Scheduling for Electricity Cost in Smart Grid*

Mihai Burcea† Wing-Kai Hon‡ Hsiang-Hsuan Liu†‡
Prudence W.H. Wong† David K.Y. Yau§

July 7, 2015

Abstract

We study an offline scheduling problem arising in demand response management in smart grid. Consumers send in power requests with a flexible set of timeslots during which their requests can be served. For example, a consumer may request the dishwasher to operate for one hour during the periods 8am to 11am or 2pm to 4pm. The grid controller, upon receiving power requests, schedules each request within the specified duration. The electricity cost is measured by a convex function of the load in each timeslot. The objective of the problem is to schedule all requests with the minimum total electricity cost. As a first attempt, we consider a special case in which the power requirement and the duration a request needs service are both unit-size. For this problem, we present a polynomial time offline algorithm that gives an optimal solution and show that the time complexity can be further improved if the given set of timeslots forms a contiguous interval.

1 Introduction

We study an offline scheduling problem arising in "demand response management" in smart grid [11, 13, 16, 26, 41]. The electrical smart grid is one of the major challenges in the 21st century [9, 37, 38]. The smart grid uses information and communication technologies in an automated fashion to improve the efficiency and reliability of production and distribution of electricity. Peak demand hours happen only for a short duration, yet makes existing electrical grid less efficient. It has been noted in [7] that in the US power grid, 10%

*A preliminary version of this paper appeared in Proceedings of the 7th International Conference on Combinatorial Optimization and Applications, 2013 [5] and some results are improved in this version.

†Department of Computer Science, University of Liverpool, UK. Email: {*m.burcea,hhliu,pwong*}*@liverpool.ac.uk*.

‡Department of Computer Science, National Tsing Hua University, Taiwan. Email: {*wkhon,hhliu*}*@cs.nthu.edu.tw*.

§Information Systems Technology and Design, Singapore University of Technology and Design, Singapore. Email: *david_yau@sutd.edu.sg*.

of all generation assets and 25% of distribution infrastructure are required for less than 400 hours per year, roughly 5% of the time [38]. *Demand response management* attempts to overcome this problem by shifting users' demand to off-peak hours in order to reduce peak load [6, 19, 25, 28, 31, 33]. This is enabled technologically by the advances in smart meters [20] and integrated communication. Research initiatives in the area include GridWise [17], the SeeLoad^{TM} system [24], EnviroGrid^{TM} [32], peak demand [36], etc.

Demand response management aids in smoothing the power demand profile of the system across time by avoiding power overload periods [19]. This helps to increase the grid sustainability by reducing the overall production cost [25]. With reduced peak demand, generation capacity can be reduced to meet the reliability requirement. In addition, off-peak demand could be increased which means the generation capacity will be utilized in a more efficient way [33]. Demand response management is not only advantageous to the supplier but also to the consumers as well. It is common that electricity supplier charges according to the generation cost, i.e., the higher the generation cost the higher the electricity price. Therefore, it is to the consumers' advantage to reduce electricity consumption at high price and hence reduce the electricity bill [33].

The smart grid operator and consumers communicate through smart metering devices. We assume that time is divided into unit integral timeslots. A consumer sends in a power request with the power requirement, required duration of service, and the time intervals that this request can be served (giving some flexibility). For example, a consumer may want the dishwasher to operate for one hour during the periods from 8am to 11am or 2pm to 4pm. The grid operator upon receiving all requests has to schedule them in their respective time intervals using the minimum energy cost. The *load* of the grid at each timeslot is the sum of the power requirements of all requests allocated to that timeslot. The *energy cost* is modeled by a convex function on the load. As a first attempt to the problem, we consider in this paper the case that the power requirement and the duration of service requested are both unit-size, a request can specify several intervals during which the request can be served, and the power cost function is any convex function.

Previous work. Koutsopoulos and Tassiulas [19] has formulated a similar problem to our problem where the cost function is piecewise linear. They show that the problem is NP-hard, and their proof can be adapted to show the NP-hardness of the general problem studied in this paper for which jobs have arbitrary duration or arbitrary power requirement (see elaboration in Section 6). They also presented a fractional solution and some online algorithms. Salinas et al. [33] considered a multi-objective problem to minimize energy consumption cost and maximize some utility. A closely related problem is to manage the load by changing the price of electricity over time, which has been considered in a game theoretic manner [6, 28, 31]. Heuristics have also been developed for demand side management [25]. Other aspects of smart grid have also been considered, e.g., communication [7, 21–23], security [23, 27]. Reviews of smart grid can be found in [11, 13, 16, 26, 41].

The combinatorial problem we defined in this paper has analogy to the tra-

ditional load balancing problem [2] in which the machines are like our timeslots and the jobs are like our power requests. The main difference is that the aim of load balancing is usually to minimize the maximum load of the machines. Another related problem is deadline scheduling with speed scaling [1, 3, 40] in which the cost function is also a convex function, nevertheless a job can be served using varying speed of the processor. Two problems that are more closely related are the minimum cost maximum flow problem [8] with convex functions [29, 35] when we have unit power requirement and unit duration for each job[1]; and the maximum-cardinality minimum-weight matching on a bipartite graph[2]. Yet, existing algorithms for the problem cater for more general input [14, 18, 30, 39]. They are more powerful and have higher time complexity than necessary to solve our problem. For example, to solve our problem, integral flow is required for the flow version of the problem making the problem hard while the matching problem can be solved in $O(n^2\tau \log(n\tau) + n^3\tau)$ [4, 12], where n is the number of jobs and τ is the number of timeslots. As we will see, this time complexity is higher than the algorithm we are going to derive.

Our contributions. In this paper we study an optimization problem in demand response management in which requests have unit power requirement, unit duration, arbitrary timeslots that the jobs can be served, and the cost function is a general convex function. We propose a polynomial time offline algorithm that gives an optimal solution. We show that the time complexity of the algorithm is $\mathcal{O}(n^2\tau)$, where n is the number of jobs and τ is the number of timeslots. We further show that if the feasible timeslots for each job to be served form a contiguous interval, we can improve the time complexity to $\mathcal{O}(n \log \tau + \min(n, \tau) n \log n)$.

Technically speaking, we use a notion of "feasible graph" to represent alternative assignments. After scheduling a job, we can look for improvement via this feasible graph. We show that we can maintain optimality each time a job is scheduled. For the analysis, we compare our schedule with an optimal schedule via the notion of "agreement graph", which captures the difference of our schedule and an optimal schedule. We then show that we can transform our schedule stepwise to improve the agreement with the optimal schedule, without increasing the cost, thus proving the optimality of our algorithm. Note that this idea may be similar to finding matching as described above but in the matching formulation, the size of graph is n times larger than the feasible graph we describe and thus our algorithm has a lower time complexity. Also the matching

[1]Reduction: In addition to a source and a sink, we create a vertex for each job and a vertex for each timeslot. A job-vertex v_j is connected, with capacity 1, to a timeslot-vertex v_t if timeslot t is a feasible timeslot for job j. The source is connected to each job-vertex with capacity 1 and each timeslot-vertex is connected to the sink with capacity n, where n is the number of jobs. The cost function of the arc from a timeslot-vertex to the sink is a convex function of the flow on the arc. A unit flow is pushed from the source to every job-vertex v_j. The flow on the arc from a timeslot-vertex v_t to the sink is the load on timeslot t.

[2]Reduction: Given n jobs, we can construct a bipartite graph $G = (U, V, E)$ where each vertex $u_j \in U$ represents a job j and each timeslot t is represented by n vertices $v_{t,i}$ in V for $1 \leq i \leq n$. If a job j is feasible at timeslot t, we add n edges $(u_j, v_{t,i})$, for $1 \leq i \leq n$, with edge weight set to $f(i) - f(i-1)$, which is indepedent of the job.

formulation does not take advantage when the intervals are contiguous.

Organization of the paper. Section 2 gives the definition of the problem and notions required. Section 3 describes our algorithm and its properties. In Section 4, we prove that our algorithm gives an optimal solution, while in Section 5 we prove its time complexity. We give some concluding remarks in Section 6.

