Scheduling for Electricity Cost in Smart Grid

Mihai Burcea[1,*], Wing-Kai Hon[2], Hsiang-Hsuan Liu[2],
Prudence W.H. Wong[1], and David K.Y. Yau[3]

[1] Department of Computer Science, University of Liverpool, UK
{m.burcea,pwong}@liverpool.ac.uk
[2] Department of Computer Science, National Tsing Hua University, Taiwan
{wkhon,hhliu}@cs.nthu.edu.tw
[3] Information Systems Technology and Design, Singapore University of Technology and Design, Singapore
david_yau@sutd.edu.sg

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 is a contiguous interval.

1 Introduction

We study an offline scheduling problem arising in "demand response management" in smart grid [7, 9, 18]. The electrical smart grid is one of the major challenges in the 21st century [6, 28, 29]. 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 [4] that in the US power grid, 10% of all generation assets and 25% of distribution infrastructure are required for less than 400 hours per year, roughly 5% of the time [29]. *Demand response management* attempts to overcome this problem by shifting users' demand to off-peak hours in order to reduce peak load [3, 12, 17, 20, 23, 25]. This is enabled technologically by the advances in smart meters [13] and integrated communication. Research initiatives in the

[*] Supported by EPSRC Studentship.

area include GridWise [10], the SeeLoad$^{\text{TM}}$ system [16], EnviroGrid$^{\text{TM}}$ [24], peak demand [27], etc.

The smart grid operator and consumers communicate through smart metering devices. We assume that time is divided into 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 [12] 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. [25] 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 [3, 20, 23]. Heuristics have also been developed for demand side management [17]. Other aspects of smart grid have also been considered, e.g., communication [4, 14, 15], security [19]. Reviews of smart grid can be found in [7, 9, 18].

The combinatorial problem we defined in this paper has analogy to the traditional 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, 31] 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 [5] with convex functions [21, 26] when we have unit power requirement and unit duration for each job; and the maximum-cardinality minimum-weight matching on a bipartite graph. Yet, existing algorithms for the problem cater for more general input [8, 11, 22, 30]. They are more powerful and have higher time complexity than necessary to solve our problem.

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 forms a contiguous interval, we can improve the time complexity to $\mathcal{O}(n\tau \log n)$.

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.

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.

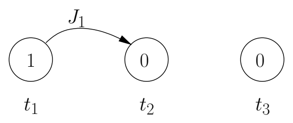
2 Preliminaries

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.

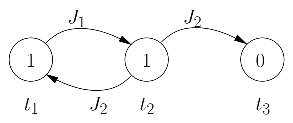
Feasible graph. Given a particular job assignment A, we define a *feasible graph* G which is a directed multi-graph that shows the potential allocation of each job in alternative assignments. In G each timeslot is represented by a vertex. If job J_i is assigned to timeslot r in A, then for all $w \in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

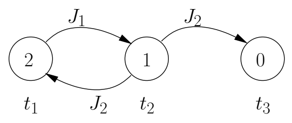
Agreement graph. We define an *agreement graph* $G_a(A, A^*)$ which is a directed multi-graph that measures the difference between a job assignment solution A and an optimal assignment A^*. In $G_a(A, A^*)$ each timeslot is represented by a vertex. For each job J_i such that J_i is assigned to different timeslots in A and A^*, we add an arc from t to t', where t and t' are the timeslots that J_i

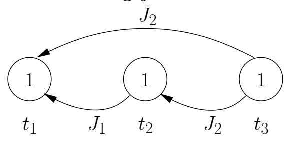


(a) The feasible graph G after adding job J_1 to t_1.



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





(c) Left: The feasible graph immediately after J_3 is added to slot t_1. The path (t_1, t_3) is a legal-path and we shift by moving J_1 to t_2 and J_2 to t_3. Right: The feasible graph after the shift, with no more legal-paths.

Fig. 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 $I_3 = \{t_1\}$. The number inside the vertices denotes their load. Suppose the algorithm schedules the jobs in order of their indices. (a) and (b) Jobs J_1 and J_2 are arbitrarily assigned their feasible minimum load slots. (c) A legal-path and the corresponding shift after assigning J_3.

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.

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 $\ell(t_1) < \ell(t_2) + 1$.*

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 6.

3 Our Algorithm

The algorithm. We propose a polynomial time offline algorithm that minimizes the total cost (Figure 1 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.

