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

Fu-Hong Liu[1], Hsiang-Hsuan Liu[2][0000−0002−0194−9360]⋆, and Prudence W.H. Wong[3][0000−0001−7935−7245]⋆⋆

[1] Dept. of Computer Science, National Tsing Hua University, Taiwan
fhliu@cs.nthu.edu.tw
[2] Dept. Information & Computing Sciences, Utrecht University, The Netherlands
Institute of Computer Science, Wroclaw University, Poland
H.H.Liu@uu.nl
[3] Dept. of Computer Science, University of Liverpool, UK
pwong@liverpool.ac.uk

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.

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

⋆ Partially supported by Polish National Science Centre grant 2016/22/E/ST6/00499. This work was partially done when Hsiang-Hsuan Liu worked in Wroclaw University, Poland.

⋆⋆ Supported by Networks Sciences & Technologies(NeST), School of EEECS, University of Liverpool.

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 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.

Smart grid scheduling. The smart grid scheduling problem arises in demand response management in electrical smart grid [16,21,23,35,48] - one of the major challenges in the 21st century [15, 44, 45]. The smart grid [17, 37] 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 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, 45]. Demand response management is to reduce peak load by shifting demand to non-peak hours [11, 26, 34, 36, 38, 41] 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 of for the supplier operating the grid [34]. On the other hand, it can reduce electricity bill for consumers as it is common that suppliers charge according to generation cost [41]. Research initiatives in the area include [24, 33, 40, 43].

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 [7] 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. load) it takes to complete if it is assigned in the corresponding machine. The objective is to minimize over the machines the total load of jobs scheduled on each machine. For the restricted assignment problem, each job is associated with a processing time (a.k.a. size) and a subset of machines that the job can be scheduled on. As pointed out in [7], 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 competitive ratio.

Roughly speaking, 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 two problems in concern, hence, the optimality holds for both problems.

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 [10]. The solution iteratively assigns each request and rearranges the assignment to maintain optimality. 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 [31], the greedy algorithm is indeed at least 3-competitive. Hence, it is still an open problem that how good or bad the greedy strategy is. Our results in this paper establish the optimality of the greedy algorithm.

For arbitrary power requirement and service duration, the problem becomes NP-hard [10, 26]. Theoretical study on this problem mainly focuses on the cost function $f(t) = \text{load}^\alpha(t)$ [12,32]. In particular, 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. Other work on demand response management can be found in [26, 34, 35, 41].

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 showed 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 [8]. For restricted assignment, Azar et al. [7] 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.

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 [42] 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, 39, 46, 47]. The (restricted and unrestricted) assignment problem has also been studied for temporary jobs that depart [2–4, 6, 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. Due to space limit, proofs are given in the full paper.

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 where the job can be assigned to. 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 where j is assigned to 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 $\mathcal{S}(J)$.

Online model. We consider the online model. The jobs are released one by one, and the released job has to be scheduled before the next one is released. And 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* [9], 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 $\mathcal{A}$, $level_{\mathcal{A}}(j)$, 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 $level_{\mathcal{A}}(j)$ means that it is the $level_{\mathcal{A}}(j)$-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, $L_{\mathcal{S}}^{(k)} := |\{j : level_{\mathcal{S}}(j) \in [1,k]\}|$.*

Theorem 2. *Given two schedules $\mathcal{S}_1$ and $\mathcal{S}_2$ which have the same number of jobs (which are not necessary of the same job instance), if $L_{\mathcal{S}_1}^{(k)} \geq L_{\mathcal{S}_2}^{(k)}$ for all $k \geq 1$, then the cost of $\mathcal{S}_1$ is at most that of $\mathcal{S}_2$.*

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

Remark: Note that Theorem 2 also holds for the objective of minimizing the maximum load over machines. This objective is equivalent to ℓ_∞ norm by viewing the loads of machines as a vector. Since ℓ_p norm for any $p \geq 1$ is a valid total cost function for the problem, the proof of Theorem 2 applies to ℓ_p norm.

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 which job in which machine. We further formalize this notion as configuration of a schedule.

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.

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 $(0, 1, 2, 2, 5)$.

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.

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 the same as $\mathcal{C}$. 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.

