Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Greedy is Optimal for Online Restricted Assignment and Smart Grid Scheduling for Unit Size Jobs

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

Received: date / Accepted: date

Abstract We study online scheduling of unit-sized jobs in two related problems, namely, restricted assignment problem and smart grid problem. The input to the two problems are in close analogy but the objective functions are different. We show that the greedy algorithm is an optimal online algorithm for both problems. Typically, an online algorithm is proved to be an optimal online algorithm through bounding its competitive ratio and showing a lower bound with matching competitive ratio. However, our analysis does not take this approach. Instead, we prove the optimality without giving the exact bounds on competitive ratio. Roughly speaking, given any online algorithm and a job instance, we show the existence of another job instance for greedy such that (i) the two instances admit the same optimal offline schedule; (ii) the cost of the online algorithm is at least that of the greedy algorithm on the respective job instance. With these properties, we can show that the competitive ratio of the greedy algorithm is the smallest possible.

Keywords Optimal online algorithm · Restricted assignment · Smart grid scheduling.

This work is partially supported by Polish National Science Centre grant 2016/22/E/ST6/00499. This work is supported by Networks Sciences & Technologies(NeST), School of EEECS, University of Liverpool. This work was partially done when Hsiang-Hsuan Liu worked in Wroclaw University, Poland. A preliminary version of this paper was published in WAOA 2019 [32].

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

Hsiang-Hsuan Liu
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Institute of Computer Science, Wroclaw University, Poland
E-mail: H.H.Liu@uu.nl

Prudence W.H. Wong
Department of Computer Science, University of Liverpool, UK
E-mail: pwong@liverpool.ac.uk

1 Introduction

In this paper, we study online scheduling of unit-sized jobs in two related problems, namely, restricted assignment problem and smart grid problem. The input to the two problems are in close analogy but the objective functions are different. We show that the greedy algorithm (precise definition to be given later) is an optimal online algorithm for both problems by showing that both objective functions have led to the same property of the greedy algorithm. The property is crucial for the optimality of the greedy algorithm. We first describe the two problems and then explain how they are related.

Smart grid scheduling. The smart grid scheduling problem arises in demand response management in electrical smart grid [16, 21, 23, 36, 50] - one of the major challenges in the 21st century [15, 45, 46]. The smart grid [17, 38] makes power generation, distribution and consumption more efficient through information and communication technologies. One of the main challenges is that peak demand hours happen only for a short duration, yet can make the electrical grid very inefficient. For example, in the US power grid, 10% of generation assets and 25% of distribution infrastructure are required for the peak hours which is roughly 5% of the whole time [13, 46]. Demand response management is to reduce peak load by shifting demand to non-peak hours [11, 26, 35, 37, 39, 42] through technological advances in smart meters [27]. It is beneficial to both the power supplier and consumers. On one hand, it can bring down the cost for the supplier operating the grid [35]. On the other hand, it can reduce electricity bill for consumers as it is common that suppliers charge according to generation cost [42]. Research initiatives in the area include [24, 34, 41, 44].

We consider online scheduling of unit-sized requests with the following input. A consumer sends in a power request j with unit power requirement, unit duration of service, and feasible timeslots $F(j)$ that j can be served. The operator of the smart grid selects a timeslot from $F(j)$ for each request j. The *load* of the grid at each timeslot t is the number of requests allocated to t. The *energy cost* is modeled by a strictly increasing convex function $f(t)$ on load(t). The objective is to minimize the total energy cost over time, i.e., minimize $\sum_t f(\text{load}(t))$.

Restricted assignment problem. The assignment problem [19, 20] and its variant restricted assignment problem [5] have been extensively studied. The assignment problem is concerned with a set of jobs and a set of machines in which each job specifies a vector of processing times (a.k.a. size/load) it takes to complete if it is assigned to the corresponding machine. The completion time of a machine is the total processing times of jobs scheduled on it and the objective is to minimize the makespan (a.k.a. the maximum load over all machines). For the restricted assignment problem, each job is associated with a processing time (a.k.a. size/load) and a subset of machines that the job can be scheduled on. As pointed out in [5], the restricted assignment problem can be applied to say a wireless communication network where customers arriving one-by-one each request a certain amount of service and must be assigned a

base-station within range to service it. We consider online scheduling of unit size jobs. This means that a job increases the load of the assigned machine by one. The objective is to minimize the maximum number of jobs assigned to any machine while satisfying the assignment restriction constraints.

Our contribution. Notice that with unit size, the input for the grid scheduling problem and the restricted assignment problem is indeed the same. Timeslots in grid scheduling is in analogy to machines in restricted assignment; feasible timeslots in analogy to subset of machines; load of timeslots in analogy to load of machines. The difference of the two problems lie in the objective functions. Our main contribution is the following theorem about both problems.

Theorem 1 *When the input to the grid scheduling problem and the restricted assignment problem is a set of unit-sized jobs, the greedy algorithm is an optimal online algorithm having the best possible competitive ratio.*

Typically, an online algorithm is proved to be an optimal online algorithm through bounding its competitive ratio and showing a lower bound with matching competitive ratio. However, our analysis does not take this approach. Instead, we prove the optimality without giving the exact bounds on the competitive ratio.

In this paper, we develop an *adversary design* technique. We design some adversarial input which is bad for the greedy strategy. Also by this technique, we design an adversary for any online algorithm that performs bad enough to be worse than the performance of a greedy strategy. By these adversarial inputs, we can show that greedy is the best online strategy for this problem.

More specifically, given any online algorithm and a load configuration (to be defined precisely later), we show the existence of two job instances J_1 and J_2 such that (i) J_1 and J_2 admit the same optimal offline schedule represented by the given load configuration; (ii) the cost of the schedule produced by the given online algorithm on J_1 is at least the cost of the schedule produced by the greedy algorithm on J_2. This means that when we consider any job instance for the greedy algorithm, there is always another job instance such that the ratio versus the (same) optimal offline schedule of the greedy algorithm is not larger than any online algorithm. Hence, we can show that the competitive ratio of the greedy algorithm is the smallest possible. The existence of the two job sets relies on a property about the relative costs of two comparable schedules (see Theorem 2). We show that this property holds for both objective functions for the grid scheduling problem and the restricted assignment problem, hence, the optimality holds for both problems.

Interestingly, our adversary design shows that there is an adversary for the greedy strategy, regardless of the cost function, as long as the cost function is strictly increasing and convex.

Related work on grid scheduling. The offline version of the grid problem with unit power requirement and unit service duration can be solved optimally in polynomial time [8]. The solution iteratively assigns request j to the timeslot with the smallest load in $F(j)$ and rearranges the other requests

to make an optimal schedule. However in the online setting where a request must be irrevocably scheduled, rearrangement is not allowed. It is interesting to study the performance of the greedy strategy without the rearrangement. A previous work [18] has studied the greedy strategy on the problem with unit power requirement, unit service duration and cost function $f(t) = \text{load}^2(t)$ and claimed that the algorithm is 2-competitive. However, as stated in [33], the greedy algorithm is indeed at least 3-competitive. Hence, it is still an open problem to determine how good or bad the greedy strategy is. Our results in this paper establish the optimality of the greedy algorithm.

For arbitrary power requirements and service durations, the problem becomes NP-hard [8, 26]. Theoretical study on this problem mainly focuses on the cost function $f(t) = \text{load}^\alpha(t)$ [12,31]. In particular, Liu et al. [31] gave the first online algorithm with competitive ratio $O(\log^\alpha \frac{w_{\max}}{w_{\min}})$ for contiguous feasible timeslots, where $w_{\max}$ and $w_{\min}$ are the maximum and minimum service duration of the requests respectively. They also proved that all deterministic online algorithms have competitive ratio at least $\Omega(\alpha^\alpha)$ for this problem. Chau et al. [12] designed a greedy algorithm based on a primal-dual approach and improved the upper bound on the competitive ratio to $O(\alpha^\alpha)$, which is asymptotically optimal. Note that the feasible timeslots of requests we considered are more flexible than that in [12, 31]. Other work on demand response management can be found in [26, 35, 36, 42].

Related work on (restricted) assignment problem. Online (restricted) assignment problem of jobs with arbitrary size has also been studied as the problem of load balancing. When jobs can be scheduled on any (unrestricted) machine, Graham [19, 20] has shown that the greedy algorithm is $(2 - \frac{1}{m})$-competitive where m is the number of machines and this has been improved to $2 - \epsilon$ in [6]. For restricted assignment, Azar et al. [5] have shown that the greedy algorithm is $(\lceil \log m \rceil + 1)$-competitive and no online algorithm can do better than $\lceil \log(m + 1) \rceil$-competitive. This implies that the greedy algorithm is very close to optimal. Our result indeed shows that the greedy algorithm is the best possible online algorithm for unit-sized jobs although the precise competitive ratio is yet to been established. The restricted assignment problem is also studied in ℓ_p norm [9, 10] and Ioannis Caragiannis [9] has shown that the greedy algorithm is an optimal online algorithm with competitive ratio $1/(2^{1/p} - 1)$. We consider arbitrary strictly increasing convex functions and show that the greedy algorithm remains optimal for unit-sized jobs.

In the offline setting, the (unrestricted) assignment problem has also been studied as scheduling on unrelated machines in which Lenstra et al. [30] have shown a 2-approximation algorithm and that approximating the problem with approximation ratio 3/2 is NP-hard. For restricted assignment, a breakthrough was made by Svensson [43] who has shown that the integrality gap of the configuration LP for the restricted assignment problem is at most 1.942. Various special cases have been studied [14, 22, 25, 29, 40, 47, 48]. The (restricted and unrestricted) assignment problem has also been studied for temporary jobs that depart [1–4, 28].