2 Preliminaries

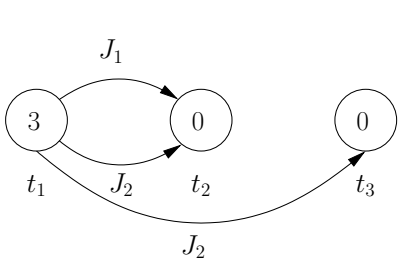
We consider an offline scheduling problem where the input consists of a set of unit-sized jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. The time is divided into integral timeslots $T = \{1, 2, 3, \ldots, \tau\}$ and each job $J_i \in \mathcal{J}$ is associated with a set of feasible timeslots $I_i \subseteq T$, in which it can be scheduled. In this model, each job J_i must be assigned to exactly one feasible timeslot from I_i. The *load* $\ell(t)$ of a timeslot t represents the total number of jobs assigned to the timeslot. We consider a general convex cost function f that measures the cost used in each timeslot t based on the load at t. The total cost used is the sum of cost over time. Over all timeslots this is $\sum_{t \in T} f(\ell(t))$. The objective is to find an assignment of all jobs in $\mathcal{J}$ to feasible timeslots such that the total cost is minimized. We first describe the notions required for discussion.

Feasible graph. Given a particular job assignment A, we define a *feasible graph* G which is a directed multi-graph that shows the potential allocation of each job in alternative assignments. In G each timeslot is represented by a vertex and the number inside the vertex denotes the load of the timeslot. If job J_i is assigned to timeslot r in A, then for all $w \in I_i \backslash \{r\}$ we add a directed arc (r, w) with J_i as its label.

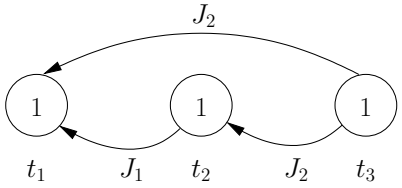
Legal-path in a feasible graph. A path (t, t') in a feasible graph G is a *legal-path* if and only if the load of the starting point t is at least 2 more than the load of the ending point t', i.e., $\ell(t) - \ell(t') \geq 2$. Note that if there is a legal-path in the feasible graph G, the corresponding job assignment is not optimal.

Agreement graph. We define an *agreement graph* $G_a(A, A^*)$ which is a directed multi-graph that measures the difference between a job assignment solution A and an optimal assignment A^*. In $G_a(A, A^*)$ each timeslot is represented by a vertex and the number inside the vertex denotes the load of the timeslot. For each job J_i such that J_i is assigned to different timeslots in A and A^*, we add an arc from t to t', where t and t' are the timeslots that J_i is assigned to by A and A^*, respectively. The arc (t, t') is labelled by the tuple $(J_i, +/-/=)$. The second value in the tuple is "+" or "−" if moving job J_i from timeslot t to timeslot t' causes the total cost of assignment A to increase or decrease, respectively. The value is "=" if moving the job does not cause any change in the total cost of assignment A.

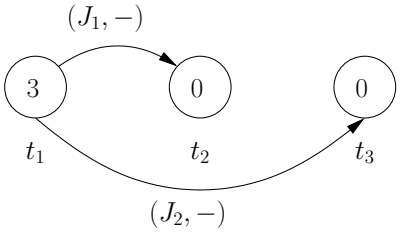
We now give two examples that demonstrate the use of notions introduced in this section.



(a) Feasible graph G for the arbitrary job assignment A.



(b) Feasible graph G^* for the optimal assignment A^*.



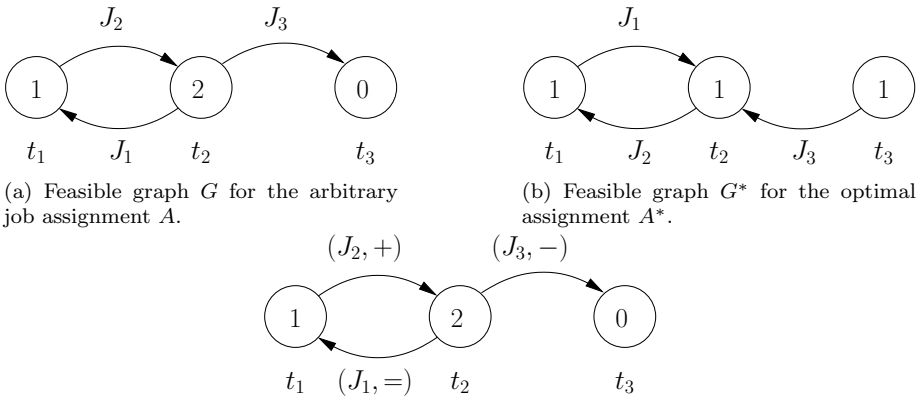
(c) The agreement graph $G_a(A, A^*)$.

Figure 1: Feasible graphs and agreement graph for Example 1.

Example 1. *Let $\mathcal{J} = \{J_1, J_2, J_3\}$, $T = \{t_1, t_2, t_3\}$, $I_1 = \{t_1, t_2\}$, $I_2 = \{t_1, t_2, t_3\}$, and $I_3 = \{t_1\}$. Figure 1(a) shows the feasible graph G for an arbitrary job assignment A, where J_1, J_2, and J_3 are all assigned to timeslot t_1. Figure 1(b) shows the feasible graph G^* for an optimal assignment A^*, where J_1, J_2, and J_3 are each assigned to timeslots t_2, t_3, and t_1, respectively. Figure 1(c) shows the agreement graph $G_a(A, A^*)$. In $G_a(A, A^*)$, J_1 and J_2 are both assigned to different timeslots in the optimal solution A^* compared to A. By moving either J_1 or J_2 to the their assigned timeslots in A^*, the overall cost of solution A would decrease.*

Example 2. *Let $\mathcal{J} = \{J_1, J_2, J_3\}$, $T = \{t_1, t_2, t_3\}$, $I_1 = \{t_1, t_2\}$, $I_2 = \{t_1, t_2\}$, and $I_3 = \{t_2, t_3\}$. Figure 2(a) shows the feasible graph G for an arbitrary job assignment A, where J_1 and J_3 are both assigned to timeslot t_2, and J_2 is assigned to timeslot t_1. Figure 2(b) shows the feasible graph G^* for an optimal assignment A^*, where J_1, J_2, and J_3 are each assigned to timeslots t_1, t_2, and t_3, respectively. Figure 2(c) shows the agreement graph $G_a(A, A^*)$. In $G_a(A, A^*)$, all jobs are assigned to different timeslots in the optimal solution A^* compared to A. By moving J_1 from t_2 to t_1 the overall cost of solution A would remain the same, by moving J_2 from t_1 to t_2 the overall cost of solution A would increase, and by moving J_3 from t_2 to t_3 the overall cost of solution A would decrease.*

Observation 1. *By moving J_i from t_1 to t_2 the overall energy cost* (i) *decreases if $\ell(t_1) > \ell(t_2) + 1$,* (ii) *remains the same if $\ell(t_1) = \ell(t_2) + 1$, and* (iii) *increases if $\ell(t_1) < \ell(t_2) + 1$.*



(a) Feasible graph G for the arbitrary job assignment A.

(b) Feasible graph G^* for the optimal assignment A^*.

(c) The agreement graph $G_a(A, A^*)$.

Figure 2: Feasible graphs and agreement graph for Example 2.

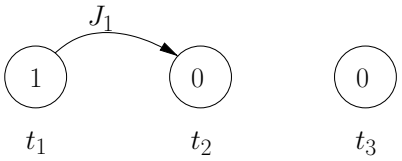
Shifting. By Observation 1, existence of a legal-path implies that the assignment is not optimal and we can execute a "shift" and decrease the total cost of the assignment. Given a legal-path P, a *shift* moves each job corresponding to an arc e along P from the original assigned timeslot to the timeslot determined by e. More precisely, if the path contains an arc (r, w) with J as its label, then job J is moved from r to w. It is easy to see from Observation 1 that such a shift decreases the cost, implying that the original assignment is not optimal.

On the other hand, when there is no legal-path, it is not as straightforward to show that the assignment is optimal. Nevertheless, we will prove this is the case in Lemma 7.

