Automatic Construction of Efficient Multiple Battery Usage Policies

Maria Fox and **Derek Long**
University of Strathclyde, UK

Daniele Magazzeni
University of Chieti-Pescara, Italy

Abstract

Efficient use of multiple batteries is a practical problem with wide and growing application. The problem can be cast as a planning problem. We describe the approach we have adopted to modelling and solving this problem, seen as a Markov Decision Problem, building effective policies for battery switching in the face of stochastic load profiles. Our solution exploits and adapts several existing techniques from the planning literature and leads to the construction of policies that significantly outperform those that are currently in use and the best published solutions to the battery management problem. We achieve solutions that achieve more than 99% efficiency compared with the theoretical limit and do so with far fewer battery switches than existing policies.

We describe the approach in detail and provide empirical evaluation demonstrating its effectiveness.

1 Introduction

In this paper we describe an application of planning to an important problem in battery management. There is a huge and growing number of systems that depend on batteries for power supply, ranging from small mobile devices to very large high-powered devices such as batteries used for local storage in electrical substations. In most of these systems, there are significant user-benefits or engineering reasons to base the supply on multiple batteries, with load being switched between batteries by a control system. Unfortunately, due to the physical and chemical properties of batteries, it is possible to extract a greater proportion of the energy stored in a single battery of capacity C than of that stored in n batteries each of capacity C/n, for $n > 1$. Throughout this paper, when we refer to the relative efficiency in the use of multiple batteries, we mean the proportion of the charge we extract from them compared with servicing the same load from a single battery with capacity equal to the combined collection of batteries and equivalent physical properties. The key to efficient use of multiple batteries lies in the design of effective policies for the management of the switching of load between them. Note that we are concerned here with a situation in which the load can be serviced entirely by a single battery at a time, so the problem is distinct from the management of cells within a single battery. An example of a context in which multiple battery use occurs is in laptops equipped with multiple battery bays. More efficient use of multiple batteries can be achieved by exploiting the phenomenon of *recovery*, which is a consequence of the chemical properties of a battery: as charge is drawn from a battery, the stored charge is released by a chemical reaction, which takes time to replenish the charge. In general, charge will be drawn from a battery faster than the reaction can replenish it and this can lead to a battery appearing to become dead when, in fact, it still contains stored charge. By allowing the battery to rest, the reaction can replenish the charge and the battery become functional once again. Thus, efficient use of multiple batteries involves carefully timing the use and rest periods. This problem can be seen as a planning problem.

The paper is organised as follows: we describe the problem in detail and then show how the problem can be seen as a planning problem. We describe the approach we have adopted in solving the problem and then compare the performance of our solution with the best policies currently considered for multiple battery management.

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2 The Multiple Battery Usage Problem

The multiple battery usage planning problem has been explored by several authors, from an electrical engineering perspective (eg (Benini et al. 2003)) and also from a scheduling perspective (Jongerden et al. 2009). Benini *et al* construct a very accurate battery model, parameterising it to capture lithium-ion, cadmium-nickel and lead-acid battery types, and show how hand constructed policies can achieve efficiency, relative to a single battery, between 70% and 97.5%. To achieve this, the policy is constructed to select a new battery whenever the voltage of the battery currently servicing a load drops below a certain threshold. The next battery is selected according to one of four alternative policies (Benini et al. 2003):

- V_{max}: select the battery pack with highest state of charge.
- V_{min}: select the battery pack with lowest state of charge.
- T_{max}: select the battery pack that has been unused for the longest time.
- T_{min}: select the battery that has been unused for the shortest time.

The authors show that V_{max} is the best of these policies, tested on up to four batteries.

Jongerden *et al* (Jongerden et al. 2009) uses a model checking strategy, based on UPPAAL, to schedule battery use given a known load profile. The approach is based on the use of a different battery model, the Kinetic Battery Model, discussed in more detail below. This is a non-linear continuous model and the authors treat it by discretisation and scheduling to a horizon. This approach allows them to find highly effective schedules, but it does not scale well because of the need to use a fine-grained discretisation of the temporal dimension. It is worth emphasising, since it contrasts with our approach, that Jongerden *et al* work with a fixed size discretisation of time, allowing them to focus on scheduling the resources (batteries) into the load periods.

