When to Reap and When to Sow:
Lowering Peak Usage With Realistic Batteries

Amotz Bar-Noy Yi Feng Matthew P. Johnson Ou Liu

Department of Computer Science
The Graduate Center of the City University of New York

Abstract. In some energy markets, large clients are charged for both total energy usage and peak energy usage, which is based on the maximum single energy request over the billing period. The problem of minimizing peak charges was recently introduced as an online problem in [4], which gave optimally competitive algorithms. In this problem, a battery (previously assumed to be perfectly efficient) is used to store energy for later use. In this paper, we extend the problem to the more realistic setting of *lossy* batteries, which lose to conversion inefficiency a constant fraction of any amount charged (e.g. 33%). For this setting, we provide efficient and optimal offline algorithms as well as possibly competitive online algorithms. Second, 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.

1 Introduction

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 request (*peak charge*, in kW). If demands are given as a sequence $(d_1, d_2, \ldots, d_n)$, then the total bill is of the form $c_1 \sum_i d_i + c_2 \max_i\{d_i\}$, i.e., a weighted sum of the total usage and the maximum usage. This means 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 charge may be on the order of 100 times the per-unit cost for total usage[1], this difference can be significant. Indeed, this is borne out in our experiments.

At least one start-up company [1] is currently marketing battery-based systems intended to reduce peak energy charges. In such a system, a battery is placed between the power company and a high-consumption client site, 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

[1] The Orlando Utilities Commission website [3], e.g. quotes rates of 6.388¢/kWh ("energy charge") and $6.50/kW ("demand charge").

in the demand curve can thus be made consistent with a relatively flat level of supplied power, yielding lower cost for the client and more tractable requests for the provider. It is interesting to note that a battery system may actually *raise* energy usage, 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 still benefit the system as a whole by easing the strain of peak demands.

In the online setting, the essential choice faced at each time is whether (and by how much) to invest in the future or to cash in on 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 needed, then it is vulnerable to spikes in demand; if it asks for much more energy than it needs, then this request could itself become a new, higher peak. The strictness of the problem lies in the fact that we want *every* request to be low, not just to minimize a total.

In [4], we gave H_n-competitive algorithms for the online lossless setting and matching lower bounds on competitiveness. (These algorithms are in fact only partially online since they depend on having the maximum demand D revealed in advance. Lacking this information, no non-trivial competitiveness is possible.) Those algorithms assume perfectly efficient batteries, however, and will fail if run on realistic, lossy batteries. In the present paper, 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. We also consider new settings and objective functions, such as total cost. Finally, we provide factor-revealing LPs, which we use to provide quasi-empirical evidence of the competitiveness of the lossy algorithms.

Background. As noted above, the problem we study was introduced in [4]. There are many related problems in commodity production, storage, warehousing, etc. More specifically, there are many inventory problems based on the Economic Lot Sizing model [6], in which demand levels for a product vary over a discrete finite time-horizon and are known in advance. See [4] for a full discussion.

The goal in the minimax work-scheduling problem [7] 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 of this in which jobs with deadlines are assigned online. In that problem, all work must be done by deadline but cannot be begun until assigned. While the problems differ in important respects (see [4]), the objectives are similar. Indeed, while the α-policy of [7] performs α *times the maximum per-unit-timeslot amount of work that OPT would have done, when running on the partial input received so far*, many of our algorithms ensure that the savings at each point is a multiple of the optimal savings so far.

2 Model and Algorithms

Definition 1. *The* demand curve *is the timeslot-indexed sequence of energy demands* $(d_1, ..., d_n)$*. The* request curve *is the timeslot-indexed sequence of energy requests* r_i*. Battery charge level* b_i *indicates the (non-negative) amount of energy present in the battery* at the start *of timeslot* i*.* D *is the revealed maximum demand* $\max_i\{d_i\}$*, and* R *is the maximum request* $\max_i\{r_i\}$*.*

The demand curve (combined with battery information) is the problem instance; the request curve is the problem solution. In the absence of battery loss and overflow/underflow, the battery level at timeslot i is simply $b_i = b_{i-1} + r_{i-1} - d_{i-1}$. It is forbidden for b_i to ever fall below 0 (underflow). That is, the request r_i and the battery level b_i must sum to at least the demand d_i at each time i.

