Energy Peak Shaving with Local Storage [1]

Matthew P. Johnson[a], Amotz Bar-Noy[b], Ou Liu[b], Yi Feng[b]

[a]Department of Computer Science and Engineering, Pennsylvania State University
[b]Department of Computer Science, City University of New York Graduate Center

Abstract

We introduce a new problem inspired by energy pricing schemes in which a client is billed for peak usage. At each timeslot the system meets an energy demand through a combination of a new request, an unreliable amount of *free source* energy (e.g. solar or wind power), and previously received energy. The added piece of infrastructure is the *battery*, which can store surplus energy for future use, and is initially assumed to be perfectly efficient or *lossless*. In a feasible solution, each demand must be supplied on time, through a combination of newly requested energy, energy withdrawn from the battery, and free source. The goal is to minimize the maximum request. In the online version of this problem, the algorithm must determine each request without knowledge of future demands or free source availability, with the goal of maximizing the amount by which the peak is reduced. We give efficient optimal algorithms for the offline problem, with and without a bounded battery. We also show how to find the optimal offline battery size, given the requirement that the final battery level equals the initial battery level. Finally, we give efficient H_n-competitive algorithms assuming the peak *effective* demand is revealed in advance, and provide matching lower bounds.

Later, we consider the setting of *lossy* batteries, which lose to conversion inefficiency a constant fraction of any amount charged (e.g. 33%). We efficiently adapt our algorithms to this setting, maintaining optimality for offline and (we conjecture) maintaining competitiveness for online. We give *factor-revealing* LPs, which provide some quasi-empirical evidence for competitiveness. Finally, we evaluate these and other, heuristic algorithms on real and synthetic data.

Keywords: online algorithms, competitive analysis, energy, peak shaving, simulation

[1] A preliminary version of this work was presented in [1, 2].

1. Introduction

There is increasing interest in saving fuel costs by use of renewable energy sources such as wind and solar power. Although such sources are highly desirable, and the power they provide is in a sense free, the typical disadvantage is unreliability: availability depends e.g. on weather conditions (it is not "dispatchable" on demand). Many companies seek to build efficient systems to gather such energy when available and store it, perhaps in modified form, for future use [3].

On the other hand, power companies charge some high-consumption clients not just for the total amount of power consumed, but also for how quickly they consume it. Within the billing period (typically a month), the client is charged for the amount of energy used (*usage charge*, in kWh) and for the maximum amount requested over time (*peak charge*, in kW).[2] If demands are given as a sequence $(d_1, d_2, \ldots, d_n)$, then the total bill is of the form $c_1 \sum_i d_i + c_2 \max_i\{d_i\}$ (for some constants $c_1, c_2 > 0$), i.e., a weighted sum of the total usage and the maximum usage. (In practice, the discrete timeslots may be 30-minute averages [4].) This means that a client who powers a 100kW piece of machinery for one hour and then uses no more energy for the rest of the month would be charged more than a client who uses a total of 100kWh spread evenly over the course of the month. Since the per-unit cost for peak charges may be on the order of 100 times the per-unit cost for total usage [5],[3] this difference can be significant. Indeed, this is borne out in our experiments.

This suggests a potential financial incentive to storing *purchased* energy for future use. Indeed, at least one start-up company has marketed such a battery-based system intended to reduce peak energy charges. In such a system, a battery is placed between the power company and a high-consumption client site, (such as a large office building or factory) in order to smooth power requests and shave the peak. The client site will charge to the battery when demand is low and discharge when demand is high. Spikes in the demand curve can thus be rendered consistent with a relatively flat level of supplied power. The result is a lower cost for the client and a more manageable request curve for the provider. (The usage charge in the billing formula above is not optimized for, since in our model all demands must be satisfied.)

It is interesting to note that a battery system may actually *raise* energy usage,

[2]In fact, some billing models are more complex.

[3]The Orlando Utilities Commission website [5], for example, quotes rates of 6.388 cents per kWh ("energy charge") and $6.50 per kW ("demand charge").

since there may be energy loss due to inefficiency in AC/DC conversion. This loss may be as much as 33% of the amount charged. Serving peak requests during periods of high demand is a difficult and expensive task for the power company, however, and the event of a black-out inflicts high societal costs. While a battery system may involve higher total energy requests, it may benefit the system as a whole by easing the strain of peak demands. Combined with alternative energy sources such as solar panels, the system could even lower the net commercial power usage. Alternative energy sources are typically low-cost but unreliable, since they depend on external events such as the weather. With a battery, this energy can be stored until needed.

We may generalize this problem of minimaxing the request to any resource which is *tenable* in the sense that it may be obtained early and stored until needed. For example, companies frequently face shortages of popular products: "Plentiful supply [of Xboxes] would be possible only if Microsoft made millions of consoles in advance and stored them without releasing them, or if it built vast production lines that only ran for a few weeks–both economically unwise strategies," a recent news story asserted [6]. A producer could smooth the product production curve by increasing production and warehousing supply until future sales. But when should the producer "charge" and "discharge"? (In some domains, there may also be an unpredictable level of volunteer help.) A third application is the scheduling of jobs composed of generic work-units that may be done in advance. Although the problem is very general, we will use the language of energy and batteries for concreteness. Many features of this production problem, including uncertainty in future demand, a bounded warehouse size and a cost for storage in the warehouse have analogs in the battery problem.

In the online version of our problem, the essential choice faced at each timeslot is whether (and by how much) to invest in the future or to cash in a prior investment. The investment in our setting is a request for more energy than is needed at the time. If the algorithm only asks for the minimum required, then it is vulnerable to spikes in demand; if it asks for much more energy than it needs, then the greater request could itself introduce a new, higher peak. The strictness of the problem lies in the fact that the cost is not cumulative: we want *every* request to be low. The offline version is solvable in polynomial time, both without and (albeit by a more complicated solution) without battery loss. Our online algorithms will be based on the optimal offline algorithms. Central to our analysis for the lossy battery settings is a mathematical function we call a *generalized average*. Our fastest offline algorithms in these settings compute a series of generalized averages with the aid of balanced binary search trees. We also show how to find the optimal offline battery size, given the requirement that the final battery level equals the initial battery level.

1.1. Contributions

We introduce a novel scheduling problem and solve several versions optimally with efficient combinatorial algorithms. We solve the offline problem for two kinds of batteries: unbounded battery in $O(n)$ time and bounded in $O(n^2)$, where n is the number of timeslots. Separately, we show how to find the optimal offline battery size, for the setting in which the final battery level must equal the initial battery level. This is the smallest battery size that achieves the optimal peak.

The online problem we study is very strict. A meta-strategy in many online problems is to balance expensive periods with cheap ones, so that the overall cost stays low [7]. The difficulty in our problem lies in its noncumulative nature: we optimize for the max, not for the average. We show that several versions of the online problem have no algorithm with nontrivial competitive ratio (i.e., better than n or $\Omega(\sqrt{n})$). Given advanced knowledge of the peak demand D, however, we give H_n-competitive algorithms for batteries bounded and unbounded. Our fastest algorithm has $O(1)$ per-slot running-time. H_n is the (optimal) competitive ratio for both battery settings.

The algorithms mentioned so far assume perfectly efficient batteries, however, and will fail if run on realistic, lossy batteries. Therefore we adapt these algorithms to the lossy setting, testing them on both synthetic and actual customer usage data. Moreover, we test more aggressive, heuristic algorithms, as well as algorithms that accept predictions, with error, of future demands. Finally, we provide factor-revealing linear programs (LPs), which provide quasi-empirical evidence of the competitiveness of the lossy algorithms.

Examples. Although there is no constant-ratio competitive algorithm for unbounded n, our intended application in fact presumes a fixed time-horizon. If the billing period is one month, and peak charges are computed as 30-minute averages, then for this setting H_n is approximately 7.84. If we assume that the battery can fully recharge at night, so that each day can be treated as a separate time period, then for a 12-hour daytime time-horizon H_n is approximately 3.76.

1.2. Related Work

There is a wide literature on commodity production, storage, warehousing, and supply-chain management (see e.g. [8, 9, 10, 11]). More specifically, there are a number of inventory problems based on the Economic Lot Sizing model [12], in which demand levels for a product vary over a discrete finite time-horizon and are known in advance. A feasible solution in these problems must obtain sufficient supply through production (sometimes construed as ordering) or through other methods, in order to meet each of the demands on time, while observing certain constraints. The nature of solution quality varies by formulation.

One such inventory problem is Single-Item Lot-Sizing, in which sufficient supplies must be ordered to satisfy each demand, while minimizing the total cost of ordering charges and holding charges. The ordering charge consists of a fixed charge per order plus a charge linear in order size. The holding charge for inventory is per-unit and per-timeslot. There is a tradeoff between these incentives since fixed ordering charges encourage large orders while holding charges discourage them. Wagner & Whitin [13] showed in 1958 that this problem can be solved in polynomial time. Under the assumption of *non-speculative costs*, in which case orders should always be placed as late as possible, the problem can be solved in linear time. Such "speculative" behavior, however, is the very motivation of our problem. There are many lot-sizing variations, including constant-capacity models that limit the amount ordered per timeslot. (See [11] and references therein.) Our offline problem differs in that our objective is minimizing this constant capacity (for orders), subject to a bound on *inventory* size, and we have no inventory charge.