In deployed systems, the standard policies are typically static, based on rapid switching between available batteries. In fact, an optimal use of multiple batteries can be achieved theoretically by switching between them at extremely high frequency, when the behaviour converges on that of a single battery. Unfortunately, this theoretical solution is not achievable in practice because of the losses in the physical process of switching between batteries, as the frequency increases. Round-robin (which is similar to T_{max} above) or best-of-n (similar to V_{max} above) policies applied at fixed frequencies are the most commonly fielded solutions, but these often achieve less than 80% efficiency (Benini et al. 2003). The efficiency of the use of multiple batteries can be assessed both by the relative lifetime compared with a single battery (to be maximised) and by the number of switches required to achieve it (to be minimised).

2.1 Objectives

In this paper our objective is to construct policies for multiple battery problems, where load is modelled probabilistically using known distributions for load size, load duration and load frequency (or equivalently, the gaps between successive loads). The best deployed solutions typically deliver less than 80% efficiency, while the best published solutions deliver less than about 95% efficiency. We show that our approach, based on construction of optimising solutions to Monte Carlo sampled problem instances and their use in the construction of appropriate policies, produces robust solutions that deliver better than 99% efficiency, while using smaller numbers of battery switches than published policies. We use a well-established continuous battery model as the basis of our construction of optimising solutions and this raises challenges in the treatment of the non-linear mixed discrete-continuous optimisation problem, as we discuss below. The model is a first-order approximation of battery behaviour, but analysis has shown that it is one of the most accurate analytic models available and is, in fact, more accurate than the model used by Benini *et al* to test their policies (Jongerden and Haverkort 2009).

2.2 The Kinetic Battery Model

The battery model we use is the Kinetic Battery Model (Manwell and McGowan 1994; Jongerden et al. 2009), in which the battery charge is distributed over two wells: the available-charge well and the bound-charge well (see Figure 1). A fraction c of the total charge is stored in the available-charge well, and a fraction $1 - c$ in the bound-charge well. The available-charge well supplies electrons directly to the load ($i(t)$), where t denotes the time, whereas the bound-charge well supplies electrons only to the available-charge well. The charge flows from the bound-charge well to the available-charge well through a "valve" with fixed conductance, k. When a load is applied to the battery, the available charge reduces, and the height difference between the two wells grows. When the load is removed, charge flows from the bound-charge well to the available-charge well until the heights are equal again.

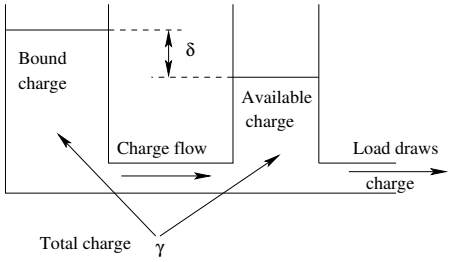


Figure 1: Kinetic Battery Model

To describe the discharge process of the battery, as in (Jongerden et al. 2009), we adopt coordinates representing the height difference between the two wells, δ, and the total charge in the battery, γ. The change in both wells is given by the system of differential equations (1), with solutions (2):

$$\frac{d\delta}{dt} = \frac{i(t)}{c} - k'\delta \qquad \frac{d\gamma}{dt} = -i(t) \tag{1}$$

$$\delta(t) = \frac{i}{c} \cdot \frac{1-e^{-k't}}{k'} \qquad \gamma(t) = C - it. \tag{2}$$

where $k' = k/(1-c)c$, $\delta(0) = 0$ and $\gamma(0) = C$, and C is the total battery capacity.

This model is less sophisticated than that used by Benini *et al* (Benini et al. 2001), but a comparison of battery models by Jongerden and Haverkort (Jongerden and Haverkort 2009) concludes that the Kinetic Battery Model (KiBaM) is the best for performance modelling.

2.3 Battery Usage Planning

The KiBaM is a deterministic non-linear continuous model of battery performance. This lends itself, in principle, to use in an optimisation problem solver that can find the best battery usage plan, given a load profile. However, in most real battery usage problems the load profile is generated by external processes, typically controlled directly or indirectly by user demands. These demands can often be modelled probabilistically, reflecting typical patterns of use. In our work we assume that the profiles are drawn from a known distribution. The consequence is that the planning problem ceases to be a deterministic optimisation problem, but a probabilistic problem in which the plan must be a policy.