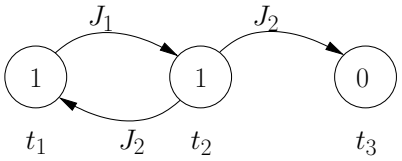
3 Our Algorithm

The algorithm. We propose a polynomial time offline algorithm that minimizes the total cost (Figure 3 shows an illustration). The algorithm arranges the jobs in $\mathcal{J}$ in arbitrary order, and runs in stages. At any Stage i, we have three steps:
(1) Assign J_i to a feasible timeslot with minimum load, breaking ties arbitrarily;
(2) Suppose J_i is assigned to timeslot t. We update the feasible graph G to reflect this assignment in the following way. If applicable, we add arcs from t labelled by J_i to any other feasible timeslots (vertices) of J_i;
(3) If there exists any legal-path in G from t to any other vertex t', the algorithm executes a shift along the legal-path (see Section 2). At the end, the algorithm updates the feasible graph G to reflect this shift.

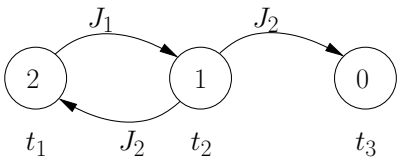
Invariants. In the next section, we show that the algorithm maintains the following two invariants. At the end of each stage:
(I1) There is no legal-path in the resulting feasible graph;
(I2) The assignment is optimal for the jobs considered so far.



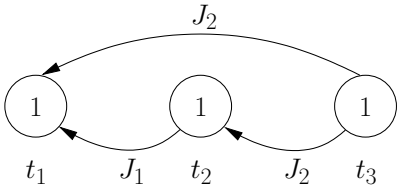
(a) The feasible graph G after assigning job J_1 to t_1.



(b) The feasible graph G after assigning job J_2 to t_2.



(c) The feasible graph G after assigning job J_3 to t_1.



(d) The feasible graph G after executing the shift of the legal-path in (c).

Figure 3: Illustration of our algorithm on Example 1. Suppose the algorithm schedules the jobs in the order of their indices. (a) – (c) Feasible graphs after J_1, J_2, and J_3 are assigned, respectively. (d) The path (t_1, t_3) is a legal-path in (c) and we shift by moving J_1 to t_2 and J_2 to t_3.

Additional notations. To ease the discussion, in the remainder of the paper, we use $\ell'_i(t)$ to represent the load of timeslot t after assigning J_i (but before the shift), $\ell_i(t)$ to represent the load of timeslot t at the end of Stage i, and $\ell'_i(s,t)$ and $\ell_i(s,t)$ to represent $\ell'_i(s) - \ell'_i(t)$ and $\ell_i(s) - \ell_i(t)$, respectively.

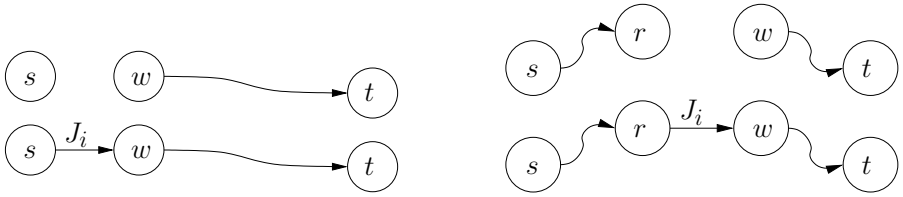
4 Correctness

Theorem 1. *Our algorithm finds an optimal assignment.*

Framework. Consider any stage. After Step (2), there may be a legal-path in the resulting feasible graph G. In Lemma 2, we show that if a legal-path exists in G after assigning J_i to timeslot r, there is at least one legal-path starting from r. Suppose the algorithm chooses the legal-path (r, t) and executes the shift along this path in Step (3). In Lemma 4, we show that if there is no legal-path in the feasible graph G before assigning a job, then after assigning a job and executing the corresponding shift by the algorithm, the resulting feasible graph has no legal-paths. Therefore, Step (3) of the algorithm needs to be applied only once and there will be no legal-path left, implying that Invariant (I1) holds. In Lemma 7, we show that if there is no legal-path in a feasible graph G, the corresponding assignment is optimal and hence Invariant (I2) holds.

4.1 Proof of Invariant (I1)

We begin by proving Lemmas 2 and 3. Lemma 3 is a technical lemma used in the proof of Lemma 4.



(a) Case 2-1 before and after assigning job J_i to timeslot s.

(b) Case 2-2 before and after assigning job J_i to timeslot r.

Figure 4: The two sub-cases of Case 2 in the proof of Lemma 3.

Lemma 2. *Suppose that before assigning job J_i to timeslot r the feasible graph G has no legal-path. If there is any legal-path after assigning J_i, there is at least one legal-path starting from r.*

Proof. Assume that there is a legal-path (s, t) after assigning J_i to timeslot r, so that $\ell'_i(s, t) \geq 2$. If $r = s$, we have obtained a desired legal-path. Otherwise, $r \neq s$, there are two cases:
Case 1. G contains an (s, t) path before assigning J_i. Since $r \neq s$, $\ell_{i-1}(s) = \ell'_i(s)$ and $\ell_{i-1}(t) \leq \ell'_i(t)$ (the latter inequality comes from the fact that r may be equal to t). This implies $\ell_{i-1}(s, t) \geq \ell'_i(s, t) \geq 2$, which contradicts the precondition that there is no legal-path before assigning J_i. Thus, Case 1 cannot occur.
Case 2. G does not contain any (s, t) path before assigning J_i. Since (s, t) becomes a path (legal-path) after assigning J_i, it must be the case that assigning J_i to timeslot r adds some new arc (r, w) (with J_i as its label) to G, which connects an existing (s, r) path and an existing (w, t) path. We know that $\ell_{i-1}(s) - \ell_{i-1}(r) \leq 1$ because there is no legal-path before assigning J_i. Also, $\ell'_i(s) = \ell_{i-1}(s)$, $\ell'_i(t) = \ell_{i-1}(t)$, and $\ell'_i(r) = \ell_{i-1}(r) + 1$ because the new job J_i is assigned to r, with $r \neq s$. Hence, $\ell'_i(r, t) \geq \ell'_i(s, t)$, so that the (r, t) subpath is also a legal-path. □

Lemma 3. *If before assigning a job the feasible graph G does not have a legal-path, then after assigning one more job there will be no legal-paths where the load of the starting point is at least 3 more than the load of the ending point. In other words, the load difference corresponding to any new legal-path, if it exists, is exactly 2.*

Proof. Suppose that before assigning a job there is no legal-path in the feasible graph G. That is, $\ell_{i-1}(s, t) \leq 1$ for any path (s, t) in G with starting point s and ending point t. Now assume on the contrary that there is a legal-path (s, t) with $\ell'_i(s, t) \geq 3$. There are two cases:
Case 1. G contains an (s, t) path before assigning job J_i. Assigning a job at timeslot r increases by one on the load difference of any path starting from r. On the other hand, the load difference of any path ending at r is decreased by one. Recall that $\ell'_i(s, t) \geq 3$. There are three situations: (i) Job J_i is assigned

to timeslot s, implying $\ell_{i-1}(s,t) \geq 2$; (ii) Job J_i is assigned to timeslot t, which implies $\ell_{i-1}(s,t) \geq 4$; (iii) Job J_i is not assigned to timeslot s or time t, which implies $\ell_{i-1}(s,t) \geq 3$. Each of these cases contradicts the fact that G has no legal-path before assigning J_i.

Case 2. G does not contain any (s,t) path before assigning job J_i. That is, assigning job J_i creates a new path (legal-path) (s,t) with $\ell'_i(s,t) \geq 3$. There are two sub-cases (see Figure 4 for an illustration):

Case 2-1. Job J_i is assigned to timeslot s. Since (s,t) becomes a new legal-path after assigning job J_i, it must be the case that assigning J_i to timeslot s adds some new arc (s,w) in G that connects s with an existing (w,t) path. The arc (s,w) means that job J_i can be assigned to timeslot s or w. Then, we have $\ell_{i-1}(s) = \ell'_i(s) - 1$, $\ell_{i-1}(t) = \ell'_i(t)$, and $\ell_{i-1}(w) = \ell'_i(w)$, and according to our assumption, $\ell'_i(s,t) \geq 3$. Since there is no legal-path before assigning job J_i, $\ell_{i-1}(w,t) \leq 1$. Hence, $\ell_{i-1}(s) - \ell_{i-1}(w) \geq 1$, which contradicts the fact that J_i is assigned to a feasible timeslot with minimum load.