Organization of the paper. We present some preliminaries in Section 2. We then present a framework of analysis in Section 3 and establish the optimality of the greedy algorithm in Section 4. Finally, we conclude in Section 5.

2 Preliminaries

Problem definition. We unify the two problems as follows. We are given a set of machines. Each job j has unit size and a set of permitted machines P_j, which is a subset of machines. A *job instance* J is a set of jobs together with their release order. Two job instances can contain the same set of jobs but with different release orders.

A schedule $\mathcal{S}(J)$ of a job instance J is an assignment assigning each job to a machine. We simply use $\mathcal{S}$ when the context is clear. We denote the machine to which j is assigned by the schedule $\mathcal{S}$ by $m_{\mathcal{S}}(j)$. A schedule $\mathcal{S}$ is *feasible* if each job $j \in J$ is assigned to one of the machines in P_j. That is, $\mathcal{S}$ is feasible if $m_{\mathcal{S}}(j) \in P_j$ for all j in the job instance. We denote by $\mathcal{A}(J)$ the schedule produced by a scheduling algorithm $\mathcal{A}$ on J. We denote the optimal offline algorithm by $\mathcal{O}$ and its schedule $\mathcal{O}(J)$.

In a schedule $\mathcal{S}$ of some job instance J, the *load* of machine i, $\text{load}_{\mathcal{S}}(i)$, is the number of jobs assigned to the machine i. That is, $\text{load}_{\mathcal{S}}(i) = |\{j : m_{\mathcal{S}}(j) = i\}|$. The *cost* of machine i, $\text{cost}_{\mathcal{S}}(i)$ is a strictly increasing convex function of the load of i and $\text{cost}_{\mathcal{S}}(i) = 0$ if the load of i is 0. We overload the notation and use $\mathcal{S}(J)$ to also denote the total cost of schedule $\mathcal{S}$ with instance J, which is the sum of $\text{cost}_{\mathcal{S}}(i)$ over the machines. The goal is to minimize the total cost $\mathcal{S}(J)$.

For unit-sized jobs, both grid scheduling problem and restricted assignment problem can be modeled as the problem above. The two problems have the same input but with different objective functions. For the grid scheduling problem, the objective is to minimize the sum of the costs over all machines, while for the restricted assignment problem, the objective is to minimize the max of the loads over all machines. To unify the objectives, we can design a cost function for restricted assignment problem that is able to grow rapidly such that the maximum load of machines dominates the sum of the total cost. One of such cost function is $\text{cost}_{\mathcal{S}}(i) = n^{\text{load}_{\mathcal{S}}(i)}$ where n is the number of jobs in $\mathcal{S}$. One can show that the minimum cost $\mathcal{S}(J)$ for the cost function implies that the maximum load over all machines in $\mathcal{S}$ is minimized. The cost function is convex and strictly increasing when $n > 1$. The case where $n = 1$ can be unified trivially. The analysis in this paper does not require the specified cost function to be known beforehand and thus the results of this work applies even if the cost function depends on n.

Online model. We consider the online model. The jobs are released one by one, and the released job has to be assigned before the next one is released. At any time, the online algorithm knows only the released jobs without any knowledge about the future. The decisions of an online algorithm are made irrevocably.

We measure the performance of online algorithms by *competitive ratio* [7], which is defined as the maximum ratio between the cost of the online algorithm and the cost of an optimal offline algorithm knowing the whole input.

The greedy algorithm $\mathcal{G}$. When a job arrives, it is assigned to the machine with the smallest number of jobs currently assigned.

A critical theorem. We first introduce a theorem which is useful when comparing the costs of two schedules.

Definition 1 Consider an algorithm $\mathcal{A}$, the *level* of job j decided by $\mathcal{A}$, $\text{level}_{\mathcal{A}}(j)$, is the number of jobs on the machine $m_{\mathcal{A}}(j)$ right after the time when j is assigned to it. That is, a job with $\text{level}_{\mathcal{A}}(j)$ means that it is the $\text{level}_{\mathcal{A}}(j)$-th job assigned to $m_{\mathcal{A}}(j)$ by $\mathcal{A}$.

Definition 2 Given a schedule $\mathcal{S}$ produced by an algorithm $\mathcal{A}$ on job instance J, the *accumulated size at level k*, $L_{\mathcal{S}}^{(k)}$, is defined as the total number of jobs with level at most k. That is, $L_{\mathcal{S}}^{(k)} := |\{j : \text{level}_{\mathcal{S}}(j) \in [1,k]\}|$. For better readability, we use L_k to represent $L_{\mathcal{S}}^{(k)}$ when the schedule $\mathcal{S}$ is clear.

Theorem 2 *Given two schedules $\mathcal{S}$ and $\mathcal{S}'$ which have the same number of jobs (which are not necessarily of the same job instance) and the accumulated size L_k and L'_k for $\mathcal{S}$ and $\mathcal{S}'$ respectively, if $L_k \geq L'_k$ for all $k \geq 1$, then the cost of $\mathcal{S}$ is at most that of $\mathcal{S}'$.*

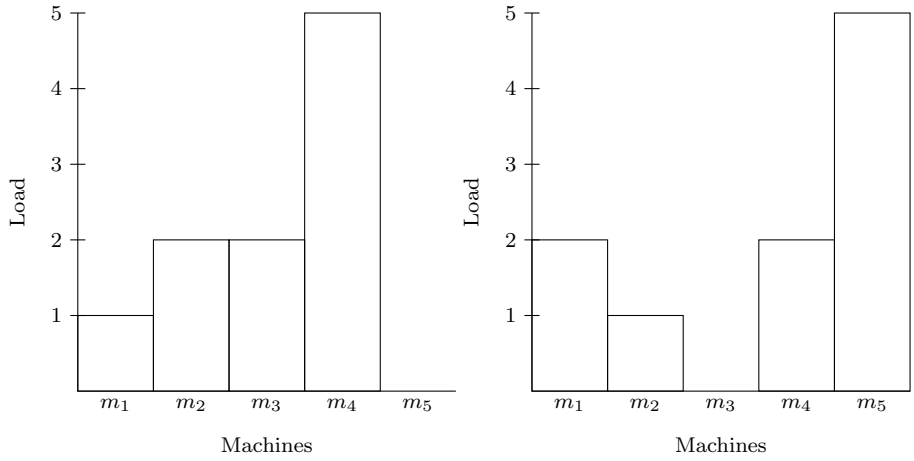
Proof Let $f(x)$ be the cost corresponding to load x. First of all, we observe that the cost of schedule $\mathcal{S}$ can be written as

$$\sum_{j \in J} \Big(f\big(\text{level}_{\mathcal{S}}(j)\big) - f\big(\text{level}_{\mathcal{S}}(j) - 1\big) \Big) \ . \tag{1}$$

We claim that we can map each job j' in $\mathcal{S}'$ to a unique job j in $\mathcal{S}$ such that $\text{level}_{\mathcal{S}'}(j') \geq \text{level}_{\mathcal{S}}(j)$. The claim can be proved inductively by first mapping jobs in $\mathcal{S}'$ at level 1 and because of $L_1 \geq L'_1$, there are enough jobs in $\mathcal{S}$ at level 1 to have a unique mapping. Then we can map jobs in $\mathcal{S}'$ at level 2 to unmapped jobs in $\mathcal{S}$ at level 1 and any jobs at level 2 because $L_2 \geq L'_2$. Since the number of jobs up to level i in $\mathcal{S}$ is always at least that in $\mathcal{S}'$, we can repeat this mapping for each level. The claim then follows. Furthermore, as the cost function f is convex, we have $f\big(\text{level}_{\mathcal{S}'}(j')\big) - f\big(\text{level}_{\mathcal{S}'}(j') - 1\big) \geq f\big(\text{level}_{\mathcal{S}}(j)\big) - f\big(\text{level}_{\mathcal{S}}(j) - 1\big)$. Summing up over all pairs of mapped jobs using Equation (1) concludes the theorem. □

3 Framework of analysis

In this section, we give a framework of the analysis and we then present the details of analysis in the next section. As proved in Theorem 2, we can compare schedules by looking at some aggregate property of the schedule instead of the precise allocation of jobs to the machines. We further formalize this notion as configuration of a schedule.



(a) A schedule with loads $(1, 2, 2, 5, 0)$. (b) A schedule with loads $(2, 1, 0, 2, 5)$.

Fig. 1: The two schedules have the same configuration $(0, 1, 2, 2, 5)$.

Given an arbitrary schedule $\mathcal{S}$, the *configuration* of $\mathcal{S}$, config($\mathcal{S}$), is defined as the multi-set of loads of the machines. Two schedules are considered as having the same configuration if they have the same multi-set of machine loads even with different order. Moreover, we represent the configuration of a schedule as the sequence of loads sorted from low to high and we can compute the *cost* of a certain configuration. We denote by cost($\mathcal{C}$) the cost of configuration $\mathcal{C}$.

Example. Consider a case with five machines and ten jobs, and two schedules $\mathcal{S}_1$ and $\mathcal{S}_2$. Let ℓ_i be the load on machine m_i. Suppose the load of $\mathcal{S}_1$ is $\ell_1 = 1$, $\ell_2 = 2$, $\ell_3 = 2$, $\ell_4 = 5$, and $\ell_5 = 0$; the load of $\mathcal{S}_2$ is $\ell_1 = 2$, $\ell_2 = 1$, $\ell_3 = 0$, $\ell_4 = 2$, $\ell_5 = 5$. The two schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ have the same configuration $(0, 1, 2, 2, 5)$. See Figure 1 for an illustration.

