mathcal{I}_2$. Hence, we have $\text{cost}(\mathcal{V}, \mathcal{I}_2) = \sum_{t \in \mathcal{I}_2} \ell(\mathcal{V}, t)^\alpha \leq 2^\alpha \cdot \sum_{t \in \mathcal{I}_2} h_{\max}(\mathcal{V}, t)^\alpha \leq 2^\alpha \cdot \text{cost}(\mathcal{O})$. □

Notice that $\text{cost}(\mathcal{V}) = \text{cost}(\mathcal{V}, \mathcal{I}_1) + \text{cost}(\mathcal{V}, \mathcal{I}_2)$ since $\mathcal{I}_1$ and $\mathcal{I}_2$ have no overlap. Together with Lemma 3 and Observation 1, we obtain the competitive ratio of $\mathcal{V}$ in the following theorem.

Theorem 2 *Algorithm $\mathcal{V}$ is $2^\alpha \cdot (R+1)$-competitive, where R is the competitive ratio of the reference* DVS *algorithm $\mathcal{R}$. If $\mathcal{R}$ is an offline algorithm with cost $R \cdot$ cost$(\mathcal{O})$, algorithm $\mathcal{V}$ is $2^\alpha \cdot (R+1)$-approximate.*

There are a number of DVS algorithms that can be used as the reference algorithm. The only requirement is that the speed of the reference algorithm within any integral interval $[t, t+1)$ for some integer t should be at most the load of the resulting online algorithm at the corresponding timeslot t. Otherwise, the feasibility of $\mathcal{V}$ cannot be guaranteed. Also, since in our online algorithm we make decisions at each integral time t, it means if the load of the reference algorithm at $i + \Delta$ is larger than $\ell(\mathcal{R}, i)$ for some $0 < \Delta < 1$, our online algorithm might not be feasible.

The speed of the $\mathcal{AVR}$ and $\mathcal{YDS}$ algorithm only change at release times or deadlines of the jobs so it is valid to use $\mathcal{AVR}$ or $\mathcal{YDS}$ as a reference. Note that if we use $\mathcal{YDS}$ as the reference, the algorithm $\mathcal{V}$ is an offline algorithm since $\mathcal{YDS}$ is an offline algorithm. Unlike $\mathcal{AVR}$ and $\mathcal{YDS}$, the speed of $\mathcal{BKP}$ within a timeslot might increase. Hence, we need to modify the $\mathcal{BKP}$ algorithm such that it can be used as the reference algorithm. In Lemma 4, we show that the speed of $\mathcal{BKP}$ in $[t, t+1)$ is bounded by a constant factor times the speed at t for any time t.

Lemma 4 *For any integral time t and a constant $0 < \Delta < 1$, $\ell(\mathcal{BKP}, t+\Delta) \leq (1+e) \cdot \ell(\mathcal{BKP}, t)$ if the release times and deadlines of jobs are integral.*

Proof Let $p(t, I)$ denotes the total work of jobs J with $I(J) \subseteq I$ and $r(J) \leq t$. The speed of $\mathcal{BKP}$ at time t, $\ell(\mathcal{BKP}, t) = \max_I e \cdot \frac{p(t,I)}{|I|}$ where $I = [t_1, t_2)$ satisfies the property that $(t_2 - t_1) : (t_2 - t) = e : 1$. To prove this lemma, we consider the interval I chosen by $\mathcal{BKP}$ at time $t + \Delta$. We prove that I can be transformed into another interval I' which is one of the candidates for $\mathcal{BKP}$ at time t. We show that $e \cdot \frac{p(t,I')}{|I'|}$ is at least $\frac{1}{1+e}$ times of the speed of $\mathcal{BKP}$ at $t + \Delta$.

Assume that at time $t + \Delta$, $\ell(\mathcal{BKP}, t + \Delta) = e \cdot \frac{p(t,I)}{|I|}$ where $I = [t_1, t_2)$ is chosen by $\mathcal{BKP}$. We can construct $I' = [t'_1, t_2)$ such that $(t_2 - t'_1) : (t_2 - t) = e : 1$ by setting $t'_1 = t_2 - e(t_2 - t)$. It is clear that $I \subset I'$ since the two intervals have the same right endpoint and I' is longer than I. In fact, $|I'| = e(t_2 - t) = e(t_2 - (t + \Delta)) + e\Delta = |I| + e\Delta \leq |I| + e$. Moreover, for any interval candidate, the length must be at least 1 if the release times and deadlines of the jobs are integral. Otherwise, the interval contains no jobs and the speed is 0. Hence, $|I'| \leq (1 + e)|I|$. By $\mathcal{BKP}$, $\ell(\mathcal{BKP}, t) \geq e \cdot \frac{p(t,I')}{|I'|} = e \cdot \frac{p(t+\Delta,I')}{|I'|}$. The later equality holds since there is no job released between t and $t + \Delta$. Since $I \subset I'$ and $|I'| \leq (1 + e)|I|$, $e \cdot \frac{p(t+\Delta,I')}{|I'|} \geq e \cdot \frac{p(t+\Delta,I)}{|I'|} \geq e \cdot \frac{p(t+\Delta,I)}{(1+e)|I|}$. Hence, $\ell(\mathcal{BKP}, t) \geq \frac{1}{1+e} \cdot \ell(\mathcal{BKP}, t + \Delta)$. □

Lemma 4 implies that, although the speeds of $\mathcal{BKP}$ change within $[t, t+1)$, the speeds are bounded by $(1+e)$ times of the speed at t. Hence, we can modify $\mathcal{BKP}$ into $\mathcal{BKP}'$ as follows: at integral time t, the speed of $\mathcal{BKP}'$, $\ell(\mathcal{BKP}', t) = (1 + e)\ell(\mathcal{BKP}, t)$; at time $t' = t + \Delta$ where t is integral and $0 < \Delta < 1$, $\ell(\mathcal{BKP}', t') = \ell(\mathcal{BKP}', t)$. By the modification, the speed of $\mathcal{BKP}'$ remains the same during any integral timeslot and $\text{cost}(\mathcal{BKP}') \leq (1 + e)^\alpha \cdot \text{cost}(\mathcal{BKP})$. As mentioned in Section 2, the $\mathcal{BKP}$ algorithm is $8 \cdot e^\alpha$-competitive. On the other hand, $\mathcal{V}$ can take an offline DVS algorithm, e.g., the optimal $\mathcal{YDS}$ algorithm, as reference and returns an offline schedule. Therefore, we have the following corollary.

Corollary 3 *$\mathcal{V}$ is $2^\alpha \cdot (8 \cdot (e + e^2)^\alpha + 1)$-competitive, $2^\alpha \cdot (\frac{(2\alpha)^\alpha}{2} + 1)$-competitive and $2^\alpha \cdot 2$-approximate when the algorithm $\mathcal{BKP}'$, $\mathcal{AVR}$ and $\mathcal{YDS}$ are referenced, respectively.*

3.2 Uniform width and arbitrary height

In this section, we consider jobs with uniform width w 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 release times or deadlines at non-integral times. To remedy this, we define a procedure AlignFI to align the feasible intervals (precisely, release times and deadlines) to the new time unit of duration w.

Let $\mathcal{J}$ be a uniform width job set. We first define the notion of "tight" and "loose" jobs. A job J is said to be *tight* if $|I(J)| < 2w$; otherwise, it is *loose*. Let $\mathcal{J}_\text{T}$ and $\mathcal{J}_\text{L}$ be the disjoint subsets of tight and loose jobs of $\mathcal{J}$, respectively.

We design different strategies for tight and loose jobs. As to be shown, 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 are multiples of w. With this alternation, we can treat the jobs as unit width and make scheduling decisions at times that are multiples of w.

Procedure AlignFI**.** Given a loose job set $\mathcal{J}_{\mathrm{L}}$ in which $w(J) = w$ and $|I(J)| \geq 2 \cdot w \ \forall J \in \mathcal{J}_{\mathrm{L}}$. 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. We denote the resulting job set by $\mathcal{J}'$.

- $r(J') \leftarrow \min_{i \in 0 \cup \mathbb{N}}\{i \cdot w \mid i \cdot w \geq r(J)\}$;
- $d(J') \leftarrow \max_{i \in 0 \cup \mathbb{N}}\{i \cdot w \mid i \cdot w \leq d(J)\}$.

Observation 3 *For any job $J \in \mathcal{J}_L$ and the corresponding J',* (i) $\frac{1}{3} \cdot |I(J)| < |I(J')| \leq |I(J)|$*;* (ii) $|I(J')| \geq w$*;* (iii) $I(J') \subseteq I(J)$*.*

Notice that after running 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 $i_1 < i_2$. By Observation 3, a feasible schedule of J' is also a feasible schedule of J. Furthermore, after running AlignFI all jobs are released at times which are multiples of w. Hence, the job set $\mathcal{J}'$ can be treated as job set with unit width, where each unit has duration w instead of 1.

As a consequence of altering the feasible intervals, we introduce two additional procedures that convert associated schedules. Given a schedule S for job set $\mathcal{J}_{\mathrm{L}}$, AlignSch converts it to a schedule S' for the corresponding job set $\mathcal{J}'$. The other procedure FreeSch takes a schedule S' for a job set $\mathcal{J}'$ and converts it to a schedule S for $\mathcal{J}_{\mathrm{L}}$.

Transformation AlignSch**.** AlignSch transforms S into S' by shifting the execution interval of every job $J \in \mathcal{J}_{\mathrm{L}}$.

- $st(S', J') \leftarrow \min\{d(J') - w(J'), \min_{i \geq 0}\{i \cdot w \mid i \cdot w \geq st(S, J)\}\}$;
- $et(S', J') \leftarrow st(S', J') + w(J')$.

Observation 4 *Consider any schedule S for $\mathcal{J}_L$ and the schedule S' for $\mathcal{J}'$ constructed by* AlignSch*. The following properties hold:* (i) *For any job $J \in \mathcal{J}_L$ and the corresponding J', $st(J') > st(J) - w$ and $et(J') < et(J) + w$;* (ii) *S' is a feasible schedule for $\mathcal{J}'$; and* (iii) *At any time t, $\ell(S', t) \leq \ell(S, t) + \ell(S, t - (w-1)) + \ell(S, t + (w-1))$.*

Proof (ii) By AlignSch, $st(S', J') \leq d(J') - w(J')$. Also, $|[st(S', J'), et(S', J'))| = w(J')$. Hence $[st(S', J'), et(S', J')) \subseteq I(J')$. That is, S' is a feasible schedule for both $\mathcal{J}'$ and J.

(iii) By (i), $st(J') > st(J) - w$ and $et(J') < et(J) + w$ for each J. Hence, for any timeslot t, for each job J with $[st(S, J), et(S, J)) \cap [t-(w-1), t+(w-1)) = \emptyset$, $t \notin [st(S', J'), et(S', J'))$. On the other hand, consider the jobs J that $[st(J), et(J)) \cap [t-(w-1), t+(w-1)) \neq \emptyset$. Since $|[st(J), et(J))| = w$, at least one of the timeslots $t-(w-1)$, t or $t+(w-1)$ is in $[st(J), et(J))$. Hence we can capture $\ell(S', t)$ by $\ell(S, t) + \ell(S, t-(w-1)) + \ell(S, t+(w-1))$. □

Corollary 4 *Using* AlignSch *to generate S' given S, we have cost$(S') \leq 3^\alpha \cdot$ cost(S).*

Proof By Observation 4 (iii), $\text{cost}(S') = \sum_t \ell(S',t)^\alpha \leq \sum_t (\ell(S,t) + \ell(S, t-(w-1)) + \ell(S, t+(w-1)))^\alpha \leq \sum_t (3 \cdot \ell(S,t))^\alpha = 3^\alpha \cdot \text{cost}(S)$. □

Lemma 5 *cost$(\mathcal{O}(\mathcal{J}')) \leq 3^\alpha \cdot$ cost$(\mathcal{O}(\mathcal{J}_L))$.*

Proof Consider set of loose jobs $\mathcal{J}_\text{L}$ with uniform width and the corresponding $\mathcal{J}'$. Given $\mathcal{O}(\mathcal{J}_\text{L})$, there exists schedule $S(\mathcal{J}')$ generated by AlignSch. By Lemma 4, $\text{cost}(S(\mathcal{J}')) \leq 3^\alpha \cdot \text{cost}(\mathcal{O}(\mathcal{J}_\text{L}))$. Hence, $\text{cost}(\mathcal{O}(\mathcal{J}')) \leq \text{cost}(S(\mathcal{J}')) \leq 3^\alpha \cdot \text{cost}(\mathcal{O}(\mathcal{J}_\text{L}))$. □

Transformation FreeSch**.** FreeSch transforms S' into S.

- $st(S,J) \leftarrow st(S',J')$;
- $et(S,J) \leftarrow et(S',J')$.

The feasibility of S' can be easily proved by Observation 3 (iii).

Lemma 6 *Using* FreeSch*, we have cost(S) = cost(S').*

Proof Since the execution intervals of J and J' are the same, $\ell(S,t) = \ell(S',t)$ for all t. Hence $\text{cost}(S) = \text{cost}(S')$. □

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 $\mathcal{J}_\text{T}$ be the set of tight jobs in $\mathcal{J}$ and $\mathcal{J}_\text{L}$ be the set of loose jobs in $\mathcal{J}$.

1. For any tight job $J \in \mathcal{J}_\text{T}$, schedule J to start at $r(J)$.
2. Loose jobs in $\mathcal{J}_\text{L}$ are converted to $\mathcal{J}'$ by AlignFI. For $\mathcal{J}'$, we run Algorithm $\mathcal{V}$, which is defined in Subsection 3.1, with $\mathcal{BKP}$ as the reference DVS algorithm. Jobs are chosen in an earliest deadline first (EDF) manner. By running Transformation FreeSch on the resulting schedule $\mathcal{V}(\mathcal{J}')$, we get the schedule for $\mathcal{J}_\text{L}$.

Note that the decisions of $\mathcal{UV}$ can be made online.

Analysis of Algorithm $\mathcal{UV}$. We analyze the tight jobs and loose jobs separately. We first give an observation.

Observation 5 *For any two job sets $\mathcal{J}_x \subseteq \mathcal{J}_y$, cost$(\mathcal{O}(\mathcal{J}_x)) \leq$ cost$(\mathcal{O}(\mathcal{J}_y))$.*

Proof Assume on the contrary that $\text{cost}(\mathcal{O}(\mathcal{J}_y)) < \text{cost}(\mathcal{O}(\mathcal{J}_x))$, we can generate a schedule $S(\mathcal{J}_x)$ by removing jobs from $\mathcal{O}(\mathcal{J}_y)$ which are not in $\mathcal{J}_x$. It follows that $\text{cost}(S(\mathcal{J}_x)) \leq \text{cost}(\mathcal{O}(\mathcal{J}_y)) < \text{cost}(\mathcal{O}(\mathcal{J}_x))$, contradicting to the fact that $\mathcal{O}(\mathcal{J}_x)$ is optimal for $\mathcal{J}_x$. □

The next lemma proves the competitive ratio separately for $\mathcal{J}_\text{T}$ and $\mathcal{J}_\text{L}$.