Case 2-2. The job J_i is assigned to timeslot r with $r \neq s$. Since (s,t) becomes a new legal-path after assigning job J_i, it must be the case that there is some new arc (r,w) added in G that connects an existing (s,r) path with an existing (w,t) path. Because there is no legal-path before assigning job J_i, $\ell_{i-1}(s,r) \leq 1$ and $\ell_{i-1}(w,t) \leq 1$. According to our assumption, $\ell'_i(s,t) \geq 3$; this implies $\ell_{i-1}(s,t) \geq 3$, so that $\ell_{i-1}(r) - \ell_{i-1}(w) \geq 1$. The latter inequality contradicts the fact that J_i is assigned to a feasible timeslot with minimum load. □

Lemma 4. *Suppose that G is a feasible graph with no legal-paths. Then after assigning a job and executing the corresponding shift by the algorithm, the resulting feasible graph has no legal-paths.*

Proof. Suppose that there were no legal-paths in G after Stage $i-1$, but there is a new legal-path in G after assigning J_i. By Lemma 2, there must be one such legal-path (s,t) where s is the timeslot assigned to J_i, and without loss of generality, let the path be the one that is selected by our algorithm to perform the corresponding shift. Let the ordering of the vertices in the path be $[s, v_1, v_2, \ldots, v_k, t]$, and P denote the set of these vertices.

We define $In(r)$ to be the set of vertices w such that a (w,r) path exists before assigning J_i, and $Out(r)$ to be the set of vertices w such that an (r,w) path exists before assigning J_i. We assume that $r \in In(r)$ and $r \in Out(r)$ for the ease of later discussion. Similarly, we define $In''(r)$ to be the set of vertices w such that a (w,r) path exists after shifting, and we define $Out''(r)$ analogously. Given a set R of vertices, let $IN(R) = \bigcup_{r \in R} In(r)$ and $OUT(R) = \bigcup_{r \in R} Out(r)$. The notation $IN''(R)$ and $OUT''(R)$ are defined analogously.

Briefly speaking, we upper bound the load of a vertex in $IN''(P)$, and lower bound the load of a vertex in $OUT''(P)$, as any legal-path that may exist after the shift must start from a vertex in $IN''(P)$ and end at a vertex in $OUT''(P)$. Based on the bounds, we shall argue that there are no legal-paths as the load difference of any path after the shift will be at most 1. Note that after the shift, only the load of t is increased by one, the load of s is decreased by one,

whereas the load of any other vertex remains unchanged. Now, concerning the legal-path (s,t), there are two cases:

Case 1. There was an arc from s to v_1 in the feasible graph G before assigning J_i. In this case, it is easy to check that $IN''(P) \subseteq IN(P)$,[3] and $OUT''(P) \subseteq OUT(P) \cup OUT(I_i)$.[4]

Suppose that $\ell_{i-1}(s) = x$. Then, $\ell_{i-1}(t) = x - 1$ because there is no legal-path before assigning J_i but there is one after assigning J_i. This implies $\ell_{i-1}(v_h) \leq x$ for any $h \in [1,k]$, or there was a legal-path (v_h, t) before assigning J_i. The load of any vertex in $IN(P)$ is at most x or there was a legal-path entering t before assigning J_i. The load of any vertex in $OUT(P)$ is at least $x-1$ or there was a legal-path leaving s before assigning J_i. For any vertex r in I_i, $\ell_{i-1}(r) \geq x$, since $s \in I_i$ has the minimum load. This implies that the load for any vertex in $OUT(I_i)$ is at least $x-1$, or there was a legal-path leaving a vertex in I_i before assigning J_i. Thus, after the shift, the load of any vertex in $IN''(P)$ is at most x, and the load of any vertex in $OUT''(P)$ is at least $x-1$, so no legal-paths will exist.

Case 2. There were no arcs from s to v_1 in the feasible graph G before assigning J_i. In this case, J_i must be involved in the shift, so that the jobs assigned to s after the shift will be the same as if J_i was not assigned. Consequently, if there is still a legal-path after the shift, the starting vertex must be from $IN''(P\backslash\{s\})$, while the ending vertex must be from $OUT''(P\backslash\{s\})$. Similar to Case 1, it is easy to check that $IN''(P\backslash\{s\}) \subseteq IN(P\backslash\{s\})$ and $OUT''(P\backslash\{s\}) \subseteq OUT(P\backslash\{s\}) \cup OUT(I_i)$. Suppose that $\ell_{i-1}(s) = x$, so that $\ell'_i(s) = x+1$. Because assigning J_i creates a new legal-path (s,t), by Lemma 3, $\ell'_i(t) = \ell_{i-1}(t) = x-1$. Thus, the load of any vertex in $IN(P\backslash\{s\})$ is at most x, since there was no legal-path entering t before assigning J_i. On the other hand, $\ell_{i-1}(v_1) \geq x$ otherwise job J_i would be assigned to v_1. However, $\ell_{i-1}(v_1) \leq x$ or there is a legal-path (v_1, t). Hence, $\ell_{i-1}(v_1) = x$. This implies that the load of any vertex in $OUT(P\backslash\{s\})$ is at least $x-1$, since there was no legal-path leaving v_1 before assigning J_i. As for the vertices in $OUT(I_i)$, we can use a similar argument as in Case 1 to show that their load is at least $x-1$. Thus, after the shift, the load of any vertex in $IN''(P\backslash\{s\})$ is at most x, and the load of any vertex in $OUT''(P\backslash\{s\})$ is at least $x-1$, so no legal-path will exist. □

4.2 Proof of Invariant (I2)

We now prove in Lemma 7 (the other key lemma for the correctness) that non-existence of legal-paths implies the assignment is optimal. The rough ideas

[3] Otherwise, let z be a vertex in $IN''(P)$ but not in $IN(P)$. Take the shortest path from z to some vertex in P after the shift. Then all the intermediate vertices of such a path are not from P. However, the jobs assigned to those intermediate vertices are unchanged, so that such a path also exists before the shift, and z is in $IN(P)$. A contradiction occurs.

[4] Otherwise, let z be a vertex in $OUT''(P)$ but not in $OUT(P) \cup OUT(I_i)$. Take the shortest path that goes to z starting from some vertex in P after the shift. Then all the intermediate vertices of such a path are not from P. If such a path does not involve vertices from I_i, then this path must exist before the shift, so that z is in $OUT(P)$. Else, z is in $OUT(I_i)$. A contradiction occurs.

are as follows. Consider an optimal assignment A^* (satisfying some constraints as to be defined). In Lemma 6, we show that there is a sequence of agreement graphs $G_a(A_1, A^*), G_a(A_2, A^*), \ldots, G_a(A_k, A^*)$ where the cost is non-increasing every step, $A_1 = A$ is the original assignment of jobs given by our algorithm, and $A_k = A^*$ is an optimal assignment. We prove Lemma 7 by contradiction, assuming there is no legal-path in the feasible graph G but the assignment A is not optimal. We then consider the sequence of agreement graphs given in Lemma 6 and show that either there is no agreement graph in the sequence involving strict decrease of overall cost (which means A is already optimal) or that there is a legal-path in the feasible graph G, leading to a contradiction.

Note that Lemma 6 considers an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic. The existence of such A^* is proved in Lemma 5.

Lemma 5. *There exists an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic.*

Proof. Consider an optimal assignment A^{**} such that $G_a(A, A^{**})$ contains directed cycles. We show that the assignment A^{**} can be transformed into an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic. Recall that each timeslot is represented by a vertex in $G_a(A, A^{**})$ and an arc from vertex s to vertex t labelled by a tuple (J_i, $+/-/=$) means that J_i is assigned to timeslot s in assignment A and timeslot t in A^{**}. For every cycle (s, t) such that $s = t$ in $G_a(A, A^{**})$, we show that the load of any vertex does not change after executing all the moves in the cycle. This implies that the total cost of A^{**} remains the same after removing all cycles from $G_a(A, A^{**})$.

