

Article

# Scheduling Non-preemptible Jobs to Minimize Peak Demand

Sean Yaw<sup>1</sup> and Brendan Mumey<sup>2</sup><sup>1</sup> Los Alamos National Laboratory, Los Alamos, NM USA; yaw@lanl.gov<sup>2</sup> Gianforte School of Computing, Montana State University, Bozeman, MT USA; mumey@montana.edu

**Abstract:** This paper examines an important problem in smart grid energy scheduling; peaks in power demand are proportionally more expensive to generate and provision for. The issue is exacerbated in local microgrids that do not benefit from the aggregate smoothing experienced by large grids. Demand-side scheduling can reduce these peaks by taking advantage of the fact that there is often flexibility in job start times. We focus attention on the case where the jobs are non-preemptible, meaning once started, they run to completion. The associated optimization problem is called the *Peak Demand Minimization* problem and has been previously shown to be NP-hard. Our results include an optimal fixed-parameter tractable algorithm, a polynomial-time approximation algorithm, as well as an effective heuristic that can also be used in an online setting of the problem. Simulation results show these methods can reduce peak demand by up to 50% versus on-demand scheduling for household power jobs.

**Keywords:** peak demand minimization; job scheduling; approximation algorithms; smart grid

## 1. Introduction

The problem of scheduling non-preemptible jobs occurs naturally in smart power grid systems, in which communication can occur between energy consumers and the energy provider. Some household appliances require instant functionality (e.g. light bulbs), but many others have more flexibility in exactly when they operate (e.g. dishwasher or water heater). Peaks in power demand are proportionally more expensive to generate and provision for (since more infrastructure is required), so it is advantageous to schedule power consuming jobs in such a way as to minimize peak demand. This problem has been previously formalized as the *Peak Demand Minimization* (PDM) problem and has been studied extensively [1–8]. The basic formulation of the problem is as follows: Each job  $j$  is non-preemptible, meaning once it begins execution, it must run to completion without any interruptions. Each job  $j$  is characterized by four parameters: an arrival time ( $a_j$ ), deadline ( $d_j$ ), and length ( $l_j$ ) that are real-valued for continuous timescales and integer-valued for discrete timescales, and a real-valued instantaneous demand ( $h_j$ ) which is conceptually the height of the job. The arrival time and deadline are within the fixed time interval  $[0, T]$  and form the *execution window* of the job. Job  $j$  is scheduled by assigning it a start time,  $s_j$ , which allows it to run in the closed interval  $[s_j, s_j + l_j]$  such that  $[s_j, s_j + l_j] \subseteq [a_j, d_j]$ . The *demand* at time  $t$  is the sum of the job heights that are scheduled to run during  $t$ :

$$H(t) = \sum_{j:t \in [s_j, s_j + l_j]} h_j$$

Then, the *peak demand*,  $H^{\max}$ , of the schedule is the maximum demand for any time  $t$  in  $[0, T]$ ,

$$H^{\max} = \max_{t \in [0, T]} H(t)$$

The PDM problem is to determine a job schedule,  $\mathcal{S} = \langle s_j \rangle$  that minimizes  $H^{\max}$ . In this work, we propose three algorithms to solve PDM:

1. An optimal dynamic programming algorithm utilizing branch-and-bound techniques that is *fixed-parameter tractable*.

2. A polynomial-time algorithm based on linear programming that provides a  $O\left(\frac{\log n}{\log \log n}\right)$ -approximation and is the first known approximation for PDM.
3. An effective and simple heuristic algorithm that can be used in either an online or offline fashion.

## 2. Related Work

Many variations on modifying demand to improve the efficiency of power grids (generically called *demand response*) have been explored. These approaches can generically be categorized as utility function optimization (commonly cost function minimization) and peak demand minimization (PDM). Utility function optimization techniques use price incentives to encourage consumers to modify their behavior and are outside the scope of this work.

PDM aims to directly schedule jobs to reduce peaks in power schedules instead of passively influencing changes with price incentives. PDM has been studied in the context of preemptible jobs (i.e. jobs that can be interrupted partway through execution) [1,2]. In this paper, we consider the scenario where jobs are non-preemptible, and thus must run to completion once started. The PDM problem has been found to be NP-hard to approximate within a ratio of 2 in this scenario [3]. Heuristics have been developed that show promise in practice, but which provide no theoretical guarantee [4,5]. Approximation algorithms have been developed for the special case where all jobs have the same start time and deadline [6,7]. Our work introduces the first general purpose approximation algorithm where no assumptions are placed on the job parameters. Finally, optimal algorithms have been introduced that work on a limited number of jobs [3,8]. Our work also introduces an optimal, fixed-parameter tractable, algorithm that is able to schedule approximately 15 times more jobs in practice than previous approaches.