Another related inventory problem is Capacity and Subcontracting with Inventory (CSI) [14], which incorporates trade-offs between production costs, subcontracting costs, holding costs, and the cost for maximum per-unit-timeslot production capacity. The goal in that problem is to choose a production capacity and a feasible production/ subcontracting schedule that together minimize total cost, whereas in our problem choosing a production capacity, subject to storage constraints, is the essential task.

In the minimax work-scheduling problem [15], the goal is to minimize the maximum amount of work done in any timeslot over a finite time-horizon. Our online problem is related to a previously studied special case in which jobs with deadlines are assigned online. In that problem, all work must be done by deadline but cannot be begun until assigned. Subject to these restrictions, the goal is to minimize the maximum work done in any timeslot. While the optimization goal is the same, our online problem differs in two respects. First, each job for us is due immediately when assigned. Second, we *are* allowed to do work (request and store energy) in advance. One online algorithm for the jobs-by-deadlines problem is the α-*policy* [15]: at each timeslot, the amount of work done is α *times the maximum per-unit-timeslot amount of work that OPT would have done, when* *running on the partial input received so far*. Our online algorithms adopt a related strategy.

2. Lossless Model and Preliminaries

Definition 2.1. *At each timeslot i for $i = 1, ..., n$, d_i is the demand, r_i is the request, b_i is the battery charge level* at the start *of the timeslot, and f_i is the amount of free source available. (b_{n+1} is the battery level following timeslot n.) By $\hat{d}_i$ we indicate the* effective demand $d_i - f_i$. *We sometimes refer to the sequence over time of one of these value types as a curve, e.g., the* demand curve. *D is the maximum effective demand* $\max_i\{\hat{d}_i\}$, *and R is the maximum request* $\max_i\{r_i\}$.

The problem instance comprises the demands, the free source curve, battery size B, initial charge b_1, and required final charge b_{n+1} (in the offline case). The problem solution consists of the request curve.

Definition 2.2. *Let* overflow *be the situation in which $r_i + f_i - d_i > B - b_i$, i.e., there is not enough room in the battery for the amount we want to charge. Let* underflow *be the situation in which $d_i - r_i - f_i > b_i$, i.e., there is not enough energy in the battery for the amount we want to discharge. Call an algorithm* feasible *if underflow never occurs.*

The goal of the problem is to minimize R (for competitiveness measures this is construed as maximizing $D - R$) while maintaining feasibility. In the absence of overflow/underflow, the battery level at *the start of* timeslot i is simply $b_i = b_{i-1} + r_{i-1} + f_{i-1} - d_{i-1}$. It is forbidden for b_i to ever fall below 0. That is, the request r_i, the free source f_i, and the battery level b_i must sum to at least the demand d_i at each timeslot i. Notice that effective demand can be negative, which means that the battery may be charged (capacity allowing), even if the request is 0. We assume D, however, is strictly positive. Otherwise, the problem instance is essentially trivial. We use the following to simplify the problem statement:

Observation 2.1. *If effective demands may be negative, then free source energy need not be explicitly considered.*

As such, we set aside the notion of free source, and for the remainder of the paper (simplifying notation) allow demand d_i to be negative.

In the energy application, battery capacity is measured in kWh, while instantaneous request is measured in kW. By discretizing we assume wlog that battery level, demand, and request values are expressed in common units. Peak charges are based linearly on the max request, which is what we optimize for. The battery can have a *maximum capacity* B or be unbounded. The problem may be online, offline, or in between; we consider the setting in which the peak demand D is revealed in advance, perhaps predicted from historical information.

Threshold algorithms. For a particular snapshot (d_i, r_i, b_i), demand d_i must be supplied through a combination of the request r_i and a change in battery $b_{i+1} - b_i$. This means that there are only three possible modes for each timeslot: request exactly the effective demand, request more than this and *charge* the difference, or request less and *discharge* the difference. We refer to our algorithms as *threshold algorithms*. Let $T_1, T_2, ..., T_n$ be a sequence of values. Then the following algorithm uses these as request thresholds:

for each timeslot i
  if $d_i < T_i$
    charge $\min(B - b_i, T_i - d_i)$
  else
    discharge $d_i - T_i$

Intuitively, the algorithm amounts to this rule: *at each timeslot i, request an amount as near to T_i as the battery constraints will allow*. Our offline algorithms are *constant threshold* algorithms, with a fixed T (though in practice an offline algorithm could naturally lower its requests to avoid overflow); our online algorithms compute T_i dynamically for each timeslot i.

A constant-threshold algorithm is specifiable by a single number. In the online setting, predicting the *exact* optimal threshold from historical data suffices to solve the online problem optimally. A small overestimate of the threshold will merely raise the peak cost correspondingly higher. Unfortunately, however, examples can be constructed in which even a small *underestimate* eventually depletes the battery before peak demand and thus produce no cost-savings at all.

The offline problem can be solved approximately, within additive error ϵ, through binary search for the minimum feasible constant threshold value T. Simply search the range $[0, D]$ for the largest value T for which the threshold algorithm has no underflow, in time $O(n \log \frac{D}{\epsilon})$. If the optimal peak reduction is $R - T$, then the algorithm's peak reduction will be at least $R - T - \epsilon$. It is straightforward to give a linear programming formulation of the *offline* problem; it can also be solved by generalized parametric max-flow [16]. Our interest here, however, is in efficient combinatorial optimal algorithms. Indeed, our combinatorial offline algorithms are significantly faster than these general techniques and lead naturally to our competitive online algorithms. Online algorithms based on such general techniques would be intractable for fine-grain timeslots.

3. Lossless Offline Problem

We now find optimal algorithms for both lossless battery settings. For unbounded, we assume the battery starts empty; for bounded, we assume the battery starts with amount B. For both, the final battery level is unspecified. We show below that these assumptions are made wlog. The two offline threshold functions, shown in Table 1, use the following definition.

Definition 3.1. *Let* $\rho(j) = \frac{1}{j}\sum_{t=1}^{j} d_t$ *be the mean demand of the prefix region* $[1, j]$*, and let* $\hat{\rho}(k) = max_{1\leq j\leq k}\rho(j)$ *be the maximum mean among the prefix regions up to* k*. Let* $\mu(i,j) = \frac{-B+\sum_{t=i}^{j} d_t}{j-i+1}$ *be the* density *of the region* $[i, j]$ *and* $\hat{\mu}(k) = max_{1\leq i\leq j\leq k}\mu(i,j)$ *be the maximum density among all subregions of* $[1, k]$.

Alg.	battery	threshold T_i	running time
1.a	unbounded	$\hat{\rho}(n)$	$O(n)$
1.b	bounded	$\hat{\mu}(n)$	$O(n^2)$

Table 1: Threshold functions used for offline algorithm settings.

Bounded capacity changes the character of the offline problem. It suffices, however, to find the peak request made by the optimal algorithm, R_{opt}. Clearly $R_{opt} \geq D - B$, since the ideal case is that a width-one peak is reduced by size B. Of course, the peak region might be wider.

Theorem 3.1. *Algorithm* 1.a *(threshold* $T_i = \hat{\rho}(n)$*, for unbounded battery) and Algorithm* 1.b *(threshold* $T_i = \hat{\mu}(n)$*, for bounded battery) are optimal, feasible, and run in times* $O(n)$ *and* $O(n^2)$*, respectively.*

Proof. First, let the battery be unbounded. For any region $[1, j]$, the best we can hope for is that requests for all demands $d_1, ..., d_j$ can be spread evenly over the first j timeslots. Therefore the optimal threshold cannot be lower than the maximum $\rho(j)$, which is Algorithm *1.a*'s threshold. For feasibility, it suffices to show that after each time j, the battery level is nonnegative. But by time j, the total input to the system will be $j \cdot \hat{\rho}(n) \geq j \cdot \rho(j) = \sum_{t=1}^{j} d_j$, which is the total output to the system up to that point. For complexity, just note that $\rho(j+1) = \frac{j \cdot \rho(j) + d_{j+1}}{j+1}$, so the sequence of ρ values, and their max, can be computed in linear time.

Now let the battery be bounded. Over the course of any region $[i, j]$, the best that can be hoped for is that the peak request will be reduced to $B/(j-i+1)$ less than the average d_i in the region, i.e., $\mu(i, j)$, so no threshold lower than Algorithm *1.b*'s is possible.

For feasibility, it suffices to show that the battery level will be nonnegative after each time j. Suppose j is the first time underflow occurs. Let $i-1$ be the last timeslot prior to j with a full battery. Then there is no underflow *or* overflow in $[i, j)$, and so for each $t \in [i, j]$ the discharge at t is $b_t - b_{t+1} = d_t - T$ (possibly negative, meaning a charge) and so the total net discharge over $[i, j]$ is $\sum_{t=i}^{j} d_t - (j-i+1)T$. Total net discharge greater than B implies $T < \frac{-B + \sum_{t=i}^{j} d_t}{j-i+1}$, which contradicts the definition of T. The densest region can be found in $O(n^2)$, with n separate linear-time passes, each of which finds the densest region beginning in some position i, since $\mu(i, j+1)$ can be computed in constant time from $\mu(i, j)$. □

3.1. Battery Level Boundary Conditions

We assumed above that the battery starts empty for the unbounded offline algorithm and starts full for the bounded offline algorithm, with the final battery level left indeterminate for both settings. A more general offline problem may require that $b_1 = \beta_1$ and $b_{n+1} = \beta_2$, i.e., the battery begins and ends at some charge levels specified by parameters β_1 and β_2. We argue here that these requirements are not significant algorithmically, since by pre- and postprocessing, we can reduce to the default cases for both the unbounded and bounded versions.