The problem can be cast as a hybrid temporal Markov Decision Process, in which the states are characterised by the states of charge of the batteries, the current load and the currently active battery. Battery switching actions are deterministic, but the events that cause load to change are not. The time between events is also governed by a stochastic process, but the timing of switch-

ing actions is controllable. More formally, for a problem with n batteries, a state is characterised by the tuple $(sb_1, sa_1, sb_2, sa_2, ..., sb_n, sa_n, B, t, L)$, where sb_i is the bound charge in battery i, sa_i is the available charge in battery i, B is the number of the battery currently servicing load ($1 \leq B \leq n$), t is the time of the state and L is the current load. Out of each state there is a deterministic action, *Use* B', which causes a transition to the state $(sb_1, sa_1, sb_2, sa_2, ..., sb_n, sa_n, B', t, L)$, in which battery B' is the battery servicing load. There is also a non-deterministic action, *wait(T)*, where T is a time interval, which causes a transition to a state in which time has advanced to time $t' \leq t + T$, the state of charge of battery B is updated according to the battery model and the load might be different (according to the probability distribution governing loads). The interpretation of the action is that it advances time to the next event, which will be when a battery is depleted of available charge, or when the load changes, or when T time has passed, whichever is first.

A variety of approaches have been proposed for solving continuous Markov Decision Processes. Meuleau *et al* propose hybrid AO* search (Meuleau et al. 2009), using a dynamic programming approach to guide heuristic search for problems involving continuous resources used by stochastic actions. This approach does not handle time-dependent resource consumption, but it appears that the above MDP could be modelled for solution by this approach. The authors give empirical data for solution of problems with up to 25,000 states. Our model, with an appropriate discretisation, contains more that 10^{86} states for 8 batteries. Mausam and Weld (Mausam and Weld 2008) describe a planner for concurrent MDPs, which are MDPs with temporal uncertainty. Again, these problems are similar to ours, although their planner does not manage continuous time-dependent resources, so is not directly applicable to our problem. Furthermore, the largest problems they consider contain 4,000,000 states and take more than an hour to solve.

In solving very large MDPs, researchers have identified a variety of techniques that can help to overcome the prohibitive cost of policy iteration or value iteration, the classical techniques for solving MDPs. In general, these techniques approximate the solution, often focussing on those parts of the policy that apply to states that are likely to be visited along the trajectory. Relevant techniques are discussed in (Bertsekas and Tsitsiklis 1996). We use a variant of hindsight optimisation (see eg (Chang, Givan, and Chong 2000)), in which we solve a deterministic sampled problem using an optimising solver, to generate an ideal trajectory for the problem instance. Using a collection of such samples as our base, we then learn a classifier that characterises the policy for the part of the space we have sampled. A classifier is simply a function from one space to another. In our case, we learn a function from the state space to the actions. This approach is similar to other work built on the use of machine learning applied to policy rollout samples, particularly due to Fern, Yoon and Givan (2004; 2006; 2007), however that work addresses propositional domains while here we are interested in continuous problems.

In general, the samples-based approach works well when the policy structure is less complex to represent than the value function for the problem space. To extend the learned classifier to a complete policy involves ensuring that an action is assigned to every possible state. This can be achieved by adding some default behaviour to cover states that are otherwise not handled by the classifier, or else by managing run-time errors in the use of an incomplete policy in a way appropriate to the application.

2.4 Our Approach

We adopt an approach based on a combination of two ideas. Firstly, we sample from the distribution of loads to arrive at a deterministic problem, which we then solve using the continuous KiBaM as our battery model. This leads to an interesting continuous non-linear optimisation problem, which we solve using a discretise and validate approach. Currently we are using UPMurphi (Della Penna et al. 2009) to solve the deterministic instances but, after discretisation, any metric temporal planner could be used. Secondly, we combine the solutions to the sample problem instances to arrive at a policy for the MDP from which the problems are drawn. Our approach is domain-specific in some respects:

- Our discretisation scheme, while based on general principles, is selected for the problem domain and load distribution.
- We use a search heuristic that, while not restricted to the battery problem alone, is not suited to all problems.
- The aggregation of solutions into a policy also makes use of an entirely general approach, but the extent to which the approach yields good policies will depend on the nature of the problem space in which it is applied.

We make use of existing tools as far as is possible, to simplify the construction of our solution.

3 Solving Deterministic Multiple Battery Problems

In this section we consider the multiple battery management problem as an optimisation problem, when faced with a known and deterministic load profile.