Non-preemptible power job scheduling is also similar to the machine minimization problem and rectangular strip packing [9–13]. In the machine minimization problem, jobs are assumed to have unit height, as opposed to power jobs that can have variable height. The main differences with rectangular strip packing are that, in the general PDM case, jobs are limited in where they can be placed in the strip and once jobs are scheduled, they do not need to remain as intact rectangles. Since job height represents the power required, each segment of a scheduled job will drop to lie on top of the job below it, instead of remaining as an intact rectangle in the strip.

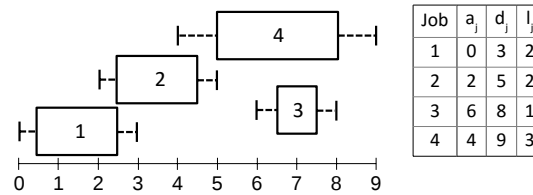
## 3. Algorithms

We first present an optimal, fixed-parameter tractable algorithm, that improves on the efficiency of the method described in [3] due to an effective branch-and-bound idea. The second algorithm described is based on linear programming relaxation and randomized rounding that is a  $O\left(\frac{\log n}{\log \log n}\right)$ -approximation algorithm for the general PDM problem. Finally, we describe a heuristic approach that can also be adapted to the online setting of the problem.

### 3.1. An Optimal Dynamic Programming Algorithm

In this section, we detail a dynamic programming algorithm for PDM, inspired by the algorithm presented in [3]. Our algorithm takes a similar approach of identifying groups of overlapping jobs, but uses a tree-based schedule representation and a branch-and-bound strategy to reduce the search space; resulting in substantially better performance (able to run instances with 15 times as many jobs). We term this algorithm *OptimalDP* in the results section.

At a high level, jobs are first ordered by increasing deadlines. For each job, we consider every possible configuration of start times of that job and as well as other earlier jobs that could overlap any job in the future. Job configurations are represented using trees where each level in the tree represents the scheduling choices of one job. We keep track of which configurations of the current job are compatible with those of the previous job using pointers. Valid schedules can be built by traversing



**Figure 1.** Sample jobs with arrival times, deadlines, lengths, and job configuration lists:  $\mathcal{L}_1 = \{1\}$ ,  $\mathcal{L}_2 = \{1,2\}$ ,  $\mathcal{L}_3 = \{2,3\}$ ,  $\mathcal{L}_4 = \{2,3,4\}$ .

these trees backwards, beginning with the final job in the list. A branch-and-bound approach is employed to limit unnecessary schedule exploration.

### Configuration Lists

We sort the jobs by increasing deadline first and then by increasing arrival time. For each job  $j$ , a list of preceding jobs  $\mathcal{L}_j$ , called the *configuration list*, is defined as the set of jobs whose execution windows either overlap  $j$ 's execution window, or overlap the execution window of a job following  $j$ . Figure 1 illustrates a sample set of jobs along with their configuration lists. We observe that each job configuration list consists of a consecutive interval in the sorted list of jobs, i.e.  $\mathcal{L}_j = [s(j), j]$ , where the head  $s(j)$  of each list has the earliest deadline (and arrival in case of ties).

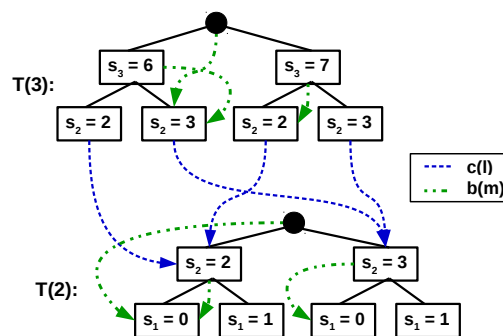
**Lemma 1.** The list heads are non-decreasing. (i.e.  $s(1) \leq s(2) \leq \dots \leq s(n)$ ).