First, consider the unbounded setting, in which the initial battery level is 0. In order to enforce that $b_1 = \beta_1$ and $b_{n+1} = \beta_2$, run the usual optimal algorithm on the sequence $(d_1 - \beta_1, d_2, ..., d_{n-1}, d_n + \beta_2)$. (Recall that negative demands are allowed.) Then b_{n+1} will be at least β_2 larger than d_n. To correct for any surplus, manually delete a total of $b_{n+1} - \beta_2$ from the final requests. For the bounded setting, the default case is $b_1 = B$ and b_{n+1} indeterminate. To support $b_1 = \beta_1 \neq B$ and $b_{n+1} = \beta_2$, modify the demand sequence as above, except with $d_1 + (B - \beta_1)$ as the first demand and then do similar postprocessing as in the unbounded case to deal with any final surplus.

3.2. Optimal Battery Size

A large component of the fixed initial cost of the system will depend on battery capacity. A related problem therefore is finding the optimal battery size B for a given demand curve d_i, given that the battery starts and ends at the same level β (which can be seen as an amount *borrowed and repaid*). The optimal peak request possible will be $\frac{1}{n}\sum_{i=1}^{n} d_i = \rho(n)$, and the goal is to find the smallest B and β that achieve peak $\rho(n)$. (A completely flat request curve is possible given a sufficiently large battery.) This can be done in $O(n)$.

Since we will have $b_1 = b_{n+1}$, $r_i = \rho$ for all i, and $\sum d_i = \sum r_i$, there must be no overflow. Let $d'_i = d_i - \rho$, i.e., the amount by which d_i is above average (positive means discharge, negative means charge). Then the minimum possible $\beta = b_1$ is the maximum prefix sum of the d' curve (which will be at least 0). It could happen that the battery level will at some point rise above b_1, however. (Consider the example $d = (0, 0, 1, 0, 0, 0)$, for which $\rho = 1/6$, $d' = (-1/6, -1/6, 5/6, -1/6, -1/6, -1/6)$ and $\beta = 1/2$.) The needed capacity B can be computed as β plus the maximum prefix sum of *the negation of* the d' curve (which will also be at least 0). (In the example, we have $B = \beta + 2/6 = 5/6$.)

Although B is computed using β, we emphasize that the computed β is the minimum possible *regardless of* B, and the computed B is the minimum possible *regardless of* β.

4. Lossless Online Problem

We consider two natural choices of objective function for the online problem. One option is to compare the peak requests, so that if ALG is the peak request of the online algorithm ALG and OPT is that of the optimal offline algorithm OPT, then a c-competitive algorithm for $c \geq 1$ must satisfy $\frac{ALG}{OPT} \leq c$ for every demand sequence. Although this may be the most natural candidate, we argue that for many settings it is uninteresting. If the peak demand is a factor k larger than the battery capacity, for example, then the trivial online algorithm that *does not use* the battery would be $k/(k-1)$-competitive. If we drop the assumption of a small battery, then no reasonable competitive factor is possible in general, *even if* D *is revealed in advance*.

Proposition 4.1. *With peak demand* D revealed in advance *and finite time horizon* n*, no online algorithm* for the problem of minimizing peak request *can have* *competitive ratio 1) better than n if the battery begins with some strictly positive charge, or 2) better than $\Omega(\sqrt{n})$ if the battery begins empty.*[4]

Proof. For part 1, suppose that the battery begins with initial charge D. Consider the demand curve $(D, 0, ..., 0, ?)$, where the last demand is either D or 0. ALG *must* discharge D at time 1, since $OPT = 0$ when $d_n = 0$. Thus ALG's battery is empty at time 2. If ALG requests nothing between times 2 and $n-1$, and $d_n = D$, then we have $OPT = D/n$ and $ALG = D$; if ALG requests some $\alpha > 0$ during any of those timeslots, and $d_n = 0$, then we have $OPT = 0$ and $ALG = \alpha$. This yields a lower bound of n.

For part 2, suppose the battery begins empty, which is a disadvantage for both ALG and OPT. Consider the demand curve $(0, 0, ..., 0, D)$, in which case $OPT = D/n$. If an algorithm is c-competitive for some $c \geq 1$, then in each of the first $n-1$ timeslots of this demand curve ALG can charge at most amount cD/n. Now suppose that the only nonzero demand, of value D, arrives possibly earlier, at some timeslot $k \in [1, n]$, following $k-1$ demands of zero, during which ALG can charge at most $(k-1)cD/n$. In this case, we have $OPT = D/k$ and $ALG \geq D - (k-1)cD/n$, which yields the competitive ratio:

$$c \geq \frac{D - (k-1)cD/n}{D/k} = \frac{1 - (k-1)c/n}{1/k} = k - k^2c/n + kc/n$$

Solving for c, and then choosing $k = \sqrt{n}$, we have:

$$c \geq \frac{k}{1 + k^2/n - k/n} = \frac{\sqrt{n}}{1 + 1 - 1/\sqrt{n}} = \Omega(\sqrt{n})$$

thus establishing the lower bound. □

Recalling the example values of n given in the Introduction, ratios of either n or $\sqrt{n}$ would be very weak. Instead, we compare the *peak shaving amount* (or *savings*), i.e., $D - R$. For a given input, let *OPT* be the peak shaving of the optimal algorithm, and let *ALG* be the peak shaving of the online algorithm. Then an online algorithm is c-competitive for $c \geq 1$ if $c \geq \frac{OPT}{ALG}$ for every problem instance. For this setting, we obtain the online algorithms (see Def. 4.1 and the proof of Corollary 4.7) shown in Table 2.

[4]If the peak is not known, a lower bound of n can be obtained also for the latter case.

Alg.	battery	threshold T_i	running time
2.a	both	$D - \frac{D-T_i^{opt}}{H_n}$	$O(n^2)$
2.b	both	$D - \frac{D-\mu(s_i,i)}{H_{n-s_i+1}}$	$O(n)$

Table 2: Threshold functions used for online algorithms.

Definition 4.1. *Let T_i^{opt} be the optimal threshold used by the appropriate optimal algorithm when run on the first i timeslots. At time i during the online computation, let s_i be the index of the most recent time prior to i with $b_{s_i} = B$ (or 1 in the unbounded setting).*

4.1. Lower Bounds for $D - R$

Since the competitiveness of the online algorithms holds for arbitrary initial battery level, in obtaining lower bounds on competitiveness, we assume particular initial battery levels.

Proposition 4.2. *With peak demand D unknown and finite time horizon n, there is no online algorithm 1) with any constant competitive ratio for unbounded battery (even with $n = 2$) or 2) with competitive ratio better than n for bounded battery.*

Proof. For part 1, assume $b_1 = 0$, and suppose $d_1 = 0$. Then if ALG requests $r_1 = 0$ and we have $d_2 = D$, then $OPT = D/2$ and $ALG = 0$; if ALG requests $r_1 = a$ (for some $a > 0$) and we have $d_2 = a$, then $OPT = a/2$ and $ALG = 0$. For part 2, let $b_1 = B$, and assume ALG is c-competitive. Consider the demand curve $(B, 0, 0, \ldots, 0)$. Then OPT clearly discharges B at time 1 (decreasing the peak by B). For ALG to be c-competitive, it must discharge at least $\frac{B}{c}$ in the first slot. Now consider curve $(B, 2B, 0, 0, \ldots, 0)$. At time 2, OPT discharges B, decreasing the peak by B, so at time 2, ALG must discharge at least $\frac{B}{c}$. (At time 1, ALG already had to discharge $\frac{B}{c}$.) Similarly, at time i for $(B, 2B, 3B, ..., iB, ..., nB)$, ALG must discharge $\frac{B}{c}$. Total discharging by ALG is then at least: $\sum_{i=1}^{n} \frac{B}{c} = \frac{nB}{c}$. Since we must have $n\frac{B}{c} \leq B$, it follows that $c \geq n$. □

The trivial algorithm that simply discharges amount B/n at each of the n timeslots and *never charges* is n-competitive (since $OPT \leq B$) and so matches the lower bound for the bounded case.

