Scheduling of electricity storage for peak shaving with minimal wearing

Thijs van der Klauw   Johann Hurink

Faculty of EEMCS, University of Twente

01-04-2014

Introduction

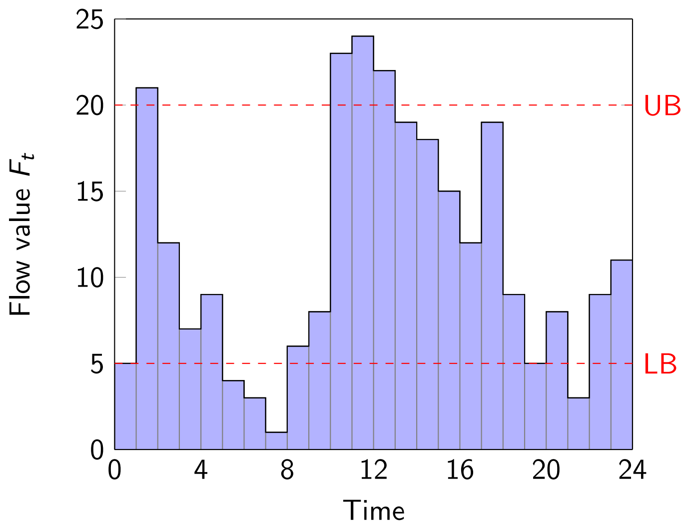
- Increasing use of renewable energy sources.
- Increasing use of electric appliances in general.
- Social and political motivation to move away from fossil fuel and nuclear power.

Introduction

- Electricity in the grid needs to be balanced to ensure grid stability; but: the aforementioned trends jeapordize this goal.
- Samrt Grids is a prommissing concept for the future energie grid.
- Storage of (electrical) energy is seen as one of the most crucial aspects.
- Apply storage for peak shaving and guaranteeing a certain base load.

Energy Flow Description

We consider a discretized time horizon: $\mathcal{T} = [1, \ldots, T]$.



Storage Description

- Given a set $\mathcal{N} = \{1, \ldots, N\}$ of storage devices
- Modelling the use of the storage device i by variables

$$s_t^i$$

representing the amount of energy that flows into (or from) the device i in time interval t.

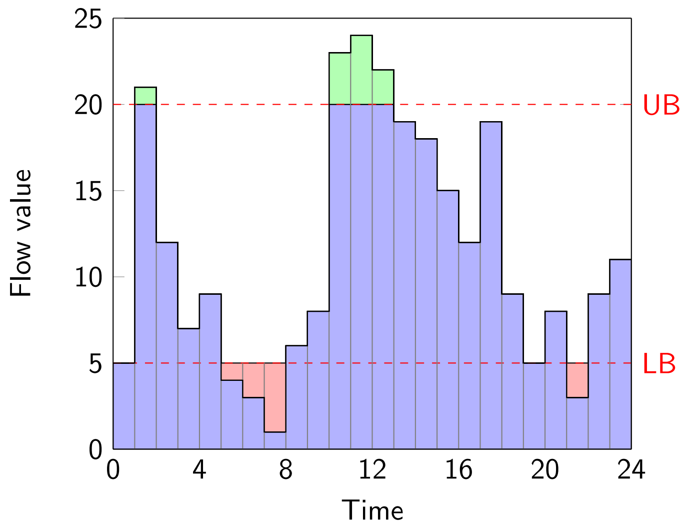
- Remaining flow after use of the storage device is:

$$f_t = F_t - \sum_{i \in \mathcal{N}} s_t^i \ \ \forall t \in \mathcal{T}$$

- Constraints to get the remainig flow between the bound (flow constraint):

$$LB_t \leq f_t \leq UB_t \ \ \forall t \in \mathcal{T}$$

Storage Description



Storage Description

- A storage device is restricted by two more constraints.
- We can't (dis)charge too much in one time interval (power constraint):

$$|s_t^i| \leq P_i \;\; \forall t \in \mathcal{T}; \forall i \in \mathcal{N}$$

- The amount of energy stored in the devices has to be nonnegative and below the given capacity bound C (SoC constraint):

$$SoC_t^i := \sum_{j \leq t} s_j^i \in [0, C_i] \;\; \forall t \in \mathcal{T}; \forall i \in \mathcal{N}$$

Objective Function

- The most direct objective is to minimize the storage usage; i.e. to minimize:

$$\sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}} |s_t^i|$$

- We can rewrite this problem easily to an LP;
  thus this objective is easily solvable.

Objective Function Minimize Charging Cycles

- The battery lifetime is usually measured/given in charging cycles; i.e. the number of times the battery changes from discharging to charging and back to discharging.
- To calculate charging cycles we use binary variables Y_t^i indicating a switch between charging and discharging.