Lemma 7 (i) *cost$(\mathcal{UV}(\mathcal{J}_T)) \leq 3^\alpha \cdot$ cost$(\mathcal{O}(\mathcal{J}))$;* (ii) *cost$(\mathcal{UV}(\mathcal{J}_L)) \leq 6^\alpha \cdot (8(e + e^2)^\alpha + 1) \cdot$ cost$(\mathcal{O}(\mathcal{J}))$.*

Proof (i) We prove that any feasible schedule S for tight jobs is 3^α-competitive. We first extend jobs $J \in \mathcal{J}_{\mathrm{T}}$ to J' such that the width of J' is the same as the length of its feasible interval. That is, $r(J') = r(J)$, $d(J') = d(J)$, $w(J') = d(J) - r(J)$ and $h(J') = h(J)$. We denote the resulting job set by $\mathcal{J}'$. Since each job in $\mathcal{J}'$ are not shiftable, there is only one feasible schedule for $\mathcal{J}'$ and it is optimal. Consider any job J and the corresponding job J'. In any feasible schedule S for $\mathcal{J}_{\mathrm{T}}$ and the optimal schedule $\mathcal{O}(\mathcal{J}')$, the execution interval of J is covered by the one of J'. Hence, $\mathrm{cost}(S(\mathcal{J}_{\mathrm{T}})) \leq \mathrm{cost}(\mathcal{O}(\mathcal{J}'))$.

Next, we prove that the load at any time t of $\mathcal{O}(\mathcal{J}')$ can be bounded by the loads of constant number of timeslots in $S(\mathcal{J}_{\mathrm{T}})$. Consider any job J and its corresponding extended job J'. For any timeslot $t \in [r(J'), d(J'))$, J' is executed as J' is not shiftable. Since J is a tight job, the length of its feasible interval is at most $2w-1$. Hence, the execution interval of J in any feasible schedule must contain either timeslot $t-(w-1)$, t or $t+(w-1)$. We can upper bound the load at any time t in $\mathcal{O}(\mathcal{J}')$: $\ell(\mathcal{O}(\mathcal{J}'), t) \leq \ell(\mathcal{O}(\mathcal{J}_{\mathrm{T}}), t-(w-1)) + \ell(\mathcal{O}(\mathcal{J}_{\mathrm{T}}), t) + \ell(\mathcal{O}(\mathcal{J}_{\mathrm{T}}), t+(w-1))$. Therefore, $\mathrm{cost}(S(\mathcal{J}_{\mathrm{T}})) \leq \mathrm{cost}(\mathcal{O}(\mathcal{J}')) \leq 3^\alpha \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J}_{\mathrm{T}}))$.

(ii) For $\mathcal{J}_{\mathrm{L}}$, we apply AlignFI and get $\mathcal{J}'_{\mathrm{L}}$, which can be viewed as unit width jobs. We then run $\mathcal{V}$ and get $\mathcal{V}(\mathcal{J}'_{\mathrm{L}})$ and get $\mathcal{UV}(\mathcal{J}_{\mathrm{L}})$ by performing FreeSch on $\mathcal{V}(\mathcal{J}'_{\mathrm{L}})$. The cost of $\mathcal{UV}(\mathcal{J}_{\mathrm{L}})$ is exactly the cost of $\mathcal{V}(\mathcal{J}'_{\mathrm{L}})$. According to Corollary 3, by choosing $\mathcal{BKP}'$ as reference algorithm, $\mathrm{cost}(\mathcal{V}(\mathcal{J}'_{\mathrm{L}})) \leq 2^\alpha \cdot (8 \cdot (e+e^2)^\alpha + 1) \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J}'_{\mathrm{L}}))$. Hence, $\mathrm{cost}(\mathcal{O}(\mathcal{J}'_{\mathrm{L}})) \leq 3^\alpha \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J}_{\mathrm{L}})) \leq 3^\alpha \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J}))$, where the first inequality is by Lemma 5 and the second is by Observation 5. Finally, $\mathrm{cost}(\mathcal{UV}(\mathcal{J}_{\mathrm{L}})) = \mathrm{cost}(\mathcal{V}(\mathcal{J}'_{\mathrm{L}})) \leq 2^\alpha \cdot (8 \cdot (e+e^2)^\alpha + 1) \cdot 3^\alpha \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J})) = 6^\alpha \cdot (8 \cdot (e+e^2)^\alpha + 1) \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J}))$. □

Theorem 6 *cost*$(\mathcal{UV}(\mathcal{J})) \leq 12^\alpha \cdot (8(e+e^2)^\alpha + 1) \cdot$ *cost*$(\mathcal{O}(\mathcal{J}))$.

Proof By definition, $\mathrm{cost}(\mathcal{UV}(\mathcal{J})) = \sum_t \ell(\mathcal{UV}(\mathcal{J}), t)^\alpha = \sum_t (\ell(\mathcal{UV}(\mathcal{J}_{\mathrm{T}}), t) + \ell(\mathcal{UV}(\mathcal{J}_{\mathrm{L}}), t))^\alpha$. By Jansen's inequality, since the convexity, the total cost is at most $2^{\alpha-1} \cdot \sum_t (\ell(\mathcal{UV}(\mathcal{J}_{\mathrm{T}}), t)^\alpha + \ell(\mathcal{UV}(\mathcal{J}_{\mathrm{L}}), t)^\alpha) = 2^{\alpha-1} \cdot (\mathrm{cost}(\mathcal{UV}(\mathcal{J}_{\mathrm{T}})) + \mathrm{cost}(\mathcal{UV}(\mathcal{J}_{\mathrm{L}})))$. By Lemma 7, $\mathrm{cost}(\mathcal{UV}(\mathcal{J})) \leq 2^{\alpha-1} \cdot (3^\alpha + 6^\alpha \cdot (8(e+e^2)^\alpha + 1)) \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J})) \leq 2^\alpha \cdot 6^\alpha \cdot (8(e+e^2)^\alpha + 1) \cdot \mathrm{cost}(\mathcal{O}(\mathcal{J}))$. □

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 optimal cost modestly. Furthermore, we show that for any nice job set $\mathcal{J}^*$, we can bound $\mathrm{cost}(\mathcal{G}(\mathcal{J}^*))$ by $\mathrm{cost}(\mathcal{O}(\mathcal{J}^*))$ and in turn by $\mathrm{cost}(\mathcal{O}(\mathcal{J}))$. Then we can establish the competitive ratio of $\mathcal{G}$.

4.1 Nice job set and transformations

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 jobs. 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^* as follows. We denote the resulting nice job set by $\mathcal{J}^*$. Suppose J is in class C_p. We modify its width, release time and deadline:

- $w(J^*) \leftarrow 2^p$;
- $r(J^*) \leftarrow r(J)$;
- $d(J^*) \leftarrow r(J^*) + \max\{d(J) - r(J), 2^p\}$.

The setting for $d(J^*)$ is due to rounding up the width. The observation below follows directly from the definition.

Observation 7 *For any job J and its nice job J^* transformed by* CONVERT*,* (i) $I(J) \subseteq I(J^*)$*;* (ii) $I(J) \neq I(J^*)$ *if and only if* $|I(J)| < 2^p$*; in this case,* $den(J) > \frac{1}{2}$ *and* $den(J^*) = 1$.

We then define two procedures that transform schedules related to nice job sets. RELAXSCH takes a schedule S for a job set $\mathcal{J}$ and converts it to a schedule S^* for the corresponding nice job set $\mathcal{J}^*$. On the other hand, SHRINKSCH takes a schedule S^* for a nice job set $\mathcal{J}^*$ and converts it to a schedule S for $\mathcal{J}$.

Transformation RELAXSCH**.** RELAXSCH transforms S into S^* by extending and necessarily shifting the execution interval of every job J.

- $st(S^*, J^*) = \min\{d(J^*) - w(J^*), st(S, J)\}$
- $et(S^*, J^*) = st(S^*, J^*) + w(J^*)$.

Observation 8 asserts that the resulting schedule S^* is feasible for $\mathcal{J}^*$ while Lemmas 8 and 9 analyze the load and cost of the schedule.

Observation 8 *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 $\mathcal{J}^*$.*

To analyze the load of the schedule S^*, we consider partial schedule $S_p^* \subseteq S^*$ (resp. $S_p \subseteq S$) which is for all the jobs of $\mathcal{J}^*$ (resp. $\mathcal{J}$) in class C_p. We prove that the load of S_p^* at any time can be bounded by the sum of the load of S_p at the current time and $2^{p-1} - 1$ timeslots before and after the current time.

Lemma 8 *At any time t, $\ell(S_p^*, t) \leq \ell(S_p, t) + \ell(S_p, t - (2^{p-1} - 1)) + \ell(S_p, t + (2^{p-1} - 1))$.*

Proof We prove that for any job J, the corresponding nice job J^* contributes to $\ell(S_p^*, t)$ only if J contributes to either $\ell(S_p, t)$, $\ell(S_p, t - (2^{p-1} - 1))$ or $\ell(S_p, t + (2^{p-1} - 1))$. There are two cases that J contributes to neither $\ell(S_p, t -$

$(2^{p-1}-1))$ nor $\ell(S_p, t+(2^{p-1}-1))$: (i) the execution interval of J is completely outside the interval $[t-(2^{p-1}-1), t+(2^{p-1}-1)]$ or (ii) $[st(J), et(J)] \subseteq (t-(2^{p-1}-1), t+(2^{p-1}-1))$.

In the case (i), $et(J) < t-(2^{p-1}-1)$ or $st(J) > t+(2^{p-1}-1)$. By the procedures of RelaxSch, since $w(J) > 2^{p-1}$, $et(J^*) \leq et(J)+(2^{p-1}-1)$ and $st(J^*) \geq st(J)-(2^{p-1}-1)$. Hence, $t \notin [st(J^*), et(J^*)]$. That is, J^* does not contribute to $\ell(S_p^*, t)$ if $et(J) < t-(2^{p-1}-1)$ or $st(J) > t+(2^{p-1}-1)$. Notice that if $et(J) = t-(2^{p-1}-1)$ or $st(J) = t+(2^{p-1}-1)$, J^* does not necessarily contribute to $\ell(S_p^*, t)$.

For case (ii), consider job J with $[st(J), et(J)] \subseteq (t-(2^{p-1}-1), t+(2^{p-1}-1))$. Since $2^{p-1} < w(J) \leq 2^p$, $t \in [st(J), et(J)]$. That is, J contributes to $\ell(S_p, t)$ no matter if J^* contributes to $\ell(S_p^*, t-(2^{p-1}-1))$ or $\ell(S_p^*, t+(2^{p-1}-1))$.

By case (i) and (ii), for any job J with $[st(J), et(J)] \cap [t-(2^{p-1}-1), t+(2^{p-1}-1)] = \emptyset$, J^* does not contribute to $\ell(S_p^*, t)$. And for any job J with $[st(J), et(J)] \subseteq (t-(2^{p-1}-1), t+(2^{p-1}-1))$, J contributes to $\ell(S_p, t)$. Hence, by assuming all jobs at timeslot $t-(2^{p-1}-1)$ or $t+(2^{p-1}-1)$ contribute to $\ell(S_p^*, t)$, $\ell(S_p^*, t)$ is bounded by $\ell(S_p, t)+\ell(S_p, t-(2^{p-1}-1))+\ell(S_p, t+(2^{p-1}-1))$. □

Lemma 9 *Using* RelaxSch*, we have* $cost(S_p^*) \leq 3^\alpha \cdot cost(S_p)$.

Proof By Lemma 8, $\text{cost}(S_p^*) = \sum_t \ell(S_p^*, t)^\alpha \leq \sum_t (\ell(S_p, t)+\ell(S_p, t-(2^{p-1}-1))+\ell(S_p, t+(2^{p-1}-1)))^\alpha \leq \sum_t (3 \cdot \ell(S_p, t))^\alpha = 3^\alpha \cdot \text{cost}(S_p)$. □

Transformation ShrinkSch**.** On the other hand, ShrinkSch converts a schedule S^* for a nice job set $\mathcal{J}^*$ to a schedule S for the corresponding job set $\mathcal{J}$. We set

- $st(S, J) \leftarrow st(S^*, J^*)$;
- $et(S, J) \leftarrow st(S, J)+w(J)$, therefore, $et(S, J) \leq et(S^*, J^*)$.

Observation 9 asserts that the resulting schedule S is feasible for J and Lemma 10 analyzes the cost of the schedule.

Observation 9 *Consider any schedule* S^* *for* $\mathcal{J}^*$ *and schedule* S *constructed by* ShrinkSch *for the corresponding* $\mathcal{J}$*. For any* J^* *and the corresponding* J*, we have* (i) $[st(S, J), et(S, J)] \subseteq [st(S^*, J^*), et(S^*, J^*)]$*;* (ii) $[st(S, J), et(S, J)] \subseteq [r(J), d(J)]$.

By Observation 9, we have the following lemma.

Lemma 10 *Using* ShrinkSch*, we have* $cost(S_p) \leq cost(S_p^*)$.

4.2 The online algorithm

Online algorithm $\mathcal{G}$. We are now ready to describe the algorithm $\mathcal{G}$ for an arbitrary job set $\mathcal{J}$. 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 Transformation ShrinkSch. Note that all these procedures can be done in an online fashion.

Using the results in Section 3 and Subsection 4.1, we can compare the cost of $\mathcal{G}(\mathcal{J})$ with $\mathcal{O}(\mathcal{J}_p^*)$ for each class C_p (see Theorem 11). It remains to analyze the cost of $\mathcal{O}(\mathcal{J}_p^*)$ and $\mathcal{O}(J)$ in the next observation.

Observation 10 *Consider any job set $\mathcal{J}$, its corresponding job set $\mathcal{J}^*$ and the corresponding job sets of each class $\mathcal{J}_p$ and $\mathcal{J}_p^*$.* (i) *cost*$(\mathcal{O}(\mathcal{J}_p^*)) \leq 3^\alpha \cdot$ *cost*$(\mathcal{O}(\mathcal{J}_p))$*;* (ii) *cost*$(\mathcal{O}(\mathcal{J}_p)) \leq$ *cost*$(\mathcal{O}(\mathcal{J}))$.

Proof (i) Given $\mathcal{O}(\mathcal{J}_p)$, there exists a schedule $S(\mathcal{J}_p^*)$ generated by RelaxSch. By Lemma 9, $\text{cost}(S(\mathcal{J}_p^*)) \leq 3^\alpha \cdot \text{cost}(\mathcal{O}(\mathcal{J}_p))$. Hence, we have $\text{cost}(\mathcal{O}(\mathcal{J}_p^*)) \leq \text{cost}(S(\mathcal{J}_p^*)) \leq 3^\alpha \cdot \text{cost}(\mathcal{O}(\mathcal{J}_p))$.

(ii) Assume on the contrary that $\text{cost}(\mathcal{O}(\mathcal{J})) < \text{cost}(\mathcal{O}(\mathcal{J}_p))$. We can generate a schedule $S(\mathcal{J}_p)$ by removing jobs from $\mathcal{O}(\mathcal{J})$ which are not in $\mathcal{J}_p$. It follows that $\text{cost}(S(\mathcal{J}_p)) \leq \text{cost}(\mathcal{O}(\mathcal{J})) < \text{cost}(\mathcal{O}(\mathcal{J}_p))$, contradicting to the fact that $\mathcal{O}(\mathcal{J}_p)$ is optimal for $\mathcal{J}_p$. □

Theorem 11 *For any job set $\mathcal{J}$, we have*

$$\textit{cost}(\mathcal{G}(\mathcal{J})) \leq (36\lceil \log K \rceil)^\alpha \cdot \left(8(e+e^2)^\alpha + 1\right) \cdot \textit{cost}(\mathcal{O}(\mathcal{J})) \ ,$$

where $K = \frac{w_{\max}}{w_{\min}}$.