Proposition 4.3. *With peak demand D known in advance and finite time horizon n, no online algorithm can have 1) competitive ratio better than H_n if the battery* *begins nonempty or 2) competitive ratio better than $H_n - 1/2$ if the battery begins empty, regardless of whether the battery is bounded or not.*

Proof. First assume the battery has initial charge b. (The capacity is either at least b or unbounded.) Suppose ALG is c-competitive. Consider the curve $(D, 0, 0, \ldots, 0)$, with $D \geq b$. Then OPT clearly discharges b at time 1 (decreasing the peak by b). For ALG to be c-competitive, it must discharge at least $\frac{b}{c}$. Now consider curve $(D, D, 0, 0, \ldots, 0)$. At times 1 and 2, OPT discharges $\frac{b}{2}$, decreasing the peak by $\frac{b}{2}$. At time 2, ALG will have to discharge at least $\frac{b/2}{c} = \frac{b}{2c}$. Similarly, at time i on $(D, D, D, ..., D)$, ALG must discharge $\frac{b}{ic}$. Total discharging by ALG is then at least: $\sum_{i=1}^{n} \frac{b}{ic} = H_n \frac{b}{c}$. Since we discharge at each timeslot and never charge, we must have $\frac{b}{c} H_n \leq b$, and so it follows that $c \geq H_n$.

Now let the battery start empty. Assume the battery capacity is at least D or is unbounded. Repeat the argument as above, except now with a zero demand inserted at the start of the demand curves, which gives both ALG and OPT an opportunity to charge. Then for each time $i \in [2, n]$, ALG must discharge at least $\frac{D}{ic}$ since OPT may discharge (and so save) $\frac{D}{i}$ (in which case it would have initially charged $D(1 - 1/i)$). ALG is then required to discharge $(H_n - 1)\frac{D}{c}$ during the last $n - 1$ timeslots. Obviously it could not have charged more than D during the first timeslot. In fact, it must charge less than this. On the sequence $(0, D, 0, 0, ..., 0)$, OPT charges $D/2$ at time 1 and discharges it at time 2, saving $D/2$. ALG must discharge $\frac{D}{2c}$ at time 2 in order to be c-competitive on this sequence, and so reduce the peak D by $\frac{D}{2c}$. Therefore at time 1, ALG cannot charge more than $D - \frac{D}{2c}$. Therefore we must have $D - \frac{D}{2c} \geq (H_n - 1)D/c$, which implies that $c \geq H_n - 1/2$. □

4.2. Bounded Battery

Our first online algorithm bases its threshold at time i on a computation of the optimal offline threshold T_i^{opt} for the demands $d_1, ..., d_i$. The second bases its threshold at time i on $\mu(s_i, i)$ (see Defs. 3.1 and 4.1). Assuming the algorithms are feasible (i.e., no battery underflow occurs), it is not difficult to show that they are competitive.

Theorem 4.4. *Algorithms* 2.a *and* 2.b *are H_n-competitive, if they are feasible, and have* per-timeslot *running times of $O(n)$ and $O(1)$, respectively.*

Proof. First observe that $\hat{\mu}(i) \geq \mu(s_i, i)$ implies $\frac{D - \hat{\mu}(i)}{H_n} \leq \frac{D - \mu(s_i, i)}{H_{n-s_i+1}}$ implies $T_i^a \geq T_i^b$ for all i. Therefore it suffices to prove competitiveness for Algorithm *2.a*. Since

T_i^{opt} is the lowest possible threshold up to time i, $D - T_i^{opt}$ is the highest possible peak shaving as of time i. Since the algorithm always saves a $1/H_n$ fraction of this, it is H_n-competitive by construction.

Since $\rho(1, i+1)$ can be found in constant time from $\rho(1, i)$, Algorithm *2.b* runs in constant-time per-slot. Similarly, Algorithm *2.a* is, recalling the proof of Theorem 3.1, linear per-slot. □

We now show that indeed both algorithms are feasible, using the following lemma, which allows us to limit our attention to a certain family of demand sequences.

Lemma 4.5. *If there is a demand sequence* $(d_1, d_2, ..., d_n)$ *in which underflow occurs for Algorithm* 2.a *or* 2.b*, then there is also a (possibly shorter) demand sequence (for the same algorithm) of length* $n' \leq n$ *in which underflow continues to the end (i.e.,* $b_{n'+1} < 0$*) and no overflow ever occurs.*

Proof. The battery is initialized to full, $b_1 = B$. Over the course of running one of the algorithms on a particular problem instance, the battery level will fall and rise, and may return to full charge multiple times. Suppose underflow were to occur at some time t, i.e. $b_t < 0$, and let s be the most recent time before t when the battery was full. We now construct a demand sequence with the two desired properties, for both algorithms.

First, suppose $s > 1$. For Algorithm *2.a*, consider the demand sequence (of length $n' = n$) obtained by copying the demands of region $[s, t]$ leftward to region $[1, t'] = [1, t-s+1]$. Since the optimal solution for region $[s, t]$ can be only lower than for region $[1, t]$, this change can only lower the thresholds used, which therefore preserves the underflow. For Algorithm *2.b*, consider the demand sequence (of length $n' = n - s + 1$) obtained by deleting the region $[1, s-1]$. Since $\mu(s, t)$ is unchanged and $H_{n'-s'+1} = H_{n-s+1}$ (where now $s' = 1$), the thresholds do not change and underflow again is preserved.

Second, since any underflow that occurs in region $[1, t']$ can be extended to the end of sequence by setting each demand after time t' to D, we can assume wlog that $t' = n' + 1$. □

First, suppose $s > 1$ and consider the demand sequence $[s, n]$ of length $n' = n - s + 1$. For Algorithm 2.a, the optimal solution for region $[s, t]$ can be only lower than for region $[1, t]$; for Algorithm 2.b, it is unchanged, since $\mu(s, t) = \mu(1, t - s + 1)$ and $H_{n-s+1} = H_{n'-s'+1}$, where $s' = 1$. Therefore deleting the region $[1, s-1]$ can only lower the thresholds used, which therefore preserves the underflow. Second, since any underflow that occurs in region $[1, t']$ can be extended to the end of sequence by setting each demand after time t' to D, we can assume wlog that $t' = n' + 1$.

Theorem 4.6. *Algorithms* 2.a *and* 2.b *are feasible.*

Proof. For a proof by contradiction, we can restrict ourselves by Lemma 4.5 to regions that begin with a full battery, underflow at the end, and have no overflow in the middle. For such a region, the change in battery-level is well behaved ($b_i - b_{i+1} = d_i - T_i$), which allows us to sum the net discharge and prove it is bounded by B. We now show that it is impossible for the battery to fall below 0 at time n, by upperbounding the *net discharge* over this region. Let $\Delta b_i = b_i - b_{i+1} = d_i - T_i$ be the amount of energy the battery discharges at step i. (Δb_i will be negative when the battery charges.) We will show that $\sum_{1 \le i \le n} \Delta b_i \le B$. Let Δb_i^a and Δb_i^b refer to the change in battery levels for the corresponding algorithms. Because as we observed above $T_i^a \ge T_i^b$, we have:

$$\Delta b_i^a = d_i - T_i^a \le \Delta b_i^b = d_i - T_i^b$$

Therefore it suffices to prove the feasibility result for Algorithm *2.b*, and so we drop the superscripts. Expanding the definition of that algorithm's threshold, we have:

$$\Delta b_i = d_i - T_i = d_i - \left(D - \frac{1}{H_n}(D - \mu(1,i))\right) = d_i - \left(D - \frac{1}{H_n}\left(D - \frac{1}{i}\left(\sum_{k=1}^{i} d_k - B\right)\right)\right) \tag{4.1}$$

By summing Eq. 1 for each i, we obtain:

$$\begin{aligned}
\sum_{i=1}^{n} \Delta b_i &= \sum_{i=1}^{n} \left(d_i - \left(D - \frac{D - (\sum_{k=1}^{i} d_k - B)/i}{H_n}\right)\right) \\
&= \sum_{i=1}^{n} \left(d_i - \left(D - \frac{D - (\sum_{k=1}^{i} d_k)/i}{H_n}\right)\right) + \sum_{i=1}^{n} \frac{B/i}{H_n} \\
&= \sum_{i=1}^{n} \left(d_i - \left(D - \frac{D - (\sum_{k=1}^{i} d_k)/i}{H_n}\right)\right) + B
\end{aligned}$$

Therefore it suffices to show that:

$$\sum_{i=1}^{n} \left(d_i - \left(D - \frac{D - \sum_{k=1}^{i} d_k/i}{H_n}\right)\right) \le 0 \tag{4.2}$$

which is equivalent to:

$$\sum_{i=1}^{n} d_i - nD + \frac{1}{H_n}\Big(nD - \sum_{i=1}^{n}\sum_{k=1}^{i} d_k/i\Big) \le 0$$

$$\Leftrightarrow \sum_{i=1}^{n} d_i + nD(\frac{1}{H_n} - 1) - \frac{1}{H_n}\Big(\sum_{i=1}^{n}\sum_{k=1}^{i} d_k/i\Big) \le 0$$

$$\Leftrightarrow \sum_{i=1}^{n} H_n d_i - \Big(\sum_{i=1}^{n}\sum_{k=1}^{i} \frac{d_k}{i}\Big) \le nD(H_n - 1) \tag{4.3}$$

With the following derivation:

$$\begin{aligned}\sum_{i=1}^{n}\sum_{k=1}^{i}\frac{d_k}{i} &= \sum_{i=1}^{n}\sum_{k=1}^{n}\frac{d_k}{i} - \sum_{i=1}^{n}\sum_{k=i+1}^{n}\frac{d_k}{i} = \sum_{k=1}^{n}\sum_{i=1}^{n}\frac{d_k}{i} - \sum_{k=1}^{n}\sum_{i=1}^{k-1}\frac{d_k}{i} \\ &= \sum_{k=1}^{n} H_n d_k - \sum_{k=1}^{n} H_{k-1} d_k\end{aligned}$$

we can rewrite Eq. 4.3 (replacing the parenthesized expression) as:

$$\sum_{i=1}^{n} H_{i-1} d_i \le n(H_n - 1)D \tag{4.4}$$

Since $d_i \le D$ and $D > 0$, it suffices to show that:

$$\sum_{i=1}^{n} H_{i-1} \le n(H_n - 1)$$

In fact, this holds with equality (see [17], Eq. 2.36). □

4.3. Unbounded Battery and Boundary Conditions

Both online algorithms, modified to call appropriate subroutines, also work for the unbounded battery setting. The algorithms are feasible in this setting since $\hat{\mu}(i) \le T_i^{opt}$ still holds, where T_i^{opt} is now the optimal threshold for the unbounded battery setting. (Recall that offline Algorithm *1.a* can "greedily" run in linear total time.) The algorithm is H_n-competitive by construction, as before.