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

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 $J_{\mathcal{G}}$, i.e., for a given $\mathcal{C}$, $J_{\mathcal{G}} = \arg\max_{J \in \mathcal{I}_{\mathcal{C}}} \mathcal{G}(J)$. 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 $J_{\mathcal{A}} \in \mathcal{I}_{\mathcal{C}}$ such that $\mathcal{A}(J_{\mathcal{A}}) \geq \mathcal{G}(J_{\mathcal{G}})$. This implies $\frac{\mathcal{G}(J_{\mathcal{G}})}{\mathcal{C}} \leq \frac{\mathcal{A}(J_{\mathcal{A}})}{\mathcal{C}} = \frac{\mathcal{A}(J_{\mathcal{A}})}{\mathcal{O}(J_{\mathcal{A}})}$, where the last equality is because that $J_{\mathcal{A}} \in \mathcal{I}_{\mathcal{C}}$. We can then bound the competitive ratio of $\mathcal{G}$ by that of $\mathcal{A}$ as follows:

$$\mathcal{R}(\mathcal{G}) = \max_{\mathcal{I}_{\mathcal{C}}} \max_{J \in \mathcal{I}_{\mathcal{C}}} \frac{\mathcal{G}(J_{\mathcal{G}})}{\mathcal{C}} \leq \max_{\mathcal{I}_{\mathcal{C}}} \max_{J \in \mathcal{I}_{\mathcal{C}}} \frac{\mathcal{A}(J_{\mathcal{A}})}{\mathcal{O}(\mathcal{A})} = \mathcal{R}(\mathcal{A}) \ .$$

Then we can conclude Theorem 1.

4 Optimality of the greedy algorithm

In this section, we construct $J_{\mathcal{G}}$ and $J_{\mathcal{A}}$ as required in the framework in Section 3.

4.1 The job instance $J_{\mathcal{G}}$ for the greedy algorithm $\mathcal{G}$

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

We find $J_{\mathcal{G}}$ by artificially designing a job instance. More specifically, $J_{\mathcal{G}}$ has the same number of jobs as the given $\mathcal{C}$, and we design the set of permitted machines of each job and the release order of the jobs. First, we transform the given $\mathcal{C}$ to schedule $\mathcal{S}_{\mathcal{G}}$ by changing the configuration. We make sure that $\mathcal{C}$ is the optimal configuration of $J_{\mathcal{G}}$ (Lemma 2) and the schedule $\mathcal{S}_{\mathcal{G}}$ is the consequence of running a greedy algorithm on $J_{\mathcal{G}}$ (Lemma 1). To achieve this, we design the set of permitted machines of each job and choose the release order carefully

Although the job instance $J_{\mathcal{G}}$ seems to be artificial, we can prove that $\mathcal{G}(J_{\mathcal{G}})$ 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 $J_{\mathcal{G}}$.

Construction of the job instance $J_{\mathcal{G}}$. 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 $\mathcal{S}_{\mathcal{G}}$. Assume that $\mathcal{C} = (v_1, v_2, \cdots, v_k)$, where $0 < v_1 \leq v_2 \leq \cdots \leq v_k$, we treat $\mathcal{C}$ as building blocks with k columns and each column i has v_i blocks (where each block corresponding to one job). The transformation runs in rounds, in each

round, we choose certain number of blocks, remove them from $\mathcal{C}$ and put them in $\mathcal{S}_\mathcal{G}$ (which is initially empty) and produce another configuration. During the process, the configuration $\mathcal{C}$ changes to reflect the removing 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 the number of non-zero terms in the configuration at the beginning of round i by n_i. Note that n_i is also the number of non-zero terms in the configuration at the end of round $i-1$.

At the beginning of round i, let $m_1, m_2, \cdots, m_{n_i}$ be the non-empty columns in the configuration and $v_1 \leq v_2 \leq \cdots \leq v_{n_i}$ be the number of blocks in the corresponding non-empty columns. We remove the jobs one by one from the lowest load non-empty column m_1 and update the number of v_1 to reflect the moving. The removing procedure stops once the number of the set of removing blocks in this round, $J_\mathcal{G}^{(i)}$, is greater than or equal to the number of non-empty columns in the current configuration (that is, the configuration after removing the jobs). Notice that by the construction, $n_{i+1} \leq |J_\mathcal{G}^{(i)}| \leq n_{i+1} + 1$.

In round i, after removing the blocks from $\mathcal{C}$, we place them in $\mathcal{S}_\mathcal{G}$ (which is initially empty at the beginning of the first round). Recall that there are n_{i+1} non-empty columns in the (updated) $\mathcal{C}$ at the end of round i. Let K'_i denote the corresponding set of these n_{i+1} non-empty columns. In $\mathcal{S}_\mathcal{G}$, the blocks in $J_\mathcal{G}^{(i)}$ are evenly placed at columns with highest load and cover all columns corresponding to K'_i (Observation 2).