**Proof.** Suppose job  $j \leq k$ . Since  $s(k) \in \mathcal{L}_k, \exists l \geq k$  s.t.  $a_l < d_{s(k)}$ . If  $j \in \mathcal{L}_k$ , then  $s(k) \in \mathcal{L}_j$  since  $a_l < d_{s(k)} \leq d_j$  and  $l \geq k > j$ . Thus  $s(j) \leq s(k)$ . If  $j \notin \mathcal{L}_k$ , then  $d_{s(j)} \leq d_j < d_{s(k)}$  and so  $s(j) < s(k)$ .  $\square$

### Configuration Trees

For each job  $j$ , we can represent each possible configuration of start times of all the jobs in its configuration list  $\mathcal{L}_j = [s(j), j]$  using a *configuration tree*,  $T(j)$ . We consider the jobs in reverse order starting from  $j$  and create a branching tree where nodes at level  $k$  in the tree correspond to the possible starting times for job  $j - (k - 1)$ . (We assume the root of the tree is at level 0). Thus, each leaf in  $T(j)$  specifies start times for all jobs in  $\mathcal{L}_j$ . Let **height**( $l$ ) be the peak demand of this particular schedule of  $\mathcal{L}_j$  job start times.

We need to keep track of compatible job configurations in order to build a consistent schedule for all jobs. We do this by maintaining a compatibility pointer,  $c(l)$ , from each leaf node  $l$  in  $T(j)$  to a compatible node in  $T(j - 1)$  ( $j > 1$ ). By Lemma 1,  $\mathcal{L}_j \cap \mathcal{L}_{j-1} = [s(j), j - 1]$ . Thus we can follow the



**Figure 2.** Sample configuration trees for two jobs with  $c(l)$  and  $b(m)$  pointers (Note:  $b(m)$  pointers from leaves to themselves are omitted). Starting times reflect the sample jobs from Figure 1. By following alternating  $b$  and  $c$  pointers, we see an optimal solution is  $\langle s_3 = 6, s_2 = 3, s_1 = 0 \rangle$ .

same path in  $T(j-1)$  that we took beginning with a node at level 2 in  $T(j)$  to reach the compatible node in  $T(j-1)$ . If  $[s(j), j-1] = \emptyset$ , then we simply make each leaf in  $T(j)$  point to the root of  $T(j-1)$ .

### Dynamic Programming

Once we have configuration trees  $T(j)$  constructed and compatibility pointers  $c(l)$  found, then we can compute the minimum overall schedule efficiently using dynamic programming as follows:

For each leaf node  $l \in T(j)$  we store  $h(l)$  which is the peak height of an optimal schedule that is compatible with  $l$ . For each node  $m \in T(j)$  we will store a pointer,  $b(m)$ , to a leaf  $l$  in the subtree rooted at  $m$  with best (i.e. lowest) peak height  $h(l)$ . The  $b(m)$  pointer for a leaf node is to itself. The  $h$  values satisfy the following recurrence,

$$h(l) = \begin{cases} \text{height}(l) & \text{if } l \in T(1) \\ \max[h(b(c(l))), \text{height}(l)] & \text{if } l \in T(j), j > 1 \end{cases} \quad (1)$$

We note that once the  $h(l)$  values are found for a given tree  $T(j)$ , it is easy to compute the  $b(m)$  pointers for the internal nodes  $m \in T(j)$  in a bottom-up fashion. Thus we can apply dynamic programming to find the  $h$  and  $b$  values for each job configuration tree in order, beginning with  $T(0)$ . An optimal schedule can be found by following alternating  $b$  and  $c$  pointers, starting with  $b(\text{root}[T(m)])$ . The peak height of this schedule will be given by  $h(b(\text{root}[T(m)]))$ . Figure 2 shows an example with two configuration trees,  $c(l)$ , and  $b(m)$  pointers.

### Branch-and-Bound Approach

To avoid unnecessary tree expansion, a branch-and-bound technique can be employed to avoid explicitly building and storing configuration trees. Configuration lists are created for each job as described above. The algorithm keeps track of the peak demand for the best schedule found so far (initialized to  $+\infty$ ).

The algorithm begins a recursive process of building and exploring paths through the configuration trees by starting with the last job in the list. It proceeds in a depth first fashion, creating subtrees of the previous job as compatibility pointers are followed. Once a viable path has been established from the last job through the first, the peak demand value is updated, and the method recurses to explore alternate paths. If at any point an alternate path realizes an  $h(l)$  value above the current best peak demand value, that subtree can be pruned and all compatibility nodes referencing it ignored.

### Fixed-Parameter Tractability

An algorithm is said to be fixed-parameter tractable if there exists some parameters,  $p_1, p_2, \dots$ , of the input or output such that the running time of the algorithm is a polynomial function of the input size times some function of the parameters:  $f(p_1, p_2, \dots) \cdot \text{poly}(n)$ . The complexity of the algorithm is driven by the size of the configuration trees  $T(j)$ , that can be bounded by their branching factor and maximum depth. The maximum branching factor is  $q = \max_j(d_j - a_j - l_j + 1)$  and the maximum depth is  $r = \max_j(|\mathcal{L}_j|)$ . Thus the maximum number of nodes in each configuration tree is  $O(q^r)$ . Since the dynamic programming time is easily seen to be linear in the size of the trees, the total running time is  $O(q^r \cdot n)$ .