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 adding 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 1, we show that if a legal-path exists in G after adding 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 3, we show that if there is no legal-path in the feasible graph G before adding a job, then after adding 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 6, we show that if there is no legal-path in a feasible graph G, the corresponding assignment is optimal and hence Invariant (I2) holds.

Proof of Invariant (I1)

Lemma 1. *Suppose that before adding job J_i to timeslot r the feasible graph G has no legal-path. If there is any legal-path after adding 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 adding 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 adding J_i. Thus, Case 1 cannot occur.
Case 2. G does not contain any (s,t) path before adding J_i. Since (s,t) becomes a legal-path after adding J_i, it must be the case that assigning J_i to timeslot

r adds some new edge (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 adding J_i. Also, $\ell'_i(s) = \ell_{i-1}(s)$ 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 2. *If before adding a job the feasible graph G does not have a legal-path, then after adding 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.*

Lemma 2 will be proved in the full paper and we proceed with Invariant (I1).

Lemma 3. *Suppose that G is a feasible graph with no legal-paths. Then after adding 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 1, there must be one such legal-path (s, t) where s is the timeslot assigned to J_i, and without loss of generality, let it 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 adding J_i, and $Out(r)$ to be the set of vertices w such that an (r, w) path exists before adding 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, 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 adding J_i. In this case, it is easy to check that $IN''(P) \subseteq IN(P)$,[§] and $OUT''(P) \subseteq OUT(P) \cup OUT(I_i)$.[‡]

[§] 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.

[‡] 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

Suppose that $\ell_{i-1}(s) = x$. Then, $\ell_{i-1}(t) = x-1$ because there is no legal-path before adding J_i but there is one after adding 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 adding J_i. The load of any vertex in $IN(P)$ is at most x or there was a legal-path entering t before adding J_i. The load of any vertex in $OUT(P)$ is at least $x-1$ or there was a legal-path leaving s before adding 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 adding 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 adding 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 added. 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 adding J_i creates a new legal-path (s, t), by Lemma 2, $\ell'_i(t) = \ell_{i-1}(t) = x-1$. Thus, the load of any vertex in $IN(P\backslash\{s\})$ is at most x, since there was no legal-path entering t before adding 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 adding 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. □

Proof of Invariant (I2)

We now prove in Lemma 6 (the other key lemma for the correctness) that non-existence of legal-paths implies the assignment is optimal. The rough ideas are as follows. Consider an optimal assignment A^* (satisfying some constraints as to be defined). In Lemma 5, 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 is the original assignment of jobs given by our algorithm, and A_k is an optimal assignment. We prove Lemma 6 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 5 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 5 considers an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic. The existence of such A^* is stated here and proved in the full paper.

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.

Lemma 4. *There exists an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic.*

Lemma 5. *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 $i \geq 1$, 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 4, $G_a(A_1, A^*)$ 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 and v_i has out-degree 0. 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 6. *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 5. 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 5 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_a and t_b. If there is no such arc, then A is already an optimal solution (since the sequence will be both non-increasing by Lemma 5 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 $\ell(A_j, t_a) = x$, then $\ell(A_j, t_b) \leq x - 2$ as the overall energy cost would be decreasing by moving a job from t_a to t_b. If $\ell(A_1, t_a) = x$ and $\ell(A_1, t_b) \leq x-2$ 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 $u_{i_0} = t_a$. 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 added 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 $v_{k_0} = t_b$, 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 prove the time complexity of our algorithm in Theorem 2 and show that this can be improved for the case where the feasible timeslots associated with each job are contiguous.

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

Proof. We add 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 1, 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 2, 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 3. Finally we update the edges 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 $n\tau$ edges in the feasible graph. If a legal-path exists after adding 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, at most $n\tau$ edges are updated in the feasible graph, taking $\mathcal{O}(n\tau)$ time. The total time for adding n jobs is thus bounded by $\mathcal{O}(n^2\tau)$. $\square$

We now consider the special case where each job $J_i \in \mathcal{J}$ is associated with an interval of contiguous timeslots $I_i = [\rho_i, \delta_i]$, for positive integers $\rho_i \le \delta_i$, and each job J_i must be assigned to exactly one feasible timeslot s_i, for $\rho_i \le s_i \le \delta_i$. We give a sketch here, while the full proof can be found in the full paper.