Now we design other parameters of the job instance $J_\mathcal{G}$. 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 $\mathcal{S}_\mathcal{G}$. That is, $P_j = \{m_\mathcal{C}(j)\} \cup \{m_{\mathcal{S}_\mathcal{G}}(j)\}$, where $m_\mathcal{C}(j)$ and $m_{\mathcal{S}_\mathcal{G}}(j)$ are the columns of block j in the configurations $\mathcal{C}$ and $\mathcal{S}_\mathcal{G}$, respectively. The release order of the jobs in $J_\mathcal{G}$ is exactly the order their corresponding blocks removed from $\mathcal{C}$. Algorithm 1 is a demonstration to find the job instance $J_\mathcal{G}$. Figure 1a gives an example configuration and Figure 1b is the corresponding $J_\mathcal{G}$ of the configuration in Figure 1a.

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

Before showing the construction produces a desired $J_\mathcal{G}$, we first show that Algorithm 1 is valid. More specifically, we prove that at the end of round i, the level of each jobs $j \in J_\mathcal{G}^{(i)}$ 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 $\mathcal{S}_\mathcal{G}$ is a greedy schedule for $J_\mathcal{G}$.

Let M_i be the subset of machines we choose to place jobs in $J_\mathcal{G}^{(i)}$, and K'_i be the subset of non-empty machines in the updated configuration $\mathcal{C}'$ at the end of round i. We first observe the relation of machines in M_i and K'_i, which then lead to the feasibility of $\mathcal{S}_\mathcal{G}$ (Lemma 1).

Algorithm 1 Find $J_{\mathcal{G}}$

Input: The given configuration $\mathcal{C} = (v_1, v_2, \cdots, v_k)$, where $0 < v_1 \leq v_2 \leq v_3 \leq \cdots \leq v_k$.

Output: Job instance $J_{\mathcal{G}}$ with job subsets $J_{\mathcal{G}}^{(1)}, J_{\mathcal{G}}^{(2)}, \cdots$
Schedule $\mathcal{S}_{\mathcal{G}}$ of $J_{\mathcal{G}}$

1: $\mathcal{C}' \leftarrow \mathcal{C}$ (we ignore all zero terms and only consider non-zero terms in $\mathcal{C}'$)
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: **while** $|J_{\mathcal{G}}^{(i)}| <$ the number of non-zero entries in $\mathcal{C}' = (v'_1, v'_2, \cdots, v'_{k'_i})$ **do**
5: Let $j \leftarrow$ be a job with lowest level at v'_1 (which is the non-zero vector with the smallest index in $\mathcal{C}'$)
6: Add j to $J_{\mathcal{G}}^{(i)}$
7: Remove j from $\mathcal{C}$; update $\mathcal{C}'$
8: **end while**
9: Let M_i be a set of $|J_{\mathcal{G}}^{(i)}|$ machines such that M_i covers all non-empty machines in $\mathcal{C}'$ (which is K'_i) and the machine $m_{\mathcal{C}}(j)$ of the last job $j \in J_{\mathcal{G}}^{(i)}$
10: Arrange the jobs in $J_{\mathcal{G}}^{(i)}$ evenly at the machines in M_i such that $\text{level}_{\mathcal{S}_{\mathcal{G}}}(j)$ are the same for all jobs $j \in J_{\mathcal{G}}^{(i)}$
11: The permitted machines of job j is $\{m_{\mathcal{C}}(j)\} \cup m_{\mathcal{S}_{\mathcal{G}}}(j)\}$, where $m_{\mathcal{C}}(j)$ and $m_{\mathcal{S}_{\mathcal{G}}}(j)$ are the machines j is assigned in $\mathcal{C}$ and $\mathcal{S}_{\mathcal{G}}$, respectively.
12: **end for**
13: The release order of jobs is the order they are removed from $\mathcal{C}$
14: **end while**

Observation 2 *There is a $\mathcal{G}$ with some tie breaking to choose a subset M_i of $|J_{\mathcal{G}}^{(i)}|$ machines such that $K'_i \subseteq M_i$.*