Corollary 4.7. *Algorithms* 2.a *and* 2.b *are feasible in the unbounded battery setting.*

Proof. The proof is similar to that of Theorem 4.6, except that b_1 (which may be 0) is plugged in for all occurrences of B (resulting in a modified μ), and overflow is no longer a concern. $\square$

We also note that the correctness and competitiveness of both algorithms (with minor preprocessing) holds for the setting of arbitrary initial battery levels. In the case of Algorithm *2.a*, each computation of T_i^{opt} is computed for the chosen b_1, as described in Section 3.1, and the online algorithm again provides a fraction $1/H_n$ of these savings. Although in the bounded setting "increasing" b_1 for the T_i^{opt} computation by modifying d_1 may raise the peak demand *in the offline algorithm's input*, the D value for the online algorithm is not changed. Indeed, examining the proof of Theorem 4.6, we note that the upperbound on the d_i is only used for $i > 1$ (see Eq. 4.4).

5. Lossy Model

We now extend the battery charge problem to the more realistic setting of *lossy* batteries, which lose to conversion inefficiency a constant fraction of any amount charged (e.g. 33%). The loss model works as follows. For each unit of energy charged, only $r = 1 - L$ units will be available for discharge, due to the combined inefficiencies of charging and discharging. This loss could be broken into separate components L_1, L_2 for charge and discharge, but since the loss does not depend on time, doing so would have essentially no effect. For simplicity, we merge these losses into a single loss that occurs instantly at the time of charging.

Threshold algorithms. In the lossy setting, *threshold algorithms* are the same as those given in Section 2 except that they modify the threshold value in order to prevent underflow, if necessary:

for each timeslot i
    if $d_i < T_i$
        charge $\min(B - b_i, T_i - d_i)$
    else
        discharge $\min(d_i - T_i, b_i)$
        if $d_i - T_i < b_i$
            $T_i \leftarrow T_i + (d_i - T_i - b_i)$

The algorithm schema again amounts to this rule: *at each timeslot i, request an amount as near to T_i as the battery constraints will allow.* The second *if* state-

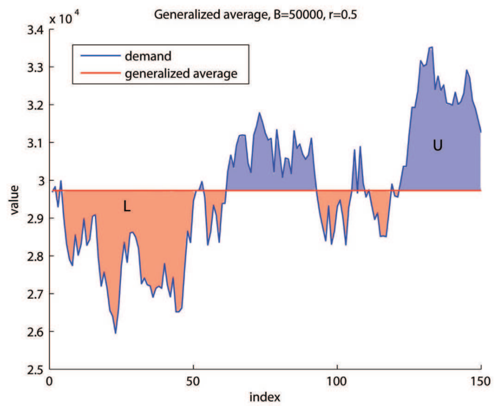


Figure 1: Generalized Average, with U shaded darker and L shaded lighter

ment of the algorithm schemas is executed only if underflow occurs (recall Def. 2.2). The competitiveness guarantee of Algorithm *2.b* for the lossless setting was achieved in Section 4 by showing that such underflow would never occur. The factor-revealing LP below provides evidence that such underflow also never occurs in the lossy setting. Our heuristic algorithms choose lower, more aggressive thresholds, with the result that such underflow does (or rather would) occur. Since meeting demand is a strict requirement, in the event of underflow, the request rises accordingly to keep the battery level nonnegative, which is what the *if* statement does.

6. GA and Lossy Battery Algorithms

The algorithms for lossy batteries are structurally similar to those for lossless, except that computations of *average* are replaced with what we call the *generalized average* (GA). In both cases, the optimal solution value for an interval will be the best possible maximum request over that interval which can be found by computing the (generalized) average of all subintervals. The generalized average of an interval corresponds to a constant request made over the course of the interval (see Fig. 1); the area below the demand curve and above the request corresponds to the amount charged; the area above the demand curve and below the average corresponds to the amount discharged. The algorithms used for the lossy bounded battery setting are shown in Table 3. Since these algorithms are the same as those for the lossless setting except with average replaced with generalized average, we maintain the same labels.

Alg.	battery	online	threshold T_i	running time
1.b	bounded	no	$\hat{\mu}(1,n)$	$O(n^2 \log n)$
2.a	bounded	yes	$D - \frac{D-\hat{\mu}(1,i)}{H_n}$	$O(n^2 \log n)$
2.b	bounded	yes	$D - \frac{D-\mu(s_i,i)}{H_{(n-s_i+1)}}$	$O(n \log n)$

Table 3: Threshold functions used for offline and online settings (see Def. 6.2).

Definition 6.1. *Given n real values $(y_1, y_2, \ldots, y_n)$ and constants $0 < r \leq 1$ and $B \geq 0$, let the* ***generalized average*** *$GA(y_1, y_2, \ldots, y_n)$ (denoted μ) be the value of a satisfying $U(a) = B + r \cdot L(a)$, where: $U(a) = \sum_{i=1}^{n} \max(y_i - a, 0)$ and $L(a) = \sum_{i=1}^{n} \max(a - y_i, 0)$. We call $U(a)$ and $L(a)$ μ's L/U* values *or a's* upper *and* lower. *Treating the input values $y_1, ..., y_n$ as a step function $y = y(x)$, they correspond to the area above a and below y (upper) and the area below a and above y (lower).*

Definition 6.2. *Let $\mu(i,j)$ be the GA of the demands over region $[i,j]$. Let $\hat{\mu}(h,k) = \max_{h \leq i \leq j \leq k} \mu(i,j)$. At time i, let s_i be the most recent time when the battery was full.*

Some intuition may be provided by considering what is likely the simplest way, in terms of coding, of computing a GA: binary search in the range $[\min_i\{d_i\} - B, \max_i\{d_i\}]$. For each candidate value μ, if $B + r \cdot L(\mu) > U(\mu)$ then shift downward and otherwise shift upward, until the two values are sufficiently close.

Note that μ need not be one of the y_i values. When $B = 0$ and $r = 1$, the generalized average is simply the mean of the values y_i; when $B = 0$ and r approaches 0, the generalized average approaches the maximum.

Computing a GA in $O(n \log n)$ is not difficult. First sort the values y_i, and let the values' subscripts now reflect their new positions. Next, set $L(y_1) = 0$, since y_1 is the smallest y_i. For each subsequent i up to n, $L(y_i)$ can be computed in constant time (we omit details due to space constraints). Similarly, compute each $U(y_i)$, starting with $U(y_n)$. Once all the L/Us are computed, μ's *neighbors* (y_i, y_{i+1}), i.e., the two nearest input values that μ lies between, can be found by inspection, and given these μ can be computed in constant time. Unlike the ordinary arithmetic mean, however, computing a GA in $O(n)$ requires more effort.

Our recursive algorithm, whose behavior we sketch in words, both is inspired by the well-known linear-time deterministic Selection Algorithm [18], and calls it as a subroutine. The bulk of the work is in finding μ's *neighbors*. Given these data points (and their L/U values), we can solve for the correct value μ in constant

```
GenAvgNbrs(A[], X_U, X_L, W_U, W_L) :
    if length(A) = 2
        return A
    else p ← Select-Median(A)                          (a)
        (A_L, A_U) ← Pivot(A,p)                        (b)
        U_p ← Upper(A,p); L_p ← Lower(A,p)             (c)
        U ← U_p + X_U + W_U · (max(A) − p)
        L ← L_p + X_L + W_L · (p − min(A))
        if U > r · L + B
            return GenAvgNbrs(A_U ∪ p, L, X_U, W_L + |A_L|, W_U)
        else if U < r · L + B
            return GenAvgNbrs(A_L ∪ p, X_L, U, W_L, W_U + |A_U|)
        else
            return p
```

Figure 2: Algorithm for computing the generalized average of an array of numbers.

time. (The cases–*not shown in the pseudocode*–when the solution μ is among the data points, and when μ is less than *all* the points can be checked as special cases.) The algorithm for finding the neighboring data points to μ takes the set of points y_i as input. Let $0 \leq r < 1$ and B be the parameters to the GA. The first parameter to the algorithm is the set of values to be averaged; all other parameters to the first (non-recursive) call are set to 0.