By discretizing we assume wlog that battery level, demand, and request values are all expressed in the same units ("kWh"). Peak charges are based linearly on the max request. We optimize for the peak charge, not for total energy usage, since the bulk of this is a fixed cost. There are several independent optional extensions, leading to many problem variants. The battery can have *maximum capacity* B or be unbounded; with some batteries, there is as already noted an automatic percentage *loss* $0 \leq \ell \leq 1$ in all charged energy, due to AC/DC conversion; the problem may be online, offline, or in between; we consider the settings in which the peak demand D is revealed in advance, or estimates of the individual demands are known, perhaps based on historical data. The loss model works as follows. For each unit of energy charged, only $r = 1 - \ell$ 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. 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 $b_i - b_{i-1}$. This means that there are only three possible modes for each timestep: request exactly the demand, request more than the demand and *charge* the difference, or request less than the demand and *discharge* the difference. Our online and offline algorithms are *threshold algorithms*. If $(T_1, T_2, ..., T_n)$ are the chosen thresholds, then the algorithms behave as follows:

```
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 ← T_i + (d_i − T_i − b_i)
```

The algorithm schema amounts to the rule: *at each timeslot i, request an amount as near to T_i as the battery constraints will allow.* Our offline algorithms use a constant 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.

Definition 2. *Let* overflow *be the situation in which $T_i - d_i > B - b_i$, i.e., there is not enough room in the battery for the amount we want to charge. Let* underflow *be the situation in which $d_i - T_i > b_i$, i.e., there is not enough energy in the battery for the amount we want to discharge. Call a threshold algorithm* feasible *if underflow never occurs (overflow merely indicates a lower effective request).*

The second *if* statement of the algorithm schemas is executed only if underflow occurs. The competitiveness guarantee of Algorithm *2.b* for the lossless setting was achieved in [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 non-negative, which is what the *if* statement does.

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.

3 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 *generalized average* (GA). In all cases, the average computed over an interval will correspond to the best possible maximum request over that interval, which can be found by examining all subintervals. The algorithms used are shown below:

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

Definition 3. *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*** *$GA(y_1, y_2, \ldots, y_n)$ be the value μ satisfying $U(a) = B + r \cdot L(a)$, where: $U(a) = \sum_{i=1}^{n} \max(y_i - a, 0)$ and $L(a) =$*

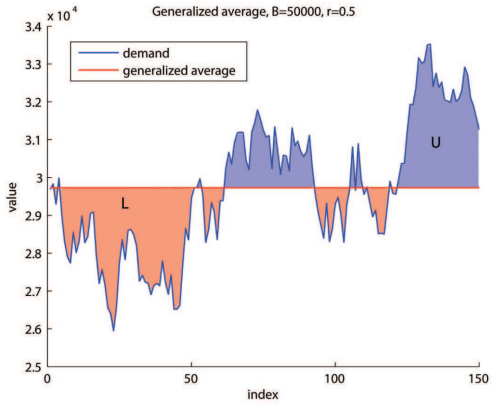


Fig. 1: Generalized Average, with U shaded darker and L shaded lighter

$\sum_{i=1}^{n} \max(a - y_i, 0)$. *We call* $U(a)$ *and* $L(a)$ μ*'s* L/U values *or* μ*'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* μ *and below* y *(upper) and the area below* μ *and above* y *(lower).*

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 $[-B, \max_i\{d_i\}]$. 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 U/Ls 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 [5], 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 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.

```
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;
```

Theorem 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 $GA[i, j + 1]$.[2] (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 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

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

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 $O(n \log n)$.