3.1 A PDDL+ Battery Model

PDDL+ (Fox and Long 2006) is an extension of the standard planning domain modelling language, PDDL, to capture continuous processes and events. The dynamics of KiBaM can be captured very easily in PDDL+. In Figure 2 we show the two processes, consume and recover, that govern the behaviour of cells and the event triggered by attempting to load a cell once its available charge is exhausted. In addition, there is a durative action of variable duration that allows the planner to use a cell over an interval. The two processes are active whenever their preconditions are satisfied, meaning that they usually execute concurrently. Together, they model both the draining of charge and the recovery that are described in the differential equation $d\delta/dt$. An event is triggered if there is ever a positive load and no active service.

The use of PDDL+ as our modelling language grants several benefits. Firstly, it allows us to use VAL (Howey, Long, and Fox 2004) to validate solutions analytically against the

```
(:process consume
 :parameters (?c - cell)
 :precondition (switchedOn ?c)
 :effect (and (decrease (gamma ?c) (* #t (load)))
    (increase (delta ?c) (* #t (/ (load) (cParam ?c))))))

(:process recover
 :parameters (?c - cell)
 :precondition (>= (delta ?c) 0)
 :effect (and
    (decrease (delta ?c) (* #t (* (kprime ?c) (delta ?c))))))

(:event cellDead
 :parameters (?c - cell)
 :precondition
    (and (switchedOn ?c)
         (<= (gamma ?c) (* (-1 (cParam ?c)) (delta ?c))))
 :effect (and (not (switchedOn ?c)) (dead ?c)))
```

Figure 2: Part of PDDL+ encoding of KiBaM dynamics

continuous model, allowing us to confirm that the discretisation we use during construction of solutions does not compromise the correctness of the plan. Secondly, it provides us with a semantics for our model in terms of a timed hybrid automaton (following from (Fox and Long 2006)). Finally, we can make use of existing tools that construct and search in spaces defined by PDDL+ models, such as UPMurphi (Della Penna et al. 2009).

3.2 The Discretise and Validate Approach

Our system is based on a discretise-and-validate approach, in which the continuous dynamics of the problem are relaxed into a discretised model, where discrete time steps and corresponding step functions for resource values are used in place of the original continuous dynamics. This relaxed problem is solved using a forward reachability analysis and then solutions are validated against the continuous model using the validator, VAL (Howey, Long, and Fox 2004). The validation process is used to identify whether a finer discretisation is required and guide remodelling of the relaxed problem. VAL provides analytic solutions to differential equations involved in the models. Although Jongerden *et al* also use a discretisation approach, they fix the granularity of the time-step in advance. In contrast, we use a variable sized discretisation, by allowing a range of alternative sized step sizes to be considered during search.

We now introduce the formal statement of the deterministic version of the problem we are interested in. A *hybrid system* is a system whose state description involves continuous as well as discrete variables. We approximate the system by discretising the continuous components of the state (which we assume to be bounded) and their dynamic behaviours so obtaining a finite number of states.

Definition 1 (Finite State Temporal System) *A* Finite State Temporal System *(*FSTS*) $\mathcal{S}$ is a 5-tuple $(S,s_0,\mathcal{A},\mathcal{D},F)$, where: S is a finite set of* states*, $s_0 \in S$ is the* initial state*, $\mathcal{A}$ is a finite set of* actions*, $\mathcal{D}$ is a finite set of* durations *and $F : S \times \mathcal{A} \times \mathcal{D} \rightarrow S$ is the* transition function*, i.e. $F(s,a,d) = s'$ iff the system can reach state s' from state s via action a having a duration d. For each state $s \in S$, we also define the set* EnAct(s)= $\{a \in \mathcal{A} | \exists d \in \mathcal{D} : F(s,a,d) \in S\}$*, as the set of all the actions* enabled *at state s.*

In an FSTS, each state $s \in S$ is assumed to contain a special *temporal variable* t denoting the time elapsed in the current path from the initial state to s. In the following we use the notation $t(s)$ for the value of variable t in state s. For all s_i, $s_j \in S$ such that $F(s_i,a,d) = s_j$, $t(s_j) = t(s_i) + d$.