Proof By definition, $\text{cost}(\mathcal{G}(\mathcal{J})) = \sum_t \ell(\mathcal{G}(\mathcal{J}), t)^\alpha = \sum_t (\sum_{p=1}^{\lceil \log K \rceil} \ell(\mathcal{G}(\mathcal{J}_p), t))^\alpha$. By Jansen's inequality and the convexity of the α-power function, the cost is at most $\lceil \log K \rceil^{\alpha-1} \sum_{p=1}^{\lceil \log K \rceil} \sum_t \ell(\mathcal{G}(\mathcal{J}_p), t)^\alpha$. For each group of jobs $\mathcal{J}_p$, we Convert it to $\mathcal{J}_p^*$, apply Algorithm $\mathcal{UV}$ on it and transform the schedule into a schedule for $\mathcal{J}_p$ by ShrinkSch. Hence, $\ell(\mathcal{G}(\mathcal{J}_p), t) \leq \ell(\mathcal{UV}(\mathcal{J}_p^*), t)$ for each t. We have $\text{cost}(\mathcal{G}(\mathcal{J})) \leq \lceil \log K \rceil^{\alpha-1} \sum_{p=1}^{\lceil \log K \rceil} \text{cost}(\mathcal{G}(\mathcal{J}_p)) \leq \lceil \log K \rceil^{\alpha-1} \cdot \sum_{p=1}^{\lceil \log K \rceil} \text{cost}(\mathcal{UV}(\mathcal{J}_p^*))$. By Theorem 6 and Observations 10 (i) and 10 (ii), $\text{cost}(\mathcal{UV}(\mathcal{J}_p^*)) \leq 12^\alpha \cdot (8(e+e^2)^\alpha + 1) \cdot \text{cost}(\mathcal{O}(\mathcal{J}_p^*)) \leq 12^\alpha \cdot (8(e+e^2)^\alpha + 1) \cdot 3^\alpha \cdot \text{cost}(\mathcal{O}(\mathcal{J}_p)) \leq 36^\alpha \cdot (8(e+e^2)^\alpha + 1) \cdot \text{cost}(\mathcal{O}(\mathcal{J}))$. Hence $\text{cost}(\mathcal{G}(\mathcal{J})) \leq 36^\alpha \cdot \lceil \log K \rceil^{\alpha-1} \cdot (8(e+e^2)^\alpha + 1) \cdot \sum_{p=1}^{\lceil \log K \rceil} \text{cost}(\mathcal{O}(\mathcal{J})) = (36\lceil \log K \rceil)^\alpha \cdot (8(e+e^2)^\alpha + 1) \cdot \text{cost}(\mathcal{O}(\mathcal{J}))$. □

Note that the logarithm in the competitive ratio comes from the number of classes defined in Section 2. Suppose we change the definition of classes such that class p includes jobs of size in the range $((1+\lambda)^{p-1}, (1+\lambda)^p]$ for some $\lambda > 0$ and Procedure Convert such that the width of jobs in class C_p is rounded up to $(1+\lambda)^p$. Then, the number of classes becomes $\lceil \log_{1+\lambda} K \rceil$. In addition, the competitive ratio depends on Lemma 8 that bounds the load at any timeslot by the load of three other timeslots. This number of timeslots is also affected by the definition of classes. In summary, the following lemma states the competitive ratio for varying λ.

Lemma 11 *For* $0 \leq \lambda \leq 0.5$, $0.5 < \lambda \leq 1$ *and* $\lambda \geq 2$, *the competitive ratio of our algorithm becomes* $(12 \times 2\lceil \log_{1+\lambda} K \rceil)^{\alpha}(8(e+e^2)^{\alpha}+1)$, $(12 \times 3\lceil \log_{1+\lambda} K \rceil)^{\alpha}(8(e+e^2)^{\alpha}+1)$ *and* $(12 \times (\lfloor 2\lambda \rfloor + 1)\lceil \log_{1+\lambda} K \rceil)^{\alpha}(8(e+e^2)^{\alpha}+1)$, *respectively.*

Proof When we use $1+\lambda$ as the factor for classifying the jobs by widths, the number of classes is $\lceil \log_{1+\lambda} K \rceil$, which replaces $\lceil \log K \rceil$ in Theorem 11. We note that this number decreases as λ increases.

Next, we observe the change of Lemma 8. In Lemma 8, we show that after the transformation RelaxSch, the load of any timeslot t in the resulting schedule S_p^* can be bounded by above by sampling a constant number of timeslots loads in the schedule S_p. When $1+\lambda \geq 2$, to bound the load at timeslot t in S_p^*, it should be sampled every λ^{p-1} timeslots in the interval $[t-(\lambda^p-1), t+(\lambda^p-1)]$. That is, $\ell(S_p^*, t) \leq \ell(S_p, t) + \ell(S_p, t-(\lambda^{p-1}-1)) + \ell(S_p, t+(\lambda^{p-1}-1)) + \ell(S_p, t-(2\lambda^{p-1}-1)) + \ell(S_p, t+(2\lambda^{p-1}-1)) + \ell(S_p, t-(3\lambda^{p-1}-1)) + \ell(S_p, t+(3\lambda^{p-1}-1)) + \cdots$. When $1.5 < 1+\lambda < 2$, there are three timeslots loads needed to be sampled: $\ell(S_p, t)$, $\ell(S_p, t-(\lambda^p-1))$ and $\ell(S_p, t+(\lambda^p-1))$. When $1 \leq 1+\lambda \leq 1.5$, it is sufficient to sample at most two timeslots loads, $\ell(S_p, t-(\lambda^p-1))$ and $\ell(S_p, t+(\lambda^p-1))$. It is because the interval $[t-(\lambda^p-1), t+(\lambda^p-1)]$ has width smaller than $3 \cdot \lambda^{p-1}$, which implies that there can be a job with width slightly greater than λ^{p-1} while the execution interval inside $[t-(\lambda^p-1)+\lambda^{p-1}, t+(\lambda^p-1)-\lambda^{p-1}]$. □

We note the competitive ratio for $\lambda < 1$ is larger than that for $\lambda = 1$ and the best competitive ratio occurs when $1 < \lambda < 2$.

4.3 Lower bound

In this section, we show lower bounds on competitive ratio for GRID problem with unit height and arbitrary width jobs by designing an adversary for the problem. The lower bounds are immediately lower bounds for the general case of GRID problem.

Although the design of the adversary is similar to Saha's design [41], our analysis is totally different from theirs. Saha's paper only consider the peak of schedule, while our paper analyzes all the loads of timeslots, calculates the total convex cost of the loads and finds the worst-case instances of competitive ratio that depends on the electricity cost function. In addition, each job in our adversary has distinct release time, while Saha's adversary accepts the identical release time for different jobs, which is ambiguous for an online algorithm to know the order of released jobs.

Our adversary constructs a set of jobs with a low cost of offline optimal schedule but a high cost of any online algorithm $\mathcal{A}$. It generates jobs one by one and assigns release times, deadlines and widths of jobs based on the previously generated jobs. The start times of jobs scheduled by $\mathcal{A}$ will be used for the job generations later. This ensures that $\mathcal{A}$ has to put a job on top of all existing jobs and results in a high energy cost. Meanwhile, the adversary will choose an

appropriate feasible interval for each job such that an optimal offline algorithm can schedule the job set with low energy cost. The following is the description of the adversary.

Adversary Λ and job instance $\mathcal{J}$. Given an online algorithm $\mathcal{A}$ and a number $\alpha > 1$, adversary Λ outputs a set of jobs $\mathcal{J}$ consisting of n jobs. Let J_i be the ith job of $\mathcal{J}$. The adversary first computes a width for each job before running $\mathcal{A}$. It sets $w(J_{n-1}) = x$ and $w(J_n) = x - 1$ for some large number x and $w(J_i) = 3w(J_{i+1}) + 1$ for $1 \leq i \leq n-2$. Then Λ releases the jobs from J_1 to J_n accordingly and computes a release time and a deadline for each job through an interaction with $\mathcal{A}$. For the first job J_1, Λ chooses any release time and deadline such that $d(J_1) - r(J_1) \geq 3w(J_1)$. For the ith job $J_i \in \mathcal{J}$ for $2 \leq i \leq n$ accordingly, Λ sets $r(J_i) = st(\mathcal{A}, J_{i-1}) + 1$ and $d(J_i) = et(\mathcal{A}, J_{i-1})$. This limits $\mathcal{A}$ to fewer choices of start times for scheduling a new job. A job can only be scheduled in the execution interval of its previous job by $\mathcal{A}$. On the other hand, no two jobs have the same release time since the feasible interval of J_i does not contain the start time of J_{i-1}. To ensure this, we require that the widths of neighboring jobs in the release order differ at least 1.

Let $w_{\max}$ and $w_{\min}$ denote by the maximum and minimum width of jobs respectively and let $\mathcal{O}$ be an optimal offline algorithm for GRID problem. We have the following results.

Lemma 12 *cost*$(\mathcal{O}(\mathcal{J})) \leq x \cdot 3^{n-1}$.

Proof By the setting of Λ, we show that $\mathcal{O}$ can schedule all the jobs $\mathcal{J}$ without overlapping and the cost of an optimal schedule is just the sum of widths of all the jobs.

For any job $J_i \in \mathcal{J}$ and $i \geq 2$, the length of its feasible interval is $d(J_i) - r(J_i) = et(\mathcal{A}, J_{i-1}) - (st(\mathcal{A}, J_{i-1}) + 1) = w(J_{i-1}) - 1 = 3w(J_i)$ and $d(J_1) - r(J_1) \geq 3w(J_1)$. This means no matter where we schedule a job, at least one of the intervals $[r(J_i), st(J_i))$ and $[et(J_i), d(J_i))$ has length at least $w(J_i)$. Algorithm $\mathcal{O}$ can schedule the subsequent jobs in the interval with length at least $w(J_i)$ such that the subsequent jobs do not overlap with J_i. This is because the sum of widths of all the subsequent jobs does not exceed $w(J_i)$. Since this argument can be applied on all the jobs, this implies that all the jobs do not overlap with each other in an optimal schedule. Thus the cost of an optimal schedule is the sum of widths of all the jobs. More precisely,

$$\begin{aligned}\text{cost}(\mathcal{O}(\mathcal{J})) &= (x-1) + x + (3x+1) + (3(3x+1)+1) + \ldots + w_{\max} \\ &\leq 2x + 2 \cdot 3x + 2 \cdot 9x + \ldots + 2 \cdot 3^{n-2}x \\ &= 2x \cdot \frac{3^{n-1}-1}{2} \leq x \cdot 3^{n-1} \ . \end{aligned}$$

□

Theorem 12 *For any deterministic online algorithm $\mathcal{A}$ for* GRID *problem with unit height and arbitrary width jobs, Adversary Λ constructs an instance*

$\mathcal{J}$ *such that* (i) *for constant* α*,*

$$\frac{cost(\mathcal{A}(\mathcal{J}))}{cost(\mathcal{O}(\mathcal{J}))} \geq \left(\frac{\lfloor\alpha\rfloor+1}{3}\right)^{\alpha} \ ;$$

and (ii) *for arbitrary* α*,*

$$\frac{cost(\mathcal{A}(\mathcal{J}))}{cost(\mathcal{O}(\mathcal{J}))} \geq \log_3^{\alpha} \frac{w_{\max}}{w_{\min}} \ .$$

Proof We first give a lower bound on $\text{cost}(\mathcal{A}(\mathcal{J}))$ and then give the lower bounds on the competitive ratio by combining $\text{cost}(\mathcal{A}(\mathcal{J}))$ with Lemma 12.

(i) In this case, we consider that Λ only releases $\lfloor\alpha\rfloor+1$ jobs, i.e., $n = \lfloor\alpha\rfloor+1$. By the setting of Λ, all the jobs scheduled by $\mathcal{A}$ overlap with each other. For ease of the computation for the cost of $\mathcal{A}$, we only consider the timeslots contained by the execution interval of the last job $J_{\lfloor\alpha\rfloor+1}$. Thus $\text{cost}(\mathcal{A}(\mathcal{J})) \geq (x-1)\cdot(\lfloor\alpha\rfloor+1)^{\alpha}$ and

$$\frac{\text{cost}(\mathcal{A}(\mathcal{J}))}{\text{cost}(\mathcal{O}(\mathcal{J}))} \geq \frac{(x-1)\cdot(\lfloor\alpha\rfloor+1)^{\alpha}}{x\cdot 3^{\lfloor\alpha\rfloor}} \geq \left(\frac{\lfloor\alpha\rfloor+1}{3}\right)^{\alpha}$$

if x is large enough.

(ii) Assume α can be arbitrarily large, say $\alpha = (y+1)n$ for some large number y. We use $w_{\max}$ and $w_{\min}$ to bound n. According to Lemma 12, we have $w_{\max} \leq \text{cost}(\mathcal{O}(\mathcal{J})) \leq x\cdot 3^{n-1}$ and thus

$$n \geq \log_3 \frac{w_{\max}}{x} + 1 \geq \log_3 \frac{w_{\max}}{3(x-1)} + 1 = \log_3 \frac{w_{\max}}{w_{\min}} \ .$$

The second inequality is due to $x \leq 3(x-1)$ when $x \geq 2$. Recall that $\mathcal{A}$ stacks all the jobs together and thus $\text{cost}(\mathcal{A}(\mathcal{J})) \geq (x-1)\cdot n^{\alpha}$. Combining with Lemma 12, we have the lower bound on the competitive ratio

$$\begin{aligned}
\frac{\text{cost}(\mathcal{A}(\mathcal{J}))}{\text{cost}(\mathcal{O}(\mathcal{J}))} &\geq \frac{(x-1)\cdot n^{\alpha}}{x\cdot 3^{n-1}} \geq \frac{(x-1)\cdot n^{\alpha}}{x\cdot 3^{n}} \\
&= \frac{n^{\alpha}}{3^{n}} \quad \text{if } x \text{ is large enough} \\
&= \frac{n^{(y+1)n}}{3^{n}} = n^{yn}\cdot\left(\frac{n}{3}\right)^{n} \\
&\geq n^{yn} \quad \text{when } n \geq 3 \ .
\end{aligned}$$

Since $\alpha = (y+1)n$, we have $yn = \alpha y/(y+1)$. Therefore, the lower bound is

$$\begin{aligned}
\frac{\text{cost}(\mathcal{A}(\mathcal{J}))}{\text{cost}(\mathcal{O}(\mathcal{J}))} &\geq n^{yn} = n^{\frac{y}{y+1}\cdot\alpha} \\
&= n^{\alpha} \quad \text{if } y \text{ is large enough} \\
&\geq \left(\log_3 \frac{w_{\max}}{w_{\min}}\right)^{\alpha} \ .
\end{aligned}$$

□

Corollary 5 *For any deterministic online algorithm for* GRID *problem, the competitive ratio is at least* (i) $(\frac{\lfloor\alpha\rfloor+1}{3})^\alpha$ *for constant* α*; and* (ii) $\log_3^\alpha \frac{w_{\max}}{w_{\min}}$ *for arbitrary* α*.*

5 Online algorithm for uniform height jobs

In this section we focus on uniform height jobs of height h and consider two special cases of the width. We first consider jobs with uniform height and unit width (Subsection 5.2) and secondly consider jobs with agreeable deadlines (Subsection 5.3).