Theorem 6.1. *$GA(y_1, ..., y_n)$ can be computed in $O(n)$ time.*

Proof. (sketch) With $|A| = n$, lines a,b,c each take time $O(n)$ since *Select-Median* uses the Selection Algorithm, *Pivot* is the usual Quicksort pivoting algorithm, and Upper and Lower are computed directly. (Min and max can be passed in separately, but we omit them for simplicity.) The function makes one recursive call, whose input size is by construction half the original input size. Hence the total running time is $O(n)$. □

The bulk of the work done by our algorithms for lossy batteries is to compute the GA for a series of ranges $[i, j]$, as i stays fixed (as e.g. 1) and j increases iteratively (e.g. from 1 to n). It is straightforward to do this in $O(n^2)$ time, by maintaining a sorted sublist of the previous elements, inserting each new y_j and computing the new GA in linear time. Unlike ordinary averages, $GA[i, j]$ and the value y_{j+1} do not together determine $GA[i, j+1]$.[5] (The GA could also be computed separately for each region $[1, j]$.) This yields offline algorithms for the lossy unbounded and bounded settings, with running times $O(n^2)$ and $O(n^3)$. Through careful use of data structures, we obtain faster algorithms, with running times $O(n \log n)$ and $O(n^2 \log n)$, respectively.

Theorem 6.2. *The values $GA[1, j]$, as j ranges from 1 to n can be computed in $O(n \log n)$.*

Proof. (sketch) A balanced BST is used to store previous work so that going from $GA[i, j]$ to $GA[i, j+1]$ is done in $O(\log n)$. Each tree node stores a y_i value plus other data (its L/U, etc.) used by GENAVGNBRS to run in $O(\log n)$. Each time a new data point y_i is inserted into the tree, its data must be computed (and the tree must be rebalanced). Unfortunately, each insertion *partly* corrupts *all other nodes' data*. Using a lazy evaluation strategy, we initially update only $O(\log n)$ values. After the insert, GENAVGNBRS is run on the tree's current set of data points, in $O(\log n)$ time, relying only on the nodes' data that is guaranteed to be correct. Running on the BST, GENAVGNBRS's subroutines (Select-Median, Pivot, and selection of the subset to recurse on) now complete in $O(\log n)$, for a total of $O(n \log n)$. □

Theorem 6.3. *For the offline/lossy setting, Algorithm* 1.b *($T_i = \hat{\mu}(1, n)$) is optimal, feasible, and runs in time $O(n^2 \log n)$.*

Proof. Within any region $[i, j]$, the battery may help in two ways. First, the battery may be able to lower the local peak by sometimes charging and sometimes discharging. Second, the battery in the best case would start with charge B at timeslot i. With battery loss percentage L, the total amount discharged from the battery over this period can be at most B plus $(1 - L)$ times the total amount charged. The optimal threshold over this region cannot be less than $GA(d_i, ..., d_j)$ with $(1 - L, B)$ chosen as its parameters (r, B).

The threshold used is $T = \hat{\mu}(1, n)$. It suffices to show that the battery will be nonnegative after each time j. Suppose j is the first time underflow occurs. Let $i-1$ be the last timeslot prior j with a full battery (or 0 if this has never occurred). Then there is no underflow *or* overflow in $[i, j)$, so the total charged in region $[i, j]$ is exactly $L(T) = \sum_{t=i}^{j} \max(T - d_t, 0)$ and the total discharged will be

[5] For example, when $B = 10$ and $r = .5$, $GA(5, 10, 15) = GA(3, 21, 3) = 7$, but $GA(5, 10, 15, 20) = 10.83 \neq GA(3, 21, 3, 20) = 11.33$.

$U(T) = \sum_{t=i}^{j} \max(d_t - T, 0)$. The amount of energy available for discharge over the entire period is $B + r \cdot L(T)$. Underflow at time j means $U(T) > B + r \cdot L(T)$, but this contradicts the definition of T.

To compute the thresholds, compute $GA[i, j]$ iteratively (varying j from i to n) for each value i. Each i value takes $O(n \log n)$, for a total of $O(n^2 \log n)$. □

Corollary 6.4. *If the battery is effectively unbounded, then a similar optimal algorithm can be obtained, which runs in time* $O(n \log n)$.

7. Evidence for Lossy Competitiveness

If no underflow occurs, then Algorithms *2.a* and *2.b* are H_n-competitive by construction. (Recall that the objective function is the peak reduction amount.) In this section, we use the factor-revealing linear program (LP) technique of Jain et al. [19] to provide some quasi-empirical evidence that no such underflow can ever occur.

A factor-revealing LP is defined based on a particular algorithm for a problem. The LP variables correspond to possible instances, of a certain size n, of the optimization problem. (We therefore have an indexed family of linear programs.) The optimal solution value of the linear program reveals something about the algorithm it is based on. In the original Facility Location application, the objective function was the ratio of the cost incurred by the approximation algorithm in covering the facility and the optimal cost (assumed wlog to be 1) of doing so, so the maximum possible value of this ratio provided an upper bound on the algorithm's approximation guarantee.

The size index of our LPs is the number of timeslots n. The objective function is the final battery level b_{n+1}. The constraints are properties describing the behavior of the algorithm; some of the constraints perform bookkeeping, including keeping track of the battery level over time. We first provide the factor-revealing LP for the lossless setting (Fig. 3), which is simpler than the lossy.

We now explain this program. The battery is initialized to B and can never supersede this level. As we argue below, we can limit ourselves without loss of generality to demand sequences in which the algorithm never wishes to charge to a level greater than B, i.e. no overflow occurs. For such inputs, the threshold scheme's first $\min$ has no effect and we always have that $b_{i+1} = b_i + T_i - d_i$. Threshold T_i is constrained in the LP to equal the expression for Algorithm *2.b*'s threshold, with opt_i lower-bounded by the closed-form expression for the analog of GA for lossless batteries (see Section 4). Moreover, this value is less than

$$
\begin{array}{l}
\textbf{\textit{min:}}\ b_{n+1} \\
\textbf{\textit{s.t.:}}\ b_{i+1} = b_i + T_i - d_i, \text{ for all } i \\
\qquad b_i \le B, \text{ for all } i \\
\qquad T_i = D - (D - opt_i)/H_n, \text{ for all } i \\
\qquad opt_i \ge (1/i)(-B + \sum_{j=1}^{i} d_j), \text{ for all } i \\
\\
\qquad b_1 = B \\
\qquad d_i \le D, \text{ for all } i \\
\qquad D \ge 0, B = 1
\end{array}
$$

Figure 3: Factor-revealing linear program for lossless batteries (LP1).

or equal to the corresponding value used in Algorithm *2.a*, with the effect that T_i is less than or equal to the corresponding threshold of Algorithm *2.a at every time* i. This in turn means that feasibility of Algorithm *2.b* implies feasibility of Algorithm *2.a*. D is included in the program for clarity.

We solved this LP, written in AMLP, with LP solvers on the NEOS server [20], for several values $n \le 100$. The solution value found was 0, consistent with the known result (i.e., Theorem 4.6 of Section 4). Lemma 4.5 of Section 4 allows us to limit our attention to a certain family of demand sequences, viz., those in which once underflow occurs it continues to the end (i.e., $b_n < 0$) and no overflow ever occurs.

Theorem 7.1. *If the optimal solution value LP1, for parameter size* n*, is at least 0, then Algorithm* 2.a *is* H_n*-competitive in the lossless setting, for problem size* n*.*

Proof. Suppose underflow were to occur at some time t, and let s be the most recent time prior to t when the battery was full. Then by the lemma, $[s, t]$ can be assumed wlog to be $[1, n]$. The assumptions that the battery starts at level B and never reaches this level again (though it may rise and fall non-monotonically) are implemented by the constraints stating that $b_1 = B$ and $b_i \le B$. Since no overflow occurs, the first $\min$ in the threshold algorithm definition has no effect, and the battery level changes based only on T_i and d_i, i.e., it rises by amount $T_i - d_i$ (which may be negative), which is stated in constraints. Since $opt_i = \hat{\mu}(1, i)$ is the max of expressions for all sequences of $[1, i]$, in particular we have that $opt_i \ge (-B + \sum_{j=1}^{i} d_i)/i$. The optimal solution value to such an LP equals the lowest possible battery level which can occur, given any possible problem instance of size n, when using any algorithm consistent with these constraints. Since in particular the behavior of Algorithms *2.a* is consistent with these constraints, the result follows. $\square$

Corollary 7.2. *If the optimal solution value LP1, for all parameter sizes $\leq n$, is at least 0, then Algorithm* 2.b *is H_n-competitive in the lossless setting, for problem sizes $\leq n$.*

Proof. In Algorithm *2.b*, threshold T_i is defined based on the region beginning after the last overflow at position $s = s_i$, shifting $[s,t]$ to $[1,t'] = [1, t-s+1]$ has *no effect*. If *2.b instead* always used H_n, then the lemma above would directly apply, since the algorithm would then perform identically on $[s,t]$ to $[1,t']$, and then the underflow could again be extended to the end. The result that LP1's optimal solution value is ≥ 0 would then imply that this *modified* Algorithm *2.b* is feasible for inputs of size n.