Definition 2 (Trajectory) *A* trajectory *in the FSTS $\mathcal{S} = (S,s_0,\mathcal{A},\mathcal{D},F)$ is a sequence $\pi = s_0a_0d_0s_1a_1d_1s_2a_2d_2 \ldots s_n$ where, $\forall i \geq 0$, $s_i \in S$ is a state, $a_i \in \mathcal{A}$ is an action, $d_i \in \mathcal{D}$ is a duration and $F(s_i,a_i,d_i) = s_{i+1}$. If π is a trajectory, we write $\pi_s(k)$, $\pi_a(k)$ and $\pi_d(k)$ to denote the state s_k, the action a_k and the duration d_k, respectively. Finally, we denote with $|\pi|$ the length of π, given by the number of actions in the trajectory, and with $\tilde{\pi}$ the duration of π, i.e. $\tilde{\pi} = \sum_{i=0}^{|\pi|-1} \pi_d(i)$.*

In order to define the planning problem for such a system, we assume that a set of *goal states* $G \subseteq S$ has been specified. Moreover, to have a finite state system, we fix a *finite temporal horizon, T,* and we require a plan to reach the goal within time T. In the case of the battery usage planning problem, this horizon is very important because it represents the target duration for the service provided by the battery. In fact, a good upper bound can be found for the battery problem, which is discussed further in section 3.3.

Definition 3 (Planning Problem on FSTS) *Let $\mathcal{S} = (S,s_0,\mathcal{A},\mathcal{D},F)$ be an FSTS. Then, a* planning problem *(PP) is a triple $\mathcal{P} = (\mathcal{S},G,T)$ where $G \subseteq S$ is the set of the goal states and T is the finite temporal horizon. A solution for $\mathcal{P}$ is a trajectory π^* in $\mathcal{S}$ s.t.: $|\pi| = n$, $\tilde{\pi} \leq T$, $\pi^*_s(0) = s_0$ and $\pi^*_s(n) \in G$.*

The constraints we add to the temporal planning problem are parameterised and can be iteratively relaxed in order to explore successively larger spaces for plans. We use a finite collection of possible durations for segments of processes (Definition 2). This set can be refined by the addition of smaller durations if successive searches fail to find a solution. Allowing different durations within the same search enables the planner to construct states that interact with executing processes at different time points, while stepping quickly along the timeline where there are no interesting features.

To use variable discretisation efficiently, we break the symmetry in the structure of the search space that arises from the possible orderings of different length action instances. Redundancy is eliminated by disallowing the use of long duration actions immediately following shorter duration versions of the same actions. Long duration actions can only be used if an event or other action has intervened since the last short action in the family. We also disallow the repeated consecutive use of short duration actions beyond the accumulated duration of the next longer duration action. The longest duration action can be repeated arbitrarily often.

3.3 The Monotonicity Property and Planning

The battery domain has an important property that supports a simple heuristic evaluation function for states: the charge in the battery monotonically decreases over time and the optimal solution is the one that gives the longest possible plan. An upper bound on the duration of the solution can be found using the observation that the optimal duration cannot exceed that of a single battery with combined capacity equal

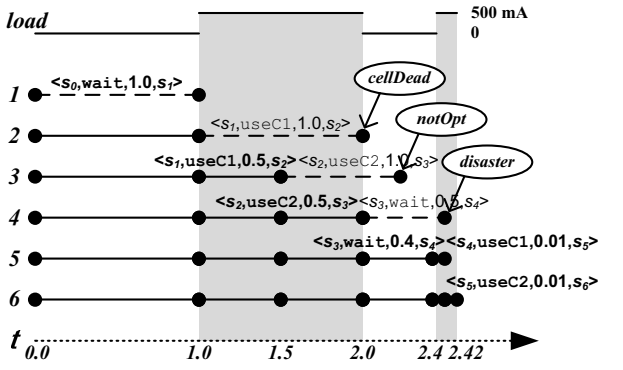


Figure 3: Example of search using variable discretisation

to the sum of the capacities of the multiple batteries (assuming the same discharging and flow behaviours). Once we have a horizon, we construct and search our discretised search space. To make this approach practical, it is essential that we have an informed heuristic to search the space. For this domain, duration of the plan to the current state plus total remaining charge is admissible, but completely uninformative, but duration plus total *available* charge is highly informative. This is also equivalent to minimising the total bound charge. We then use a best-first search to efficiently explore the reachable space.

This heuristic is suitable for a class of domains: any domain where there is a monotonically decreasing resource, and the longest plan is required (such as the satellite domain against a finite amount of resources), a heuristic that sums plan duration with available resource will be informative.