To ease the discussion, we refine a notation we defined before. For any algorithm $\mathcal{A}$ for a job set $\mathcal{J}$ and a time interval $\mathcal{I}$, we denote by $\mathcal{A}(\mathcal{J},\mathcal{I})$ the schedule of $\mathcal{A}$ on $\mathcal{J}$ over the time interval $\mathcal{I}$.

5.1 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 $\mathcal{I}_{\leq h}$, 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 a case, we compare directly to the optimal solution. Combining the two cases, we have Lemma 13, which holds for any job set. In Subsections 5.2 and 5.3, we show how we can use Lemma 13 to obtain algorithms for the special cases. Notice that the number $\lceil\frac{\text{avg}(t)}{h}\rceil$ is the minimum number of jobs needed to make the load at t at least $\text{avg}(t)$.

Lemma 13 *Suppose we have an algorithm* $\mathcal{A}$ *for a any job set* $\mathcal{J}$ *such that for some* c *and* c' (i) $\ell(\mathcal{A},t) \leq c\cdot h\cdot\lceil\frac{avg(t)}{h}\rceil$ *for all* $t\in\mathcal{I}_{>h}$ *and* (ii) $\ell(\mathcal{A},t)\leq c'\cdot h$ *for all* $t\in\mathcal{I}_{\leq h}$*. Then we have* $cost(\mathcal{A}(\mathcal{J})) \leq (\frac{(4c\alpha)^\alpha}{2}+c'^\alpha)\cdot cost(\mathcal{O}(\mathcal{J}))$*.*

Proof Let $\ell(\mathcal{AVR},t)$ denote the speed of $\mathcal{AVR}$ at t. We are going to prove that (a) $\text{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{>h})) \leq \frac{(4c\alpha)^\alpha}{2}\cdot\text{cost}(\mathcal{O}(\mathcal{J}))$ and (b) $\text{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{\leq h}))\leq c'^\alpha\cdot\text{cost}(\mathcal{O}(\mathcal{J}))$. Hence, the total cost $\text{cost}(\mathcal{A}(\mathcal{J}))\leq(\frac{(4c\alpha)^\alpha}{2}+c'^\alpha)\cdot\text{cost}(\mathcal{O}(\mathcal{J}))$ since $\mathcal{I}_{>h}$ and $\mathcal{I}_{\leq h}$ partition the entire time horizon during which jobs are available.

(a) We compare $\ell(\mathcal{A},t)$ to $\ell(\mathcal{AVR},t)$ for each timeslot t in $\mathcal{I}_{>h}$. By the assumption of $\mathcal{A}$, $\ell(\mathcal{A},t)\leq c\cdot h\cdot\lceil\frac{\text{avg}(t)}{h}\rceil < c\cdot h\cdot(\frac{\text{avg}(t)}{h}+1) = c\cdot(\text{avg}(t)+h)\leq 2c\cdot\text{avg}(t)$ since $\text{avg}(t) > h$. Hence, $\text{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{>h})) = \sum_{t\in\mathcal{I}_{>h}}\ell(\mathcal{A},t)^\alpha \leq \sum_{t\in\mathcal{I}_{>h}}(2c\cdot\text{avg}(t))^\alpha$. Recall that $\ell(\mathcal{AVR},t)=\text{avg}(t)$ for each t. By Corollary 1, $\text{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{>h}))\leq(2c)^\alpha\cdot\text{cost}(\mathcal{AVR}(\mathcal{J},\mathcal{I}_{>h}))\leq(2c)^\alpha\cdot\text{cost}(\mathcal{AVR}(\mathcal{J}))\leq\frac{(4c\alpha)^\alpha}{2}\cdot\text{cost}(\mathcal{O}(\mathcal{J}))$.

(b) By the assumption of $\mathcal{A}$, $\text{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{\leq h}))\leq|\mathcal{I}_{\leq h}|\cdot(c'h)^\alpha$. The length of $\mathcal{I}_{\leq h}$ can be upper bounded by the sum of width of all jobs. Hence, we

have $\mathrm{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{\le h})) \le \sum_{J\in\mathcal{J}} w(J)\cdot(c'h)^\alpha$. By convexity, $\mathrm{cost}(\mathcal{O}(\mathcal{J})) \ge \sum_{J\in\mathcal{J}} w(J)\cdot h(J)^\alpha = \sum_{J\in\mathcal{J}} w(J)\cdot h^\alpha$. Hence, $\mathrm{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{\le h})) \le c'^\alpha\cdot\mathrm{cost}(\mathcal{O}(\mathcal{J}))$.

The totol cost $\mathrm{cost}(\mathcal{A}(\mathcal{J}))$ is the sum of $\mathrm{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{>h}))$ and $\mathrm{cost}(\mathcal{A}(\mathcal{J},\mathcal{I}_{\le h}))$, which in total is at most $(\frac{(4c\alpha)^\alpha}{2}+c'^\alpha)\cdot\mathrm{cost}(\mathcal{O}(\mathcal{J}))$ and the theorem follows. □

5.2 Uniform height and unit width

In this section we consider job sets where all jobs have uniform height and unit width, i.e., $w(J) = 1$ and $h(J) = h$ for all jobs J. Note that such a case is a subcase discussed in Subsection 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 Subsection 3.1 when $\alpha < 3.22$.

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

The next theorem asserts that the algorithm gives a feasible schedule and states its competitive ratio.

Theorem 13 (i) *The schedule constructed by Algorithm $\mathcal{UU}$ is feasible.* (ii) *Algorithm $\mathcal{UU}$ is $(\frac{(4\alpha)^\alpha}{2}+1)$-competitive.*

Proof (i) Assume on the contrary that t is the first time at which a job J_m misses its deadline in the schedule $\mathcal{UU}$. That is, $d(J_m) = t$ and there is no space for executing J_m at t. Let t_0 denote the latest timeslot where the load of $\mathcal{UU}$ is lower than $\lceil\frac{\mathrm{avg}(t)}{h}\rceil$ (if there is no such timeslot, $t_0 = 0$). Since $\mathcal{UU}$ applies the EDF principle, the jobs executed at timeslots $t_0+1, t_0+2, \cdots, t$ have their feasible intervals lying completely in inside the interval $[t_0+1, t]$.

For any timeslot $t_i \in [t_0+1, t]$, the work executed by $\mathcal{AVR}$ in interval $[t_0+1, t_i]$ is $\sum_{t'=t_0+1}^{t_i}\mathrm{avg}(t')$. On the other hand, the total work done by $\mathcal{UU}$ in the same interval is $\sum_{t'=t_0+1}^{t_i} h\cdot\lceil\frac{\mathrm{avg}(t')}{h}\rceil \ge \sum_{t'=t_0+1}^{t_i}\mathrm{avg}(t)$. The work done by $\mathcal{UU}$ within interval $[t_0+1, t]$ is at least the work done by $\mathcal{AVR}$ in both cases. Hence, $\mathcal{AVR}$ is not feasible since $\mathcal{UU}$ is not feasible, which is a contradiction.

(ii) We note that $\ell(\mathcal{UU}, t) \le h\cdot\lceil\frac{\mathrm{avg}(t)}{h}\rceil$. To use Lemma 13, we can set $c' = 1$ for $t \in \mathcal{I}_{\le h}$ by the definition of $\mathcal{I}_{\le h}$. Furthermore, we can set $c = 1$ for $t \in \mathcal{I}_{>h}$. □

5.3 Uniform-height, arbitrary width and agreeable deadlines

In this section we consider jobs with agreeable deadlines. We first note that even for jobs with common feasible intervals, simply scheduling $\lceil\frac{\mathrm{avg}(t)}{h}\rceil$ number of jobs may not return a feasible schedule.

Example 1 Consider four jobs each job J with $r(J) = 0$, $d(J) = 5$, $h(J) = h$, $w(J) = 3$. Note that $\text{avg}(t) = 2.4 \cdot h$ for all t. If we schedule at most $\lceil \frac{\text{avg}(t)}{h} \rceil = 3$ jobs at any time, we can complete three jobs but the remaining job cannot be completed. To schedule all jobs feasibly, we need at least one timeslot where all jobs are being executed.

To schedule these jobs, we first observe in Lemma 14 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. For jobs with agreeable deadlines, a greedy schedule in EDF manner is feasible if the total density of the jobs is at most h. Roughly speaking, we keep the current ending time of all available jobs. 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. Lemma 14 asserts that such a scheduling is feasible and maintains the load at any time to be at most h.

Our algorithm $\mathcal{AD}$ for jobs with uniform-height, arbitrary width and agreeable deadlines is based on this observation. We first partition the jobs into "queues" each of which has sum of densities at most h (***InsertQueue***). Each queue $\mathcal{Q}_i$ is scheduled independently and the resulting schedule is to "stack up" all these schedules (***SetStartTime***). 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 density stay at most h; (ii) otherwise, the current queue is "closed" and a new queue $\mathcal{Q}_{q+1}$ is created as open.

Lemma 14 *Given any set of jobs of uniform-height h, arbitrary-width and agreeable deadlines. If the sum of densities of all these jobs is at most h, then it is feasible to schedule all of them using a maximum load h at any time. That is, the jobs can be scheduled feasibly without overlapping execution intervals.*

Proof Suppose there are k jobs $J_1, J_2, \cdots, J_k$ such that $\sum_{1 \le i \le k} \text{den}(J_i) \le h$. Without loss of generality, we assume that $d(J_i) \le d(J_j)$ and $r(J_i) \le r(J_j)$ for $1 \le i < j \le k$. We show that it is a feasible schedule to execute the jobs one by one, in the EDF manner. More specifically, we set $[st(J_i), et(J_i))$ to $[\max\{r(J_i), et(J_{i-1})\}, st(J_i) + w(J_i))$ for all $1 < i \le k$ and $[st(J_1), et(J_1)) = [r(J_1), r(J_1) + w(J_1))$

We observe that $\text{den}(J_1) \le h$ since $\sum_i \text{den}(J_i) \le h$. It is feasible to set $[st(J_1), et(J_1))$ to $[r(J_1), r(J_1)+w(J_1))$ since the input is feasible. Next, we prove that for all $i \ge 2$, $[st(J_i), et(J_i)) = [\max\{r(J_i), et(J_{i-1})\}, st(J_i) + w(J_i)) \subseteq [r(J_i), d(J_i))$. Since $st(J_i) = \max\{r(J_i), et(J_{i-1})\}$, we have $st(J_i) \ge r(J_i)$. Assume that $\cup_{g \le i} I(J_g)$ is a contiguous interval. Since $\sum_{g \le i} \text{den}(J_g) \le h$, $\sum_{g \le i} \frac{w(J_g)}{d(J_g) - r(J_g)} \le 1$. Since the jobs have agreeable deadlines and jobs with earlier deadlines are released earlier, $1 \ge \sum_{g \le i} \frac{w(J_g)}{d(J_g) - r(J_g)} \ge \sum_{g \le i} \frac{w(J_g)}{d(J_i) - r(J_1)}$. Hence, $\sum_{g \le i} w(J_g) \le d(J_i) - r(J_1)$. Therefore J_i can be finished before $d(J_i)$. On the other hand, if $\cup_{g \le i} I(J_g)$ is not contiguous. The proof above shows

that for each contiguous, each of the involving jobs can be finished by its deadline. □

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 $\text{den}(J) + \sum_{J' \in \mathcal{Q}_q} \text{den}(J') \leq h$, 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 $r(J) \leq E$, 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.

By Lemma 14, the schedule returned by $\mathcal{AD}$ is feasible. We then analyze its load and hence, derive its competitive ratio. Recall the definition of $\mathcal{I}_{>h}$ and $\mathcal{I}_{\leq h}$

Lemma 15 *Using $\mathcal{AD}$, we have* (i) $\ell(\mathcal{AD}, t) \leq 2 \cdot h \cdot \lceil \frac{avg(t)}{h} \rceil$ *for* $t \in \mathcal{I}_{>h}$*;* (ii) $\ell(\mathcal{AD}, t) \leq 2h$ *for* $t \in \mathcal{I}_{\leq h}$*.*

Proof We first prove the following claim: given $r > \lfloor \frac{n-1}{2} \rfloor$ where n is a natural number and r is a real number, $n \leq 2\lceil r \rceil$. If n is odd, $\lceil r \rceil \geq r > \lfloor \frac{n-1}{2} \rfloor = \frac{n-1}{2}$. Since n is an natural number, $n \leq 2\lceil r \rceil$. If n is even, $\lceil \frac{n-1}{2} \rceil > \frac{n-1}{2} > \lfloor \frac{n-1}{2} \rfloor$. Since $\lceil r \rceil \geq r > \lfloor \frac{n-1}{2} \rfloor$, $\lceil r \rceil \geq \lceil \frac{n-1}{2} \rceil > \frac{n-1}{2}$. Therefore, $n \leq 2\lceil r \rceil$ since n is a natural number.

For timeslot t, suppose there are k queues $(\mathcal{Q}_{s_1}, \mathcal{Q}_{s_2}, \cdots, \mathcal{Q}_{s_k})$ which contain jobs available at t, where $s_1 < s_2 < \cdots < s_k$. According to our algorithm, $\ell(\mathcal{AD}, t) \leq k \cdot h$.

Let D_i be the sum of densities of jobs in $\mathcal{Q}_{s_i}$. Consider $t \in \mathcal{I}_{>h}$. By the InsertQueue procedure, $D_i + D_{i+1} > h$ for $1 \leq i < k-1$. Therefore, if $k \geq 3$, $\text{avg}(t) = \sum_{1 \leq i \leq k} D_i > \sum_{1 \leq i \leq k-1} D_i \geq \lfloor \frac{k-1}{2} \rfloor \cdot h$. By our claim, $k \leq 2 \cdot \lceil \frac{\text{avg}(t)}{h} \rceil$. Therefore, $\ell(\mathcal{AD}, t) = k \cdot h \leq 2 \cdot h \cdot \lceil \frac{\text{avg}(t)}{h} \rceil$ for $t \in \mathcal{I}_{>h}$. On the other hand, if $k < 3$, $\ell(\mathcal{AD}, t) \leq 2h < 2 \cdot \text{avg}(t) \leq 2 \cdot h \cdot \lceil \frac{\text{avg}(t)}{h} \rceil$.

For $t \in \mathcal{I}_{\leq h}$, $\text{avg}(t) \leq h$ by definition. That is, the sum of densities of all available jobs at t is no more than h. By the InsertQueue procedure all jobs will be in at most two adjacent queues. Hence, $\ell(\mathcal{AD}, t) \leq 2h$ for $t \in \mathcal{I}_{\leq h}$. □

By Lemma 15 and Lemma 13, we have Theorem 14 by setting $c = 2$ and $c' = 2$.

Theorem 14 *For jobs with uniform height, arbitrary width and agreeable deadlines, $\mathcal{AD}$ is $(\frac{(8\alpha)^\alpha}{2} + 2^\alpha)$-competitive.*

6 Exact Algorithms

In this section, we propose exact algorithms and derive lower bounds on the running time of exact algorithms. Table 2 gives a summary of our results.

Width	Height	Time complexity
Arbitrary	Arbitrary	${w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m} \cdot O(n^2)$
Arbitrary	Arbitrary	$(4m \cdot {w_{\max}}^2)^{2m} \cdot O(n^2)$
Unit	Arbitrary	$2^{O(N)}$

Table 2 Summary of our exact algorithms.