We consider a cycle that contains the vertices $[s, v_1, v_2, \ldots, v_k, t]$, for $s = t$. There are arcs from s to v_1, v_1 to v_2, and so on, until the last arc from v_k to $t = s$. An arc denotes the moving of a distinct job each step. As we move one job from s to v_1, $\ell(s)$ decreases by one and $\ell(v_1)$ increases by one. However, $\ell(v_1)$ returns to the original value as we move the respective job from vertex v_1 to v_2. Thus, $\ell(v_i)$, for $1 \leq i < k$ remains unchanged. As we move the last job from vertex v_k to $t = s$, both $\ell(v_k)$ and $\ell(s)$ return to their original value. Clearly, the load of all vertices remains the same even for cycles of size 2. Thus, the cost of A^{**} remains the same after removing all cycles from $G_a(A, A^{**})$ and we denote the corresponding agreement graph by $G_a(A, A^*)$. □

To illustrate the previous proof, we refer to Example 2 and Figure 2, and Figure 5. In Figure 2(c), the agreement graph $G_a(A, A^*)$ contains a cycle, yet an alternative optimal assignment A' exists such that $G_a(A, A')$ contains no cycles, as depicted in Figure 5.

Lemma 6. *Suppose A is not optimal and A^* is an optimal assignment such that $G_a(A, A^*)$ is acyclic. Then we can have a sequence of agreement graphs $G_a(A_1, A^*), G_a(A_2, A^*), \ldots, G_a(A_k, A^*)$ such that $A_1 = A$, $A_k = A^*$, and the cost is non-increasing every step.*

Proof. Consider the agreement graph $G_a(A_i, A^*)$, for $i \geq 1$, starting from $A_1 = A$. In each step, from $G_a(A_i, A^*)$ to $G_a(A_{i+1}, A^*)$, one arc is removed. For

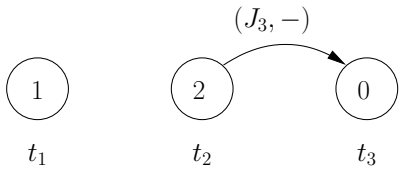


Figure 5: Let A be the arbitrary job assignment of Example 2, illustrated in Figure 2(a). The assignment A', where J_1, J_2, and J_3 are each assigned to t_2, t_1, and t_3, respectively, is an alternative optimal assignment such that the agreement graph $G_a(A, A')$ contains no cycles.

$i \geq 1$, we consider in $G_a(A_i, A^*)$ any arc labelled with either a "$-$" or an "$=$" and we execute the move corresponding to this arc. Through this move, we remove one arc, and thus we do not introduce any new arcs. However, the $+/-/=$ label of other arcs may change. If the resulting graph $G_a(A_{i+1}, A^*)$ does not contain any more "$-$" or "$=$" arcs, we stop. Otherwise, we repeat the process.

Note that the cost is non-increasing in every step. By the time we stop, if the resulting graph, say, $G_a(A_h, A^*)$, does not contain any more arcs, we have obtained the desired sequence of agreement graphs. Otherwise, we are left only with "$+$" labelled arcs in $G_a(A_h, A^*)$; however, in the following, we shall show that such a case cannot happen, thus completing the proof of the lemma.

Firstly, $cost(A_h) \geq cost(A^*)$ since A^* is an optimal assignment. Next, by Lemma 5, $G_a(A_1, A^*)$ is acyclic and the resulting graph $G_a(A_h, A^*)$ by removing all "$-$" and "$=$" labelled arcs is also acyclic. Thus, in $G_a(A_h, A^*)$, there must exist at least one vertex with in-degree 0 and one vertex with out-degree 0. We look at all such (v_1, v_i) paths in $G_a(A_h, A^*)$, where v_1 has in-degree 0, v_i has out-degree 0, and $v_1 \neq v_i$. For any such (v_1, v_i) path, we show that by executing all moves of the path (i) the overall cost is increasing, and (ii) the labels of all arcs not contained in the (v_1, v_i) path remain "$+$". After executing all moves of the path, all arcs of the (v_1, v_i) path are removed.

(i) Suppose the vertices of the path are $[v_1, v_2, \ldots, v_i]$ and $\ell(v_1) = x$. As all arcs in (v_1, v_i) are labelled with "$+$" (i.e., the cost is increasing), $\ell(v_j) \geq x$, for $j > 1$. By executing all moves in the path, $\ell(v_1) = x - 1$, $\ell(v_j)$ is unchanged, for $1 < j < i$, and $\ell(v_i)$ is increased by one. Thus, the overall cost is increasing.

(ii) We show that the labels of all arcs not contained in the (v_1, v_i) path remain "$+$". There may be out-going arcs from v_1 to other vertices not in the (v_1, v_i) path initially labelled by "$+$". Before executing all the moves in the (v_1, v_i) path, the load of all other vertices is at least x as we assume $\ell(v_1) = x$. After the move, $\ell(v_1) = x - 1$ and out-going arcs from v_1 point to vertices with load at least x. Thus, an arc from v_1 to any other vertex denotes a further increase in the cost and the labels of the arcs do not change. For vertices v_j, for $1 < j < i$, the load of v_j remains unchanged and thus the labels of the arcs incoming to or outgoing from v_j remain the same. For v_i, there may be incoming arcs. Suppose $\ell(v_i) = y$ before executing all the moves in the (v_1, v_i)

path. Then the load of all other vertices pointing to v_i is at most y and the arcs are labelled by "+". After executing all the moves in the (v_1, v_i) path, $\ell(v_i) = y+1$, and thus any subsequent moves from vertices pointing to v_i cause further increases in the cost, i.e., the labels do not change.

Thus, the overall cost is increasing. We repeat this process until there are no more such (v_1, v_i) paths. We end up with $cost(A_k) > cost(A^*)$, which contradicts the fact that $cost(A_k) = cost(A^*)$ as $A_k = A^*$. Thus, the case where we are left only with "+" labelled arcs in $G_a(A_h, A^*)$ cannot happen, and the lemma follows. □

Lemma 7. *If there is no legal-path in the feasible graph G, the corresponding assignment is optimal.*

Proof. Suppose by contradiction there is no legal-path in the feasible graph G, but the corresponding assignment A is not optimal. Let A^*, $A_1 = A, A_2, \ldots,$ $A_k = A^*$ be the assignments as defined in Lemma 6. Note that each arc in the agreement graph $G_a(A_1, A^*)$ corresponds to an arc in the feasible graph G (since G captures all possible moves). Because the sequence of agreement graphs in Lemma 6 only involves removing arcs, each arc in all of $G_a(A_i, A^*)$ corresponds to an arc in G.

Suppose $G_a(A_j, A^*)$ is the first agreement graph in which a "−" labelled arc is considered between some timeslots t_α and t_β. If there is no such arc, then A is already an optimal solution (since the sequence will be both non-increasing by Lemma 6 and non-decreasing as no "−" labelled arc is involved). Otherwise, if there is such an arc in $G_a(A_j, A^*)$, we show that there must have existed a legal-path in the feasible graph G, leading to a contradiction. We denote by $\ell(A_i, t)$ the load of timeslot t in the agreement graph $G_a(A_i, A^*)$. Suppose $\ell(A_j, t_\alpha) = x$, then $\ell(A_j, t_\beta) \leq x-2$ as the overall energy cost would be decreasing by moving a job from t_α to t_β. If $\ell(A_1, t_\alpha) = x$ and $\ell(A_1, t_\beta) \leq x-2$ in the original assignment, then there is a legal-path in G, which is a contradiction. Otherwise, we claim that there are some timeslots u_{i_y} and v_{k_z} such that $\ell(A_1, u_{i_y}) \geq x$ and $\ell(A_1, v_{k_z}) \leq x-2$, and there is a path from u_{i_y} to v_{k_z} in G. This forms a legal-path in G, leading to a contradiction.

To prove the claim, we first consider finding u_{i_y}. We first set $i_0 = j$ and $u_{i_0} = t_\alpha$. If $\ell(A_1, u_{i_0}) \geq x$, we are done. Else, since $\ell(A_j, u_{i_0}) = x$ and $\ell(A_1, u_{i_0}) < x$, there must be some job that is moved to u_{i_0} before A_j. Let $i_1 < i_0$ be the latest step such that a job is assigned to u_{i_0} and the job is moved from u_{i_1}. Note that since this move corresponds to an arc with label "=", $\ell(A_{i_1}, u_{i_1}) = x$ and $\ell(A_{i_1}, u_{i_0}) = x-1$. If $\ell(A_1, u_{i_1}) \geq x$, we are done. Otherwise, we can repeat the above argument to find u_{i_2} and so on. The process must stop at some step $i_y < i_0$ where $\ell(A_1, u_{i_y}) \geq x$. Similarly, we set $k_0 = j$ and $v_{k_0} = t_\beta$, so that we can find a step $k_z < k_0$ such that $\ell(A_1, v_{k_z}) \leq x-2$. Recall that since each arc in $G_a(A_1, A^*)$ corresponds to an arc in the feasible graph G and in all subsequent agreement graphs we only remove arcs, there is a path from u_{i_y} and v_{k_z} in G. Therefore, we have found a legal-path from u_{i_y} to v_{k_z} in G. □

5 Time Complexity

We analyze the time complexity of our algorithm in Section 5.1 and show in Section 5.2 that this can be improved when the feasible timeslots associated with each job form a contiguous interval.

