A Modular Battery System Controller

Dhirendra Singh
RMIT University, Melbourne, Australia
dhirendra.singh@rmit.edu.au

Geoff James
CSIRO, Sydney, Australia
geoff.james@csiro.au

August 19, 2010

1 Introduction

Energy storage helps to enable increasing levels of renewable energy in our electricity system, and the rapidly maturing supply chains for several battery technologies encourages electricity utilities, generators, and customers to consider using large battery systems. Large battery systems usually comprise multiple modules and in many installations these may be controlled independently. Modules may be operated in synchrony but often there are strategic reasons to keep some modules in a different state to others. For example, if it is undesirable to change the direction of power flow between charging and discharging too frequently, a subset of modules may be used for each direction until it is necessary to change their roles. Also, some technologies have specific requirements, such as the zinc-bromine flow battery for which a complete discharge at regular intervals is desirable to strip the zinc plating and ensure irregularities never have an opportunity to accumulate. Where they exist these requirements place further constraints on module control.

Given, then, an input signal which is the requested rate of charging and discharging for a large battery installation, as a function of time, we would like a control algorithm for the set of component modules that implements the requested rate as the sum over the module rates of charging and discharging. The input signal will be different every day but will have many features that are diurnal or nearly so, due to typical variations of electricity demand and solar and wind energy generation sources, and the repetitive patterns that may be seen over several days of the input signal suggest that a learning algorithm may be appropriate. Our problem is to develop a method for on-line learning that will result in a useful control regime for a modular battery system, when installed at a new site and provided with an input signal derived from the electricity demand and renewable supply at that site.

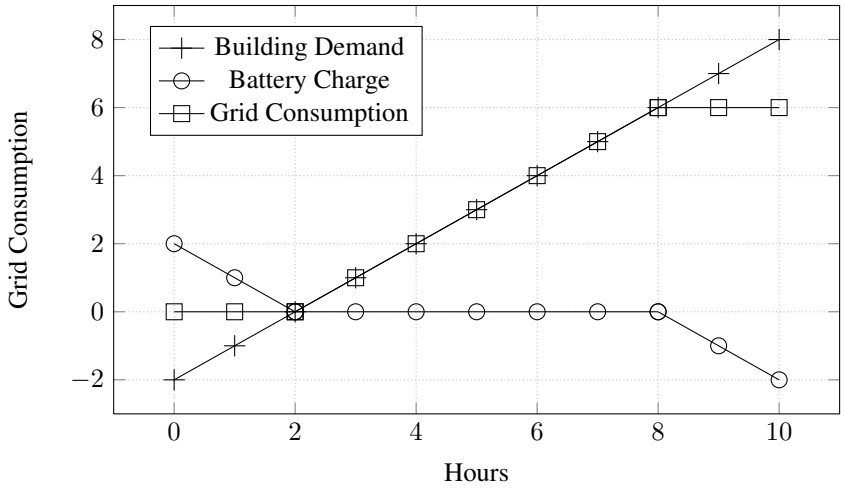

Figure 1: Use case scenario for the battery system during a 10 hour period. The aim is to maintain grid consumption within the range $[0 : 6]$ units. Building Demand is the sum of individual loads and renewable generation and may therefore fall outside this range. By suitably ordering the charge or discharge of the battery system (Battery Charge signal), the net consumption (Grid Consumption signal) of the system may be maintained within range.