### 3.2. An Approximation Algorithm

In this section we detail a novel algorithm for the PDM problem that provides the first known approximation guarantee for the general PDM problem. The algorithm is based on relaxing an Integer Linear Program (ILP) to allow real-valued solutions and then rounding the real-valued solution back to an integer solution.

## Integer Linear Programming Formulation

The PDM problem can be formulated as an ILP using the following notation and variables:

$J$  - Set of jobs.

$I_j$  - A finite set of valid execution intervals for job  $j \in J^1$

$h_j$  - Height of job  $j$ .

$L$  - Set of all left hand time points of intervals in  $\cup_j I_j$ .

$H^{\max} \in \mathbb{R}$  - Peak demand.

$x_{i,j} \in \{0, 1\}$  - Indicates if interval  $i \in I_j$  is scheduled.

The ILP is:

$$\min H^{\max}$$

subject to:

$$\sum_{i \in I_j} x_{i,j} = 1 \quad \forall j \quad (2)$$

$$\sum_{i \in \cup_j I_j: t \in i} h_j x_{i,j} \leq H^{\max} \quad \forall t \in L \quad (3)$$

Constraint (2) ensures that exactly one interval will be selected for each job, and (3) ensures that  $H^{\max}$  will denote the peak height of the schedule. Only elements of  $L$  need to be considered in (3) as a consequence of Lemma 2.

**Lemma 2.** *The maximum height of any schedule must occur for some  $t \in L$ .*

**Proof.** Schedule height only increases at the moment a new job begins processing, thus the peak height of a schedule must be initiated at the start time of some job. Since  $L$  is the collection of all possible start times for all jobs, some point in  $L$  must correspond to the arrival time of the peak height of the schedule.  $\square$

## A Randomized Rounding Algorithm

The ILP presented above cannot be efficiently solved due to the restriction of  $x_{i,j}$  to integer values. To address this, we construct a relaxed LP in the exact same fashion as the ILP, except the  $x_{i,j}$  variables may take on any values in the range  $[0, 1]$ . After the relaxed LP is solved, a specific interval is selected for each job  $j$  by randomly selecting interval  $i$  with probability  $x_{i,j}$ . This linear programming rounding scheme is presented in Algorithm 1. We note the running time is polynomial, since linear programs can be solved in polynomial time.

---

### Algorithm 1 RoundLP

---

Step 1 Solve the LP as described in Section 3.2, but with real-valued  $x_{i,j} \in [0, 1]$ .

Step 2 Determine a start time for each job as follows:

**forall** jobs  $j$   
     Pick a random  $i \in I_j$  with probability  $x_{i,j}$   
     Set  $s_j$  to be the start time of interval  $i$   
**endforall**

---

<sup>1</sup> An interval  $[s_j, s_j + l_j)$  is valid for job  $j$  iff  $a_j \leq s_j$  and  $s_j + l_j \leq d_j$ .

## Continuous timescales

The ILP detailed above is polynomial in size for the case of discrete timescales, where each  $I_j$  is finite, but not for continuous timescales in which jobs can be scheduled anywhere in the arrival-deadline window (so  $I_j$  is infinite). Using a technique similar to the one introduced in [9] for the unit-height machine minimization problem, we show that any continuous instance can be transformed into a corresponding discrete instance such that an optimal solution to the discrete instance provides a solution to the continuous instance that is within a factor of two from optimal.

First, schedule a subset of the jobs,  $J'$  while concurrently identifying a set of time points,  $P$ , such that each interval of every remaining job contains at least one element of  $P$ . This is accomplished by scheduling as many jobs as possible horizontally, without any overlaps. We start at the first time point and proceed in this fashion: As soon as no job is being serviced, schedule the job that will complete the earliest from all remaining jobs.  $P$  can be constructed from this initial scheduling by adding the start and end times from all the scheduled intervals in  $J'$  and by adding the release times and deadlines for each remaining job in  $J \setminus J'$ . Thus,  $|P| \leq 2n$ . Then, each interval of every job in  $J \setminus J'$  must contain at least one point of  $P$ . If some interval of job  $j \in J \setminus J'$  did not contain a point of  $P$ , then either it is nested inside a scheduled interval and thus would have been scheduled since it ends sooner, or it is totally separated from any scheduled intervals and thus would also have been scheduled as well.