6.1 Fixed parameter algorithms

In parameterized complexity theory, the complexity of a problem is not only measured in terms of the input size, but also in terms of parameters. The theory focuses on situations where the parameters can be assumed to be small and the time complexity depends mainly on these small parameters. The problems having such small parameters are captured by the concept "fixed-parameter tractability". An algorithm with parameters $p_1, p_2, \ldots$ is said to be an *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 parameterized problem is *fixed-parameter tractable* if it can be solved by a fixed parameter algorithm. In this subsection, we show that the general case of GRID problem, jobs with arbitrary release times, deadlines, width and height, is fixed-parameter tractable with respect to a few small parameters.

The number of timeslots τ in the horizon is usually very large. Unlike the brute force algorithm that needs $O(\tau^n)$ time to find an optimal schedule where n is the number of jobs, our algorithm designs remove the dependency on τ in the time complexities. Instead, we use the width of jobs, which is much smaller than τ, to bound the running time. We exploit the concept of "window" and design a fixed parameter algorithm of which the running time depends on the length of windows and the width of jobs. Later, in the second fixed parameter algorithm, we further remove the dependency on the length of windows and leave the width of jobs as the main parameter.

6.1.1 Key notions

We design two fixed parameter algorithms that are based on a dynamic program. Roughly speaking, we divide the timeline into k contiguous windows in a specific way, where each window D_i represents a time interval $[b_i, b_{i+1})$ for $1 \leq i \leq k$. The algorithm visits all windows accordingly from the left to the right and maintains a candidate set of schedules for the visited windows in such a way that no optimal solution is deleted from the set. In the first fixed parameter algorithm, the parameters of the algorithm are the maximum width of jobs, the maximum number of overlapping feasible intervals and the maximum size of windows, where the latter two can be observed on the *interval graph*. We will drop out the last parameter in the second algorithm. All parameters do not increase necessarily as the number of jobs grows and can be assumed to be small in practice. For example, a width of a job is a requested amount of

time to run an appliance and the running time is usually a few hours, which is small when we make a timeslot to be an hour. The number of overlapping feasible intervals is at most the number of appliances, which is also small with respect to one house.

Interval graph. A graph $G = (V, E)$ is an *interval graph* if it is the intersection graph of a set of intervals on the real line. Formally, for each $v \in V$, we can associate v to an interval I_v on the real line such that (u, v) is in E if and only if $I_u \cap I_v \neq \emptyset$. It has been shown in [19, 20] 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 feasible interval $I(J)$, we create a vertex $v(J) \in V$ and there is an edge between $v(J)$ and $v(J')$ if and only if $I(J)$ intersects $I(J')$. The release times and deadlines of the jobs divide the real line into $O(|V|^2)$ intervals. By considering these $O(|V|^2)$ intervals, we can obtain a set of maximal cliques in linear order C_1, C_2, $\cdots$ and C_k, where k denotes the number of maximal cliques. The parameter of our algorithm, the maximum number of overlapping feasible intervals, is the maximum size of these maximal cliques.

Boundaries and windows. Based on the maximal cliques described above, we define some "windows" D_1, D_2, $\cdots$, D_k with "boundaries" b_1, b_2, $\cdots$, b_{k+1}. We first give the definition of boundaries for the first algorithm. This definition will be generalized in Subsubsection 6.1.4 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*. The i-th window D_i is defined as $[b_i, b_{i+1})$.

Example. Figure 1 is an example of a set of jobs, its corresponding interval graph and the corresponding maximal cliques. The cliques are put in such a way that any vertex appears consecutively if there is two or more of it. The boundaries of windows are determined by the leftmost vertex of the maximal cliques.

6.1.2 Framework of the algorithms

We propose two exact algorithms, both of which run in k stages corresponding to each of the k windows. We maintain a table T_{left} that stores all "valid" configurations of jobs in all the windows that have been considered so far. A configuration of a job corresponds to a segment (sub-interval) of its execution interval. A row in the table consists of the configurations of all the jobs. In addition, for each window D_i, we compute a table T_{right_i} to store all possible configurations of jobs available in D_i by keeping their start and end times. The configurations in T_{right_i} would then be "concatenated" to some configurations in T_{left} that are "compatible" with each other. These merged configurations

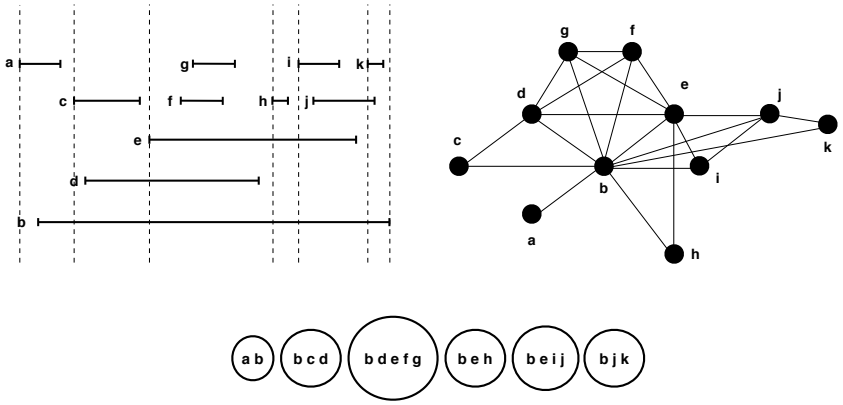

Fig. 1 The figure on the left is a set of jobs, where the horizontal line segments are the feasible time intervals of jobs and the vertical lines are boundaries of windows. The figure on the right is an interval graph of the corresponding job set. The figure at the bottom is a set of all the maximal cliques in the interval graph.

will be filtered to remove the non-optimal ones. The remaining configurations will become the new T_{left} for the next window. To describe the details of the algorithm, we explain several notions below. We denote by D_{left} the union of the windows corresponding to T_{left}. More formally, in the i-th stage, $D_{\text{left}} = \cup_{j<i} D_j$. We use T_{right} to denote T_{right_i} when the context is clear.

Configurations. A *configuration* $F_i(J)$ of job J in window D_i is an execution segment, denoted by $[st_i(J), et_i(J))$ contained completely in D_i. That is, $st_i(J) \in \{b_i, b_i+1, \cdots, b_{i+1}-1\}$ and $et_i(J) \in \{b_i+1, b_i+2, \cdots, b_{i+1}\}$. We also deal with the special cases where jobs are finished before D_i or started after D_i by setting the execution segment $[b_i-1, b_i)$ or $[b_{i+1}, b_{i+1}+1)$, respectively. Also, setting the execution segment as $[b_i - 1, b_{i+1} + 1)$ means J starts execution before D_i, crosses the whole window D_i and ends execution after D_i. We say that *the job J is executed in D_i* if $st_i(J) \in [b_i, b_{i+1})$ and $et_i(J) \in (b_i, b_{i+1}]$. For a collection C of jobs, we use $F_i(C)$ to denote the set of configurations of all jobs in C and $F_{\text{left}}(J)$ and $F_{\text{left}}(C)$ for the counterparts corresponding to T_{left}. The cost of $F_i(C)$ is the cost corresponding to the execution segments in $F_i(C)$. That is, $\text{cost}(F_i(C)) = \sum_{t \in D_i} (\sum_{J \in C : t \in F_i(J)} h(J))^{\alpha}$.

Validity. A configuration $F_i(J)$ is *invalid* if one of the following conditions holds: (i) $st_i(J) \geq et_i(J)$; (ii) $et_i(J) > st_i(J) + w(J)$ meaning that the length of execution segment of J is larger than the width of J; (iii) $((et_i(J) - st_i(J)) < \min\{w(J), b_{i+1} - b_i\}) \wedge (st_i(J) \geq b_i) \wedge (et_i(J) \leq b_{i+1})$ meaning that the length of execution segment of J is strictly shorter than the width of J and the width of the window, which is not acceptable since preemption is not allowed; (iv) $(st_i(J) < r(J)) \wedge (st_i(J) < b_{i+1})$ meaning that the start time of J is earlier than the release time of J; (v) $(et_i(J) > d(J)) \wedge (et_i(J) > b_i)$ meaning that the end time of J exceeds the deadline of J. For $F_{\text{left}}(J)$, the validity is defined on the boundaries b_1 (instead of b_i) and b_{i+1}. For T_{left},

$F_{\text{left}}(J)$ is also invalid if $st_{\text{left}}(J) = b_1 - 1$ since there is no window on the left of D_{left}. Similarly, $F_k(J)$ is invalid if $et_k(J) = b_{k+1} + 1$. A configuration $F_i(C)$ is *invalid* if there exists $J \in C$ such that $F_i(J)$ is invalid.

Compatibility. For job J, the two configurations $F_{\text{left}}(J)$ and $F_i(J)$ are *compatible* if (i) J is executed in D_{left} according to $F_{\text{left}}(J)$ and J is executed before D_i according to $F_i(J)$; (ii) J starts execution in D_{left} and ends execution after D_{left} according to $F_{\text{left}}(J)$ and J starts execution before D_i and ends execution either in D_i or after D_i according to $F_i(J)$; (iii) J is executed completely after D_{left} according to $F_{\text{left}}(J)$ and J does not start before D_i according to $F_i(J)$.

Concatenating configurations. To concatenate two configurations $F_{\text{left}}(J)$ and $F_i(J)$, we create a new $F_{\text{left}}(J)$ by the following setting based on the three types of compatible configurations described in the previous paragraph: for type (i), $st_{\text{left}}(J)$ and $et_{\text{left}}(J)$ stay unchanged; for type (ii), $st_{\text{left}}(J)$ leaves unchanged and set $et_{\text{left}}(J) \leftarrow et_i(J)$; and for type (iii), set $st_{\text{left}}(J) \leftarrow st_i(J)$ and $et_{\text{left}}(J) \leftarrow et_i(J)$. *Concatenating* $F_{\text{left}}(C)$ and $F_i(C)$ is to concatenate the configurations of each job in C. The corresponding cost is simply adding the cost of the two configurations.

Uncertainty and equivalence. A configuration $F_i(J)$ is *uncertain* if $et_i(J) = b_{i+1} + 1$ meaning that the end time of J is not determined yet and we are not sure at the i-th stage whether $F_i(J)$ will be valid after concatenating $F_i(J)$ and $F_{i+1}(J)$. Two configurations $F_i(C)$ and $F'_i(C)$ are *equivalent with respect to uncertain jobs* if (i) $F_i(J)$ is uncertain if and only if $F'_i(J)$ is uncertain for all jobs $J \in C$; and (ii) the start time of $F_i(J)$ is equal to the start time of $F'_i(J)$ for all uncertain configuration $F_i(J)$ and $J \in C$. That is, we only consider the differences among the start times of those jobs with uncertain configurations when we distinguish two configurations of a set of jobs.

6.1.3 An algorithm with three parameters

Algorithm $\mathcal{E}$. The algorithm consists of three components: ListConfigurations, ConcatenateTables and FilterTable. In the algorithm, 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 D_i. We start with T_{left} containing the only configurations which sets $st_0(J) = b_1$ and $et_0(J) = b_1 + 1$ for all jobs J. That is, the configuration treats all the jobs to be not yet executed. Then we visit the windows from the left to the right.

ListConfigurations: For window D_i and jobs in C_i, we construct T_{right} storing all configurations of $J \in C_i$. We enumerate all $st_i(J) \in [b_i, b_{i+1})$ and $et_i(J) \in (b_i, b_{i+1}]$ for each job $J \in C_i$, list all the combinations of all the jobs J with all of its start times and end times and store the results in T_{right} in the way that one row is for one configuration $F_i(C_i)$. In other words, T_{right} stores all the combinations of execution segments in D_i for all jobs $J \in C_i$. We also list the configurations indicating the job being executed before or after D_i. For each configuration $F_i(C_i)$, we also store its cost contribution $\text{cost}(F_i(C_i))$ together. We also check each of the configurations and delete the invalid ones.

ConcatenateTables: We then concatenate compatible configurations in T_{left} and T_{right}. The resulting table is the new T_{left}. More specifically, for each configuration $F_{\text{left}}(C)$ in T_{left} and each configuration $F_{\text{right}}(C)$ in T_{right}, we concatenate $F_{\text{left}}(C)$ and $F_{\text{right}}(C)$ if they are compatible and store the result to a new row in T_{left}. We also check each of the configurations in the new T_{left} and delete those invalid ones.

FilterTable: After concatenation, we filter non-optimal configurations. We select only one representative for the configurations in T_{left} with equivalence relation. More precisely, we only leave the configuration with the lowest cost (choosing anyone to break tie if any) among equivalent configurations. In the current T_{left}, no two configurations are equivalent with respect to uncertain jobs.

After processing all windows, the only configuration in the final T_{left} is returned as the solution. Algorithm 1 is the pseudocode of this algorithm.

Algorithm 1 The fixed parameter algorithm $\mathcal{E}$

Input: a set of job $\mathcal{J}$
Output: an optimal configuration of $\mathcal{J}$
$G \leftarrow$ the interval graph transformed from $\mathcal{J}$
$\{C_i\}_{i=1}^{k} \leftarrow$ the maximal cliques obtained from the consecutive clique arrangement of G
$\{b_i\}_{i=1}^{k+1} \leftarrow$ the boundaries where
  $b_i = \min\{t \mid t = r(J) \text{ and } J \in C_i \setminus (\cup_{s=1}^{i-1} C_s)\}$ and
  $b_{k+1} = \max\{d(J) \mid J \in \mathcal{J}\}$
$\{D_i\}_{i=1}^{k} \leftarrow$ the windows where D_i is bounded by b_i (inclusively) and b_{i+1} (exclusively)
$T_{\text{left}} \leftarrow$ a configuration that labels all jobs $J \in \mathcal{J}$ to be not yet executed
for i from 1 to k **do**
  $T_{\text{right}} \leftarrow$ ListConfigurations(D_i, C_i)
  $T_{\text{left}} \leftarrow$ ConcatenateTables(T_{left}, T_{right})
  $T_{\text{left}} \leftarrow$ FilterTable(T_{left})
return any configuration in T_{left}

Lemma 16 *Algorithm $\mathcal{E}$ outputs an optimal solution.*

Proof In each stage, we list all possible configurations. A configuration is deleted only when it is invalid or it is equivalent with respect to uncertain jobs to another configuration with lower cost. An invalid configuration cannot be optimal since either it is not executed within its feasible interval (invalidity condition (i), (iv) and (v)), it has an execution interval longer than its width (invalidity condition (iii)) or it is preempted (invalidity condition (iii)).

In the rest of the proof we focus on the other case, where a configuration is equivalent with respect to uncertain jobs to another configuration with lower cost. Given two equivalent with respect to uncertain jobs configurations $F_{\text{left}}(C)$ and $F'_{\text{left}}(C)$ with $\text{cost}(F_{\text{left}}(C)) < \text{cost}(F'_{\text{left}}(C))$, we show that $F'_{\text{left}}(C)$ cannot be optimal. Suppose there is an optimal solution F^* containing $F'_{\text{left}}(C)$, which means each execution segment $F'_{\text{left}}(J)$ in $F'_{\text{left}}(C)$ is completely contained by the corresponding execution interval of J in F^*. Since $F_{\text{left}}(C)$ and $F'_{\text{left}}(C)$ are equivalent with respect to uncertain jobs, the start times of J are the same

in the two configurations for all uncertain jobs J. In D_{left}, this means the uncertain jobs do not make the costs of the two configurations to be different and the jobs $\mathcal{J}_c$ that are not uncertain do. Note that $\mathcal{J}_c$ consists of the jobs with their end times being determined. This means we can replace the configurations of $\mathcal{J}_c$ in $F'_{\text{left}}(C)$ by the configurations of $\mathcal{J}_c$ in $F_{\text{left}}(C)$ and this action will not affect the procedures in the algorithm thereafter. However, this also results in a solution of lower cost and contradicts the assumption that F^* is optimal. Thus $F'_{\text{left}}(C)$ cannot be optimal. Therefore, none of the deleted configuration can be part of an optimal schedule. That is, no optimal schedule would be removed through out the whole process. □

Let n be the number of jobs, $w_{\max}$ be the maximum width of jobs, m be the maximum size of cliques, $D_{\max}$ be the maximum length of windows and k be the number of windows. We analyze in the following the time complexity of Algorithm $\mathcal{E}$. We first compute the time complexities for the three components of the algorithm and then compute the total time complexity.