Objective Function Minimize Charging Cycles

- Every charging cycle icontains exactly 2 switches between charging and discharging.
- Thus minimizing charging cycles is equivalent with minimizing:

$$\sum_{i \in \mathcal{N}} \sum_{t \in \mathcal{T}} Y_t^i$$

- Integrating the binary variables into the model, leads to an MILP; i.e., this problem may be harder to solve.

Multiple Devices

Theorem

The battery scheduling problem when minimizing charging cycles for multiple devices is NP-hard

- For 2 devices we use PARTITION for the reduction.
- If we consider the number of devices as input, we have a reduction from 3-PART.
- Both reductions hold even when all devices are the same

Theorem

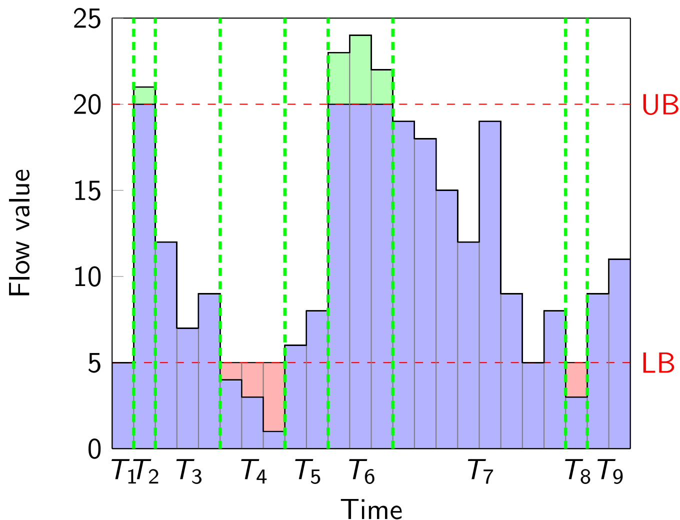
Theorem

The battery scheduling problem when minimizing charging cycles for a single device can be solved in polynomial time

- Grouping the intervals into subsets
- No Lower Bound Violation on Flow
- First SoC constraint violation
- Decoupling points
- General Algorithm

Grouping The Intervals into Subsets

We distinguish between intervals on which we are forced to charge or discharge by the flow constraint and intervals on which we are not.



The Naive Schedule

- In the remainder we omit superscript i since we have only one device

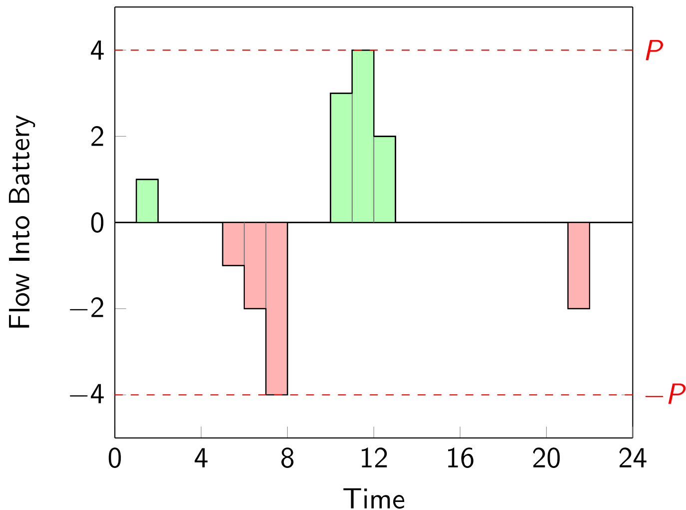
Definition

The naive schedule (possibly infeasible) (dis)charge for every time interval the minimal amount to satisfy the flow constraint and is given by:

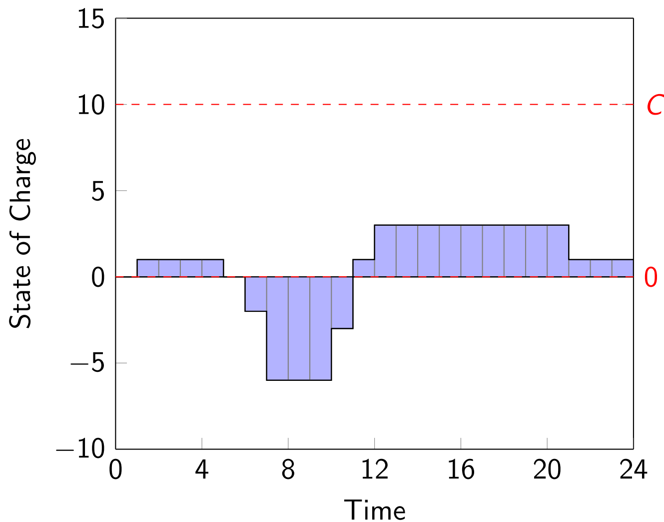
$$s_t = \begin{cases} F_t - UB_t & \text{if } F_t \geq UB_t \\ F_t - LB_t & \text{if } F_t \leq LB_t \\ 0 & \text{else} \end{cases}$$

- The naive schedule can only violate the constraints induced by the storage device.
- If it violates the power constraint: no feasible solution exists.