Theorem 3. *We can find the optimal schedule in $\mathcal{O}(n\tau \log n)$ time for the case where the feasible timeslots associated with each job are contiguous.*

Proof (Sketch). For the special case, we use data structure techniques for the speed up. For each timeslot $t_i \in T$, we use two balanced binary search trees that contain the feasible intervals for all jobs assigned to t_i. For each job J_j with $I_j = [\rho_j, \delta_j]$ assigned to t_i, the first binary tree keeps the value of ρ_j, while the second binary tree keeps the value of δ_j. The binary trees are updated whenever a job is moved to and from t_i accordingly, and each such update takes $\mathcal{O}(\log n)$ time. We can query a minimum and a maximum value of the two trees, respectively, in order to establish the *directly reachable interval* of timeslot t_i, i.e., the other timeslots that jobs from t_i can be moved to. Because of the contiguous property of the feasible intervals, the set of timeslots is contiguous. We denote this interval of timeslots by $[\alpha_i, \beta_i]$ and we have that $\alpha_i \le t_i \le \beta_i$.

We further find the set of the ending vertices of all the paths of length at most $\tau - 1$ that start from t_i, which we call *reachable interval.* Note that the ending vertices of paths of length 2 from t_i can be found by checking the binary search trees of each timeslot in $[\alpha_i, \beta_i]$, which can then be used to find vertices at distance 3 from t_i and so on. Finding the reachable interval requires $\mathcal{O}(\tau)$ time. We can then identify any legal path in $\mathcal{O}(\tau)$ time.

In summary, adding a job to the feasible graph takes $\mathcal{O}(\log n)$ time. Finding the reachable interval and legal path takes $\mathcal{O}(\tau)$ time. Shifting of jobs along the legal path found takes $\mathcal{O}(\tau \log n)$ time. Thus the time taken to add one job is bounded by $\mathcal{O}(\tau \log n)$. The overall time for adding all n jobs is thus bounded by $\mathcal{O}(n\tau \log n)$. $\square$

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 [12] 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. An obvious research direction is to develop approximation algorithms for the general problem. It would be also interesting to consider online algorithms for the problem.

Acknowledgement: We would like to thank the reviewers for very helpful comments leading to improvement in the time complexities of our algorithms.

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. S. Caron and G. Kesidis. Incentive-based energy consumption scheduling algorithms for the smart grid. In *IEEE Smart Grid Comm.*, pages 391–396, 2010.
4. 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.
5. J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, Apr. 1972.
6. European Commission. Europen smartgrids technology platform. `ftp://ftp.cordis.europa.eu/pub/fp7/energy/docs/smartgrids_en.pdf`, 2006.
7. K. Hamilton and N. Gulhar. Taking demand response to the next level. *Power and Energy Magazine, IEEE*, 8(3):60–65, 2010.
8. 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.
9. A. Ipakchi and F. Albuyeh. Grid of the future. *IEEE Power and Energy Magazine*, 7(2):52–62, 2009.
10. 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.
11. 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.

12. 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.
13. R. Krishnan. Meters of tomorrow [in my view]. *IEEE Power and Energy Magazine*, 6(2):96–94, 2008.
14. H. Li and R. C. Qiu. Need-based communication for smart grid: When to inquire power price? *CoRR*, abs/1003.2138, 2010.
15. 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.
16. Lockheed Martin. SEELoad^{TM}Solution. `http://www.lockheedmartin.co.uk/us/products/energy-solutions/seesuite/seeload.html`.
17. 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.
18. T. Lui, W. Stirling, and H. Marcy. Get smart. *IEEE Power and Energy Magazine*, 8(3):66–78, 2010.
19. 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.
20. 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.
21. M. Minoux. A polynomial algorithm for minimum quadratic cost flow problems. *European Journal of Operational Research*, 18(3):377 – 387, 1984.
22. 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.
23. 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.
24. REGEN Energy Inc. ENVIROGRID^{TM}SMART GRID BUNDLE. `http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/`.
25. 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.
26. 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.
27. Toronto Hydro Corporation. Peaksaver Program. `http://www.peaksaver.com/peaksaver_THESL.html`.
28. 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.
29. US Department of Energy. The Smart Grid: An Introduction. `http://www.oe.energy.gov/SmartGridIntroduction.htm`, 2009.
30. 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.
31. 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.