Lemma 17 *The running time of ListConfigurations in Algorithm $\mathcal{E}$ is $O((D_{\max}+1)^{2m+1} \cdot n)$.*

Proof For ListConfigurations, there are $O((D_{\max}+1)^{2m})$ configurations in the output table T_{right}, since there are at most $D_{\max}+1$ possible start times and end times respectively and at most m jobs that should be considered in the current window. For each configuration, it takes $O(n)$ time for construction and validity checking. It also takes $O(nD_{\max})$ to compute the cost of a configuration. So, the time complexity for ListConfigurations is

$$O((D_{\max}+1)^{2m} \cdot nD_{\max}) = O((D_{\max}+1)^{2m+1} \cdot n) \ . \qquad \square$$

Before computing the time complexities of the other components, we focus on the number of configurations of T_{left} at the end of each iteration in the algorithm.

Lemma 18 *There are $O({w_{\max}}^m)$ configurations of T_{left} at the end of each iteration in Algorithm $\mathcal{E}$.*

Proof Since T_{left} is filtered to have no equivalent with respect to uncertain jobs configurations, the number of configurations can be upper bounded. For the jobs with determined end times, their configurations are filtered to have at most 1 configuration for each of the jobs. For the uncertain jobs, the number of configurations depends on the number of different start times. There are at most m uncertain jobs and for each of such the jobs, the number of start times is at most $w_{\max}$, otherwise the difference between the job's start time and end time is larger than its width, which is invalid. Note that the end times of these jobs are all set to be later than the current window and will not affect the number of configurations. So the number of configurations of T_{left} at the end of each iteration is $O({w_{\max}}^m)$, which is the combination of configurations of all the jobs. □

Lemma 19 *The running time of ConcatenateTables in Algorithm $\mathcal{E}$ is $O({w_{\max}}^m \cdot (D_{\max}+1)^{2m} \cdot n)$.*

Proof For ConcatenateTables, there are $O({w_{\max}}^m \cdot (D_{\max}+1)^{2m})$ configurations in the output table T_{left}. This is because for each configuration in the input T_{left} (with $O({w_{\max}}^m)$ configurations in total), we need to compare it with all the configurations in T_{right} (with $O((D_{\max}+1)^{2m})$ configurations in total) for compatibility checking. For each configuration, it takes $O(n)$ time for compatibility checking, concatenation and validity checking. Thus the time complexity for ConcatenateTables is $O({w_{\max}}^m \cdot (D_{\max}+1)^{2m} \cdot n)$. □

Lemma 20 *The running time of FilterTable in Algorithm $\mathcal{E}$ is $O({w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m} \cdot n)$.*

Proof For FilterTable, the number of configurations in the output table T_{left} is at most the number of configurations output by ConcatenateTables. Also, the number of equivalence groups is at most its number of configurations. Thus it takes

$$O([{w_{\max}}^m \cdot (D_{\max}+1)^{2m}]^2 \cdot n) = O({w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m} \cdot n)$$

time for classification, which is quadratic time in terms of the size of the table and $O(n)$ time for each configuration. In addition, it takes $O({w_{\max}}^m \cdot (D_{\max}+1)^{2m})$ time for deletion. So the time complexity for FilterTable is $O({w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m} \cdot n)$. □

Since there are k iterations, the total time complexity is $O(k \cdot {w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m} \cdot n)$. Thus we obtain the following theorem.

Theorem 15 *Algorithm $\mathcal{E}$ computes an optimal solution in $O(k \cdot {w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m} \cdot n)$ time, where n is the number of jobs, $w_{\max}$ is the maximum width of jobs, m is the maximum size of cliques, $D_{\max}$ is the maximum length of windows and k is the number of windows.*

There are $O(n)$ windows. So Algorithm $\mathcal{E}$ also runs in $f(w_{\max}, m, D_{\max}) \cdot O(n^2)$ time where $f(w_{\max}, m, D_{\max}) = {w_{\max}}^{2m} \cdot (D_{\max}+1)^{4m}$.

Corollary 6 GRID *problem is fixed parameter tractable with respect to the maximum width of jobs, the maximum number of overlapping feasible intervals and the maximum length of windows.*

6.1.4 An algorithm with two parameters

This subsubsection describes how to drop out the parameter $D_{\max}$ in the previous algorithm by generalizing the definitions of windows and boundaries.

At the beginning of Algorithm $\mathcal{E}$, we transform a set of jobs to its corresponding interval graph and obtain a sequence of windows by the set of maximal cliques in the interval graph. We require in the algorithm that all the

cliques should be maximal. However, the algorithm is still optimal and has parameterized bound of time complexity if we divide a maximal clique into multiple non-maximal cliques in a specific way. Given a maximal clique C_i and its corresponding window D_i, we divide D_i into a set of contiguous windows $D_{i_1}, D_{i_2}, \ldots$ such that $D_i = \cup_j D_{i_j}$. Note that the set of jobs C_{i_j} corresponding to D_{i_j} is a clique in the interval graph since C_i is a clique and $C_{i_j} \subseteq C_i$. In this way, the number of jobs in the window D_{i_j} is still at most m. Furthermore, since this window division does not affect the proof of Lemma 16, the algorithm is still optimal. Thus we have the following observation.

Observation 16 *If the windows $\{D_i\}_{i=1}^{k}$ in Algorithm $\mathcal{E}$ cover all the jobs and is contiguous and the jobs in each D_i represents a clique (not necessarily maximal) in the interval graph of the input jobs, then (1) the algorithm outputs an optimal solution; (2) the number of jobs in each D_i is at most the maximum number of overlapping feasible intervals.*

To drop out the parameter $D_{\max}$ in the previous algorithm, we divide windows into smaller ones such that the number of configurations in a window can be bounded by $w_{\max}$ and m. In the new algorithm, we set the locations of boundaries to the release times and deadlines of all the jobs and construct the windows based on these boundaries. In this way, there is no job being released or having its deadline in the middle of a window. In the worst case, all jobs in a window are scheduled such that no job overlaps another and these jobs consume at most $m \cdot w_{\max}$ timeslots. Thus, the number of used timeslots is at most $m \cdot w_{\max} + 2(w_{\max} - 1)$. In addition, we need to consider the cases that a job's start time is earlier than the window or its deadline is later than the window. Both cases consume at most $w_{\max} - 1$ timeslots respectively. Note that this new window division results in a set of windows whose sizes are smaller than their original counterparts and thus Observation 16 can be applied. Based on this new window division, we have the following algorithm.

Algorithm $\mathcal{E}^+$. This algorithm is similar to Algorithm $\mathcal{E}$ except the definitions of boundaries and the component 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. Let k denote the number of windows. There are k stages for the algorithm. At the i-th stage, the algorithm runs ListConfigurations, ConcatenateTables and FilterTable accordingly as Algorithm $\mathcal{E}$ does. It finally outputs the only configuration in T_{left}. For the component ListConfigurations, we only consider to schedule jobs on the timeslots used instead of enumerating all possibilities of start times and end times. The algorithm tries all $m \cdot w_{\max}$ timeslots (the worst case described in the previous paragraph) as the start time of a job and also the $2(w_{\max} - 1)$ schedules that a job is partially executed in the window. In addition, the component shall include the cases that either a job is completely executed before the window, it is completely executed after the window or it crosses the window.

Algorithm 2 The fixed parameter algorithm $\mathcal{E}^+$

Input: a set of job $\mathcal{J}$
Output: an optimal configuration of $\mathcal{J}$
$(b_i)_{i=1}^{k+1} \leftarrow$ the non-decreasing boundaries consisting of $\{r(J) \mid J \in \mathcal{J}\} \cup \{d(J) \mid J \in \mathcal{J}\}$
$\{D_i\}_{i=1}^{k} \leftarrow$ the windows where D_i is bounded by b_i (inclusively) and b_{i+1} (exclusively)
$\{C_i\}_{i=1}^{k} \leftarrow$ the collection of job sets where C_i is the set of jobs in D_i
$T_{\text{left}} \leftarrow$ a configuration that labels all jobs $J \in \mathcal{J}$ to be not yet executed
for i from 1 to k **do**
  $T_{\text{right}} \leftarrow$ ListConfigurations(D_i, C_i)
  $T_{\text{left}} \leftarrow$ ConcatenateTables(T_{left}, T_{right})
  $T_{\text{left}} \leftarrow$ FilterTable(T_{left})
return any configuration in T_{left}

Theorem 17 *Algorithm $\mathcal{E}^+$ computes an optimal solution in $f(w_{\max}, m) \cdot O(n^2)$ time, where n is the number of jobs, $w_{\max}$ is the maximum width of jobs, m is the maximum size of cliques and $f(w_{\max}, m) = (4m \cdot {w_{\max}}^2)^{2m}$.*

Proof As in the proof of Theorem 15, we compute the running time of the three components and then the total time complexity. For the component ListConfigurations, there are at most $(m \cdot w_{\max} + 2(w_{\max} - 1) + 3)^m$ output configurations, since there are at most $m \cdot w_{\max} + 2(w_{\max} - 1) + 3$ schedules for a job (see the description in the previous paragraph) and at most m jobs in a window. It takes $O(n(m \cdot w_{\max} + 2(w_{\max} - 1))) \leq O(n \cdot m \cdot w_{\max})$ time to compute the cost for each configuration. Thus the time complexity for ListConfigurations is at most

$$O((m \cdot w_{\max} + 2(w_{\max} - 1) + 3)^m \cdot (n \cdot m \cdot w_{\max})) \leq O((4m \cdot w_{\max})^{m+1} \cdot n) \ .$$

The time complexities of ConcatenateTables and FilterTable are similar to that in the proof of Theorem 15 except the number of output configurations. For ConcatenateTables and FilterTable, both the number of output configurations are at most ${w_{\max}}^m \cdot (4m \cdot w_{\max})^m$. Thus their running time is at most $O({w_{\max}}^{2m} \cdot (4m \cdot w_{\max})^{2m} \cdot n)$. Since there are $k = O(n)$ iterations, the total time complexity of the algorithm is at most

$$O((4m \cdot {w_{\max}}^2)^{2m} \cdot n^2) = f(w_{\max}, m) \cdot O(n^2) \ . \qquad \square$$

Corollary 7 GRID *problem is fixed parameter tractable with respect to the maximum width of jobs and the maximum number of overlapping feasible intervals.*

6.2 An exact algorithm without parameter

For the case with unit width and arbitrary height jobs of GRID problem, we can use Algorithm $\mathcal{E}$ to design an exact algorithm whose time complexity is only measured in the size of the input.

Algorithm $\mathcal{E}^{\mathcal{V}}$. In Algorithm $\mathcal{E}$, we maintain two tables T_{left} and T_{right} for each stage. At each stage, the core operations are to construct T_{right}, merge

T_{left} and T_{right} and filter the resulting table. In the case with unit width and arbitrary height jobs, one may observe that the functionalities of these core operations are not affected by the length of the windows representing T_{left} and T_{right}. For example, we can restrict the window length to be a constant but not be related to the cliques in the interval graph and the algorithm still works correctly. By fixing the lengths of all windows, a new exact algorithm is obtained. Without loss of generality, we assume that the number of timeslots τ is even. We enforce all windows to have length 2, i.e., we have $\tau/2$ windows in total. By this setting, the new algorithm runs in $O((\tau/2) \cdot 4^{2n} \cdot n)$ time where n is the number of jobs. This is because the numbers of configurations for ListConfigurations, ConcatenateTables and FilterTable are at most 4^n, 4^{2n} and 4^n respectively. Consider a window with length 2, there are 4 possible configurations for each job. We can schedule a job either at the first timeslot, at the second timeslot, consider the job to be executed before the window or be executed after the window. 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.

Algorithm 3 The fixed parameter algorithm $\mathcal{E}^{\mathcal{V}}$

Input: a set of job $\mathcal{J}$
Output: an optimal configuration of $\mathcal{J}$
$\{b_i\}_{i=1}^{\tau/2+1} \leftarrow$ the boundaries $\{0, 2, 4, 6, \ldots, \tau\}$
$\{D_i\}_{i=1}^{\tau/2} \leftarrow$ the windows where D_i is bounded by b_i (inclusively) and b_{i+1} (exclusively)
$\{C_i\}_{i=1}^{\tau/2} \leftarrow$ the collection of job sets where C_i is the set of jobs in D_i
$T_{\text{left}} \leftarrow$ a configuration that labels all jobs $J \in \mathcal{J}$ to be not yet executed
for i from 1 to $\tau/2$ **do**
    $T_{\text{right}} \leftarrow$ ListConfigurations(D_i, C_i)
    $T_{\text{left}} \leftarrow$ ConcatenateTables(T_{left}, T_{right})
    $T_{\text{left}} \leftarrow$ FilterTable(T_{left})
return any configuration in T_{left}

Theorem 18 *There is an exact algorithm running in $2^{O(N)}$ time for the* GRID *problem with unit width and arbitrary height jobs where N is the length of the input encoding.*

Our algorithm is highly more efficient than a brute force search. Such a naive method would enumerate all possible schedules and check if they are feasible and optimal, which requires $O(\tau^n n)$ time. The running time can be rewritten as $2^{O(Nn)}$ or more clearly, $(2^{O(N)})^n$. The exact algorithm modified from our fixed parameter algorithm indeed crosses out an 'n' in the exponent.

6.3 Lower bound on the running time

This subsection provides two lower bounds on the running time of the GRID problem under a certain condition.

Jansen et al. [23] 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. By a reduction from PARTITION, we obtain the following theorem.

Theorem 19 *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.*

Proof We design a strong reduction from PARTITION to the decision version of GRID problem with unit width and arbitrary height jobs. Recall that PARTITION is a decision problem that is to decide if a given set S of integers can be partitioned into two disjoint subsets such that the two subsets have equal sum. The reduction works as follows. For each integer $s \in S$, 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 J can be scheduled with cost at most $2(\sum_{s\in S} s)^{\alpha}$. Note that the specified cost appears when jobs can be put into two timeslots with equal loads. By setting the length of the input as the parameter, we observe that the parameter increases at most linearly from PARTITION to our problem. (Note that a strong reduction from PARTITION to the case with unit height and arbitrary width jobs can be done similarly and the results also apply to that case.) Furthermore, we can choose the number of jobs as a parameter of the problem. Note that the reduction above does not increase this parameter with respect to the parameter of PARTITION, which is the number of input integers. □

7 Minimizing peak and non-preemptive machine minimization

In this section, we investigate extension of our solutions to other objectives and other problems. In particular, we consider the objective of minimizing peak electricity cost in the GRID model and we name it the $\mathsf{GRID}_{\mathsf{peak}}$ problem. This objective is equivalent to a limit case of the GRID model where α approaches to infinity. We also consider the classical non-preemptive machine minimization problem denoted as MACHINE.