5.1 General Case

In this section, we consider the general case and show that the time complexity is $\mathcal{O}(n^2\tau)$.

Theorem 8. *We can find the optimal schedule in $\mathcal{O}(n^2\tau)$ time.*

Proof. We assign jobs one by one. Each round when we assign the job J_i to timeslot t, we add arcs (t, w) labelled by J_i for all vertices w that $w \in I_i$ in the feasible graph. By Lemma 2, there is a legal-path starting from t if there is a legal-path after assigning J_i to timeslot t. When J_i is assigned to t, we start breadth-first search (BFS) at t. By Lemma 3, if there is a node w which can be reached by the search and the number of jobs assigned to w is two less than the number of jobs assigned to t, it means that there is a legal-path (t, w). Then we shift the jobs according to the (t, w) legal-path. After shifting there will be no legal-paths anymore by Lemma 4. Finally we update the arcs of the vertices on the legal-path in the feasible graph.

Adding J_i to the feasible graph needs $\mathcal{O}(|I_i|)$ time. Because $|I_i|$ is at most the total number of timeslots in T, $|I_i| = \mathcal{O}(\tau)$ where τ is the number of timeslots. The BFS takes $\mathcal{O}(\tau + n\tau)$ time because there are at most $n\tau$ arcs in the feasible graph. If a legal-path exists after assigning J_i and its length is l, the shifting needs $\mathcal{O}(l)$ time, which is $\mathcal{O}(\tau)$ because there are at most τ vertices in the legal-path. After the shift, the final step to update the arcs of the vertices on the legal-path takes at most $\mathcal{O}(n\tau)$ time because there are at most $n\tau$ arcs in the feasible graph. The total time for assigning n jobs is thus bounded by $\mathcal{O}(n^2\tau)$. □

5.2 Contiguous Intervals

In this section, we consider the special case where each job $J_i \in \mathcal{J}$ is associated with an interval of contiguous timeslots $I_i = [a_i, b_i]$, for positive integers $a_i \leq b_i$. We show that we can improve the time complexity to $\mathcal{O}(n \log \tau + \min(n, \tau) n \log n)$. Let s_i be the timeslot J_i is assigned to. Then we require that $a_i \leq s_i \leq b_i$.

Framework. Recall that our algorithm first assigns a job J_p to a feasible timeslot with minimum load and then executes a shift if there is a legal-path in the resulting feasible graph. When the feasible timeslots of a job form a contiguous interval, we will use several data structures to help finding a legal-path. In particular, we first find a path from s_p such that the end point has the minimum load. If this path is a legal-path, we execute the shift, otherwise,

there is no legal-path from s_p. To find such a path, we exploit the notion of l-reachable intervals (to be defined). To compute reachable intervals, we maintain two heaps for each timeslot t to store a_i and b_i of jobs J_i that are assigned to t. To find the timeslot with minimum load in a reachable interval, we use a data structure that supports dynamic range minimum query (RMQ). Before we give the detailed analysis, we first define a few notions on a feasible graph.

Reachable interval. For every timeslot t_i, the *l-reachable interval* of t_i, denoted by $\mathcal{R}_i^{(l)}$, is defined to be the set of timeslots t such that there is a path from t_i to t with length at most l. We define $\mathcal{R}_i^{(0)} = \{t_i\}$ and $\mathcal{R}_i^{(-1)} = \emptyset$. Note that $\mathcal{R}_i^{(1)} = \cup_{j:s_j=t_i} I_j$. We call $\mathcal{R}_i^{(1)}$ the *directly reachable interval* of t_i. $\mathcal{R}_i^{(1)}$ is a contiguous interval containing t_i because the feasible timeslots of each job form a contiguous interval and the feasible timeslots of any job that is assigned to t_i must contain t_i.

Notice that $\mathcal{R}_i^{(l+1)}$ is the union of the directly reachable intervals of each timeslot in $\mathcal{R}_i^{(l)}$. For any $t \in \mathcal{R}_i^{(l)}$, as observed above, the directly rearchable interval of a timeslot t must contain t and thus is a contiguous interval overlapping $\mathcal{R}_i^{(l)}$. Therefore, $\mathcal{R}_i^{(l+1)}$ forms a contiguous interval and $\mathcal{R}_i^{(l)} \subseteq \mathcal{R}_i^{(l+1)}$ for $l \geq 0$. We denote the interval $\mathcal{R}_i^{(l)}$ as $[\alpha_i^{(l)}, \beta_i^{(l)}]$. In particular, for directly reachable intervals, $\alpha_i^{(1)} = \min_{j:s_j=t_i} a_j$ and $\beta_i^{(1)} = \max_{j:s_j=t_i} b_j$.

Depth. We note that if $\mathcal{R}_i^{(l+1)}$ is the same as $\mathcal{R}_i^{(l)}$, then for any $k \geq l$, $\mathcal{R}_i^{(k)}$ is also the same as $\mathcal{R}_i^{(l)}$. We define the *depth* D_i of t_i to be the smallest integer l such that $\mathcal{R}_i^{(l+1)} = \mathcal{R}_i^{(l)}$. Note that the depth D_i is the longest length of the shortest paths starting from t_i to any other vertex in the feasible graph. Furthermore, the D_i-reachable interval of t_i is the set of all timeslots such that there is a path from t_i.

Path-finder-job. Consider any $1 \leq l \leq D_i$. The definition of D_i implies that $\mathcal{R}_i^{(l)} \supsetneq \mathcal{R}_i^{(l-1)}$. We define the *left-path-finder-job* (*right-path-finder-job* resp.) of $\mathcal{R}_i^{(l)}$ as the job J_p (with smallest job index) such that $s_p \in \mathcal{R}_i^{(l-1)}$ and $a_p = \alpha_i^{(l)}$ ($b_p = \beta_i^{(l)}$ resp.). We denote them by $\text{lpfj}(\mathcal{R}_i^{(l)})$ and $\text{rpfj}(\mathcal{R}_i^{(l)})$, respectively. Then we have the following property about path-finder-jobs, which then leads to a bound on D_i in Property 10.

Property 9. *For $1 \leq l \leq D_i$, $\text{lpfj}(\mathcal{R}_i^{(l)})$ or $\text{rpfj}(\mathcal{R}_i^{(l)})$ is assigned to a timeslot in $\mathcal{R}_i^{(l-1)} \setminus \mathcal{R}_i^{(l-2)}$.*

Proof. By the definition of D_i, $\mathcal{R}_i^{(l)} \supsetneq \mathcal{R}_i^{(l-1)}$, implying that $\alpha_i^{(l)} < \alpha_i^{(l-1)}$ or $\beta_i^{(l)} > \beta_i^{(l-1)}$. Consider the former case. Let $J_p = \text{lpfj}(\mathcal{R}_i^{(l)})$; i.e., $a_p = \alpha_i^{(l)}$. We claim that $s_p \in \mathcal{R}_i^{(l-1)} \setminus \mathcal{R}_i^{(l-2)}$; otherwise, $s_p \in \mathcal{R}_i^{(l-2)}$ implying $\alpha_i^{(l-1)} \leq a_p$, which is contradicting to $a_p = \alpha_i^{(l)}$ and $\alpha_i^{(l)} < \alpha_i^{(l-1)}$. Using a similar argument, the latter case implies that $\text{rpfj}(\mathcal{R}_i^{(l)})$ is assigned to a timeslot in $\mathcal{R}_i^{(l-1)} \setminus \mathcal{R}_i^{(l-2)}$. Combining the two cases, the property holds. □