Next, expand each interval of each remaining job  $j$  by moving its left and right endpoints to the nearest points in  $P$  to the left and right of the original endpoints, respectively. This results in a finite number (at most  $|P|$ ) of possible intervals for job  $j$ .

Consider an optimal solution to the continuous instance. When this solution is transformed to a discrete instance by expanding the intervals as above, the overall height of the solution may increase, as two intervals that did not overlap now overlap. It is easy to see that these intervals must have originally overlapped two consecutive points from  $P$ . Thus, the height of the schedule can at most double after interval expansion. This implies that the height of an optimal schedule for the discrete instance is at most twice the optimal height of the continuous instance.

Finally, observe that in the discrete instance we only need to look at the height at times  $t \in L$  to determine the overall height of the schedule; this is because we can slide any time  $t$  a little bit to the left until it touches the left endpoint of some interval without changing the overall height of the schedule above  $t$ . By our construction,

$$|L| \leq |P| \leq 2n \quad (4)$$

The following theorem shows that Algorithm 1 provides a polynomial-time approximation algorithm for PDM (with a continuous timescale).

**Theorem 3.** *The schedule generated by the relaxed LP has a height of at most  $O\left(\frac{\log n}{\log \log n}\right)$  times the height of the optimal ILP's schedule, with probability at least  $1 - O(1/n)$ .*

**Proof.** For each  $t \in L$ , define the random variables  $Z_t$  and  $Y_t$  as,

$$Z_t = \sum_{i \in \cup_j I_j: t \in i} h_j x_{i,j}$$

$$Y_t = \sum_{i \in \cup_j I_j: t \in i} \frac{h_j}{h} x_{i,j}, \text{ where } h = \max_j \{h_j\}$$

Bounding the probability that  $Y_t$  exceeds some factor  $\alpha$  times the optimal height ( $H^{\text{opt}}$ ) divided by  $h$  results in the same bound for the probability that  $Z_t$  exceeds  $\alpha$  times the optimal schedule height:

$$\begin{aligned}\Pr[Z_t > \alpha H^{\text{opt}}] &= \Pr[hY_t > \alpha H^{\text{opt}}] \quad (\text{since } Z_t = hY_t, \forall t) \\ &= \Pr\left[Y_t > \alpha \frac{H^{\text{opt}}}{h}\right]\end{aligned}\quad (5)$$

We use a Chernoff-type bound [14] to bound the deviation of  $Y_t$  above  $\frac{H^{\text{opt}}}{h}$ . Note that  $E[Y_t] \leq \frac{H^{\text{opt}}}{h}$  by (3). Since  $x_{i,j}$  are independent Bernoulli trials, for any given  $t$ , with  $E[x_{i,j}] = p_{i,j}$  and  $0 < \frac{h_j}{h} \leq 1$  for all jobs, the following bound exists for all  $\delta > 0$ :<sup>2</sup>

$$\Pr\left[Y_t > (1 + \delta) \frac{H^{\text{opt}}}{h}\right] < \left[\frac{e^\delta}{(1 + \delta)^{(1 + \delta)}}\right]^{\frac{H^{\text{opt}}}{h}}$$

Substituting  $\alpha = 1 + \delta$  and gathering terms on the right hand of the expression into one exponential results in,

$$\begin{aligned}\Pr\left[Y_t > \alpha \frac{H^{\text{opt}}}{h}\right] &< \exp\left[-\left(\frac{H^{\text{opt}}}{h}\right)(\alpha \ln \alpha - \alpha + 1)\right] \\ &\leq \exp\left[-\alpha \ln \alpha + \alpha - 1\right] \quad (\text{since } \frac{H^{\text{opt}}}{h} \geq 1) \\ &< \exp\left[-\alpha(\ln \alpha - 1)\right],\end{aligned}\quad (6)$$

for all  $\alpha > 1$ . Next, choose  $\alpha = \frac{4 \log n}{\log \log n}$  and consider the value of  $\ln \alpha - 1$ :

$$\begin{aligned}\ln \alpha - 1 &= \ln\left(\frac{4 \log n}{\log \log n}\right) - 1 \geq \ln \log n - \ln \log \log n \\ &= \frac{\log \log n}{\log e} - \frac{\log \log \log n}{\log e} \quad (\text{change of base}) \\ &\geq \frac{1}{\log e}(\log \log n - \frac{1}{2} \log \log n) = \frac{\log \log n}{2 \log e}.\end{aligned}\quad (7)$$