Proof. First we notice that $|M_i| = |J_{\mathcal{G}}^{(i)}|$ and $|J_{\mathcal{G}}^{(i)}| \geq |K'_i|$ by the construction (Line 4 in Algorithm 1). By the construction (Line 9), $M_i = K'_i$ or $M_i = K'_i \cup m_{\mathcal{C}}(j)$. The second case is the situation where $m_{\mathcal{C}}(j) \notin K'_i$, that is, the removing of j from $\mathcal{C}$ creates another empty machine. In this case, $|J_{\mathcal{G}}^{(i)}| = |K'_i| + 1$. □

Observation 3 *For all jobs $j \in J_{\mathcal{G}}^{(i)}$, $m_{\mathcal{C}}(j) \in K'_{i-1}$. ($K'_0$ is defined as the whole set of machines.)*

Proof. In the updated $\mathcal{C}'$, in the beginning of round i, the job $j \in J_{\mathcal{G}}^{(i)}$ is at one of the non-empty machines. That is, the position of job j in $\mathcal{C}$, $m_{\mathcal{C}}(j)$ is one of the machines in K'_{i-1}. □

Lemma 1. *$\mathcal{S}_{\mathcal{G}}$ is a greedy schedule for $J_{\mathcal{G}}$.*

Lemma 2. *$J_{\mathcal{G}}$ is a job instance in $\mathcal{I}_{\mathcal{C}}$. That is, the optimal configuration of $J_{\mathcal{G}}$ is $\mathcal{C}$.*

The job instance $J_{\mathcal{G}}$ is the most troublesome among $\mathcal{I}_{\mathcal{C}}$ for $\mathcal{G}$. Now we show that the job instance $J_{\mathcal{G}}$ has the highest greedy cost among all job

instances in $\mathcal{I}_{\mathcal{C}}$. Recall that the schedule $\mathcal{S}_{\mathcal{G}}$ produced by Algorithm 1 is $\mathcal{G}(J_{\mathcal{G}})$. We compare $\mathcal{S}_{\mathcal{G}}$ with any greedy schedule of job instances in $\mathcal{I}_{\mathcal{C}}$.

Lemma 3. *Given any job instance J where $J, J_{\mathcal{G}} \in \mathcal{I}_{\mathcal{C}}$ for some $\mathcal{I}_{\mathcal{C}}$, $L^{(k)}_{\mathcal{G}(J)} \geq L^{(k)}_{\mathcal{S}_{\mathcal{G}}}$ for all k.*

By Lemma 3 and Theorem 2,

Corollary 1. *Given any job instance J where $J, J_{\mathcal{G}} \in \mathcal{I}_{\mathcal{C}}$ for some $\mathcal{I}_{\mathcal{C}}$, $\mathcal{G}(J) \leq \mathcal{G}(J_{\mathcal{G}})$*

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

Given an arbitrary 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 $J_{\mathcal{A}} \in \mathcal{I}_{\mathcal{C}}$ for $\mathcal{A}$ such that $\mathcal{A}(J_{\mathcal{A}}) \geq \mathcal{G}(J_{\mathcal{G}})$.

Find a job instance for any online algorithm $\mathcal{A}$. Similar to the construction of $J_{\mathcal{G}}$, we aim to construct a job instance which has a high cost for the online algorithm $\mathcal{A}$. However, unlike the greedy strategy, we have no knowledge about the behavior of $\mathcal{A}$. Hence we reference $\mathcal{A}$ as an oracle and design the job instance such that every decision made by $\mathcal{A}$ makes some trouble for itself in the future. Note that 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.

Given an optimal configuration $\mathcal{C} = (v_1, v_2, \cdots, v_k)$ where $0 < v_1 \leq v_2 \leq \cdots \leq v_k$. In each round i, we release the set of jobs at column corresponding to v_i as $J^{(i)}_{\mathcal{A}}$. The permitted machines of jobs $j \in J^{(i)}_{\mathcal{A}}$ is decided by the simulation of $\mathcal{A}$ on jobs released in previous rounds. The number of these permitted machines is $k-i+1$. Note that we can make the simulation since $\mathcal{A}$ is an online algorithm. Algorithm 2 is a detailed instruction of finding $J_{\mathcal{A}}$. In the end, let $\mathcal{S}_{\mathcal{A}}$ be the schedule returned by running $\mathcal{A}$ on $J_{\mathcal{A}}$.

Figure 1 is a demonstration to find the job instance $J_{\mathcal{A}}$. Figure 1c is the corresponding $J_{\mathcal{A}}$ for some online algorithm $\mathcal{A}$ of the configuration in Figure 1a.