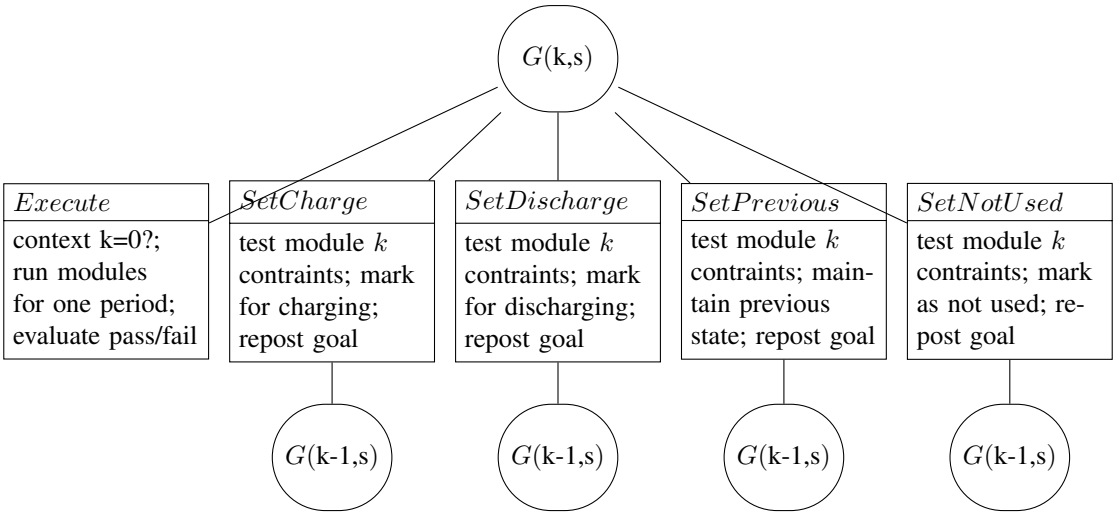

Figure 2: Goal-plan hierarchy for the battery system with k modules.

Consider the example scenario of a smart office building that consists of a set of individual loads (appliances in the building), some renewable generators (solar panels on the roof and a local wind turbine), and a modular battery system. The building is connected to the main grid, and the economics govern that the grid power consumption of the building be maintained within the range $[0 : 6]$. Since there is little control over the demand in the building and certainly no control over the renewable generation, then for some period in the day it is possible that the power consumption of the building will fall outside this range. For instance, if the renewable generation is higher than the demand, then the consumption will be negative. Similarly, if the demand is higher than the renewable generation then the building consumption may rise above P. In Figure 1 the Building Demand curve shows the sum of consumption and generation in the building for a 10 hour window some given day.

While there is little control over the consumption and generation in the building (Building Demand), we do have full control over how the battery system is used. So for instance, by suitably ordering the battery system to charge (act as a load) or discharge (act as a generator) at determined rates throughout the period we may influence the net demand in the building. Figure 1 shows how the appropriate battery response (Battery Charge curve) added to the building consumption and generation (Building Demand curve) ensures that the power drawn from the grid (Grid Consumption) is maintained within the desired range.

2 Approach

The battery system consists of a number of internal modules that may all have individual constraints. Moreover, the charge/discharge characteristics of the modules may change over time. As such, it is desirable that the battery control algorithm be able to adapt to these changes over time. Figure 2 shows a possible learning BDI agent controller for a battery system with k modules.

The top level goal $G(k, s)$ is posted by the system at the beginning of each period of deliberation. The controller then responds by operating the battery modules for that period in a suitable configuration that resolves the goal. The parameter k is initially set to the number of modules in the system[1]. The parameter s specifies the desired response from the system and lies in the normalised range $[-1.0 : +1.0]$ where -1.0 indicates a maximum discharge rate and $+1.0$ indicates a maximum charge rate.

The resolution of the battery system determines how closely it can match the desired response and depends on the number of modules n. For simplicity, we will assume homogeneous capacity of the modules (but with possibly different chemical properties and constraints), such that each module has a capacity c and $c * n = 1.0$. Each internal module in turn may be in one of three states: charging (i.e $+c$), discharging (i.e. $-c$) or not in use (i.e. 0). The sum of these values gives the net response of the system. In other words, the response of the battery system may be adjusted in multiples of c.

[1]Note that the battery design may contain more internal modules than the advertised number to provide for a operational buffer. In this case the parameter k should reflect this actual count.

The BDI controller works by recursively setting the operational state of each module in the system using the $Set*$ plans, and then finally running the selected configuration for one period and evaluating the result using the $Execute$ plan. The execution therefore always consists of the selection of k high level $Set*$ plans followed by the $Execute$ leaf plan. Using the BDI learning framework from [3] the pass/fail result is then recorded in the chain of active $Set*$ plans. By training over the outcomes of each plan selection in each situation, the system over time learns correct plan selection at each recursive level for the set of possible top-level requests.

Each plan in the goal-plan hierarchy is further explained below:

$SetCharge$: Plan to set the operational state of module k to *Charge* for this period. The plan first checks that the internal constraints of module $k > 0$ will not be violated by this operation. If a violation is expected, then the plan is aborted, otherwise the state is updated and Goal G reposted for module $k - 1$.