Definition 4. *Let $\mu(i,j)$ be the GA of the demands over region $[i,j]$. Let $\hat{\mu}(h,k) = \max_{h \le i \le j \le k} \mu(i,j)$. At time i, let s_i be the most recent time when the battery was full.*

Theorem 3. *For the offline/lossy setting, Algorithm 1 ($T_i = \hat{\mu}(1,n)$) 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 timestep i. With battery loss percentage ℓ, the total amount discharged from the battery over this period can be at most B plus $(1-\ell)$ times the total amount charged. The optimal threshold over this region cannot be less than $GA(d_i, ..., d_j)$ with $(1-\ell, B)$ 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 timestep 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 $U(T) = \sum_{t=i}^{j} \max(T - d_t, 0)$ and the total discharged will be $L(T) = \sum_{t=i}^{j} \max(d_t - T, 0)$. The amount of energy available for discharge over the entire period is $B + r \cdot L(T)$. Overflow 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 1. *If the battery is effectively unbounded, then a similar optimal algorithm can be obtained, which runs in time $O(n \log n)$.*

4 Factor-revealing LPs

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 LP technique of Jain et al. [8] 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,

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

Fig. 2: Factor-revealing linear program for lossless batteries (LP1)

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 timesteps 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 book-keeping, including keeping track of the battery level over time. We first provide the factor-revealing LP for the lossless setting (Fig. 2), 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 [4]. Moreover, this value is less than 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 [2], for several values $n \le 100$. The solution value found was 0, consistent with the known result [4]. We now state the following lemma, which allows us to limit our attention to a certain family of demand sequences.

Lemma 1 ([4]). *If there is a demand sequence $d_1, d_2, ..., d_n$ in which underflow occurs for Algorithm* 2.a*, then there is also a demand sequence in which underflow continues to the end (i.e., $b_n < 0$) and no overflow ever occurs.*

Theorem 4. *If the optimal solution value LP1, for parameter size n, is at least 0, then Algorithms* 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 $b_i \geq B$. 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 $opt_i \geq (-B + \sum_{j=1}^{i} d_i)/i$. 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 the these constraints. Since in particular the behavior of Algorithms *2.a* is consistent with these constraints, the result follows.

Corollary 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 non-negative, Algorithm *2.b* is feasible.

The more complicated Factor-revealing program for the lossy setting is shown in Fig. 3. 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, defined for a fixed constant efficiency L. Unfortunately, the resulting QP is not convex.

We then have the following results.

Theorem 5. *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*.*

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

Fig. 3: Factor-revealing quadratically-constrained LP for lossy batteries (LP2)

Corollary 3. *If the optimal solution value LP2, for all parameter sizes $\le n$, is at least 0, then Algorithm* 2.b *is H_n-competitive in the lossy setting, for problem sizes $\le n$.*

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

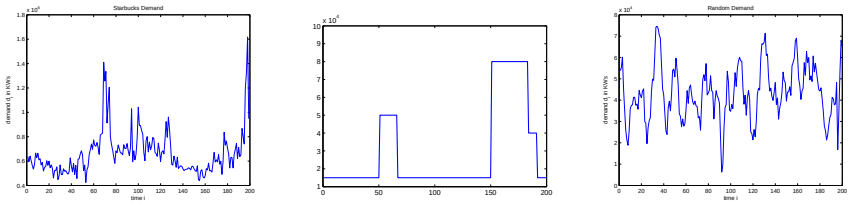
5 Performance Evaluation

5.1 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. 4. The parameters in our simulation 1) battery size B (typically $B = 500K$), 2) battery charging loss factor L (typically $L = 0.33$), 3) aggressiveness c (with $c = 0$ for *2.b* and $c = 1$ for *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.