Plugging (7) and  $\alpha = \frac{4 \log n}{\log \log n}$  into (6) yields,

$$\begin{aligned}\Pr\left[Y_t > \frac{4 \log n}{\log \log n} \frac{H^{\text{opt}}}{h}\right] &< \exp\left[\frac{-4 \log n \log \log n}{\log \log n \cdot 2 \log e}\right] \\ &= \exp\left[\frac{-4 \ln n \log e \log \log n}{\log \log n \cdot 2 \log e}\right] \\ &= \exp\left[-2 \ln n\right] = 1/n^2.\end{aligned}\quad (8)$$

<sup>2</sup> Theorem 1 in [15] proves that  $\Pr[X > (1 + \delta)\mu] < \left[\frac{e^\delta}{(1 + \delta)^{(1 + \delta)}}\right]^\mu$  for  $\mu = E[X]$ , but it is easy to see that this holds for all  $\mu \geq E[X]$ .

Finally, we use (5), (8), (4) and Lemma 2 to give a probabilistic bound on the height of rounded schedule,  $H^{\text{rnd}}$ :

$$\begin{aligned} \Pr \left[ H^{\text{rnd}} > \frac{4 \log n}{\log \log n} H^{\text{opt}} \right] &\leq \sum_{t \in L} \Pr \left[ Z_t > \frac{4 \log n}{\log \log n} H^{\text{opt}} \right] \\ &= \sum_{t \in L} \Pr \left[ Y_t > \frac{4 \log n}{\log \log n} \frac{H^{\text{opt}}}{h} \right] \\ &< \sum_{t \in L} 1/n^2 \\ &\leq |L|/n^2 \\ &\leq 2n/n^2 = 2/n \in O(1/n). \end{aligned}$$

So, with probability  $1 - O(1/n)$ ,  $H^{\text{rnd}} \leq \frac{4 \log n}{\log \log n} H^{\text{opt}}$ .  $\square$

### 3.3. A Greedy Heuristic

Lastly, we also consider a greedy approach for PDM that schedules each job to have a start time with the smallest contribution to the peak energy required by the schedule. We consider both an offline and online version of the algorithm. In the online version, each job must be scheduled as soon as it arrives. The online algorithm schedules each job sequentially by finding the first starting time that minimizes the resulting peak demand. The complete algorithm is presented as Algorithm 2.

---

#### Algorithm 2 MinFit-Online

---

Step 1 Define  $height(\mathcal{S}, j, s_j)$  as the height of schedule  $\mathcal{S}$  with new job  $j$  scheduled to start at  $s_j$ .  
 Step 2 Build schedule,  $\mathcal{S}$ , by determining start times as follows:  
   **forall** jobs,  $j$  (in the order that jobs arrive)  
      $s = a_j$   
     **for** time  $t = a_j + 1, \dots, d_j - l_j + 1$   
       **if**  $height(\mathcal{S}, j, t) < height(\mathcal{S}, j, s)$   
          $s = t$   
       **endif**  
     **endfor**  
      $start_j = s$   
**endforall**

---

The offline heuristic algorithm schedules individual jobs in the same fashion as the online one, but the jobs that have tight execution windows are scheduled first while jobs with more space in their execution windows are scheduled later. The complete algorithm is presented as Algorithm 3 (note that  $w_j = \frac{l_j}{d_j - a_j} \in (0, 1]$  measures the execution window tightness of job  $j$ ; values closer to 1 indicate tighter jobs).

---

#### Algorithm 3 MinFit-Offline

---

Step 1 Sort jobs by decreasing  $w_j = \frac{l_j}{d_j - a_j}$  values.  
 Step 2 Execute Algorithm 2 on this ordered list of jobs.

---

## 4. Results

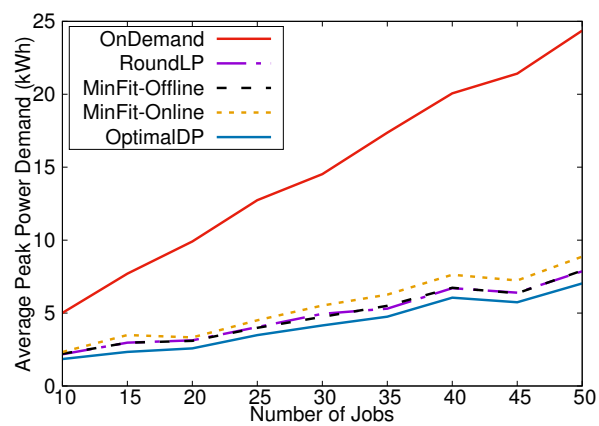
Simulations were conducted using the *OptimalDP*, *RoundLP*, *MinFit-Online*, and *MinFit-Offline* algorithms. For a baseline comparison, we also compare to an on-demand algorithm, *OnDemand*,



that schedules jobs to start on their arrival times. Realistic household power consuming jobs were created using appliance specific data from six residences [16]. We identified appliances (e.g. washing machine) likely to have flexible timelines and determined their height, length, and arrival time distributions within a 24 hour period. Deadlines were set to be uniformly distributed between the minimum possible deadline ( $a_j + l_j - 1$ ) and the arrival time plus four times the average job length of that appliance.