The high level idea of the analysis is roughly as follows. We attempt to find some "bad" instances for the greedy algorithm $\mathcal{G}$ and show that for each such bad instance we can always find another bad instance for every other online algorithm $\mathcal{A}$ such that the ratio of $\mathcal{G}$ to $\mathcal{O}$ on its bad instance is no more than the ratio of $\mathcal{A}$ to $\mathcal{O}$ on its own bad instance. We can then bound the competitive ratio of $\mathcal{G}$ by that of $\mathcal{A}$. We are going to find these bad instances through characterizing the job instances by the configuration of their optimal schedules. Figure 2 is an illustration of the framework.

Let $\mathcal{I}$ be the set of all possible job instances. We partition $\mathcal{I}$ according to the optimal configuration of job instances. Job instances J and J' are in the same partition $\mathcal{I}_\mathcal{C}$ if and only if they both have the optimal configuration $\mathcal{C}$. That is, config($\mathcal{O}(J)$) = config($\mathcal{O}(J')$) = $\mathcal{C}$. The following are some properties of $\mathcal{I}_\mathcal{C}$.

Observation 1 *Consider a partition $\mathcal{I}_{\mathcal{C}}$ and the corresponding optimal configuration $\mathcal{C}$.*

(1) Since the cost function is strictly increasing and convex, any two different configurations have different cost. Hence, for each job instance J, there is exactly one $\mathcal{I}_{\mathcal{C}}$ such that $J \in \mathcal{I}_{\mathcal{C}}$, i.e., the partition is well defined.

(2) By definition, for any job instance $J \in \mathcal{I}_{\mathcal{C}}$, config$(\mathcal{O}(J)) = \mathcal{C}$.

(3) For any two job instances $J_1, J_2 \in \mathcal{I}_{\mathcal{C}}$, consider their optimal schedules $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. Although config$(\mathcal{O}_1)$ = config$(\mathcal{O}_2)$, $\mathcal{O}_1$ may not be a feasible schedule for J_2, and neither the other way round.

With the above partition, we can express the competitive ratio of $\mathcal{G}$, denoted by $\mathcal{R}(\mathcal{G})$, as follows.

$$\mathcal{R}(\mathcal{G}) = \max_{J \in \mathcal{I}} \frac{\mathcal{G}(J)}{\mathcal{O}(J)} = \max_{\mathcal{I}_{\mathcal{C}}} \max_{J \in \mathcal{I}_{\mathcal{C}}} \frac{\mathcal{G}(J)}{\mathcal{O}(J)} = \max_{\mathcal{I}_{\mathcal{C}}} \max_{J \in \mathcal{I}_{\mathcal{C}}} \frac{\mathcal{G}(J)}{\text{cost}(\mathcal{C})} \ .$$

This means that we can characterize the competitive ratio by considering the job instance in each $\mathcal{I}_{\mathcal{C}}$ with the highest greedy cost. We denote this job instance as J^*, i.e., for a given $\mathcal{C}$, $J^* = \arg\max_{J \in \mathcal{I}_{\mathcal{C}}} \mathcal{G}(J)$. It is not clear how to find such job instances directly and instead we try to find their counter parts (bad instances) for any online algorithm $\mathcal{A}$ which share the same $\mathcal{C}$. Precisely, for any online algorithm $\mathcal{A}$, we show the existence of a job instance $J' \in \mathcal{I}_{\mathcal{C}}$ such that $\mathcal{A}(J') \geq \mathcal{G}(J^*)$. This implies $\frac{\mathcal{G}(J^*)}{\text{cost}(\mathcal{C})} \leq \frac{\mathcal{A}(J')}{\text{cost}(\mathcal{C})} = \frac{\mathcal{A}(J')}{\mathcal{O}(J')}$, where the last equality is because $J' \in \mathcal{I}_{\mathcal{C}}$. We can then bound the competitive ratio of $\mathcal{G}$ by that of $\mathcal{A}$ as follows:

$$\mathcal{R}(\mathcal{G}) = \max_{\mathcal{I}_{\mathcal{C}}} \max_{J \in \mathcal{I}_{\mathcal{C}}} \frac{\mathcal{G}(J)}{\text{cost}(\mathcal{C})} = \max_{\mathcal{I}_{\mathcal{C}}} \frac{\mathcal{G}(J^*)}{\text{cost}(\mathcal{C})} \leq \max_{\mathcal{I}_{\mathcal{C}}} \frac{\mathcal{A}(J')}{\mathcal{O}(J')} \leq \max_{\mathcal{I}_{\mathcal{C}}} \max_{J \in \mathcal{I}_{\mathcal{C}}} \frac{\mathcal{A}(J)}{\mathcal{O}(J)} = \mathcal{R}(\mathcal{A}) \ .$$

Then we can conclude Theorem 1. Section 4 is devoted to finding J^* (Section 4.1) and J' (Section 4.2).

4 Optimality of the greedy algorithm

In this section, we construct J^* and J' as required in the framework in Section 3.

4.1 The job instance J^* for the greedy algorithm $\mathcal{G}$

4.1.1 Overview

Given an optimal configuration $\mathcal{C}$ and the corresponding set of job instances $\mathcal{I}_{\mathcal{C}}$, we aim to find a job instance $J^* \in \mathcal{I}_{\mathcal{C}}$ such that J^* is the most troublesome job instance for $\mathcal{G}$ among all job instances in $\mathcal{I}_{\mathcal{C}}$.

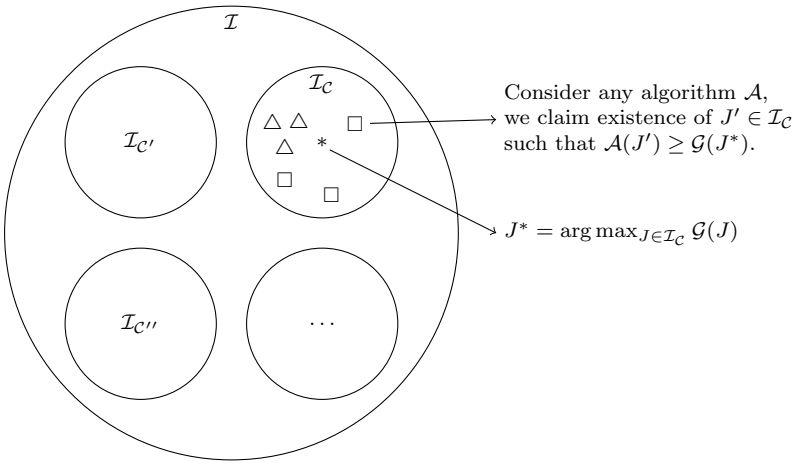


Fig. 2: An illustration of the framework. Each partition, say $\mathcal{I}_{\mathcal{C}}$, has exactly one corresponding J^* in which we are interested and J^* makes $\mathcal{G}$ to have the largest cost. In a partition, each J' may be different for different online algorithm $\mathcal{A}$ but the cost of $\mathcal{A}$ on J' must be at least the cost of $\mathcal{G}$ on J^*. There are other job instances represented by $\triangle$ that are neither J^* nor J' and are not crucial to the analysis.

We will find J^* by artificially designing a job instance and transform $\mathcal{C}$ to schedule $\mathcal{S}^*$ as follows. First, J^* will have the same number of jobs as in the given $\mathcal{C}$ and we will impose the set of permitted machines of each job and the release order of the jobs in the construction. Second, during the construction, $\mathcal{C}$ will be transformed to a schedule $\mathcal{S}^*$ by changing the configuration. The construction will ensure the following properties:

- $\mathcal{C}$ is the optimal configuration of J^* (Lemma 3);
- $\mathcal{S}^*$ is the schedule of running a greedy algorithm on J^* (Lemma 2);
- for any job instance $J \in \mathcal{I}_{\mathcal{C}}$, we will have $\mathcal{G}(J^*) \geq \mathcal{G}(J)$ (Corollary 1).

To achieve this, we design the set of permitted machines of each job and choose the release order carefully.

Although the job instance J^* seems to be artificial, we can prove that $\mathcal{G}(J^*)$ is the highest among all job instances in $\mathcal{I}_{\mathcal{C}}$ (Corollary 1). That is, consider any job set and any release order, as long as the job set with the release order has optimal configuration $\mathcal{C}$, its greedy cost is no greater than the greedy cost of J^*.

4.1.2 Construction of the job instance J^*

We aim to construct a job instance with high greedy cost, i.e., we want the greedy schedule for the job instance to have as few jobs at each level as possible. This can be done by setting a small set of permitted machines. However, this

may result in a high optimal cost as well and the ratio between the greedy cost and the optimal cost is still small. Hence we have to balance the greedy schedule and the feasibility of optimal configuration of the job instance.

First we explain how to transform the given optimal configuration $\mathcal{C}$ to schedule $\mathcal{S}^*$. Assume that $\mathcal{C} = (v_1, v_2, \cdots, v_h)$, where $0 < v_1 \leq v_2 \leq \cdots \leq v_h$, we treat $\mathcal{C}$ as building blocks with h columns and each column i has v_i blocks (where each block corresponds to one job). The transformation runs in rounds, in each round, we choose a certain number of blocks, remove them from $\mathcal{C}$ and put them in $\mathcal{S}^*$ (which is initially empty) and produce another configuration. During the process, the configuration $\mathcal{C}$ changes to reflect the removal of blocks. Hence, the number of non-zero terms in the configuration changes over the process as well. The number of non-zero terms in the configuration in each round takes an important role in our construction. We denote by $NE(i)$ the set of machines with non-empty load in the end of round i. By default, $NE(0)$ is the set of non-empty machines in $\mathcal{C}$.