3.4 Plan Search with Variable Discretisation

We now illustrate the way in which the range of differently sized duration intervals can lead to significant benefits in the size of the set of visited nodes in the search space, compared with using a fixed duration increment.

Consider the load profile shown at the top of Figure 3. The planning problem for two cells is defined according to definitions 1 and 3, with $G = \{s \in S | t(s) = 2.42\}$, i.e. the goal is to service the whole load profile. The temporal horizon T is set to the duration of the profile as well. The definition of the FSTS is straightforward: the set of actions is $\mathcal{A} = \{\texttt{useC1}, \texttt{useC2}, \texttt{wait}\}$ where the former actions refer to the cell being used while the latter one is applicable when there is no active service. The set of durations we use for this example is $\mathcal{D} = \{0.01, 0.4, 0.5, 1.0\}$ (measured in minutes). In practice, to define the set of durations we start with a minimum value given by the time required for the decision making process, then we add exponentially increasing values up to a maximum duration given by the longest interval between different events (i.e., load variations). In particular, the smallest duration is included in order to handle very sensitive interactions.

In the initial state s_0 there is no load and no active service and both cells have a limited initial capacity. In this setting, the plan search with variable discretisation proceeds as follows:

1. No cell is used for a period of 1 minute (when the load is idle). The corresponding transition is shown in Figure 3.
2. After one minute a load is applied and cell 1 is used. This corresponds to transition $< s_1, \texttt{useC1}, 1.0, s_2 >$. However, for sake of simplicity, let us assume that, due to their limited capacity, cells cannot be used continuously for 1 minute. The transition is thus not valid and a shorter duration has to be considered.
3. Cell 1 is used for 0.5 minute. Then, since a load is still applied, the second cell is used. As before, the transition $< s_2, \texttt{useC2}, 1.0, s_3 >$ can be considered, but in this case there would be an active service and no load.
4. Cell 2 is used for 0.5 minute. In the next period no load is applied, then no cell is used. The transition $< s_3, \texttt{wait}, 0.5, s_4 >$ is considered, but it would lead to a positive load and no active service, so the duration of action `wait` has to be reduced to 0.4.
5. To service the last load period of 0.02 minute, cell 1 could be used. However, in this sample instance let us assume that the remaining charge in cell 1 allows it to service only 0.01 minute. So, finally, cell 2 is used until the end of the load profile.

The validity of a transition is dynamically checked during the search since invalid transitions trigger specific events (e.g. event `cellDead` is triggered at step 2 and event `disaster` is triggered at step 4) which, in turn, violates the invariant conditions of corresponding actions (a cell must not die during use). Moreover, with variable discretisation only 6 states have to be visited in order to reach the goal, while using a uniform discretisation it is necessary to explore at least 242 states since the finest discretisation of 0.01 must be used in order to correctly handle the interactions in steps 5 and 6.

A further benefit of the use of differently sized durations in the discretisation is that favouring longer durations reduces the number of switches in the solutions we generate, leading to solutions that are better in practical terms than those based on a high frequency switching between batteries, as is shown in subsequent results.

3.5 Performance on Deterministic Load Problems

We now present a first set of experimental results to show the performance of our solver on the deterministic battery usage optimisation problem. We use the same case study proposed in (Jongerden et al. 2009), where two types of jobs are considered, a low current job (250 mA) and a high current job (500 mA), according to the following load profiles:

- *continuous* loads: one load with only low current jobs (CL_250), one with only high current jobs (CL_500) and one alternating between a low current job and a high current job (CL_alt);
- *intermittent* loads with *short idle periods* of one minute between the jobs: one with only low current jobs (ILs_250), one with only high current jobs (ILs_500), and one alternating between a low current job and a high current job (ILs_alt);