In fact, the actual Algorithm *2.b* uses H_{n-s+1}, with $s = s_i$, at time i. In effect, Algorithm *2.b* treats the demand suffix $d_{s+1}, ..., d_n$ as an independent problem instance of size $n - s + 1$. Each time the battery overflows, the harmonic number subscript is modified, and there is a new subregion and a new possibility for underflow. If all sizes of overflow-free subregions are underflow-free, then the algorithm is feasible. Therefore, if LP1's optimal solution is nonnegative, Algorithm *2.b* is feasible. $\square$

The more complicated factor-revealing program for the lossy setting is shown in Fig. 4. The additional difficulty here is that the program has quadratic constraints.

We omit a full description of this program and only remark that the two main difficulties are 1) that there is an essential asymmetry in the battery behavior, which complicates the first constraints; and 2) that since we have no closed formula for GA, (a lower bound on) the optimal threshold must be *described* rather than *computed*.

There are three sets of quadratic constraints, indicated by stars. In fact, it is possible to remove these and convert LP2 to a linearly-constrained quadratic program (QP), defined for a fixed constant efficiency ratio r (unlike LP2, in which r is a variable). Unfortunately, the resulting QP is not convex.

We then have the following results.

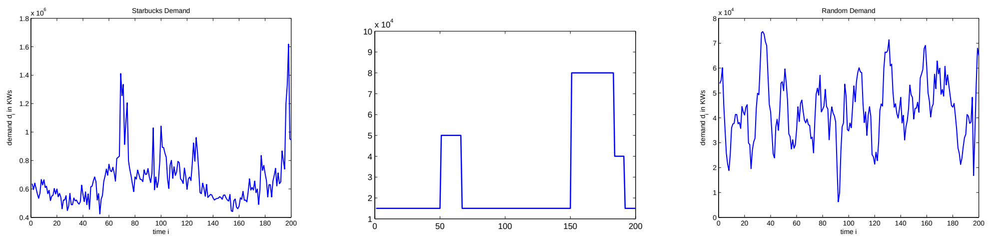
Theorem 7.3. *If the optimal solution value LP2, for parameter size n, is at least 0, then Algorithms* 2.a *is H_n-competitive in the lossy setting, for problem size n.*

$$
\begin{array}{ll}
\textbf{\textit{min:}} & b_{n+1} \\
\textbf{\textit{s.t.:}} & b_{i+1} = b_i + r \cdot ch_i - dis_i, \text{ for all } i \\
& ch_i \cdot dis_i = 0 \qquad \text{(*)} \\
& b_i \leq B, \text{ for all } i \\
& ch_i, dis_i \geq 0, \text{ for all } i \\
& \\
& b_1 = B \\
& D \geq d_i, \text{ for all } i \\
& B = 1, D \geq 0 \\
& \\
& T_i = D - (D - opt_i)/H_n \\
& T_i = d_i - dis_i + ch_i \\
& opt_i \geq ga_i, \text{ for all } i \\
& \\
& B + r \cdot (\sum_{j=1}^{i} cho_{i,j}) = \sum_{j=1}^{i} diso_{i,j}, \text{ for all } i \qquad \text{(*)} \\
& cho_{i,j} \cdot diso_{i,j} = 0, \text{ for all } (j,i) : j \leq i \qquad \text{(*)} \\
& ga_i = d_j - diso_{i,j} + cho_{i,j}, \text{ for all } (j,i) : j \leq i \\
& cho_{i,j}, diso_{i,j} \geq 0, \text{ for all } (j,i) : j \leq i
\end{array}
$$

Figure 4: Factor-revealing mathematical program for lossy batteries (LP2).

Corollary 7.4. *If the optimal solution value LP2, for all parameter sizes $\leq n$, is at least 0, then Algorithm* 2.b *is H_n-competitive in the lossy setting, for problem sizes $\leq n$.*

We solved the second program, implemented in AMLP, using several solvers (MINLP, MINOS, SNOPT) on the NEOS server [20], for several values $n \leq 100$. (The number of variables is quadratic in n, and there are limits to the amount of memory NEOS provides.) In all cases, we found the solution value found was (within a negligible distance of) nonnegative. Although these solvers do not guarantee a globally optimal solution (at least not for non-convex nonlinear programs), we believe this performance provides some "quasi-empirical" evidence for the correctness of Algorithms *2.a* and *2.b*.



(a) Starbucks.   (b) Resident.   (c) Random.

Figure 5: Input data: demand versus time.

8. Performance Evaluation

8.1. Experiment Setup

We performed experiments on three datasets: a regular business day's demand from an actual Starbucks store, a simulated weekday demand of a residential user, and a randomly generated demand sequence. Each dataset is of length $n = 200$. The demand curves are shown in Fig. 5. The parameters in our simulation are: 1) battery size B (typically $B = 500K$), 2) battery charging loss factor L (typically $L = 0.33$), and 3) an aggressiveness c discussed below (where $c = 0$ corresponds to Algorithm *2.b* and $c = 1$ to Algorithm *2.b-opt*).

Since the objective is peak minimization, we modify the algorithms so that the requests are monotonically increasing (except when prevented by overflow). Since the peak must be at least $D - B$, we similarly force this to be the minimum request (again barring overflow). Although the underlying offline algorithm assumes that $b_1 = B$, other lower initial battery levels can be simulated by artificially increasing the initial demand(s). In the next subsection, we discuss a sample of the experiments performed.

8.2. Simulation Results

We know that in the lossless setting, our algorithms are H_n-competitive in terms of *peak reduction*, since no underflow occurs. We first wish to test the performance of the corresponding lossy algorithms, as well as other, heuristic algorithms.

Test 1 - battery size. In this test, we measured the peaks produced by the different algorithms running with various battery sizes, for settings including lossy and lossless and initial battery levels of $b_1 = 0$ and $b_1 = B$. We observe the same general patterns throughout. For random input, performance is averaged over 50 runs. We observe (Fig. 6) that increasing the battery size reduces the peak in our optimal algorithm; we also see that Algorithm *2.b* constantly outperforms Algorithm *2.a*, and that they both are within H_n of opt, as expected. We include the heuristic Algorithm *2.b-opt*, for comparison, which at each point attempts threshold $opt_i = \hat{\mu}(1, i)$, i.e., to have, at all times, the same peak as opt would have had so far. This is a very bold algorithm. We see that it can perform badly with too large a battery since its aggressiveness can then have greater effect, increasing the likelihood of underflow.

In our next test, we seek a middle-ground between the conservativeness of *2.b* and the boldness of *2.b-opt*.

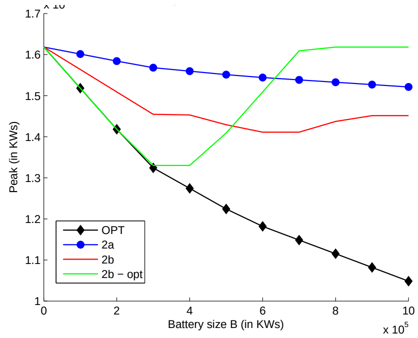
Test 2 - aggressiveness. We vary the boldness in an algorithm based on *2.b* by using a threshold $T_i = D - \frac{D-\hat{\mu}(1,i)}{1+(H_{n-s+1}-1)c}$, with parameter c. When $c = 1$, the algorithm is *2.b* (most conservative); when $c = 0$, it is *2.b-opt* (most aggressive). In this test, we measure the performance as c varies from 0 to 1 with increment of 0.1. We compare the performance of Algorithm *2.b* as a reference. We used two battery sizes on the scale of battery size in the Starbucks installation. We observe (Fig. 6) that increased aggressiveness improves performance, but only up to a point, for reasons indicated above. We note that the best aggressive factor c can depend on both battery size B and the input data.

Although we naturally find that too much *unmotivated* boldness can be damaging, there are potential situations in which significant boldness can be justified.

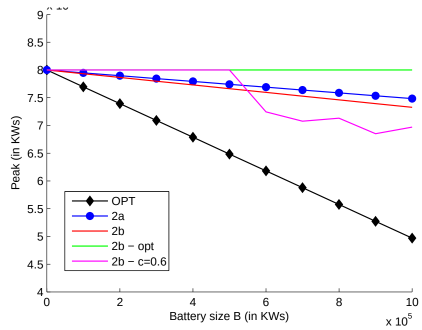
Test 3 - predictions. Suppose we are given error-prone but reasonably accurate predictions of future demands, based e.g. on historical data. In this test, we test two prediction-based algorithms. Let p_i be the predicted demand sequence. Let error $e \in [-1, 1]$ be the prediction error level, with p_i uniformly distributed in $[d_i, d_i(1 + e)]$ or $[d_i(1 + e), d_i]$ if $e < 0$. First, the *oblivious* algorithm simply runs the optimal offline algorithm on the p_i values. The *update* version runs the offline algorithm on demand sequence $< d_1, \ldots, d_i, p_{i+1}, p_n >$, in which the d_i are the actual past demands and the p_i are the future predictions. We compare the performances of prediction algorithms with optimal offline algorithm, Algorithms *2.b* and *2.b-opt* as references. We vary the prediction values from most optimistic $e = -1$ to most conservative $e = 1$.