At the beginning of round i, let $m_1, m_2, \cdots, m_{|NE(i-1)|}$ be the non-empty columns in the configuration. We remove the jobs one by one from the least-loaded non-empty column m_1. The removing procedure stops once the size of the set of removed blocks in this round, denoted as J_i^*, is greater than or equal to the number of non-empty columns in the updated configuration (that is, the configuration after removing the jobs). Notice that by the construction,

$$|NE(i)| \leq |J_i^*| \leq |NE(i)| + 1. \tag{2}$$

In round i, after removing the blocks from $\mathcal{C}$, we update $\mathcal{S}^*$ by placing each of them at $|J_i^*|$ distinct machines with the highest loads in current $\mathcal{S}^*$ (note that $\mathcal{S}^*$ is initially empty at the beginning of the first round). We call this set of machines as $B(i)$. We will prove later that the machines in $B(i)$ cover all the machines in $NE(i)$ (Observation 3) and the blocks we placed in each round are on the same level (Lemma 1), which means they are placed evenly.

Now we design other parameters of the job instance J^*. As mentioned before, each block is corresponding to one job. For each job j, its permitted machines are the machines corresponding to the columns the block was in $\mathcal{C}$ and $\mathcal{S}^*$. That is, $P_j = \{m_{\mathcal{C}}(j)\} \cup \{m^*(j)\}$, where $m_{\mathcal{C}}(j)$ and $m^*(j)$ are the columns of block j in the configurations $\mathcal{C}$ and $\mathcal{S}^*$, respectively. The release order of the jobs in J^* is exactly the order their corresponding blocks removed from $\mathcal{C}$. Algorithm 1 is a demonstration to find the job instance J^*. Figure 3a gives an example configuration and Figure 3b is the corresponding J^* of the configuration in Figure 3a.

4.1.3 $\mathcal{C}$ is optimal and $\mathcal{S}^*$ is greedy for J^*

We have to show that $J^* \in \mathcal{I}_{\mathcal{C}}$. Moreover, we show that $\mathcal{S}^*$ generated during the construction process is a greedy schedule for J^*. That is, there is a greedy algorithm for the input job set and the release order generating the schedule $\mathcal{S}^*$ (with a certain tie breaking).

Algorithm 1 Find J^*

Input: The given configuration $\mathcal{C} = (v_1, v_2, \cdots, v_h)$, where $0 < v_1 \leq v_2 \leq v_3 \leq \cdots \leq v_h$.
Output: Job instance J^* with job subsets $J_1^*, J_2^*, \cdots$
Schedule $\mathcal{S}^*$ of J^*

1: Updated configuration $\mathcal{C}' \leftarrow \mathcal{C}$
2: **while** there is at least one job in the updated configuration $\mathcal{C}'$ **do**
3: **for** round $i = 1, 2, 3, \cdots$ **do**
4: $J_i^* \leftarrow \emptyset$
5: **while** $|J_i^*| <$ the number of non-zero entries in $\mathcal{C}'$ **do**
6: $v_1' \leftarrow$ the non-zero vector with the smallest load in $\mathcal{C}'$
7: $j \leftarrow$ the job with lowest level at v_1'. $J_i^* \leftarrow J_i^* \cup \{j\}$
8: $\mathcal{C}' \leftarrow \mathcal{C}' \setminus \{j\}$
9: **end while**
10: Update $\mathcal{S}^*$ by assigning the jobs in J_i^* evenly to the machines with highest loads, denoted as $B(i)$
11: The permitted machines P_j of job j is $\{m_{\mathcal{C}}(j), m^*(j)\}$
12: **end for**
13: The release order of jobs is the order they are removed from $\mathcal{C}'$
14: **end while**

Recall that in round i, $B(i)$ is the set of machines chosen to (evenly) assign the chosen jobs J_i^*, and $NE(i)$ is the set of machines in $\mathcal{C}'$ which are non-empty after removing J_i^*. We first observe the relation of machines in $B(i)$ and $NE(i)$, and show that Algorithm 1 is valid. More specifically, we prove that all machines in $NE(i)$ can be covered by the highest load machines. Hence, at the end of round i, the machines in $B(i)$ are equally-loaded and the level of each jobs $j \in J_i^*$ is equal to i (that is, in Algorithm 1, Line 10 is achievable). This property of the construction is essential for proving that the resulting schedule $\mathcal{S}^*$ is a greedy schedule for J^*.

Observation 2 *For all jobs $j \in J_i^*$, $m_{\mathcal{C}}(j) \in NE(i-1)$.*

Proof According to Algorithm 1, Line 7, we choose jobs from the updated optimal configuration $\mathcal{C}'$. Also, once a job is chosen, it is removed from $\mathcal{C}'$ (Line 8). Hence, $m_{\mathcal{C}}(j) \in NE(i-1)$. □

Observation 3 *There is a $\mathcal{G}$ with some tie breaking to choose a subset $B(i)$ of $|J_i^*|$ machines such that $NE(i) \subseteq B(i)$.*

Proof By Inequality 2, $|J_i^*| = |B(i)| \geq |NE(i)|$. There are two cases in the i-th round, $|J_i^*| = |NE(i)|$ or $|J_i^*| = |NE(i)| + 1$. If $|J_i^*| = |NE(i)|$, we can choose $B(i)$ as $NE(i)$ and $NE(i) \subseteq B(i)$.

The case $|J_i^*| = |NE(i)| + 1$ comes from the situation when $m_{\mathcal{C}}(j) \notin NE(i)$ where j is the last job in J_i^*. That is, the removal of j causes a new empty column in the updated $\mathcal{C}$. In this case, $m_{\mathcal{C}}(j) \in NE(i-1) \setminus NE(i)$. We can choose $B(i)$ to be the union of $NE(i)$ and $m_{\mathcal{C}}(j)$. Hence, $NE(i) \subseteq B(i)$. □

Lemma 1 *At the end of round i, (a) $B(i) \subseteq B(i-1)$, and (b) the machines in $B(i)$ have equal load. Note that $B(0)$ is defined as the set of non-empty machines in the given optimal configuration $\mathcal{C}$.*

Proof We prove it by induction in rounds. The argument holds for the first round, since $\mathcal{S}^*$ is initially empty and all machines have load 0. Now suppose that the argument holds for the first $k-1$ rounds. It implies that by the construction, $|B(k)| \leq |B(k-1)|$ since $|J_k^*| \leq |J_{k-1}^*|$. Also by the construction, $NE(k) \subseteq NE(k-1)$.

There are two cases in the k-th round, $|J_k^*| = |NE(k)|$ or $|J_k^*| = |NE(k)|+1$. If $|J_k^*| = |NE(k)|$, we choose $B(k)$ as $NE(k) \subseteq NE(k-1)$.

In the case $|J_k^*| = |NE(k)| + 1$, we choose $B(k)$ to be the union of $NE(k)$ and $m_{\mathcal{C}}(j)$. Hence, $B(k) = NE(k) \cup \{m_{\mathcal{C}}(j)\} \subseteq NE(k-1)$.

In both cases, by Observation 3, $B(k) \subseteq NE(k-1) \subseteq B(k-1)$. By the inductive hypothesis, all machines in $B(k-1)$ have the same load (because every job in J_{k-1}^* has the same level $k-1$). The jobs in J_k^* can be assigned to machines $B(k)$ evenly and have the same level k (Line 10). Hence the statement is proven. □

Lemma 2 *$\mathcal{S}^*$ is a greedy schedule for J^*.*

Proof We prove that for all $j \in J^*$, when j arrives (according to the release order of J^*), $\text{load}_{\mathcal{S}^*}(m_{\mathcal{C}}(j)) \geq \text{load}_{\mathcal{S}^*}(m^*(j))$. Consider job $j \in J_k^*$, by Observation 2 and 3, $m_{\mathcal{C}}(j) \in NE(k-1) \subseteq B(k-1)$. Also, by Lemma 1(a), $m^*(j) \in B(k) \subseteq B(k-1)$. Since $m_{\mathcal{C}}(j)$ and $m^*(j)$ are both in $B(k-1)$, by Lemma 1(b), $\text{load}_{\mathcal{S}^*}(m_{\mathcal{C}}(j)) = \text{load}_{\mathcal{S}^*}(m^*(j))$. Note that by the construction, $\{m_{\mathcal{C}}(j), m^*(j)\}$ are the permitted machines of j and it implies $\mathcal{S}^*$ assigns j to the least-loaded machine among j's permitted machines. Thus the lemma follows. □

Lemma 3 *J^* is a job instance in $\mathcal{I}_{\mathcal{C}}$. That is, the optimal configuration of J^* is $\mathcal{C}$.*

Proof We prove this lemma by using the legal-path argument in [8]. For the completeness, we explain the idea of the legal-path argument first.

Based on the machine assignment, we construct a directed multi-graph $G = (V, E)$ and define the legal-path in G. Each machine i is represented by a vertex u_i in V. For each pair of vertices u_i and u_k, there is an edge from u_i to u_k if there is a job j which is assigned to u_i, and u_k is also a permitted machine of j. Any directed path in G, $u_{i_1}, u_{i_2}, \cdots, u_{i_k}$ indicates a sequence of job shiftings such that the machines u_{i_1} and u_{i_k} are the only two machines of which the load is changed by the rescheduling.

Definition 3 ([8]) A directed path in the graph G is a *legal-path* if and only if the load of the machine represented by the starting point is at least 2 more than the load of the one representing the ending point.

The authors of [8] also proved that as long as there is no legal-path in G, the corresponding assignment is optimal:

Claim ([8] Lemma 7) If there is no legal-path in the construction based on the assignment $\mathcal{S}$, $\mathcal{S}$ is optimal.

We claim that $\text{load}_{\mathcal{C}}(m_{\mathcal{C}}(j)) \leq \text{load}_{\mathcal{C}}(m^*(j))$ for any job j. It follows that in the constructed graph G, every edge points from a lower-load vertex to a higher-load vertex. Hence, in G, for any directed path, the last vertex has higher load than the first vertex. Therefore, there is no legal path in G and the lemma is proved.