Property 10. *For every timeslot t_i,* (i) *$D_i \leq \min(n, \tau)$;* (ii) *the number of timeslots in $\mathcal{R}_i^{(D_i)}$ that have jobs assigned to them is at most $\min(n, \tau)$.*

Proof. (i) We observe that $D_i \leq \tau$ because $\mathcal{R}_i^{(l)} \supsetneq \mathcal{R}_i^{(l-1)}$ for $1 \leq l \leq D_i$. On the other hand, by Property 9, there is at least one job assigned to a timeslot in $\mathcal{R}_i^{(l-1)} \setminus \mathcal{R}_i^{(l-2)}$ for every $1 \leq l \leq D_i$ and each job is only assigned to one timeslot, therefore, $D_i \leq n$. (ii) is trivial. □

Analysis. To compute reachable intervals, we have to know the minimum of a_j and maximum of b_j of jobs assigned to each timeslot. We use two heaps for each timeslot to keep this information: a min-heap (max-heap resp.) keeps the starting timeslot a_j (ending timeslot b_j resp.) of all jobs assigned to the timeslot. Using these two heaps we can compute the directly reachable interval of any timeslot in $\mathcal{O}(1)$ time. When a job is assigned to or moved away from a timeslot, the corresponding heaps have to be updated and each such update takes $\mathcal{O}(\log n)$ time since the size of the heap is bounded by the total number of jobs. By Property 10 (i), the update time for each newly assigned job is bounded by $\mathcal{O}(\min(n, \tau) \log n)$.

Lemma 11. *For each timeslot t_i, we can compute D_i-reachable intervals in $\mathcal{O}(\min(n, \tau))$-time.*

Proof. As described above, we can compute directly reachable interval in $\mathcal{O}(1)$ time. By Property 9, to compute $\mathcal{R}_i^{(l)}$, we only need to consider timeslots in $\mathcal{R}_i^{(l-1)} \setminus \mathcal{R}_i^{(l-2)}$ by checking the corresponding heaps; hence, each timeslot in the D_i-reachable interval needs to be considered in the computation of one l-reachable interval only. The number of timeslots in the D_i-reachable interval could be τ. However, we only need to consider those "occupied" timeslots that have jobs assigned to them. We can keep links among occupied timeslots by a doubly linked list. Each occupied timeslot t is linked to two nearest occupied timeslots $t_l < t$ and $t_r > t$. In this way, we can skip non-occupied timeslots and only check occupied timeslots in $\mathcal{R}_i^{(l-1)} \setminus \mathcal{R}_i^{(l-2)}$ when we compute $\mathcal{R}_i^{(l)}$. By Property 10 (ii), the number of occupied timeslots in the D_i-reachable interval is at most $\min(n, \tau)$ and the overall computation takes $\mathcal{O}(\min(n, \tau))$ time. □

Using Lemma 11, we can analyze the overall time complexity which takes into account also the time taken to update various data structures.

Theorem 12. *We can find the optimal schedule in $\mathcal{O}(n \log \tau + \min(n, \tau) n \log n)$-time for the case where the feasible timeslots associated with each job form a contiguous interval.*

Proof. For each newly considered job, we first compute the D_i-reachable interval $\mathcal{R}_i^{(D_i)}$ in $\mathcal{O}(\min(n, \tau))$ time. We then check the existence of a legal-path from t_i by examining the timeslot t in $\mathcal{R}_i^{(D_i)}$ with the minimum load. If the load $\ell(t_i) - \ell(t) \geq 2$, then there is a legal-path, otherwise, there is no legal-path. This timeslot t can be found by using a simple balanced binary tree structure that supports dynamic range minimum query (RMQ). We store the load of the timeslots 1, 2, $\cdots$, τ in the leaves from left to right. Each internal node maintains the minimum load in its subtree. Using this data structure, we can

return the minimum load in any time interval $[x, y]$ in $\mathcal{O}(\log \tau)$ time. The value from the root to a leaf needs to be modified when the load of a leaf is changed, and such update takes $\mathcal{O}(\log \tau)$ time. When a new job is assigned and a possible shift takes place, at most two timeslots have their load changed. Therefore, the update of the dynamic RMQ structure takes $\mathcal{O}(\log \tau)$ time for each job assigned.

If there exists a legal-path from t_i to t, we construct a legal-path with length at most D_i as follows. Suppose $t \in \mathcal{R}_i^{(l)} \setminus \mathcal{R}_i^{(l-1)}$ for some $l \leq D_i$. We construct a legal-path $[t_i = v_0, v_1, \cdots, v_l = t]$ in a bottom-up fashion. If v_l is on the right extension of $\mathcal{R}_i^{(l-1)}$, i.e., $v_l > \beta_i^{(l-1)}$, then we set v_{l-1} to be the timeslot that $\text{rpfj}(\mathcal{R}_i^{(l)})$ is assigned to; otherwise, i.e., $v_l < \alpha_i^{(l-1)}$, we set v_{l-1} to be the timeslot that $\text{lpfj}(\mathcal{R}_i^{(l)})$ is assigned to. By the definition of path-finder-jobs, v_{l-1} has jobs assigned to it and one of these jobs ($\text{rpfj}(\mathcal{R}_i^{(l)})$ or $\text{lpfj}(\mathcal{R}_i^{(l)})$ accordingly) has a feasible interval covering v_l, hence the arc (v_{l-1}, v_l) exists in the feasible graph and the arc is labelled by $\text{rpfj}(\mathcal{R}_i^{(l)})$ or $\text{lpfj}(\mathcal{R}_i^{(l)})$ correspondingly. Inductively, we can define v_{j-1} from v_j, for $j = l, l-1, \cdots, 1$, until we reach t_i.

Given the legal-path found, we execute a shift along the path. We need to update the heaps of at most D_i timeslots, thus taking $\mathcal{O}(\min(n, \tau) \log n)$ time, by Property 10 (i). The doubly linked list in the proof of Lemma 11 can be updated in $\mathcal{O}(1)$ time when a job is added or removed from a timeslot, and hence updating at most D_i timeslots takes $\mathcal{O}(\min(n, \tau))$ time.

In summary the time taken for assigning a new job (including update of data structures) is bounded by $\mathcal{O}(\log \tau + \min(n, \tau) \log n)$. Therefore, the overall time complexity for assigning n jobs is $\mathcal{O}(n \log \tau + \min(n, \tau) n \log n)$ and the theorem follows. □

6 Conclusion

In this paper we study an offline scheduling problem arising in demand response management in smart grid. We focus on the particular case where requests have unit power requirement and unit duration. We give a polynomial time offline algorithm that gives an optimal solution. Natural generalization extends to arbitrary power requirement and arbitrary duration. The problem where requests have unit power requirement and arbitrary duration has been shown to be NP-hard [19] by a reduction from the bin packing problem. Using a similar idea, it can be shown that the problem where requests have arbitrary power requirement and unit duration is also NP-hard.

For requests with arbitrary power requirement and unit duration, our problem has equivalence to load balancing on unrelated machines. In this setting, the set of unit integral timeslots $T = \{1, 2, \ldots, \tau\}$ corresponds to the set of machines $M = \{M_1, M_2, \ldots M_m\}$, for $\tau = m$. In load balancing on unrelated machines, each job J_j has a (possibly different) given processing time (length) on each machine M_i. In our model, each job J_j with a power requirement p_j has a set of feasible timeslots $I_j \in T$ in which it can be assigned. For equivalence with load balancing on unrelated machines, this means that the processing time

of J_j on all machines M_i for $i \in I_j$ is equal to p_j, and infinity otherwise. In [34], a $(2 - 1/m)$-approximation algorithm is given for load balancing on unrelated machines, m being the number of machines, for minimizing the makespan. We note the following remark about using the approximation algorithm in the model with arbitrary power requirement and unit duration where the cost function is $f(x) = x^\alpha$, for some $\alpha > 1$.