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

Fig. 4: Input data: demand versus time

5.2 Simulation results

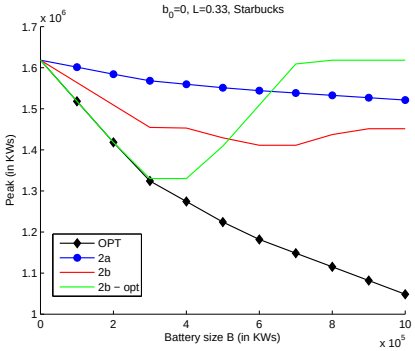
We know that in the lossless setting, our algorithms are H_n-competitive in terms of *peak reduction*, since no underflow occurs. We first wish to test the performance of the corresponding lossy algorithms, as well as other, heuristic algorithms.

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. 5) 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.

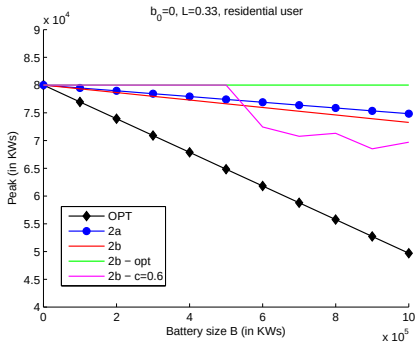
In our next test, we seek a middle-ground between the conservativeness of *2.b* and the boldness of *2.b-opt*.

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 $c = 1$, 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. 5) 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.

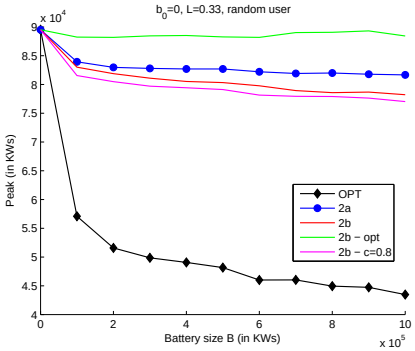
Although we naturally find that too much *unmotivated* boldness can be damaging, there are potential situations in which significant boldness can be justified.