We see (Fig. 7) that the performance of the prediction algorithm varies in roughly inversely proportion to the error level. If the prediction error is less than 20%, both prediction algorithms outperform the two online algorithms. As e approaches zero, the performance naturally converges to the optimal.

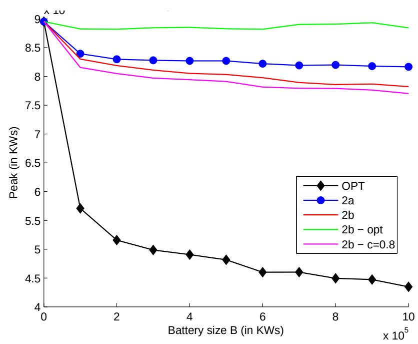
Test 4 - lost energy. As noted in the introduction, the use of lossy batteries increases the total energy used. In this test, we compare the lost energy during charging process with the total energy demand (Fig. 8). We verify that the amount



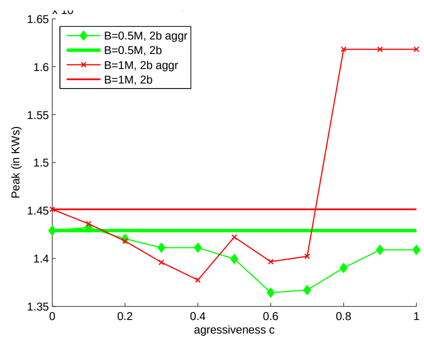
(a) Starbucks.



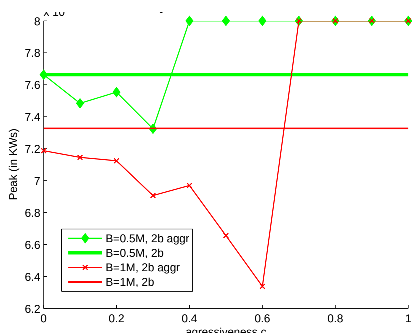
(b) Resident.



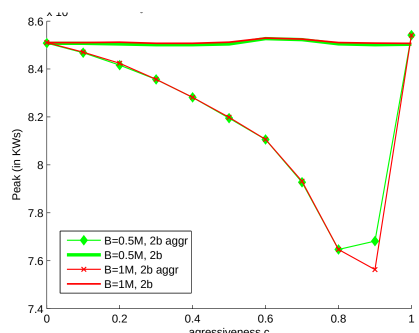
(c) Random, 50 average.



(d) Starbucks.

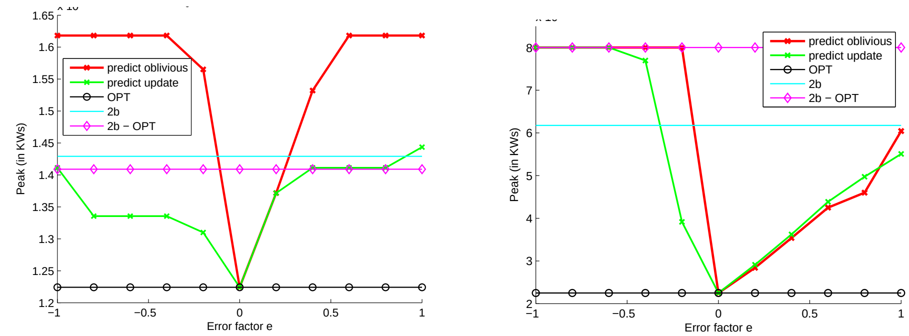


(e) Resident.



(f) Random, 50 average.

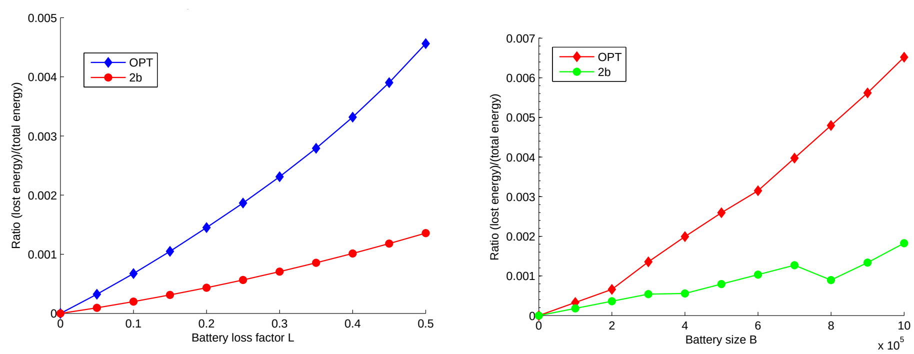
Figure 6: Test 1: peak versus B, $b_1 = 0$, $L = 0.33$ (a) (b) (c); Test 2: peak versus aggressiveness c, $b_1 = 0$, $L = 0.33$ (d) (e) (f).



(a) Starbucks.

(b) Resident, 50 average.

Figure 7: Test 3: peak versus prediction error e, $b_1 = B = 500k$, $L = 0.33$.



(a) Varying L; B=500k.

(b) Varying B; $L = 0.33$.

Figure 8: Test 4: ratio of energy loss and energy demand, $b_1 = 0$, Starbucks.

of lost energy is negligible compared with the total energy demand. We naturally find, however, that larger B and larger loss factor L increase energy loss. We believe that the facts that the fraction of lost energy is small and that the per-unit energy charge is significantly lower than the per-unit peak charge vindicate our choice to focus on peak charge.

9. Conclusion

In this paper, we formulated a novel peak shaving problem, and gave efficient optimal offline algorithms and optimally competitive online algorithms. We also presented optimal offline algorithms and both heuristic and possibly competitive online algorithms for the peak reduction problem with lossy batteries. The factor-revealing mathematical program for the lossy setting presently provide only quasi-empirical evidence for competitiveness. The ability to solve such programs optimally would provide computer-aided proof of such competitiveness, at least for instances of bounded size. The primary open problem suggested by this work is to prove this competitiveness analytically. Several additional future extensions suggest themselves:

- limited battery charging/discharging speed
- battery loss over time ("self-discharge')
- multi-dimensional demands and resulting complex objective functions

Acknowledgements. This work was supported by grants from the NSF (grant number 0332596) and the New York State Office of Science, Technology and Academic Research. We thank Deniz Sariöz and Ted Brown for useful discussions. We also thank Ib Olsen of Gaia for posing the problem and for providing the Starbucks dataset.

References

[1] A. Bar-Noy, M. P. Johnson, O. Liu, Peak shaving through resource buffering, in: WAOA, 2008, pp. 147–159.

[2] A. Bar-Noy, Y. F. 0002, M. P. Johnson, O. Liu, When to reap and when to sow - lowering peak usage with realistic batteries, in: WEA, 2008, pp. 194–207.

[3] M. L. Wald, Storing sunshine, The New York Times (www.nytimes.com/2007/07/16/business/16storage.html).

[4] ConEd electricity rates document (www.coned.com/documents/elec/043-059h.pdf).

[5] Orlando Utilities Commission website (www.ouc.com/account/rates/electric-comm.htm).

[6] T. Harford, The great Xbox shortage of 2005, Slate (www.slate.com/id/2132071/).

[7] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.

[8] M.-K. Lee, E. A. Elsayed, Optimization of warehouse storage capacity under a dedicated storage policy, Int J Prod Res Vol. 43, No. 9.

[9] Y.-W. Zhou, A multi-warehouse inventory model for items with time-varying demand and shortages, Computers and Operations Research Vol. 30, Issue 14.

[10] M. Goh, O. Jihong, T. Chung-Piaw, Warehouse sizing to minimize inventory and storage costs, Naval Research Logistics Vol. 48, Issue 4.

[11] Y. Pochet, L. Wolsey, Production Planning by Mixed Integer Programming, Springer, 2006.

[12] M. Florian, J. Lenstra, A. R. Kan, Deterministic production planning: algorithms and complexity, Management Science Vol. 26.

[13] H. Wagner, T. Whitin, Dynamic version of the economic lot size model, Management Science Vol. 5.

[14] A. Atamturk, D. S. Hochbaum, Capacity acquisition, subcontracting, and lot sizing, Management Science Vol. 47, No. 8.

[15] B. Hunsaker, A. J. Kleywegt, M. W. P. Savelsbergh, C. A. Tovey, Optimal online algorithms for minimax resource scheduling, SIAM J. Discrete Math.

[16] R. Ahuja, T. Magnanti, J. Orlin, Network Flows, Prentice Hall, 1993.

[17] R. L. Graham, D. E. Knuth, O. Patashnik, Concrete Mathematics: A Foundation for Computer Science, 2nd Edition, Addison-Wesley Professional, 1994.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press and McGraw-Hill, 2001.

[19] K. Jain, M. Mahdian, E. Markakis, A. Saberi, V. Vazirani, Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP, Journal of the ACM 50(6) (2003) 795–824.

[20] NEOS server, Argonne National Lab (www-neos.mcs.anl.gov/neos/solvers/).