Remark 1. *The $(2-1/m)$-approximation algorithm in [34] for minimizing the makespan for the problem of load balancing on m unrelated machines results in a $((2 - 1/\tau)^\alpha \tau)$-approximation algorithm for minimizing the total cost of scheduling jobs with arbitrary power requirement and unit duration in our model, for τ being the number of timeslots and the cost function being $f(\ell(t)) = (\ell(t))^\alpha$, for a timeslot t with load $\ell(t)$ and some $\alpha > 1$.*

On the other hand, we can extend our optimal algorithm for unit power requirement to an approximation algorithm for arbitrary power requirement by classifying the jobs according to their power requirement into classes of ranges in powers of two, i.e., a job is in class i if its power requirement is in the range $(2^{i-1}, 2^i]$. The power requirement of the jobs in the same class i are then rounded up to 2^i. Each class can be scheduled independently of other classes using the optimal algorithm. The final schedule is effectively obtained by stacking up the schedules of all the classes. Using standard technique (e.g., [3]), one can show that when the cost function is $\ell(t)^\alpha$, the approximation ratio is $O(\log \frac{p_{\max}}{p_{\min}})^\alpha$ through the rounding up and classification, where $p_{\max}$ and $p_{\min}$ are the maximum and minimum power requirement of the jobs, respectively. While this approximation depends on the max-min ratio of power requirement, it does not depend on the total number of timeslots.

Remark 2. *For minimizing the total cost of scheduling jobs with arbitrary power requirement and unit duration in our model where the cost function is $f(\ell(t)) = (\ell(t))^\alpha$, there is an $O(\log \frac{p_{\max}}{p_{\min}})^\alpha$-approximation algorithm, where $p_{\max}$ and $p_{\min}$ are the maximum and minimum power requirement of the jobs, respectively.*

An obvious research direction is to develop better approximation algorithms for the general problem with both arbitrary power requirements and arbitrary job duration. One approach is to consider an approximation algorithm that uses a dual classification of jobs based on their power requirement and duration, and then schedules jobs in different classes independently of each other. Nevertheless, this would create a non-constant gap between the maximum load of the approximation algorithm and the optimal algorithm, and the overall impact on the total electricity cost is yet to be investigated. Similar to Remark 2 we expect the gap is logarithmic in terms of the maximum and minimum power requirements and durations of jobs.

In addition, one may consider different electricity cost function over time, e.g., [10] considers different coefficients on the cost of electricity at different time. We may alter our electricity cost function in a similar way. In [19], online algorithms are considered for the general problem, where power requirements

and durations are stochastic, and the goal is to minimize long-term average cost. It would be also interesting to consider competitive worst-case analysis for online algorithms. Some preliminary work has been announced recently [15].

Acknowledgements

Mihai Burcea is supported by a studentship from the Engineering and Physical Sciences Research Council, UK. Hsiang-Hsuan Liu is supported by the UoL-NTHU Dual-PhD programme studentship.

References

[1] S. Albers. Energy-efficient algorithms. *Communication ACM*, 53(5):86–96, 2010.

[2] Y. Azar. On-line load balancing. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms*, volume 1442 of *LNCS*, pages 178–195. Springer, 1998.

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

[4] P. Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004.

[5] M. Burcea, W.-K. Hon, H.-H. Liu, P. W. H. Wong, and D. K. Y. Yau. Scheduling for electricity cost in smart grid. In P. Widmayer, Y. Xu, and B. Zhu, editors, *Combinatorial Optimization and Applications*, volume 8287 of *LNCS*, pages 306–317. Springer, 2013.

[6] S. Caron and G. Kesidis. Incentive-based energy consumption scheduling algorithms for the smart grid. In *IEEE Smart Grid Comm.*, pages 391–396, 2010.

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

[8] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, Apr. 1972.

[9] European Commission. Europen smartgrids technology platform. `ftp://ftp.cordis.europa.eu/pub/fp7/energy/docs/smartgrids_en.pdf`, 2006.

[10] K. Fang, N. A. Uhan, F. Zhao, and J. W. Sutherland. Scheduling on a single machine under time-of-use electricity tariffs. In *The 12th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, 2015.

[11] X. Fang, S. Misra, G. Xue, and D. Yang. Smart grid – the new and improved power grid: A survey. *Communications Surveys Tutorials, IEEE*, 14(4):944–980, 2012.

[12] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[13] K. Hamilton and N. Gulhar. Taking demand response to the next level. *Power and Energy Magazine, IEEE*, 8(3):60–65, 2010.

[14] D. S. Hochbaum and J. G. Shanthikumar. Convex separable optimization is not much harder than linear optimization. *J. ACM*, 37(4):843–862, Oct. 1990.

[15] W.-K. Hon, H.-H. Liu, and P. W. Wong. Online nonpreemptive scheduling for electricity cost in smart grid. In *The 12th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, 2015.

[16] A. Ipakchi and F. Albuyeh. Grid of the future. *IEEE Power and Energy Magazine*, 7(2):52–62, 2009.

[17] L. D. Kannberg, D. P. Chassin, J. G. DeSteese, S. G. Hauser, M. C. Kintner-Meyer, R. G. Pratt, L. A. Schienbein, and W. M. Warwick. GridWiseTM: The benefits of a transformed energy system. *CoRR*, nlin/0409035, Sept. 2004.

[18] A. V. Karzanov and S. T. McCormick. Polynomial methods for separable convex optimization in unimodular linear spaces with applications. *SIAM J. Comput.*, 26(4):1245–1275, Aug. 1997.

[19] I. Koutsopoulos and L. Tassiulas. Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In *Proc. e-Energy*, pages 41–50, 2011.

[20] R. Krishnan. Meters of tomorrow [in my view]. *IEEE Power and Energy Magazine*, 6(2):96–94, 2008.

[21] H. Li and R. C. Qiu. Need-based communication for smart grid: When to inquire power price? *CoRR*, abs/1003.2138, 2010.

[22] Z. Li and Q. Liang. Performance analysis of multiuser selection scheme in dynamic home area networks for smart grid communications. *IEEE Trans. Smart Grid*, 4(1):13–20, 2013.

[23] A. Llaria, J. Jiménez, and O. Curea. Study on communication technologies for the optimal operation of smart grids. *Transactions on Emerging Telecommunications Technologies*, to appear. `http://dx.doi.org/10.1002/ett.2625`.

[24] Lockheed Martin. SEELoad^{TM} Solution. `http://www.lockheedmartin.co.uk/us/products/energy-solutions/seesuite/seeload.html`.

[25] T. Logenthiran, D. Srinivasan, and T. Z. Shun. Demand side management in smart grid using heuristic optimization. *IEEE Trans. Smart Grid*, 3(3):1244–1252, 2012.

[26] T. Lui, W. Stirling, and H. Marcy. Get smart. *IEEE Power and Energy Magazine*, 8(3):66–78, 2010.

[27] C. Y. T. Ma, D. K. Y. Yau, and N. S. V. Rao. Scalable solutions of markov games for smart-grid infrastructure protection. *IEEE Trans. Smart Grid*, 4(1):47–55, 2013.

[28] S. Maharjan, Q. Zhu, Y. Zhang, S. Gjessing, and T. Basar. Dependable demand response management in the smart grid: A stackelberg game approach. *IEEE Trans. Smart Grid*, 4(1):120–132, 2013.

[29] M. Minoux. A polynomial algorithm for minimum quadratic cost flow problems. *European Journal of Operational Research*, 18(3):377 – 387, 1984.

[30] M. Minoux. Solving integer minimum cost flows with separable convex cost objective polynomially. In G. Gallo and C. Sandi, editors, *Netflow at Pisa*, volume 26 of *Mathematical Programming Studies*, pages 237–239. Springer, 1986.

[31] A.-H. Mohsenian-Rad, V. Wong, J. Jatskevich, and R. Schober. Optimal and autonomous incentive-based energy consumption scheduling algorithm for smart grid. In *Innovative Smart Grid Technologies (ISGT)*, 2010.

[32] REGEN Energy Inc. ENVIROGRID^{TM} SMART GRID BUNDLE. http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/.

[33] S. Salinas, M. Li, and P. Li. Multi-objective optimal energy consumption scheduling in smart grids. *IEEE Trans. Smart Grid*, 4(1):341–348, 2013.

[34] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127 – 133, 2005.

[35] P. T. Sokkalingam, R. K. Ahuja, and J. B. Orlin. New polynomial-time cycle-canceling algorithms for minimum-cost flows. *Networks*, 36(1):53–63, 2000.

[36] Toronto Hydro Corporation. Peaksaver Program. http://www.peaksaver.com/peaksaver_THESL.html.

[37] 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.

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

[39] L. A. Végh. Strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 27–40, New York, NY, USA, 2012. ACM.

[40] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[41] 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.