The $\mathsf{GRID}_{\mathsf{peak}}$ problem. The input is the same as the GRID problem. The goal is to find a feasible non-preemptive schedule such that the maximum load over the time horizon is minimized. $\mathsf{GRID}_{\mathsf{peak}}$ has been proven to be NP-hard [44] and approximation algorithms are known for requests having

common feasible interval with approximation ratio 4 [50] and for requests having agreeable deadlines with approximation ratio $O(\log \frac{w_{\max}}{w_{\min}})$ [50].

The MACHINE problem. The input is a set of jobs each with a processing time $p(J)$, release time $r(J)$ and deadline $d(J)$. Each job has to be scheduled non-preemptively on one of the (infinite number of) machines. For each machine, at most one job can be executed at any time. The goal is to minimize the number of machines used.

The MACHINE problem can be seen as a special case of the $\mathsf{GRID}_{\mathsf{peak}}$ problem where jobs have unit height. A lower bound of $\log_3 \frac{w_{\max}}{w_{\min}}$ on the competitive ratio of any online algorithm has been shown in [41]. As a result, this lower bound also applies to $\mathsf{GRID}_{\mathsf{peak}}$. The author also provided an $O(\log \frac{p_{\max}}{p_{\min}})$-competitive algorithm for the MACHINE problem. Our results are similar to theirs. However, for the case of tight jobs, their algorithm blows up 10 times the optimal cost, whereas our algorithm blows up only 3 times the optimal cost.

In this section, we show that our online algorithm solves the $\mathsf{GRID}_{\mathsf{peak}}$ problem with a best-possible competitive ratio in an asymptotical sense and provide an alternative asymptotically best-possible competitive algorithm for the MACHINE problem.

7.1 Online algorithms

In this section, we prove that the online algorithm $\mathcal{G}$ proposed in Section 4 is asymptotically best-optimal for the $\mathsf{GRID}_{\mathsf{peak}}$ problem. We first state two properties, one for $\mathcal{BKP}'$ that $\mathcal{G}$ is based on and the other for the optimal solution w.r.t. the peak objective. Let function $\mathrm{peak}(S)$ denote the maximum load (cf. speed) of any schedule S, i.e., $\mathrm{peak}(S) = \max_t \ell(S,t)$. Combining Lemma 4 and the fact that $\mathcal{BKP}$ is e-competitive w.r.t. maximum speed [4], we have the following observation.

Observation 20 *The $\mathcal{BKP}'$ algorithm is $e(1+e)$-competitive with respect to maximum speed.*

Proof By Lemma 4, for any integral t and $0 < \Delta < 1$, $\ell(\mathcal{BKP}, t+\Delta) \leq (1+e) \cdot \ell(\mathcal{BKP}, t)$. Also, by our modification for $\mathcal{BKP}'$, $\ell(\mathcal{BKP}', t) = (1+e)\ell(\mathcal{BKP}, t)$. Hence, $\mathrm{peak}(\mathcal{BKP}') \leq (1+e) \cdot \mathrm{peak}(\mathcal{BKP})$. □

On the other hand, $\mathcal{YDS}$ guarantees that the maximum speed is minimized [4]. Similar to Observation 1, $\mathcal{YDS}$ gives a lower bound for the $\mathsf{GRID}_{\mathsf{peak}}$ problem.

Observation 21 *Let $\mathcal{O}_D$ and $\mathcal{O}_G$ be the optimal schedule for the* DVS *and* $\mathsf{GRID}_{\mathsf{peak}}$ *problem, respectively. Suppose a job set $\mathcal{J}_G$ for the* $\mathsf{GRID}_{\mathsf{peak}}$ *problem is given. Let $\mathcal{J}_D$ denote the job set after converting $\mathcal{J}_G$ into a job set for the* DVS *problem. Then, peak$(\mathcal{O}_D(\mathcal{J}_D)) \leq$ peak$(\mathcal{O}_G(\mathcal{J}_G))$.*

Proof Since the DVS problem allows preemption while the $\mathsf{GRID}_{\mathsf{peak}}$ problem does not, any feasible schedule of the $\mathsf{GRID}_{\mathsf{peak}}$ problem with input job set $\mathcal{J}_G$ can be transform to a feasible schedule S_D of the DVS problem with the corresponding job set $\mathcal{J}_D$. Hence, $\text{peak}(\mathcal{O}_G(\mathcal{J}_G)) = \text{peak}(S_D) \geq \text{peak}(\mathcal{O}_D(\mathcal{J}_D))$.

Recall that in Sections 3 and 4, we have presented three algorithms $\mathcal{V}$, $\mathcal{UV}$ and $\mathcal{G}$ for unit width jobs, uniform width jobs and arbitrary width jobs, respectively. Here, we analyze their performance w.r.t. $\mathsf{GRID}_{\mathsf{peak}}$. In summary, we show that for $\mathsf{GRID}_{\mathsf{peak}}$, we have

- $\mathcal{V}$ is $2(e+e^2)$-competitive for uniform width job sets (Theorem 22);
- $\mathcal{UV}$ is $(6(e+e^2)+3)$-competitive for unit width job sets (Theorem 23); and
- $\mathcal{G}$ is $(18(e+e^2)+9)\cdot\lceil\log\frac{w_{\max}}{w_{\min}}\rceil$-competitive for arbitrary width job sets (Theorem 24).

7.1.1 Unit width jobs.

Recall that for each timeslot t, $\mathcal{V}$ schedules jobs to start at t such that $\ell(\mathcal{V},t)$ is at least $\ell(\mathcal{BKP}',t) = (1+e)\cdot\ell(\mathcal{BKP},t)$ or until all available jobs have been scheduled. We prove that although $\ell(\mathcal{V},t)$ might be higher than $\ell(\mathcal{BKP}',t)$, the peak of $\mathcal{V}$ is no more than $2(e+e^2)$ times of the peak of the optimal.

Theorem 22 *For any job set $\mathcal{J}$ where each job has unit width, peak$(\mathcal{V}(\mathcal{J})) \leq 2(e+e^2)\cdot$ peak$(\mathcal{O}(\mathcal{J}))$.*

Proof Let $h_{\max}(\mathcal{V},t)$ be the maximum height of jobs scheduled at t by $\mathcal{V}$; we set $h_{\max}(\mathcal{V},t)=0$ if $\mathcal{V}$ assigns no job at t. We classify each timeslot t into two types: (i) $h_{\max}(\mathcal{V},t) < \ell(\mathcal{BKP}',t)$ and (ii) $h_{\max}(\mathcal{V},t) \geq \ell(\mathcal{BKP}',t)$. 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.)

We first prove that for any job set $\mathcal{J}$ where for each job $J\in\mathcal{J}$, $w(J)=1$, $\text{peak}(\mathcal{V}(\mathcal{J}),\mathcal{I}_1) \leq 2(e+e^2)\cdot\text{peak}(\mathcal{O}(\mathcal{J}))$ and $\text{peak}(\mathcal{V}(\mathcal{J}),\mathcal{I}_2) \leq 2\cdot\text{peak}(\mathcal{O}(\mathcal{J}))$. For every timeslot $t\in\mathcal{I}_1$, $\ell(\mathcal{V},t) < \ell(\mathcal{BKP}',t) + h_{\max}(\mathcal{V},t) \leq 2\cdot\ell(\mathcal{BKP}',t) \leq 2(1+e)\cdot\ell(\mathcal{BKP},t)$. Hence, for any $t\in\mathcal{I}_1$, $\ell(\mathcal{V},t) \leq 2(1+e)\cdot\ell(\mathcal{BKP},t) \leq 2e(1+e)\cdot\text{peak}(\mathcal{O})$ by Observations 21 and 20.

For every timeslot $t\in\mathcal{I}_2$, $\ell(\mathcal{V},t) < \ell(\mathcal{BKP}',t) + h_{\max}(\mathcal{V},t) \leq 2\cdot h_{\max}(\mathcal{V},t)$. In the optimal schedule, the job with height $h_{\max}(\mathcal{V},t)$ has to be scheduled somewhere or the schedule is not feasible, so $\text{peak}(\mathcal{O}) \geq h_{\max}(\mathcal{V},t)$. Hence, $\ell(\mathcal{V},t) \leq 2h_{\max}(\mathcal{V},t) \leq 2\cdot\text{peak}(\mathcal{O})$ for any $t\in\mathcal{I}_2$.

Since $\mathcal{I}_1$ and $\mathcal{I}_2$ are disjoiont, $\text{peak}(\mathcal{V}) = \max\{\text{peak}(\mathcal{V},\mathcal{I}_1),\text{peak}(\mathcal{V},\mathcal{I}_2)\}$. Therefore, we have $\text{peak}(\mathcal{V}) = \max\{2(e+e^2)\cdot\text{peak}(\mathcal{O}), 2\cdot\text{peak}(\mathcal{O})\} = 2(e+e^2)\cdot\text{peak}(\mathcal{O})$. □

7.1.2 Uniform-width jobs.

In this subsubsection, we consider the jobs with uniform width w. Recall that in the algorithm for GRID for handling uniform-width jobs, we classify jobs

into tight and loose jobs. Let $\mathcal{J}^*$ denote the input set with uniform width jobs, $\mathcal{J}_{\mathrm{T}}^*$ and $\mathcal{J}_{\mathrm{L}}^*$ denote the set of tight jobs and loose jobs in $\mathcal{J}^*$, respectively. We first prove that any feasible schedule for tight jobs is 3-competitive due to the "inflexibility" (Lemma 21). Then, we prove that $\mathcal{UV}$ is $O(1)$-competitive for loose jobs (Lemma 22).

Lemma 21 *For any feasible schedule S, peak$(S(\mathcal{J}_T^*)) \leq 3 \cdot$ peak$(\mathcal{O}(\mathcal{J}^*))$.*

Proof We prove it by showing that even if the execution intervals of jobs are considered as the whole feasible interval, the peak is not too much larger than the peak in the optimal schedule.

We first *extend* jobs $J \in \mathcal{J}_{\mathrm{T}}^*$ to J^+ as follows: $r(J^+) = r(J)$, $d(J^+) = d(J)$, $w(J^+) = d(J) - r(J)$ and $h(J^+) = h(J)$. That is, every job has its width as the length of its feasible interval. We denote the resulting job set by $\mathcal{J}^+$. Since each job in $\mathcal{J}^+$ are not shiftable, there is only one feasible schedule for $\mathcal{J}^+$ and it is optimal. It is clear that $\mathrm{peak}(S(\mathcal{J}_{\mathrm{T}}^*)) \leq \mathrm{peak}(\mathcal{O}(\mathcal{J}^+))$.

Similar to Lemma 7 (i), we bound the load at time t of $\mathcal{O}(\mathcal{J}^+)$ by the total load of a constant number of timeslots in $S(\mathcal{J}_{\mathrm{T}}^*)$. Consider the job J corresponding to J^+, the execution interval of J in any feasible schedule must contain either timeslot $t-(w-1)$, $t+(w-1)$ or t. Hence, we can upper bound the load at time t in $\mathcal{O}(\mathcal{J}^+)$ as follows: $\ell(\mathcal{O}(\mathcal{J}^+), t) \leq \ell(\mathcal{O}(\mathcal{J}_{\mathrm{T}}^*), t-(w-1)) + \ell(\mathcal{O}(\mathcal{J}_{\mathrm{T}}^*), t+(w-1)) + \ell(\mathcal{O}(\mathcal{J}_{\mathrm{T}}^*), t)$. Hence, $\mathrm{peak}(S(\mathcal{J}_{\mathrm{T}}^*)) \leq \mathrm{peak}(\mathcal{O}(\mathcal{J}^+)) \leq 3 \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}_{\mathrm{L}}^*))$. □

Lemma 22 *For sets of loose jobs $\mathcal{J}_L^*$ where jobs have uniform width, peak$(\mathcal{UV}(\mathcal{J}_L^*))$ $\leq 6(e+e^2) \cdot$ peak$(\mathcal{O}(\mathcal{J}^*))$.*

Proof According to $\mathcal{UV}$, the job set $\mathcal{J}_{\mathrm{L}}^*$ is transformed into a job set $\mathcal{J}'$ by AlignFI and $\mathcal{V}$ is run on $\mathcal{J}'$. Then, the schedule $\mathcal{V}(\mathcal{J}')$ is transformed to a schedule of $\mathcal{J}_{\mathrm{L}}^*$ by Transformation FreeSch. Hence, by Theorem 22, $\mathrm{peak}(\mathcal{UV}(\mathcal{J}_{\mathrm{L}}^*)) \leq \mathrm{peak}(\mathcal{V}(\mathcal{J}')) \leq 2(e+e^2) \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}'))$.

Given an optimal schedule of $\mathcal{J}_{\mathrm{L}}^*$, $\mathcal{O}(\mathcal{J}_{\mathrm{L}}^*)$, let S' be the resulting schedule by running Transformation AlignSch on $\mathcal{J}_{\mathrm{L}}^*$. By Observation 4 (iii), the load at any time in the schedule S' is at most the sum of the load of the schedule $\mathcal{O}(\mathcal{J}_{\mathrm{L}}^*)$ at three timeslots. Therefore, $\mathrm{peak}(\mathcal{O}(\mathcal{J}')) \leq \mathrm{peak}(S') \leq 3 \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}_{\mathrm{L}}^*))$.

In summary, $\mathrm{peak}(\mathcal{UV}(\mathcal{J}_{\mathrm{L}}^*)) \leq 2(e+e^2) \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}')) \leq 6(e+e^2) \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}_{\mathrm{L}}^*))$. □

Theorem 23 *For any jobs set $\mathcal{J}^*$ where jobs have uniform width, peak$(\mathcal{UV}(\mathcal{J}^*))$ $\leq (6(e+e^2)+3) \cdot$ peak$(\mathcal{O}(\mathcal{J}^*))$.*

Proof By definition, $\mathrm{peak}(\mathcal{UV}(\mathcal{J}^*)) \leq \mathrm{peak}(\mathcal{UV}(\mathcal{J}_{\mathrm{T}}^*)) + \mathrm{peak}(\mathcal{UV}(\mathcal{J}_{\mathrm{L}}^*))$. By Lemma 21 and 22, $\mathrm{peak}(\mathcal{UV}(\mathcal{J}^*)) \leq 3 \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}^*)) + 6(e+e^2) \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}^*)) = (6(e+e^2)+3) \cdot \mathrm{peak}(\mathcal{O}(\mathcal{J}^*))$. □

7.1.3 Arbitrary width jobs.

Finally, we bound the competitive ratio of $\mathcal{G}$.

Theorem 24 *For any job set $\mathcal{J}$, peak$(\mathcal{G}(\mathcal{J})) \leq (18(e+e^2)+9) \cdot \lceil \log \frac{w_{\max}}{w_{\min}} \rceil \cdot$ peak$(\mathcal{O}(\mathcal{J}))$.*

Proof Recall that $\mathcal{G}(\mathcal{J})$ partitions jobs into subsets $\mathcal{J}_p$ such that in each $\mathcal{J}_p$ jobs have bounded widths. For each $\mathcal{J}_p$, it is transformed into $\mathcal{J}_p^*$ by Convert and $\mathcal{UV}$ is applied independently on each class. Then, $\mathcal{UV}(\mathcal{J}_p^*)$ is transformed into a schedule for $\mathcal{J}_p$ by Transformation ShrinkSch. By Observation 9, $\ell(\mathcal{G}(\mathcal{J}_p), t) \leq \ell(\mathcal{UV}(\mathcal{J}_p^*), t)$. Hence, $\ell(\mathcal{G}(\mathcal{J}), t) = \sum_p \ell(\mathcal{G}(\mathcal{J}_p), t) \leq \sum_p \ell(\mathcal{UV}(\mathcal{J}_p^*), t)$ for all t.