$SetDischarge$: Plan to set the operational state of module k to *Discharge* for this period. The plan first checks that the internal constraints of module $k > 0$ will not be violated by this operation. If a violation is expected, then the plan is aborted, otherwise the state is updated and Goal G reposted for module $k - 1$.

$SetPrevious$: Plan to keep the operational state of module k unchanged from the previous period. The plan first checks that the internal constraints of module $k > 0$ will not be violated by this operation. If a violation is expected, then the plan is aborted, otherwise Goal G is reposted for module $k - 1$.

$SetNotUsed$: Plan to set the operational state of module k to *NotUsed* for this period. This means that the module will remain disconnected from use for this period. The plan first checks that the internal constraints of module $k > 0$ will not be violated by this operation. If a violation is expected, then the plan is aborted, otherwise the state is updated and the Goal G reposted for module $k - 1$.

$Execute$: The leaf plan that actually performs the operations on the battery modules. The plan operates only when $k = 0$ which implies that all modules have been configured. The battery modules are then operated for this period according to their assigned states. At the end of the period, the plan will evaluate the battery response against the goal and determine the $Pass$ or $Fail$ status. Normally, a pass would mean that the actual battery response was within tolerance of the desired response.

The $Set*$ plans in the hierarchy may fail when selected if the constraints for module k are violated for that period. For instance, plan $SetCharge$ might fail because module k is only allowed to change charge directions once every four periods say, and charging it in this period will violate that constraint. Similarly, plan $SetDischarge$ might fail because the module may have critically low charge and further discharge for a full period is not possible. Finally, plan $SetNotUsed$ might fail because the module has a requirement to be fully discharged once every day, and disconnecting it in this period will violate that constraint.

Since violation checks are performed prior to taking any action, then BDI failure recovery may be performed to select a different $Set*$ plan until all internal constraints are satisfied prior to execution. Note that failure recovery is not allowed for the $Execute$ plan because it runs for a full period and that is the limit for the decision making. In other words, only one try is allowed per period. So functionally, the $Execute$ plan must always succeed, even though the evaluation against the goal (for learning purposes) may differ.

Finally, the system only learns a response to the immediate request i.e. how to resolve the top level goal. It does not learn any temporal relationship in the sequence of top level goals. For instance, the input signal may have some diurnal pattern, however the proposed system does not attempt to learn this pattern.

3 Implementation Notes

The system is (purposely) similar in design to the Towers of Hanoi problem of [3]. In some respects, it is simpler because the solutions are always at recursive depth k. Moreover, the non-leaf $Set*$ plans do not have side-effects when they fail and leave the initial state unaltered. Finally, a solution always consists of a single $Execute$ action whereas in the Hanoi problem it consists of possibly several $Move$ actions[2].

[2]One point of difference is that this is not a universal library where a solution can always be found. For some requests, no solution will be possible given the internal state of the modules.

Nonetheless, the problem captures a real world problem where it is not straightforward to hand-craft a functional hierarchy and where learning is justified. We do not have a solution to begin with. Then, the size of the problem is still significant - the flow battery in CSIRO Newcastle has 10 internal modules, so with three possible module states that represents $3^{10} = 59049$ possible solutions for a given request. Finally, it offers a realistic scenario for re-learning a solution due to significant changes in the environment. For instance, if an internal module were to fail and had to be replaced, then prior learning may no longer work effectively, and the system will have to adjust and re-learn it's response based on the new characteristics of the updated module.

4 Assumptions

Geoff, you mentioned earlier that we want to be able to rotate modules around so that they don't develop a bias. I can't remember what the exact problem was. Does that still apply to this solution? Perhaps it is something that can be captured as an internal constraints per module? Or for simplicity we might defer this to future work. In any case, we will likely start with no constraints to make sure we can get the system working first.

References

[1] S. Airiau, L. Padgham, S. Sardina, and S. Sen. Enhancing Adaptation in BDI Agents Using Learning Techniques. In *International Journal of Agent Technologies and Systems*, 2009.

[2] D. Singh, S. Sardina, L. Padgham, S. Airiau. Learning Context Conditions for BDI Plan Selection. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[3] D. Singh, S. Sardina, L. Padgham. Extending BDI Plan Selection to Incorporate Learning from Experience. Robotics and Autonomous Systems, 2010.