load profile	best-of-two lifetime		UPPAAL-KiBaM lifetime		DD-KiBaM lifetime (visited states)		8 batteries B_2 lifetime (number of switches)		
	B_1	B_2	B_1	B_2	B_1	B_2	best-of-8	DD	DD-Policy
CL_250	12.16	46.92	12.04	N/A	**12.14** (194)	**46.91** (691)	310.6 (31072)	**307.6 (485)**	**307.6 (992)**
CL_500	4.59	12.16	4.58	N/A	**4.59** (116)	**12.14** (194)	134.7 (13472)	**133.4 (266)**	**133.4 (571)**
CL_alt	7.03	21.26	6.48	N/A	**7.03** (136)	**21.2** (350)	192.8 (19280)	**190.8 (355)**	**190.8 (806)**
ILs_250	44.79	132.8	40.80	N/A	**44.76** (552)	**132.7** (1068)	660.7 (33076)	**654.1 (495)**	**654.1 (904)**
ILs_500	10.82	44.79	10.48	N/A	**10.8** (131)	**44.76** (552)	308.7 (15476)	**305.7 (293)**	**305.7 (513)**
ILs_alt	16.95	72.75	16.91	N/A	**16.92** (159)	**72.55** (599)	424.8 (21280)	**420.6 (357)**	**420.6 (614)**
ILl_250	84.91	216.9	78.96	N/A	**84.88** (488)	**216.8** (1123)	1008.9 (33692)	**998.8 (471)**	**998.8 (822)**
ILl_500	21.86	84.91	18.68	N/A	**21.85** (173)	**84.88** (488)	480.9 (16090)	**476.1 (295)**	**476.1 (597)**

Table 1: System lifetime (in minutes) for all load profiles according to different battery usages

- *intermittent* loads with *long idle periods* of two minutes between the jobs: one with only low current jobs (ILl_250) and one with only high current jobs (ILl_500).

As a first step, we used these load profiles to validate our variable-range discretisation KiBaM model (DD-KiBaM), and to find an appropriate discretisation for the continuous variables involved in the system dynamics (i.e. variables γ and δ and process durations). To do this we used VAL to validate solutions for the discretised model against the continuous model, using single cell batteries. As in (Jongerden et al. 2009), we considered two battery types, one with capacity 5.5 Ahr (B_1) and one with capacity 11 Ahr (B_2). Both battery types have the same parameters: $c = 0.166$ and $k' = 0.122\text{hr}^{-1}$. We discretised γ and δ, rounding them to 0.00001, and, for all the load profiles above and for both battery types, we obtained the same lifetimes computed with the original KiBaM and validated in (Jongerden and Haverkort 2008).

To generate the scheduling plans for multi-cell batteries, we used the approach described in sections 3.2 and 3.3 and the set of durations $\mathcal{D} = \{0.01, 0.02, 0.05, 0.1, 0.25, 0.5, 1.0\}$. We compared our solutions to those obtained using the UPPAAL-based approach. The resulting lifetimes are shown in Table 1 where the second column shows the theoretical upper bound given by an extremely high-frequency switching. In all load profiles considered we observe that our approach outperforms significantly the UPPAAL-based one, providing solutions that achieve more than 99% efficiency compared with the theoretical limit. The key points described in the preceding parts of this section allow the resulting search to efficiently prune the state space and quickly find the solutions. In particular, by using variable discretisation it is possible to consider a much finer discretisation for variables γ and δ than is used in (Jongerden et al. 2009) and to handle very sensitive interactions. This is crucial, particularly when the available charge in the cells is almost exhausted. Jongerden *et al* (2009) describe their plans as optimal, but it is important to note that this is only with respect to the discretisation that they use; a finer-grained discretisation offers the opportunity for a higher quality solution to be found at the cost of a much larger state space. Despite the very large state space our model creates, the solver visits a very small collection of states (as shown in the table). These problems are all solved in less than a second.

Moreover, when dealing with larger batteries of type B_2, the state space becomes so large that any exhaustive approach is infeasible. Indeed, in (Jongerden et al. 2009; Jongerden and Haverkort 2008) the authors were not able to handle this second case. We also found high quality solutions for batteries of type B_2, an example is shown in Figure 4 compared with the V_{max} solution, showing the huge improvement we can obtain over this policy. Note that the slicing of the load periods occurs towards the end of the plan, and this is a phenomenon we have observed in all our plans.

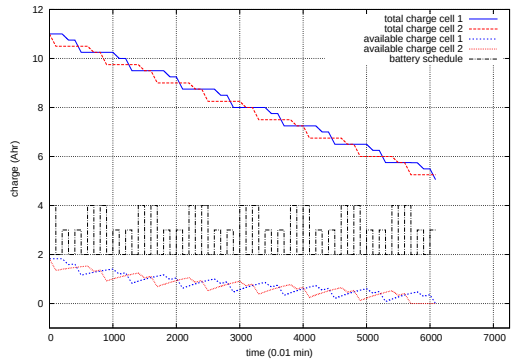
Our results also show performance on a 8 battery system (see Figure 5), showing that we can scale effectively to much larger problems. Notice that the number of switches we use to produce the results is very significantly smaller than the best-of-8 policy, however the resulting solutions achieve more than 99% efficiency. The final column, labelled *DD-Policy*, shows the performance of the policies we discuss in the next section, applied to these load profiles. These generate slightly worse performance in switches, but maintain the lifetime performance.