By Algorithm 1, every job j has at most two permitted machines, $m_{\mathcal{C}}(j)$ and $m^*(j)$. The final step is to prove the claim is true that $\text{load}_{\mathcal{C}}(m_{\mathcal{C}}(j)) \leq \text{load}_{\mathcal{C}}(m^*(j))$. Consider the jobs j in J_k^* (that is, the jobs chosen in the k-th round) for each k. Let j_ℓ be the last chosen job in J_k^*. By Observation 3, there exists a greedy schedule $\mathcal{G}$ by choosing $B(k) = NE(k)$ or $B(k) = NE(k) \cup \{m_{\mathcal{C}}(j_\ell)\}$ (in the second case, removing the job j_ℓ causes a new empty column in the updated $\mathcal{C}$). By the construction, we choose jobs from the least-loaded machines, and $\text{load}_{\mathcal{C}}(m_{\mathcal{C}}(j)) \leq \text{load}_{\mathcal{C}}(m)$ for any $m \in NE(k)$. Therefore, for any job j with $m^*(j) \in NE(k)$, $\text{load}_{\mathcal{C}}(m_{\mathcal{C}}(j)) \leq \text{load}_{\mathcal{C}}(m^*(j))$. For the job j with $m^*(j) = m_{\mathcal{C}}(j_\ell) \notin NE(k)$, $\text{load}_{\mathcal{C}}(m_{\mathcal{C}}(j)) \leq \text{load}_{\mathcal{C}}(m_{\mathcal{C}}(j_\ell)) = \text{load}_{\mathcal{C}}(m^*(j))$ since j_ℓ is the last chosen job in J_k^*. □

4.1.4 The job instance J^* is the most troublesome among $\mathcal{I}_{\mathcal{C}}$ for $\mathcal{G}$

Now we show that the job instance J^* has the highest greedy cost among all job instances in $\mathcal{I}_{\mathcal{C}}$. Recall that the schedule $\mathcal{S}^*$ produced by Algorithm 1 is $\mathcal{G}(J^*)$. We compare $\mathcal{S}^*$ with greedy schedule of any job instances in $\mathcal{I}_{\mathcal{C}}$.

Lemma 4 *Consider a configuration $\mathcal{C}$, the job instance J^* in $\mathcal{I}_{\mathcal{C}}$, and any job instance $J \in \mathcal{I}_{\mathcal{C}}$. For all $k \geq 1$, we have $L_k \geq L_k^*$ where L_k^* and L_k are the accumulated sizes at level k for the schedule $\mathcal{S}^*$ and $\mathcal{G}(J)$ respectively.*

Proof We prove the lemma by induction on k. Without loss of generality, we set $L_0 = 0$ and $L_0^* = 0$. Thus the base case $L_0 \geq L_0^*$ holds.

Assume that $L_{k-1} \geq L_{k-1}^*$ for some $k > 0$, we prove that $L_k \geq L_k^*$. Let J_k denote the set of jobs at level k in schedule $\mathcal{G}(J)$, i.e., $J_k = \{j \in J \mid \text{level}_{\mathcal{G}(J)}(j) = k\}$. Recall in Lemma 1 that J_k^* are the jobs at level k in $\mathcal{S}^*$. Suppose on the contrary that $L_k < L_k^*$. By combining this inequality with the inductive hypothesis $L_{k-1} \geq L_{k-1}^*$, we have

$$|J_k| < |J_k^*| \ . \tag{3}$$

We consider the set of machines M to which jobs in $J \setminus J_1 \cup J_2 \cup \cdots \cup J_k$ are assigned by $\mathcal{O}(J)$, that is, M is the non-empty machines in $\mathcal{C}$ that remain after the jobs $J_1 \cup J_2 \cup \cdots \cup J_k$ are removed. We will show in the following paragraphs that

$$|J_k^*| \leq |M| \ . \tag{4}$$

Combining Inequality 3 and Inequality 4, we have

$$|J_k| < |M| \ . \tag{5}$$

Since $\mathcal{C}$ is a feasible configuration for job instance J, after removing the jobs $J_1 \cup J_2 \cup \cdots \cup J_k$, the updated $\mathcal{C}$ is also feasible for $J \setminus J_1 \cup J_2 \cup \cdots \cup J_k$. Recall

that $|M|$ is the number of non-empty machines in $\mathcal{C}$ that remain after the jobs $J_1 \cup J_2 \cup \cdots \cup J_k$ are removed. This means that the jobs $J \setminus J_1 \cup J_2 \cup \cdots \cup J_k$ can be scheduled to at least $|M|$ machines. However, by Inequality 5, $|J_k| < |M|$, which means that the number of jobs at level k in $\mathcal{G}(J)$ is strictly less than the number of machines to which $J \setminus J_1 \cup J_2 \cup \cdots \cup J_k$ can be scheduled. It implies the existence of a job $j \in J \setminus J_1 \cup J_2 \cup \cdots \cup J_k$ that can be assigned to one of the machines with load k or lower, while $\mathcal{G}(J)$ assigned j to a machine with load higher than k. It contradicts to the fact that $\mathcal{G}(J)$ is a greedy algorithm. Thus we have $L_k \geq L_k^*$ and the lemma follows. We show in the following paragraphs that Inequality 4 holds.

A property for proving Inequality 4. Before proving Inequality 4, we describe a property of Algorithm 1. In the algorithm, we have an order of job removal. We remove the jobs from the least-loaded machine in $\mathcal{C}$. The order of job removal decreases the number of non-empty machines in a rapid way. In fact, the job removal has the least number of non-empty machines among all job removals with the same number of removed jobs. This is because we can change other orders to the order without decreasing the number of empty machines. It is either both orders have the same set of empty machines or the other order has an empty machine whose original load is larger than the original loads of all empty machines in our order. For both the cases, we can replace the other order by our order without decreasing the number of empty machines. Recall that $|M|$ is the number of non-empty machines in $\mathcal{C}$ that remain after the jobs $J_1 \cup J_2 \cup \cdots \cup J_k$ are removed. The number of removed jobs is $L_k = |J_1| + |J_2| + \cdots + |J_k|$. Let M^* be the non-empty machines in $\mathcal{C}$ after L_k jobs are removed according to the removing order of Algorithm 1. Since both M^* and M consider removing the same number of jobs, we have

$$|M^*| \leq |M| \ . \tag{6}$$

Proof of Inequality 4. To show Inequality 4, i.e., $|J_k^*| \leq |M|$, we consider all two cases of Inequality 2. Case 1: $|J_k^*| = |NE(k)|$; and Case 2: $|J_k^*| = |NE(k)| + 1$.

Case 1: $|J_k^*| = |NE(k)|$. Recall that $|M^*|$ is the number of non-empty machines in $\mathcal{C}$ that remain after L_k jobs are removed in the order we used in Algorithm 1. The number of removed jobs is L_k. Recall the assumption $L_k < L_k^*$ we used for the proof by contradiction. We remove more than L_k jobs when generating $\mathcal{S}^*$ up to level k. We defined $|NE(k)|$ to be the number of non-empty machines that remain after L_k^* jobs are removed from $\mathcal{C}$ in the order. Since we remove more jobs when considering $NE(k)$ than considering M^*, we have $|NE(k)| \leq |M^*|$. By combining this inequality, the case assumption and Inequality 6, we have $|J_k^*| = |NE(k)| \leq |M^*| \leq |M|$. Inequality 4, i.e., $|J_k^*| \leq |M|$, is proved for this case.

Case 2: $|J_k^*| = |NE(k)|+1$. Let j be the last considered job in J_k^* in the order we used in Algorithm 1. The case $|J_k^*| = |NE(k)| + 1$ comes from the situation when j is the last removed job on machine $m_{\mathcal{C}}(j)$. That is, the removal of j causes a new empty column in $\mathcal{C}$. Recall that $|NE(k)|$ is the number of non-empty machines that remain after L_k^* jobs are removed from $\mathcal{C}$ in the order.

In the case where $|J_k^*| = |NE(k)| + 1$, these non-empty machines are full, i.e., no job is removed from these machines. This means if we add back (undo job removal) any number of jobs, the number of non-empty machines increases by at least 1. Recall that $|M^*|$ is the number of non-empty machines in $\mathcal{C}$ that remain after L_k jobs are removed in the order. Also recall the assumption $L_k < L_k^*$ we used for the proof by contradiction. We remove fewer jobs when considering M^* than considering $NE(k)$. This means $|M^*|$ is at least one more than $|NE(k)|$ (since j is the last removed job on $m_{\mathcal{C}}(j)$), i.e., $|M^*| \geq |NE(k)| + 1$. By combining this inequality, the case assumption and Inequality 6, we have $|J_k^*| = |NE(k)| + 1 \leq |M^*| \leq |M|$. Inequality 4, i.e., $|J_k^*| \leq |M|$, is also proved for this case. □

By Lemma 4 and Theorem 2, we have the following statement.

Corollary 1 *Given a configuration $\mathcal{C}$, the job instance J^* in $\mathcal{I}_{\mathcal{C}}$, and any job instance $J \in \mathcal{I}_{\mathcal{C}}$, we have $\mathcal{G}(J) \leq \mathcal{G}(J^*)$.*

4.2 A job instance J' for an online algorithm $\mathcal{A}$

Given an arbitrary deterministic online algorithm $\mathcal{A}$ and an optimal configuration $\mathcal{C}$ with the corresponding set of job instances $\mathcal{I}_{\mathcal{C}}$, we prove that there is a bad instance $J' \in \mathcal{I}_{\mathcal{C}}$ for $\mathcal{A}$ such that $\mathcal{A}(J') \geq \mathcal{G}(J^*)$.