#### 4.1. Performance vs. Optimal

The first scenario we considered was a simple scenario meant to compare our algorithms against an optimal solution. Instead of generating jobs as described above, we randomly generated jobs with an arrival time of 0 to simulate a single peak. Figure 3 shows the average results of running *OptimalDP*, *RoundLP*, *MinFit-Online*, *MinFit-Offline*, and *OnDemand* on 20 iterations of a variable number of jobs, each from this simplified data generation method. *RoundLP*, *MinFit-Online*, and *MinFit-Offline* all produced near-optimal solutions at each job set size, in stark contrast to the solutions provided by *OnDemand*.

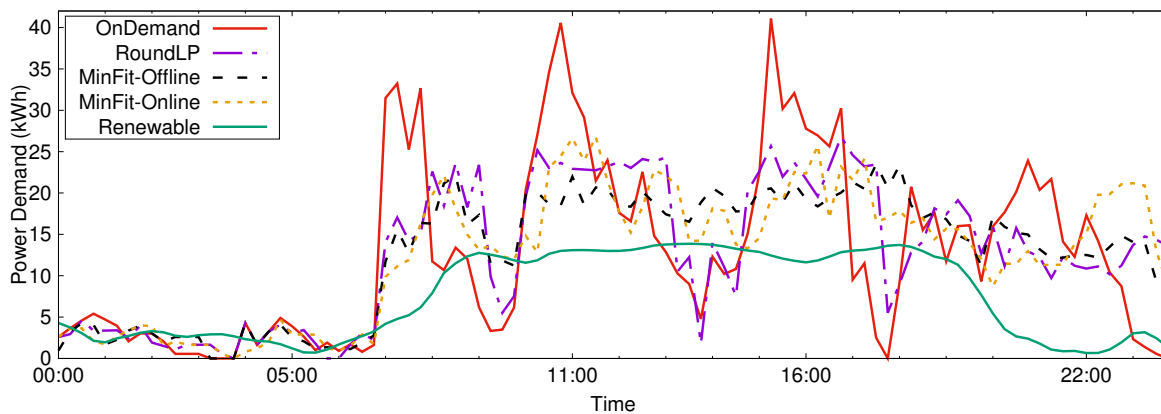


**Figure 3.** Performance of PDM scheduling algorithms in a smaller scenario where the optimal solution can be computed.

#### 4.2. Scheduling with Renewable Energy

The second scenario we considered was the performance of the *RoundLP*, *MinFit-Online*, and *MinFit-Offline* algorithms with a large number of realistically generated jobs as well as the inclusion of available renewable energy. Integrating with renewable energy sources is becoming increasingly important for microgrids and provides an additional mechanism for controlling the non-renewable consumption. It is straightforward to modify all of the algorithms considered so that instead of minimizing overall peak demand, scheduled jobs at time  $t$  are first provisioned using the available renewable energy at that time,  $R(t)$ , and the goal is to minimize the peak non-renewable energy demand of the schedule.

Figure 4 shows the average power demand versus time of day for the *OnDemand*, *RoundLP*, *MinFit-Online*, and *MinFit-Offline* algorithms when scheduling 100 instances of 500 jobs over a 24 hour period, as well as the renewable generation curve, which was found using the National Renewable Energy Laboratory's Wind Integration National Dataset [17]. We sampled locations in the dataset nearest Bozeman, Montana and selected several days with representative wind generation curves. The inclusion of a renewable energy source changes the resulting schedules as it enables the algorithms to shift demand and fill up the "free" energy that the renewable source provides. The peak non-renewable demands in kWh were 28.6 for *OnDemand*, 14.7 for *RoundLP*, 12.9 for *MinFit-Offline*, and 19.1 for *MinFit-Online*.



**Figure 4.** Performance of PDM scheduling algorithms in a larger scenario with available renewable energy. The available renewable energy curve,  $R(t)$ , is shown in green. Total demand over time is plotted for each algorithm.