For any timeslot t, $\ell(\mathcal{G}(\mathcal{J}), t) \leq \sum_p \ell(\mathcal{UV}(\mathcal{J}_p^*), t) \leq \sum_p \text{peak}(\mathcal{UV}(\mathcal{J}_p^*))$. By Theorem 23, we have $\text{peak}(\mathcal{G}(\mathcal{J})) \leq \sum_p (6(e+e^2)+3) \cdot \text{peak}(\mathcal{O}(\mathcal{J}_p^*))$.

Now we prove that $\text{peak}(\mathcal{O}(\mathcal{J}_p^*)) \leq 3 \cdot \text{peak}(\mathcal{O}(\mathcal{J}))$. Consider the optimal schedule $\mathcal{O}(\mathcal{J}_p)$, there exists schedule $S(\mathcal{J}_p^*)$ generated by Transformation RelaxSch where $\mathcal{J}_p^*$ is the job set corresponding to $\mathcal{J}_p$ generated by Convert. By Lemma 8, the load of any timeslot in $S(\mathcal{J}_p^*)$ is no more than the sum of loads of three timeslots in $\mathcal{O}(\mathcal{J}_p)$ and hence no more than three times the peak in $\mathcal{O}(\mathcal{J}_p)$. Therefore, $\text{peak}(\mathcal{O}(\mathcal{J}_p^*)) \leq \text{peak}(S(\mathcal{J}_p^*)) \leq 3 \cdot \text{peak}(\mathcal{O}(\mathcal{J}_p)) \leq 3 \cdot \text{peak}(\mathcal{O}(\mathcal{J}))$.

In summary, we have $\text{peak}(\mathcal{G}(\mathcal{J})) \leq \sum_p (6(e+e^2)+3) \cdot \text{peak}(\mathcal{O}(\mathcal{J}_p^*)) \leq \sum_p (6(e+e^2)+3) \cdot 3 \cdot \text{peak}(\mathcal{O}(\mathcal{J})) \leq (18(e+e^2)+9) \cdot \lceil \log \frac{w_{\max}}{w_{\min}} \rceil \cdot \text{peak}(\mathcal{O}(\mathcal{J}))$.□

Since the MACHINE problem is a special case of the $\mathsf{GRID}_{\mathsf{peak}}$ problem where jobs have uniform height, we have the following corollary:

Corollary 8 *The algorithm $\mathcal{G}$ is $(18(e+e^2)+9)$-competitive for the* MACHINE *problem.*

7.2 The interval graph approach on the $\mathsf{GRID}_{\mathsf{peak}}$ problem

In Subsection 6.1, we introduced an exact algorithm $\mathcal{E}$ using the *linear clique arrangement* property of the interval graphs. The linear property of the consecutive clique arrangement of interval graphs gives a direction to design a dynamic programming algorithm, which breaks down a problem into overlapped subproblems until the subproblems are simple enough to be solved. In the following of this section, we show that there is an exact algorithm $\mathcal{E}_{peak}$ for solving the $\mathsf{GRID}_{\mathsf{peak}}$ problem by adapting $\mathcal{E}$. In fact, $\mathcal{E}_{peak}$ is almost the same as $\mathcal{E}$ except the cost entries in the DP table: in $\mathcal{E}_{peak}$, each configuration is associated with the highest load in the corresponding window. The following is a formal description of $\mathcal{E}_{peak}$, where the parts different from $\mathcal{E}$ are emphasized.

Algorithm $\mathcal{E}_{peak}$ for the $\mathsf{GRID}_{\mathsf{peak}}$ problem (also see Subsection 6.1). The jobs are considered as time intervals and the time horizon is chopped into "windows". The algorithm visits all windows accordingly from the left to the right. In Stage i, the i-th window is visited and the algorithm maintains a candidate set of schedules for the visited windows that no optimal solution is deleted from the set. In each Stage i, the algorithm consists of three procedures: ListConfigurations, ConcatenateTables and FilterTable.

The ListConfigurations procedure lists all possible configurations (i.e., execution segments) of the jobs in C_i within D_i. The invalid configurations will be deleted. The valid configurations together with *the peak load within* D_i will be stored in a table.

The ConcatenateTables procedure concatenates the configurations in the current window D_i and the configurations in the windows which have been seen so far. If the execution interval after concatenation is not valid, it is deleted from the table. *The peak load of the new configuration is simply the maximum among the peaks of the two concatenated configurations.*

The FilterTable procedure filters non-optimal configurations. The idea is, given a configuration of the jobs in C_i, there must be a best decision of the jobs in $\bigcup_{k=1}^{i-1} C_k \setminus C_i$ which has minimum *peak load* within the intervals $[0, b_{i+1})$, where b_{i+1} is the right boundary of the window D_i. For each configuration, we only keep the (partial) schedule with the minimum *peak load*.

After processing all the windows, the schedule with minimum *peak load* can be found in the final table.

It can be seen that we list all possible configurations. A configuration is deleted only when it is invalid or it is identical to another configuration with lower peak. Hence in the end we get an optimal schedule. It also shows that the $\mathsf{GRID}_{\mathsf{peak}}$ problem is fixed parameter tractable with respect to the maximum width of jobs and the maximum number of overlapped feasible intervals and the maximum length of windows.

Corollary 9 *The* $\mathsf{GRID}_{\mathsf{peak}}$ *problem is fixed parameter tractable with respect to the maximum width of jobs and the maximum number of overlapped feasible intervals.*

8 Conclusion

We develop the first online algorithm with polylog-competitive ratio and the first FPT algorithms for non-preemptive smart grid scheduling problem in general case. We also derive a matching lower bound for the competitive ratio. Constant competitive online algorithms are presented for several special input instances. We remark that recently another work [8] has studied the problem with constant α and the authors show an $O(\alpha^\alpha)$-competitive algorithm for this case, matching our lower bound for constant α. In this case this algorithm has better competitive ratio than our algorithm yet our algorithm is more efficient as the scheduling decision of each job takes constant time whereas the algorithm in [8] needs to enumerate all its possible start times and the latter can be large for loose job with long feasible interval.

There are quite a few directions for extending the problem setting: different cost functions perhaps to capture varying electricity cost over the time of day; jobs with varying power requests during its execution (it is a constant value in this paper); other objectives like response time.

Acknowledgements The work is partially supported by Networks Sciences and Technologies, University of Liverpool. Hsiang-Hsuan Liu is partially supported by a studentship from the University of Liverpool-National Tsing-Hua University Dual PhD programme.

References

1. Alamdari, S., Biedl, T., Chan, T.M., Grant, E., Jampani, K.R., Keshav, S., Lubiw, A., Pathak, V.: Smart-grid electricity allocation via strip packing with slicing. In: WADS, pp. 25–36. Springer (2013)
2. Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5), 86–96 (2010)
3. Azar, Y.: On-line load balancing. In: Online Algorithms, pp. 178–195. Springer (1998)
4. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM **54**(1) (2007)
5. Bell, P.C., Wong, P.W.H.: Multiprocessor speed scaling for jobs with arbitrary sizes and deadlines. J. Comb. Optim. **29**(4), 739–749 (2015)
6. Burcea, M., Hon, W., Liu, H., Wong, P.W.H., Yau, D.K.Y.: Scheduling for electricity cost in a smart grid. J. Scheduling **19**(6), 687–699 (2016)
7. Caron, S., Kesidis, G.: Incentive-based energy consumption scheduling algorithms for the smart grid. In: SmartGridComm, pp. 391–396. IEEE (2010)
8. Chau, V., Feng, S., Thang, N.K.: Competitive algorithms for demand response management in smart grid. In: M.A. Bender, M. Farach-Colton, M.A. Mosteiro (eds.) LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 10807, pp. 303–316. Springer (2018). DOI 10.1007/978-3-319-77404-6_23. URL `https://doi.org/10.1007/978-3-319-77404-6\_23`
9. Chen, C., Nagananda, K.G., Xiong, G., Kishore, S., Snyder, L.V.: A communication-based appliance scheduling scheme for consumer-premise energy management systems. IEEE Trans. Smart Grid **4**(1), 56–65 (2013)
10. Chen, L., Megow, N., Schewior, K.: An $O(\log m)$-competitive algorithm for online machine minimization. In: SODA, pp. 155–163. ACM (2016)
11. Chuzhoy, J., Guha, S., Khanna, S., Naor, J.S.: Machine minimization for scheduling jobs with interval constraints. In: FOCS, pp. 81–90. IEEE (2004)
12. Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B., Widmayer, P.: Scheduling with release times and deadlines on a minimum number of machines. In: IFIP TCS, pp. 209–222. Springer (2004)
13. Djurović, M.Ž., Milačić, A., Kršulja, M.: A simplified model of quadratic cost function for thermal generators. In: DAAAM, pp. 25–28. DA (2012)
14. European Commission: Europen smartgrids technology platform. `ftp://ftp.cordis.europa.eu/pub/fp7/energy/docs/smartgrids_en.pdf` (2006)
15. Fang, K., Uhan, N.A., Zhao, F., Sutherland, J.W.: Scheduling on a single machine under time-of-use electricity tariffs. Annals OR **238**(1-2), 199–227 (2016)
16. Fang, X., Misra, S., Xue, G., Yang, D.: Smart grid – the new and improved power grid: A survey. IEEE Communications Surveys and Tutorials **14**(4), 944–980 (2012)
17. Farhangi, H.: The path of the smart grid. IEEE Power and Energy Mag. **8**(1), 18–28 (2010)
18. Feng, X., Xu, Y., Zheng, F.: Online scheduling for electricity cost in smart grid. In: COCOA, pp. 783–793. Springer (2015)
19. Fulkerson, D., Gross, O.: Incidence matrices and interval graphs. Pacific Journal of Mathematics **15**(3), 835–855 (1965)
20. Halin, R.: Some remarks on interval graphs. Combinatorica **2**(3), 297–304 (1982)
21. Hamilton, K., Gulhar, N.: Taking demand response to the next level. IEEE Power and Energy Mag. **8**(3), 60–65 (2010)
22. Ipakchi, A., Albuyeh, F.: Grid of the future. IEEE Power & Energy Mag. **7**(2), 52–62 (2009)
23. Jansen, K., Land, F., Land, K.: Bounding the running time of algorithms for scheduling and packing problems. SIAM J. Discrete Math. **30**(1), 343–366 (2016)

24. Kannberg, L., Chassin, D., DeSteese, J., Hauser, S., Kintner-Meyer, M., (U.S.), P.N.N.L., of Energy, U.S.D.: GridWise: The Benefits of a Transformed Energy System. Pacific Northwest National Laboratory (2003)
25. Karbasioun, M.M., Shaikhet, G., Kranakis, E., Lambadaris, I.: Power strip packing of malleable demands in smart grid. In: ICC, pp. 4261–4265. IEEE (2013)
26. Koutsopoulos, I., Tassiulas, L.: Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In: e-Energy, pp. 41–50. ACM (2011)
27. Krishnan, R.: Meters of tomorrow [in my view]. IEEE Power and Energy Mag. **6**(2), 96–94 (2008)
28. Li, H., Qiu, R.C.: Need-based communication for smart grid: When to inquire power price? In: GLOBECOM, pp. 1–5. IEEE (2010)
29. Li, Z., Liang, Q.: Performance analysis of multiuser selection scheme in dynamic home area networks for smart grid communications. IEEE Trans. Smart Grid **4**(1), 13–20 (2013)
30. Liu, F., Liu, H., Wong, P.W.H.: Optimal nonpreemptive scheduling in a smart grid model. In: ISAAC, pp. 53:1–53:13. LIPIcs (2016)
31. Llaria, A., Jiménez, J., Curea, O.: Study on communication technologies for the optimal operation of smart grids. Trans. Emerging Telecommunications Technologies **25**(10), 1009–1019 (2014)
32. Lockheed Martin: SEELoad^{TM} Solution. `http://www.lockheedmartin.co.uk/us/products/energy-solutions/seesuite/seeload.html`
33. Logenthiran, T., Srinivasan, D., Shun, T.Z.: Demand side management in smart grid using heuristic optimization. IEEE Trans. Smart Grid **3**(3), 1244–1252 (2012)
34. Lui, T., Stirling, W., Marcy, H.: Get smart. IEEE Power & Energy Mag. **8**(3), 66–78 (2010)
35. Ma, C.Y.T., Yau, D.K.Y., Rao, N.S.V.: Scalable solutions of markov games for smart-grid infrastructure protection. IEEE Trans. Smart Grid **4**(1), 47–55 (2013)
36. Maharjan, S., Zhu, Q., Zhang, Y., Gjessing, S., Basar, T.: Dependable demand response management in the smart grid: A stackelberg game approach. IEEE Trans. Smart Grid **4**(1), 120–132 (2013)
37. Masters, G.M.: Renewable and efficient electric power systems. John Wiley & Sons (2013)
38. Mohsenian-Rad, A.H., Wong, V., Jatskevich, J., Schober, R.: Optimal and autonomous incentive-based energy consumption scheduling algorithm for smart grid. In: ISGT, pp. 1–6. IEEE (2010)
39. Mohsenian-Rad, A.H., Wong, V.W., Jatskevich, J., Schober, R., Leon-Garcia, A.: Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. IEEE Trans. Smart Grid **1**(3), 320–331 (2010)
40. REGEN Energy Inc: ENVIROGRID^{TM} SMART GRID BUNDLE. `http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/`
41. Saha, B.: Renting a cloud. In: FSTTCS, pp. 437–448. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2013)
42. Salinas, S., Li, M., Li, P.: Multi-objective optimal energy consumption scheduling in smart grids. IEEE Trans. Smart Grid **4**(1), 341–348 (2013)
43. Samadi, P., Mohsenian-Rad, A.H., Schober, R., Wong, V.W., Jatskevich, J.: Optimal real-time pricing algorithm based on utility maximization for smart grid. In: SmartGridComm, pp. 415–420. IEEE (2010)
44. Tang, S., Huang, Q., Li, X.Y., Wu, D.: Smoothing the energy consumption: Peak demand reduction in smart grid. In: INFOCOM, pp. 1133–1141. IEEE (2013)
45. Toronto Hydro Corporation: Peaksaver Program. `http://www.peaksaver.com/peaksaver_THESL.html`
46. UK Department of Energy & Climate Change: Smart grid: A more energy-efficient electricity supply for the UK. `https://www.gov.uk/smart-grid-a-more-energy-efficient-electricity-supply-for-the-uk` (2013)
47. US Department of Energy: The Smart Grid: An Introduction. `http://www.oe.energy.gov/SmartGridIntroduction.htm` (2009)
48. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382. IEEE (1995)

49. Yaw, S., Mumey, B.: An exact algorithm for non-preemptive peak demand job scheduling. In: COCOA, pp. 3–12. Springer (2014)
50. Yaw, S., Mumey, B., McDonald, E., Lemke, J.: Peak demand scheduling in the smart grid. In: SmartGridComm, pp. 770–775. IEEE (2014)
51. Zpryme Research & Consulting: Power systems of the future: The case for energy storage, distributed generation, and microgrids. `http://smartgrid.ieee.org/images/features/smart_grid_survey.pdf` (2012)