4.2.1 Construction of the job instance J'

Similar to the construction of J^*, we aim to construct a job instance J' which has a high cost for a given online algorithm $\mathcal{A}$. However, unlike the greedy strategy, we have no knowledge about the behavior of $\mathcal{A}$. Instead, we reference a *simulation* of $\mathcal{A}$ as an oracle and design the job instance such that every decision made by $\mathcal{A}$ makes some troubles for itself in the future. Since $\mathcal{A}$ is an online algorithm, it is practicable for us to make use of the history of $\mathcal{A}$ and design the next group of released jobs such that the previous decisions of $\mathcal{A}$ become bad choices. The construction is detailed in Algorithm 2 and explained below.

Consider an optimal configuration $\mathcal{C} = (v_1, v_2, \cdots, v_h)$ where $0 < v_1 \leq v_2 \leq \cdots \leq v_h$. The job instance J' is constructed in rounds where in round i, for $1 \leq i \leq h$, we construct J'_i, which contains v_i jobs corresponding to the jobs at column v_i in $\mathcal{C}$ — this association to jobs in $\mathcal{C}$ and hence to J^* is crucial in the analysis to be shown later. All the jobs in J'_i have the same set of $h - (i-1)$ permitted machines, which we denote by $B(i)$. The machines in $B(i)$ is determined based on the simulation [1] of $\mathcal{A}$ on jobs in $J'_1, \cdots, J'_{i-1}$. Precisely, $B(i)$ is chosen to contain the **highest-load** machines in $\mathcal{A}$, ties are broken by taking machine with smaller machine index. The reason to choose the highest loaded machines as permitted machines is that we want to make

[1] We note that we can refer to the simulation of $\mathcal{A}$ since $\mathcal{A}$ is an online algorithm.

Algorithm 2 Find J'

Input: The given configuration $\mathcal{C} = (v_1, v_2, \cdots, v_h)$, where $0 < v_1 \leq v_2 \leq v_3 \leq \cdots \leq v_h$.
Online scheduling algorithm $\mathcal{A}$
Output: Job instance J' with job subsets $J'_1, J'_2, \cdots, J'_h$
Schedule $\mathcal{S}'$ of J'

1: **for** round $i = 1, 2, \cdots, h$ **do**
2: $B(i) \leftarrow$ the first $h - (i-1)$ machines with the highest loads in $\mathcal{A}(\bigcup_{k=1}^{i-1} J'_k)$.
3: $J'_i \leftarrow$ jobs at machine v_i in $\mathcal{C}$
4: For each job $j \in J'_i$, the permitted machines $P'_j \leftarrow B(i)$
5: The jobs in J'_i are released after J'_{i-1}.
6: For jobs within J'_i, the jobs with lower level in $\mathcal{C}$ are released before the jobs with higher level.
7: **end for**
8: $\mathcal{S}' \leftarrow \mathcal{A}(\bigcup_{i=1}^{h} J'_i)$.

the load of $\mathcal{A}$ continue to accumulate on the higher load machines (versus the situation the configuration $\mathcal{C}$ that the corresponding jobs are on the relatively lower load machines). At the end of constructing J'_h, we set $J' = \cup_{1 \leq i \leq h} J'_i$ and we denote the schedule of $\mathcal{A}$ on J' by $\mathcal{S}'$.

To further illustrate the construction, for round 1, J'_1 contains v_1 jobs and the permitted machines $B(1)$ contains machines 1 to h. For round 2, J'_2 contains v_2 jobs and $B(2)$ contains $h-1$ machines with the highest loads after $\mathcal{A}$ schedules J'_1. Figure 3 gives an example demonstrating how we find the job instance J'. In particular, Figure 3c is the corresponding J' for some online algorithm $\mathcal{A}$ of the configuration in Figure 3a.

To complete the construction, we also need to specify the release order of jobs (Line 6 in Algorithm 2). From the discussion before, it is required that the jobs in J'_i are released after J'_{i-1}. As for jobs in J'_i, the jobs with lower level in $\mathcal{C}$ are released before the jobs with higher level.

4.2.2 $J' \in \mathcal{I}_{\mathcal{C}}$

In this section, we prove that $J' \in \mathcal{I}_{\mathcal{C}}$, i.e., $\mathcal{O}(J')$ has the same configuration as $\mathcal{C}$ as stated in the following lemma.

Lemma 5 *$\mathcal{O}(J')$ has the same configuration as $\mathcal{C}$.*

Proof We prove this lemma by arguing that $\mathcal{C}$ is an optimal schedule of J'. We use the critical interval argument in [49] and claim that the schedule returned by the YDS algorithm on J' is the same as $\mathcal{C}$.

The YDS algorithm solves the dynamic voltage scheduling (DVS) problem. Our problem resembles the DVS problem. The machines in our problem is in analogy to time points in the DVS problem. The permitted machines in our problem is in analogy to feasible time points in the DVS problem. In the DVS problem, feasible time points are contiguous, but in our problem, permitted machines can be incontiguous. Nevertheless, we will show that J' has a property that we can rearrange the machines such that the permitted

machines for each job is contiguous. The objective of the DVS problem is

$$\sum_t (\text{ load of time point } t)^\alpha$$

for some constant $\alpha > 1$. The analysis of the YDS algorithm only needs the cost function to be convex and not necessarily the α-th power, so the analysis applies on general convex functions. This means the YDS algorithm also solves the objective

$$\sum_t f(\text{ load of time point } t) \tag{7}$$

for some convex function f. Formula 7 is the same as the objective in our problem.

For completeness, we state the YDS algorithm [49] with adaptions:

Definition 4 Given a set of machines S, the *density* is defined as the following:

$$d(S) = \frac{|\{j : P_j \subseteq S\}|}{|S|} \ ,$$

where P_j is the permitted machines of job j.

Definition 5 A *critical set* is a set of machines such that the density is maximized.

The YDS algorithm. Repeat the following steps until there are no jobs:

1) Find the critical set S and evenly assign jobs j with permitted machines $P_j \subseteq S$ to machines in S.

2) Remove S from the set of machines and remove the jobs which are already assigned.

YDS is proven to be optimal for minimizing the total energy consumption [49]. By the convexity of the cost function in our problem, it implies that the YDS algorithm optimally solves our problem. That is, if in each critical set S, only jobs j with $P_j \subseteq S$ are executed, the schedule is optimal.

Note that to adapt the critical interval argument, we need to rearrange the machines so that the set of permitted machines of each job forms a contiguous interval. It can be done since by our construction, the permitted machines of any two subset of jobs, J'_x and J'_y, $B(x) \subseteq B(y)$ if $x > y$. Also, since in our problem each of the jobs are unit-size, there is no *preemption* issue as the analogue of the dynamic voltage scheduling problem.

In the following, we are going to prove that the schedule generated by the YDS algorithm on job instance J' is the same as $\mathcal{C}$. Recall that J'_i is the set of jobs on the $(h-(i-1))$-th highest load machine in $\mathcal{C}$, and $B(i)$ is the set of permitted machines of jobs in J'_i. Denote S_i as $B(i) \setminus B(i+1)$ (let $B(h+1)$ be ϕ), the set $S_{h-(i-1)}$ is the i-th critical set and there is only one machine inside, which is the i-th highest load machine (with load $v_{h-(i-1)}$) in $\mathcal{C}$. The reason is that by the construction of J', for each $1 < i \leq h$, $|J'_i| \geq |J'_{i-1}|$ and $B(i-1) = B(i) \cup \{M_i\}$ where M_i is the machine with the $(h-(i-2))$-th

highest load in $\mathcal{C}$. Hence $d(B(i)) \geq d(B(i-1))$. By induction, S_h is the first critical set, S_{h-1} is the second critical set, and so on. Furthermore, in the YDS schedule on the input J', J'_i is the set of jobs assigned to the machine in S_i.

By the construction in Algorithm 2, J'_i is the set of jobs assigned to the machine with the $(h-(i-1))$-th highest load in $\mathcal{C}$, which is the machine in $B(i) \setminus B(i+1)$ (let $B(h+1)$ be ϕ). Hence, the schedule constructed by the YDS algorithm is the same as $\mathcal{C}$. □

4.2.3 $\mathcal{A}(J') \geq \mathcal{G}(J^*)$

Recall that the schedule $\mathcal{S}'$ produced by Algorithm 2 is $\mathcal{A}(J')$. We compare $\mathcal{S}'$ with the greedy schedule $\mathcal{S}^* = \mathcal{G}(J^*)$ produced by Algorithm 1.

Observation 4 *Given an optimal configuration $\mathcal{C}$, there exists constructions of J^* and J' such that for any job in $\mathcal{C}$, the corresponding jobs in J^* and J' have the same position in the release orders in J^* and J'.*

Proof The release orders are the same due to Line 6 in Algorithm 1 and Line 6 in Algorithm 2. □

To compare $\mathcal{S}'$ and $\mathcal{S}^*$, we construct J' and J^* simultaneously with the same release order and compare their accumulated number of jobs L'_k and L^*_k (for $\mathcal{S}'$ and $\mathcal{S}^*$ respectively) for each k.