By the construction, the output schedule $\mathcal{S}_{\mathcal{A}}$ is the result of running $\mathcal{A}$ on job set $J_{\mathcal{A}}$ (Line 4 in Algorithm 2). Now we need to prove that the job instance $J_{\mathcal{A}}$ satisfied the requirement that $J_{\mathcal{A}} \in \mathcal{I}_{\mathcal{C}}$.

Lemma 4. *$\mathcal{O}(J_{\mathcal{A}})$ and $\mathcal{C}$ have the same configuration.*

The property that $\mathcal{A}(J_{\mathcal{A}})$ is at least $\mathcal{G}(J_{\mathcal{G}})$. Recall that the schedule $\mathcal{S}_{\mathcal{A}}$ produced by Algorithm 2 is $\mathcal{A}(J_{\mathcal{A}})$. We compare $\mathcal{S}_{\mathcal{A}}$ with the greedy schedule $\mathcal{S}_{\mathcal{G}} = \mathcal{G}(J_{\mathcal{G}})$ produced by Algorithm 1.

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

Algorithm 2 Find $J_{\mathcal{A}}$

Input: The given configuration $\mathcal{C} = (v_1, v_2, \cdots, v_k)$, where $0 < v_1 \leq v_2 \leq v_3 \leq \cdots \leq v_k$.
Online scheduling algorithm $\mathcal{A}$
Output: Job instance $J_{\mathcal{A}}$ with job subsets $J_{\mathcal{A}}^{(1)}, J_{\mathcal{A}}^{(2)}, \cdots$
Schedule $\mathcal{S}_{\mathcal{A}}$ of $J_{\mathcal{A}}$
1: **for** round $i = 1, 2, \cdots, k$ **do**
2: Let $t_1, t_2, \cdots, t_{k-i+1}$ be the first $k-(i-1)$ machines with highest load in $\mathcal{A}(\bigcup_{j=1}^{i-1} J_{\mathcal{A}}^{(j)})$.
3: $J_{\mathcal{A}}^{(i)} \leftarrow$ jobs at machine v_i in $\mathcal{C}$, where for each job $j \in J_{\mathcal{A}}^{(i)}$, $P_j = \{t_1, t_2, \cdots, t_{k-i+1}\}$
4: $\mathcal{S}_{\mathcal{A}} \leftarrow \mathcal{A}(\bigcup_{j=1}^{i} J_{\mathcal{A}}^{(j)})$.
5: **end for**
6: The jobs in $J_{\mathcal{A}}^{(i)}$ released after $J_{\mathcal{A}}^{(i-1)}$. For jobs within $J_{\mathcal{A}}^{(i)}$, the jobs with lower level in $\mathcal{C}$ are released before the jobs with higher level.

Proof. The release orders are the same due to the Line 5 in Algorithm 1 and the Line 6 in Algorithm 2. □

Lemma 5. *Consider the schedules $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{S}_{\mathcal{G}}$, $L_{\mathcal{S}_{\mathcal{G}}}^{(k)} \geq L_{\mathcal{S}_{\mathcal{A}}}^{(k)}$ for all k.*

By Lemma 5 and Theorem 2, we have

Corollary 2. *Given any online algorithm $\mathcal{A}$, $\mathcal{G}(J_{\mathcal{G}}) \leq \mathcal{A}(J_{\mathcal{A}})$.*

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. Another direction of research is to consider ℓ_p norm. The assignment problem and restricted assignment problem have been studied in ℓ_p norm [1, 5]. As far as we are aware, the general grid problem with arbitrary duration and arbitrary power requirement has not been studied in ℓ_p norm and it would be an interesting direction.