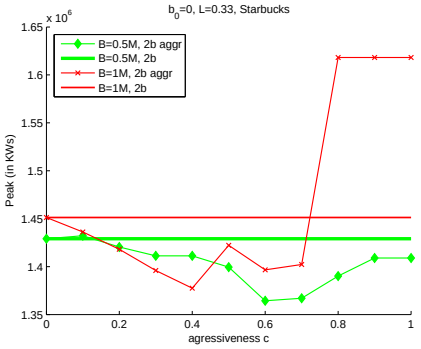
(a) Starbucks



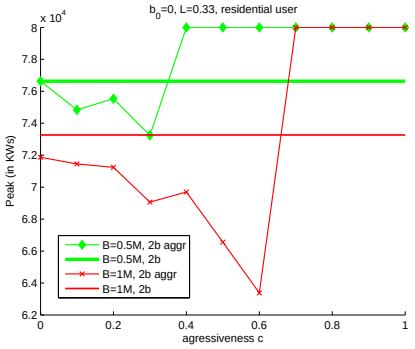
(b) Resident



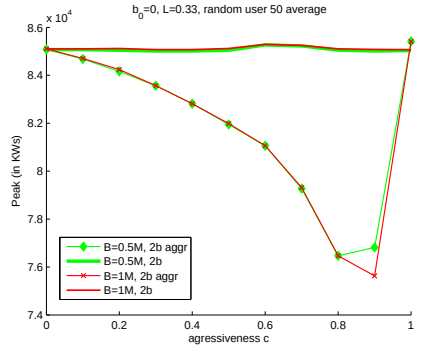
(c) Random, 50 average



(d) Starbucks

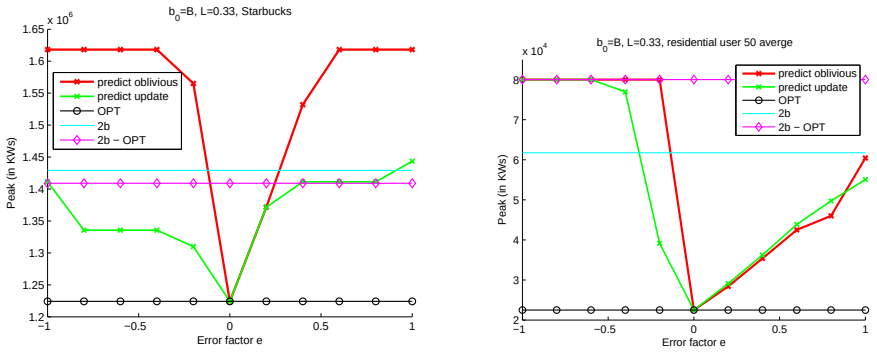


(e) Resident



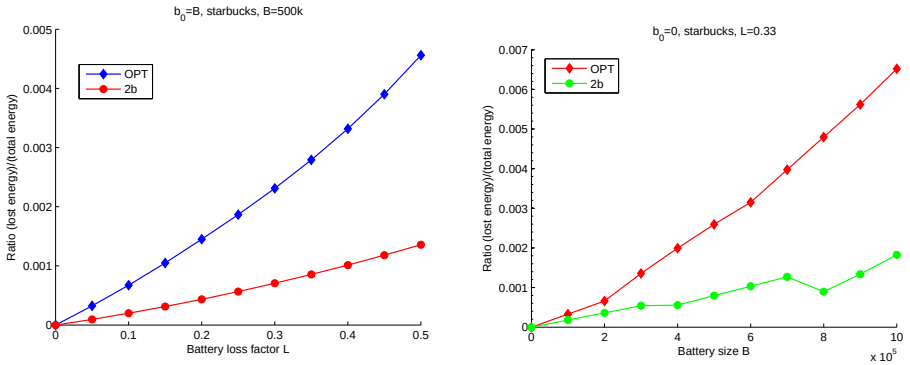
(f) Random, 50 average

Fig. 5: 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

Fig. 6: Test 3: Peak versus prediction error e, $b_1 = B = 500k$, $L = 0.33$



(a) vary L (b) vary B

Fig. 7: Test 4: Ratio of energy loss and energy demand, $b_1 = 0$, Starbucks

Test 3 - predictions. Suppose we are give 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 is 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. 6) 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. 7). We verify that the amount 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.

6 Conclusion

In this paper, we presented optimal offline algorithms and both heuristic and possibly competitive online algorithms for the peak reduction problem with lossy batteries. The factor-revealing LPs for the lossy setting presently provide only quasi-empirical evidence for competitiveness. The potential future availability of global quadratically-constrained LP solvers, however, could provide computer-aided proof of such competitiveness, at least for instances of bounded size. Several additional future extensions suggest themselves:

- additional free but unreliable energy sources (e.g. solar power)
- 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. Gaia Power Technologies. gaiapowertech.com.
2. NEOS server, Argonne National Lab. www-neos.mcs.anl.gov/neos/solvers/.
3. Orlando Utilities Commission website. www.ouc.com/account/rates/electric-comm.htm.
4. A. Bar-Noy, M. Johnson, and O. Liu. Peak shaving through resource buffering. Technical Report TR-2007018, CUNY Graduate Center, Dept. of Computer Science, November 2007.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* MIT Press and McGraw-Hill, 2001.
6. M. Florian, J. Lenstra, and A. R. Kan. Deterministic production planning: algorithms and complexity. *Management Science*, Vol. 26, 1980.
7. B. Hunsaker, A. J. Kleywegt, M. W. P. Savelsbergh, and C. A. Tovey. Optimal online algorithms for minimax resource scheduling. *SIAM J. Discrete Math.*, 2003.
8. K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *J. ACM*, 50(6):795–824, 2003.