Consider the output J^* of Algorithm 1 given configuration $\mathcal{C}$. We denote by q the number of levels in $\mathcal{S}^*$, which means $J^* = \bigcup_{i=1}^{q} J^*_i$. According to Algorithm 1, $J^*_1, J^*_2, \cdots, J^*_q$ is the release order of J^*. According to Observation 4, there exists the same release order of jobs for finding J^* and J' under the same given configuration $\mathcal{C}$. We denote by Q the sequence of jobs representing the common release order of J^* and J'. The sequence Q is defined to only involve the order of the jobs but not their permitted machines. We partition Q by the sizes $|J^*_1|, |J^*_2|, \cdots, |J^*_q|$ and denote by $Q(i)$ for $1 \leq i \leq q$ the subsequence of Q starting from the L^*_{i-1}-th job (exclusively) and ending with the L^*_i-th job (inclusively). It implies that $|Q(i)| = |J^*_i|$ and q is the number of subsequences we partition Q into, i.e., $Q = \bigcup_{i=1}^{q} Q(i)$. When we release the jobs Q to Algorithm 1, the jobs $Q(i)$ are the jobs at level i in $\mathcal{S}^*$. On the other hand, when we release the jobs Q to Algorithm 2, we do not know where $\mathcal{A}$ will schedule these jobs. Instead, we consider the permitted machines of these jobs. To indicate the permitted machines, we use P'_j to point out that the permitted machine of job $j \in Q$ is defined by Algorithm 2. We denote by $P'(i)$ the union of the permitted machines (defined by Algorithm 2) of jobs $Q(i)$, i.e., $P'(i) = \bigcup_{j \in Q(i)} P'_j$.

Lemma 6 *Consider the schedules $\mathcal{S}'$ and $\mathcal{S}^*$ and the corresponding accumulated size L'_k and L^*_k, we have $L^*_k \geq L'_k$ for all $k \geq 1$. The inequalities hold even if $k > q$, where q is the number of levels in $\mathcal{S}^*$.*

Proof By releasing the common sequence of jobs Q (obtained from Observation 4 with a configuration $\mathcal{C}$) to Algorithm 1 and Algorithm 2, we obtain $\mathcal{S}^*$ and $\mathcal{S}'$ respectively. To further observe the properties of $\mathcal{S}'$, we consider partial schedules of $\mathcal{S}'$. A partial schedule $\mathcal{S}'(i)$ of $\mathcal{S}'$ is the schedule obtained by releasing $Q(1), Q(2), \cdots, Q(i)$ to Algorithm 2. In other words, $\mathcal{S}'(i+1)$ is obtained by releasing $Q(i+1)$ to Algorithm 2 based on schedule $\mathcal{S}'(i)$. We note that the jobs in $Q(i+1)$ may not be released in the same round in Algorithm 2. But we can obtain the same schedule $\mathcal{S}'$ by releasing the jobs $Q(1), Q(2), \cdots, Q(q)$ accordingly. We denote by $l'_k(i)$ the set of jobs at level k of schedule $\mathcal{S}'(i)$, i.e., $l'_k(i) = \{j \in \bigcup_{x=1}^{i} Q(x) \mid \text{level}_{\mathcal{A}}(j) = k\}$. Since $\mathcal{S}'(i+1)$ is obtained by adding jobs to $\mathcal{S}'(i)$ without modifying any existing job in $\mathcal{S}'(i)$, we have $l'_k(i) \subseteq l'_k(i+1)$. We denote by $L'_k(i)$ the accumulated number of jobs up to $Q(i)$, i.e., $L'_k(i) = \sum_{x=1}^{k} |l'_x(i)|$. It implies $L'_k = L'_k(q)$. Since $l'_k(i) \subseteq l'_k(i+1)$, we have $L'_k(i) \leq L'_k(i+1)$. We prove the lemma by induction on two parameters, index i and level k. For each inductive step, we increase the index by one, to be $i+1$, and consider the levels up to level $(i+1)$, namely, $k \leq (i+1)$. More precisely, we show that if $L^*_k \geq L'_k(i)$ for all $k \leq i$, then $L^*_k \geq L'_k(i+1)$ for all $k \leq (i+1)$. Thus we obtain $L^*_k \geq L'_k(q)$ for all $k \leq q$. Since $|Q|$ is the total number of jobs, we have $L'_k(q) \leq |Q|$ for all k. And since the total number of levels in $\mathcal{S}^*$ is q, we have $L^*_q = |Q| \geq L'_k(q)$ for all k. Thus we have $L^*_k \geq L'_k(q)$ for all k (even for $k > q$, where $\mathcal{S}'$ has more levels than $\mathcal{S}^*$). Since $L'_k = L'_k(q)$, the lemma follows.

For the base case, we show that $L^*_1 \geq L'_1(1)$. By the definition of L^*_i, we have $L^*_1 = |J^*_1| = |Q(1)|$. Since there are only $|Q(1)|$ jobs released, $L'_1(1) \leq |Q(1)|$. By combining the two results, we have $L^*_1 = |Q(1)| \geq L'_1(1)$.

For the inductive step, we show that if $L^*_k \geq L'_k(i)$ for all $k \leq i$, then $L^*_k \geq L'_k(i+1)$ for all $k \leq (i+1)$. We first show that $L^*_{i+1} \geq L'_{i+1}(i+1)$. For index $(i+1)$, we release the job set $Q(i+1)$. We do not know in which levels $\mathcal{A}$ schedules these jobs. However, we only need to consider the jobs being scheduled in level 1 to $(i+1)$. In the worst case, $\mathcal{A}$ schedules all the jobs in $Q(i+1)$ in level 1 to $(i+1)$ and no job is scheduled in higher levels. It implies that $L'_{i+1}(i+1) \leq L'_i(i) + |Q(i+1)|$. Since $L^*_i \geq L'_i(i)$ by the inductive assumption, we have

$$\begin{aligned} L^*_{i+1} &= L^*_i + |J^*_{i+1}| = L^*_i + |Q(i+1)| \\ &\geq L'_i(i) + |Q(i+1)| \geq L'_{i+1}(i+1) \ . \end{aligned}$$

Before proving the next case, we recall the properties of Algorithm 1 and 2. From Inequality 2, we have $|J^*_i| \geq |NE(i)|$. Since J^*_i is the job set at the i-th level of $\mathcal{S}^*$, by the definition of levels, $|J^*_{i-1}| \geq |J^*_i|$. Thus we have $|J^*_k| \geq |NE(i)|$ for all $k \leq i$.

On the other hand, we relate Algorithm 1 and Algorithm 2 by relating $NE(i)$ in Algorithm 1 to $P'(i+1)$ regarding the corresponding P'_j in Algorithm 2. More precisely, we claim that $|NE(i)|$ is an upper bound for $|P'(i+1)|$ as explained below. By Algorithm 2, if a job j is released in round x, then the number of permitted machines of j is $h - x + 1$ where h is the total number

of columns in $\mathcal{C}$. By removing all the jobs released before j from $\mathcal{C}$, $h-x+1$ is also the number of non-empty columns in the modified $\mathcal{C}$. This argument holds no matter where j is in the common sequence Q. For ease of analysis, we assume that j is the first (released) job in $Q(i+1)$. In Algorithm 1, by removing all the jobs released before j from $\mathcal{C}$, i.e., removing all the jobs $Q(1), Q(2), \cdots, Q(i)$, $NE(i)$ is defined to be the non-empty columns in the modified $\mathcal{C}$. Thus $h-x+1 \leq |NE(i)|$. Since in Algorithm 2, the permitted machines of the jobs released after j are chosen from the machines with the highest loads, these permitted machines are subsets of j's permitted machines, namely, subsets of the $h-x+1$ permitted machines. Thus we have $|P'(i+1)| \leq h-x+1 \leq |NE(i)|$.

Now we show that $L_k^* \geq L'_k(i+1)$ for all $k \leq i$. Recall that $|P'(i+1)| \leq |NE(i)|$, which means the number of permitted machines of jobs in $Q(i+1)$ is at most $|NE(i)|$. For each level k, we consider two cases: Case 1, $|l'_k(i+1)| \geq |NE(i)|$; and Case 2, $|l'_k(i+1)| < |NE(i)|$. For Case 1, since the $|P'(i+1)|$ permitted machines are chosen from the machines with the highest loads, $\mathcal{A}$ cannot add more jobs to some level k for $k \leq i$ if level k already has at least $|P'(i+1)|$ jobs. Now we have $|l'_k(i+1)| \geq |NE(i)|$ and $|P'(i+1)| \leq |NE(i)|$, which means $|l'_k(i+1)| \geq |P'(i+1)|$. Thus $\mathcal{A}$ cannot add jobs into level k and we have $|l'_k(i+1)| = |l'_k(i)|$.

For Case 2, since the jobs in $Q(i+1)$ have only $|P'(i+1)|$ permitted machines, $\mathcal{A}$ can increase the number of jobs for some level k for $k \leq i$ up to $|P'(i+1)|$ if level k has less than $|P'(i+1)|$ jobs. Since $|P'(i+1)|$ is upper bounded by $|NE(i)|$ and level k may have fewer jobs than $|P'(i+1)|$, the number of jobs for level k may be increased by $\mathcal{A}$ but must not be more than $|NE(i)|$. Thus we have $|l'_k(i+1)| \leq |NE(i)|$. Recall from the previous paragraphs that $|J_i^*| \geq |NE(i)|$. We have $|l'_k(i+1)| \leq |J_i^*|$. Since $|J_i^*| \leq |J_k^*|$ for $k \leq i$, we have $|l'_k(i+1)| \leq |J_k^*|$ for $k \leq i$. On the other hand, by the definition of levels, we have $|l'_k(i+1)| \leq |l'_{k-1}(i+1)|$. Since the number of jobs decreases when k increases for level k, the levels in both the cases are contiguous. This means there exists a level z such that the levels from 1 to z are in Case 1 and the levels from $(z+1)$ to k are in Case 2. By combining the two cases and the inductive assumption, for all $k \leq i$, we have

$$\begin{aligned}
L'_k(i+1) &= \sum_{x=1}^{k} |l'_x(i+1)| \\
&= \left(\sum_{x=1}^{z} |l'_x(i+1)|\right) + \left(\sum_{x=z+1}^{k} |l'_x(i+1)|\right) \\
&\leq \left(\sum_{x=1}^{z} |l'_x(i)|\right) + \left(\sum_{x=z+1}^{k} |J_x^*|\right) \\
&= L'_z(i) + \sum_{x=z+1}^{k} |J_x^*| \leq L_z^* + \sum_{x=z+1}^{k} |J_x^*| = L_k^* \ .
\end{aligned}$$

□

By Lemma 6 and Theorem 2, we have the following statement.