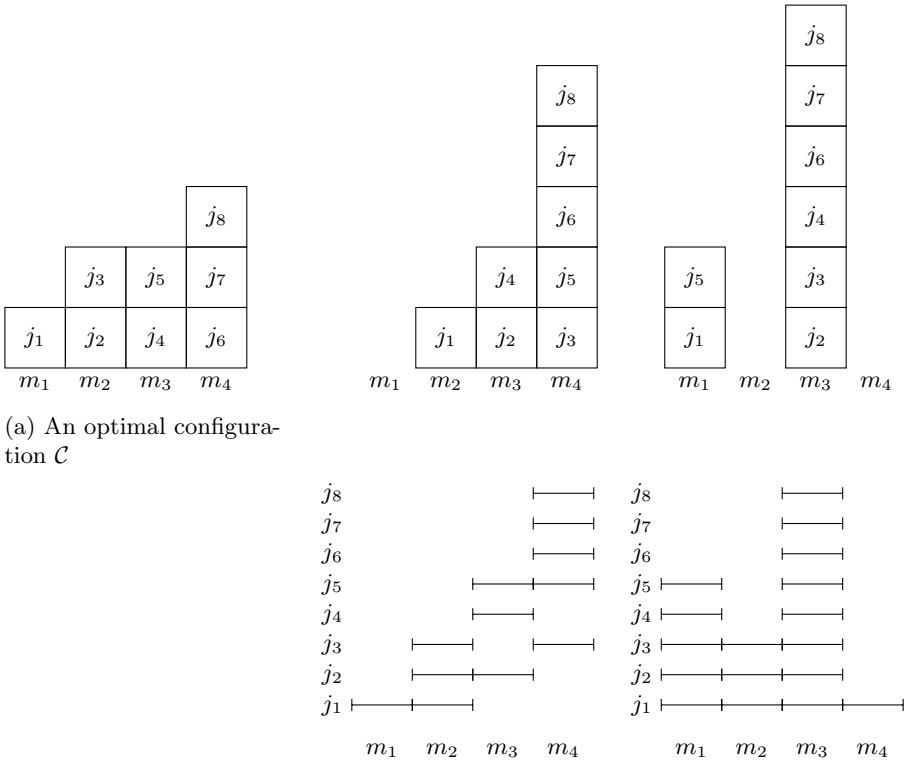
Acknowledgment. The authors would like to thank Marcin Bienkowski for helpful discussion.

References

1. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling. In: SODA. pp. 493–500 (1997)
2. 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)

3. Azar, Y.: On-line load balancing. In: Fiat, A., Woeginger, J. (eds.) Online Algorithms: The State of the Art, pp. 178–195. Springer (1998)
4. Azar, Y., Broder, A.Z., Karlin, A.R.: On-line load balancing. Theor. Comput. Sci. **130**(1), 73–84 (1994)
5. Azar, Y., Epstein, L., Richter, Y., Woeginger, G.J.: All-norm approximation algorithms. J. Algorithms **52**(2), 120–133 (2004)
6. 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)
7. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**(2), 221–237 (1995)
8. 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)
9. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
10. 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)
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. CoRR **abs/1602.06659** (2016)
32. 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)
33. Lockheed Martin: SEELoad^{TM} Solution. http://www.lockheedmartin.co.uk/us/products/energy-solutions/seesuite/seeload.html
34. 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)
35. Lui, T., Stirling, W., Marcy, H.: Get smart. IEEE Power & Energy Mag. **8**(3), 66–78 (2010)
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.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)
39. Muratore, G., Schwarz, U.M., Woeginger, G.J.: Parallel machine scheduling with nested job assignment restrictions. Oper. Res. Lett. **38**(1), 47–50 (2010)
40. REGEN Energy Inc: ENVIROGRID^{TM} SMART GRID BUNDLE. http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/
41. Salinas, S., Li, M., Li, P.: Multi-objective optimal energy consumption scheduling in smart grids. IEEE Trans. Smart Grid **4**(1), 341–348 (2013)
42. Svensson, O.: Santa claus schedules jobs on unrelated machines. SIAM J. Comput. **41**(5), 1318–1341 (2012)
43. Toronto Hydro Corporation: Peaksaver Program. http://www.peaksaver.com/peaksaver_THESL.html
44. 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)
45. US Department of Energy: The Smart Grid: An Introduction. http://www.oe.energy.gov/SmartGridIntroduction.htm (2009)
46. Verschae, J., Wiese, A.: On the configuration-lp for scheduling on unrelated machines. J. Scheduling **17**(4), 371–383 (2014)
47. Wang, C., Sitters, R.: On some special cases of the restricted assignment problem. Inf. Process. Lett. **116**(11), 723–728 (2016)
48. 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 $J_{\mathcal{G}}$

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

Fig. 1: An example of finding $J_{\mathcal{G}}$ and $J_{\mathcal{A}}$. (a) is a configuration with 8 jobs and 4 machines. To obtain $J_{\mathcal{G}}$, we first remove from (a) the jobs j_1, j_2 and j_3, which are in the lowest-load 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 $J_{\mathcal{G}}$ respectively. Then we remove j_4 and j_5 from (a) and assign them evenly on the second level of $J_{\mathcal{G}}$. After that, we stack the remaining jobs onto $J_{\mathcal{G}}$ 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 $J_{\mathcal{A}}$ for some $\mathcal{A}$, we first release j_1, which is at the lowest-load 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.