Storage Requirements Naive Schedule

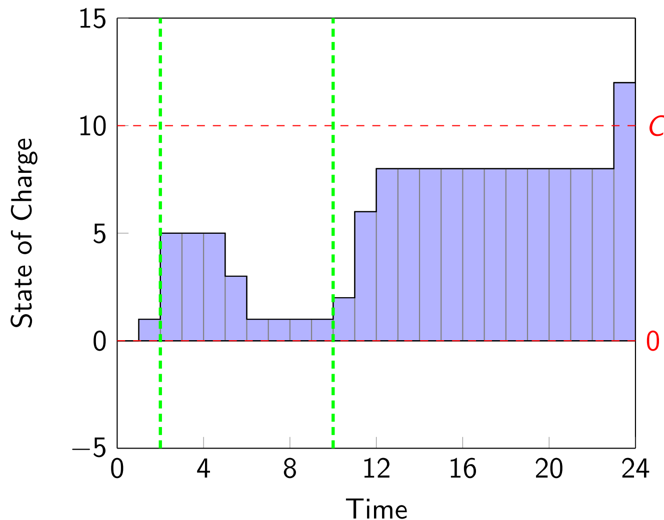


SoC Naive Schedule



First SoC constraint Violation -1-

Consider the naive schedule up to the first violation of the SoC constraint:



First SoC constraint Violation -2-

Apply extra decharging until SoC violation is reduced to 0 as follows:

1. First use subsets which introduce no extra charging cycle (in an arbitrary order).
2. Calculate for each subset the amount which can be discharged within this subset (depends on current schedule).
   Note: each discharge now increases number of switches by 2.
3. Pick the best possible subset, i.e. with the highest potential, and discharge within this subset.
4. Update the discharge capability of all other subsets and continue iteratively with step 2.

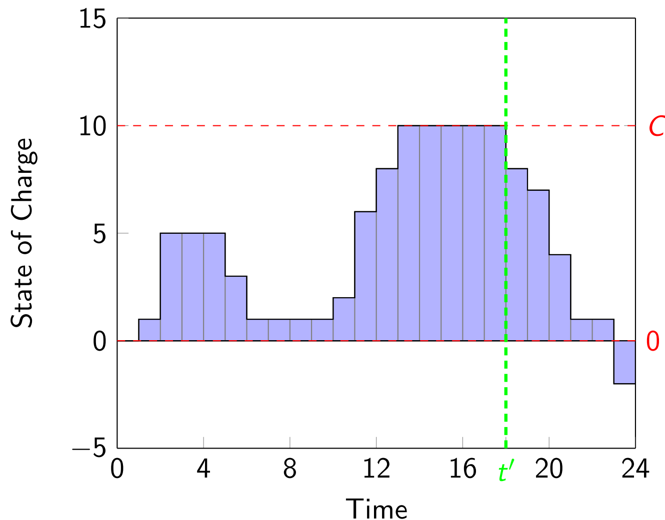
Decoupling Point

- We are able to solve the problem up to the first violation in our naive schedule.

- Now: consider the problem up to the next violation.
- Two cases: violation at the same bound or the oposite bound.
- Same bound: Starting with the schedule we got by solving the previous violation, apply the same method.

Decoupling Point

Other case: the second violation is at the oposite bound.



Algorithm

Summary Algorithm:

1. Construct the naive schedule as the starting point.
2. Consider the subproblem up to the first violation in the naive schedule.
3. Change the schedule until this violation is 'solved' by:
   - First iteratively (dis)charge on the best subset which do not induce switches.
   - Then iteratively (dis)charge on the best subset which do induce switches.
4. Iteratively consider the next violation, until there are no more SoC constraint violations:
   - If the violation is at the oposite bound, introducing a decoupling point at previous violation.
   - Applying step 3 to the schedule from the latest decoupling point to the current considered violation.

Future Work

- Integrate this method into a general framework for Demand Side Management (DSM).
- Currently we assume perfect knowledge of the future;
  Aim: account for fluctuations in the energy flow by useing techniques from e.g. Robust Optimization.
- Account for device specific constraints.

Questions?