Corollary 2 *Given any deterministic online algorithm $\mathcal{A}$, $\mathcal{G}(J^*) \leq \mathcal{A}(J')$.*

5 Conclusion

We have shown the optimality of greedy algorithm for online grid scheduling and restricted assignment problem for unit-sized jobs. Nevertheless, we have not been able to derive the precise competitive ratio of the greedy algorithm. It is therefore of immediate interest to find the competitive ratio. As mentioned in the introduction, in the restricted assignment problem for arbitrary sized jobs, the greedy algorithm is almost the best online algorithm. Deriving a similar result for the grid scheduling problem would be of interest.

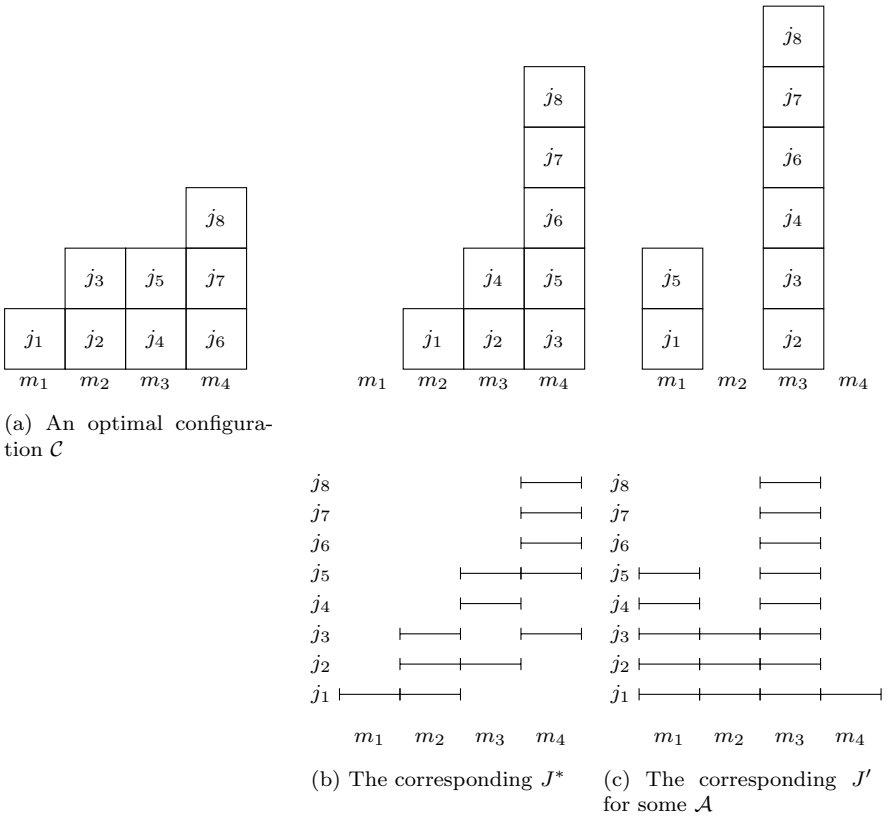
Acknowledgements The authors would like to thank Marcin Bienkowski for helpful discussion.

References

1. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S.A., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (1997)
2. Azar, Y.: On-line load balancing. In: A. Fiat, J. Woeginger (eds.) Online Algorithms: The State of the Art, pp. 178–195. Springer (1998)
3. Azar, Y., Broder, A.Z., Karlin, A.R.: On-line load balancing. Theor. Comput. Sci. **130**(1), 73–84 (1994)
4. Azar, Y., Kalyanasundaram, B., Plotkin, S.A., Pruhs, K., Waarts, O.: On-line load balancing of temporary tasks. J. Algorithms **22**(1), 93–110 (1997)
5. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**(2), 221–237 (1995)
6. Bartal, Y., Fiat, A., Karloff, H.J., Vohra, R.: New algorithms for an ancient scheduling problem. J. Comput. Syst. Sci. **51**(3), 359–366 (1995)
7. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
8. 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)
9. Caragiannis, I.: Better bounds for online load balancing on unrelated machines. In: SODA, pp. 972–981 (2008)
10. Caragiannis, I., Flammini, M., Kaklamanis, C., Kanellopoulos, P., Moscardelli, L.: Tight bounds for selfish and greedy load balancing. Algorithmica **61**(3), 606–637 (2011)
11. Caron, S., Kesidis, G.: Incentive-based energy consumption scheduling algorithms for the smart grid. In: SmartGridComm, pp. 391–396. IEEE (2010)
12. Chau, V., Feng, S., Thang, N.K.: Competitive algorithms for demand response management in smart grid. In: LATIN, pp. 303–316 (2018)
13. 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)
14. Ebenlendr, T., Krcál, M., Sgall, J.: Graph balancing: A special case of scheduling unrelated parallel machines. Algorithmica **68**(1), 62–80 (2014)
15. European Commission: Europen smartgrids technology platform. `ftp://ftp.cordis.europa.eu/pub/fp7/energy/docs/smartgrids_en.pdf` (2006)

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. Graham, R.L.: Bounds for certain multiprocessing anomalies. The Bell System Technical Journal **45**(9), 1563–1581 (1966)
20. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal of Applied Mathematics **17**(2), 416–429 (1969)
21. Hamilton, K., Gulhar, N.: Taking demand response to the next level. IEEE Power and Energy Mag. **8**(3), 60–65 (2010)
22. Huo, Y., Leung, J.Y.: Fast approximation algorithms for job scheduling with processing set restrictions. Theor. Comput. Sci. **411**(44-46), 3947–3955 (2010)
23. Ipakchi, A., Albuyeh, F.: Grid of the future. IEEE Power & Energy Mag. **7**(2), 52–62 (2009)
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. Kolliopoulos, S.G., Moysoglou, Y.: The 2-valued case of makespan minimization with assignment constraints. Inf. Process. Lett. **113**(1-2), 39–43 (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. Lam, T.W., Ting, H., To, K., Wong, P.W.H.: On-line load balancing of temporary tasks revisited. Theor. Comput. Sci. **270**(1-2), 325–340 (2002)
29. Lee, K., Leung, J.Y., Pinedo, M.L.: A note on graph balancing problems with restrictions. Inf. Process. Lett. **110**(1), 24–29 (2009)
30. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**, 259–271 (1990)
31. 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)
32. Liu, F., Liu, H., Wong, P.W.H.: Greedy is optimal for online restricted assignment and smart grid scheduling for unit size jobs. In: WAOA, pp. 217–231 (2019)
33. Liu, F., Liu, H., Wong, P.W.H.: Non-preemptive scheduling in a smart grid model and its implications on machine minimization. Algorithmica **82**(12), 3415–3457 (2020)
34. Lockheed Martin: SEELoad^{TM} Solution. `http://www.lockheedmartin.co.uk/us/products/energy-solutions/seesuite/seeload.html`
35. 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)
36. Lui, T., Stirling, W., Marcy, H.: Get smart. IEEE Power & Energy Mag. **8**(3), 66–78 (2010)
37. 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)
38. Masters, G.M.: Renewable and efficient electric power systems. John Wiley & Sons (2013)
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. Muratore, G., Schwarz, U.M., Woeginger, G.J.: Parallel machine scheduling with nested job assignment restrictions. Oper. Res. Lett. **38**(1), 47–50 (2010)
41. REGEN Energy Inc: ENVIROGRID^{TM} SMART GRID BUNDLE. `http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/`
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. Svensson, O.: Santa claus schedules jobs on unrelated machines. SIAM J. Comput. **41**(5), 1318–1341 (2012)
44. Toronto Hydro Corporation: Peaksaver Program. http://www.peaksaver.com/peaksaver_THESL.html
45. 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)
46. US Department of Energy: The Smart Grid: An Introduction. http://www.oe.energy.gov/SmartGridIntroduction.htm (2009)
47. Verschae, J., Wiese, A.: On the configuration-lp for scheduling on unrelated machines. J. Scheduling **17**(4), 371–383 (2014)
48. Wang, C., Sitters, R.: On some special cases of the restricted assignment problem. Inf. Process. Lett. **116**(11), 723–728 (2016)
49. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995, pp. 374–382 (1995)
50. 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)



(a) An optimal configuration $\mathcal{C}$

(b) The corresponding J^*

(c) The corresponding J' for some $\mathcal{A}$

Fig. 3: An example of finding J^* and J'. (a) is a configuration with 8 jobs and 4 machines. To obtain J^*, we first remove from (a) the jobs j_1, j_2 and j_3, which are in the least-loaded machines and the number of such jobs is at least the number of non-empty machines: m_3 and m_4. These 3 jobs are assigned to m_2, m_3 and m_4 in J^* respectively. Then we remove j_4 and j_5 from (a) and assign them evenly to the second level of J^*. After that, we stack the remaining jobs onto J^* such that each job occupies a level since the current number of non-empty machines in (a) is at most 1. The bottom part of (b) shows the permitted machines of each job which is the union of machines the job assigned to in (a) and (b). On the other hand, for finding J' for some $\mathcal{A}$, we first release j_1, which is at the least-loaded machine in (a), with all machines being permitted and get that $\mathcal{A}$ schedules j_1 to m_1. Then we release j_2 and j_3 with the permitted machines of the 3 highest-load machines in the current J_A, which are m_1, m_2 and m_3. And then we release j_4 and j_5 with the permitted machines m_1 and m_3. Finally we release j_6, j_7 and j_8 with the current highest-load machine as their permitted machine.