## 5. Conclusions

In this work we presented the first approximation algorithm for the PDM problem for scheduling non-preemptible jobs, as well as an optimal dynamic programming algorithm that is fixed-parameter tractable and a heuristic approach that can be adapted to the online version of the problem. The problem occurs naturally in scheduling household power consuming jobs in a smart grid power system. Simulation results indicate that there is substantial opportunity to reduce the peak demand through deferred scheduling vs. on-demand. Our final simulation scenario also considered available renewable energy and used slightly modified versions of the presented algorithms to reduce the peak non-renewable energy demand. There are several further research directions to scheduling with renewable energy sources. First, the available renewable energy curve may not be known with certainty and so forecasts must be used. This introduces a random aspect to the performance of any schedule and so one might seek the schedule with the lowest expected non-renewable peak demand, etc. A second consideration is that it is also important to fully utilize all of the available renewable energy and not just minimize the non-renewable peak demand. Solutions of interest would be on the *Pareto front* of minimizing peak non-renewable demand and maximizing renewable energy use. Finally, the online version of PDM for non-preemptible jobs has not been explored in great depth.

**Author Contributions:** S.Y. and B.M. jointly conceived and designed the algorithms; S.Y. performed the experiments.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Bibliography

1. Koutsopoulos, I.; Tassioulas, L. Optimal control policies for power demand scheduling in the smart grid. *IEEE Journal on Selected Areas in Communications* **2012**, *30*, 1049–1060.
2. Fathi, M.; Bevrani, H. Adaptive Energy Consumption Scheduling for Connected Microgrids Under Demand Uncertainty. *IEEE Transactions on Power Delivery* **2013**, *28*, 1576–1583.
3. Yaw, S.; Mumey, B. An Exact Algorithm for Non-preemptive Peak Demand Job Scheduling. In *Combinatorial Optimization and Applications*; Springer International Publishing, 2014; Vol. 8881, *Lecture Notes in Computer Science*, pp. 3–12.
4. Logenthiran, T.; Srinivasan, D.; Shun, T.Z. Demand Side Management in Smart Grid Using Heuristic Optimization. *IEEE Transactions on Smart Grid* **2012**, *3*, 1244–1252.
5. Huang, Q.; Li, X.; Zhao, J.; Wu, D.; Li, X.Y. Social Networking Reduces Peak Power Consumption in Smart Grid. *IEEE Transactions on Smart Grid* **2015**, *6*, 1403–1413.

6. Tang, S.; Huang, Q.; Li, X.Y.; Wu, D. Smoothing the energy consumption: Peak demand reduction in smart grid. *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 1133–1141.
7. Yaw, S.; Mumey, B.; McDonald, E.; Lemke, J. Peak demand scheduling in the Smart Grid. *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, 2014, pp. 770–775.
8. Roh, H.T.; Lee, J.W. Residential Demand Response Scheduling With Multiclass Appliances in the Smart Grid. *IEEE Transactions on Smart Grid* **2016**, *7*, 94–104.
9. Chuzhoy, J.; Guha, S.; Khanna, S.; Naor, J. Machine minimization for scheduling jobs with interval constraints. *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, 2004, pp. 81–90.
10. Cieliebak, M.; Erlebach, T.; Hennecke, F.; Weber, B.; Widmayer, P. Scheduling With Release Times and Deadlines on A Minimum Number of Machines. In *Exploring New Frontiers of Theoretical Informatics*; Levy, J.J.; Mayr, E.; Mitchell, J., Eds.; Springer US, 2004; Vol. 155, *IFIP International Federation for Information Processing*, pp. 209–222.
11. Ortmann, F.G.; Ntene, N.; van Vuuren, J.H. New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. *European Journal of Operational Research* **2010**, *203*, 306–315.
12. Gu, X.; Chen, G.; Xu, Y. Average-Case Performance Analysis of a 2D Strip Packing Algorithm - NFDH. *Journal of Combinatorial Optimization* **2005**, *9*, 19–34.
13. Baker, B.S.; Schwarz, J.S. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing* **1983**, *12*, 508–525.
14. Raghavan, P.; Tompson, C.D. Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs. *Combinatorica* **1987**, *7*, 365–374.
15. Raghavan, P. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Foundations of Computer Science, 1986., 27th Annual Symposium on. IEEE*, 1986, pp. 10–18.
16. Kolter, J.Z.; Johnson, M.J. Redd: A public data set for energy disaggregation research. *SustKDD workshop on Data Mining Applications in Sustainability*, 2011.
17. Draxl, C.; Clifton, A.; Hodge, B.M.; McCaa, J. The Wind Integration National Dataset (WIND) Toolkit. *Applied Energy* **2015**, *151*, 355 – 366.