4 From Plans to Policies

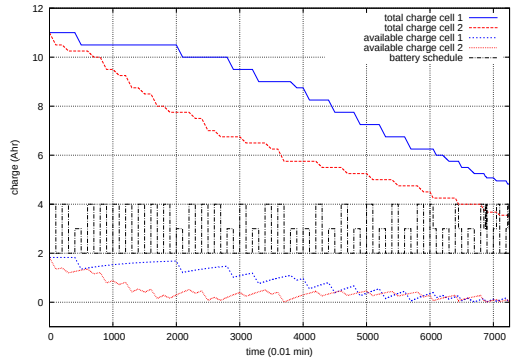
Having shown how to generate high quality plans for deterministic multiple battery management problems, we now turn our attention to the stochastic problem we are really interested in solving. In general, we cannot know in advance what will be the load profile applied to the batteries, but we assume that a probability distribution characterising typical use of the batteries is available. To manage the batteries in this setting we need a policy. Our policy will be a function that determines which battery to use when load must be serviced, using the current states of charge of the available batteries as the basis for making the decision.

One way to approach this problem is to see the mapping as a classification, where the state of the batteries is mapped to a class corresponding to the correct choice of battery. We can use the solutions to the determinised problems as the basis of a classifier construction problem and use an existing machine learning approach to build a good classifier.

Several important observations can be made. Firstly, the successful construction of a classifier depends on there being exploitable structure in the space defined by the solutions to the determinised problems. Secondly, the states are described by continuous variables: we discretise these for the purpose of building the classifier. Thirdly, our solution



(a) V_{max} (based on the feasible frequency switching used in (Jongerden et al. 2009))



(b) DD-KiBaM

Figure 4: ILs_alt load test with two batteries of type B_2

set will generally not cover the whole space of reachable states, so it is important that we complete the policy with a sensible default rule when the input state is too distant from any of the previously encountered states. In our case, the default rule is a best-of-n rule, which is the best of the published hand-constructed policies for this problem. Finally, we note that deployment of the final policies will require that they can be efficiently implemented in cheap hardware. Simple classifier rule systems can be very effectively implemented in look-up tables, which are ideal for implementation on Field Programmable Gate Arrays (FPGAs) or as purpose-built hardware.

4.1 Policy Learning through Classification

WEKA (Hall et al. 2009) is a machine learning framework, developed at the University of Waikato, that provides a set of classification and clustering algorithms for data-mining tasks. WEKA takes as input a training set, comprising a list of instances sharing a set of attributes. In order to perform the classification on the battery usage problem data, we consider inputs with the form $\tau = (\sigma_1, \gamma_1, \ldots, \sigma_N, \gamma_N, B, L)$, where σ_i and γ_i denote the available charge and total charge of the ith battery, respectively, B is the currently active battery and L is the current load (this is essentially the state, as described in section 2.3, but without the time label, since we want our policy to operate independently of time). In this setting, the attribute used as the class is the battery B.

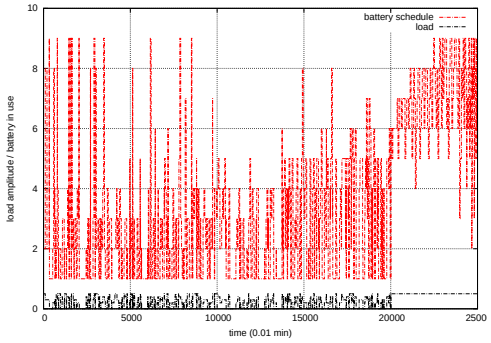


Figure 5: DD policy for 8 batteries with a stochastic load

We used stochastic load profiles with distributions:

- the load amplitude $l \in [100 \ldots 750]$ mA;
- the load/idle period duration $d \in [0.1 \ldots 5]$ min;
- the load frequency $f \in [0.3 \ldots 0.7]$.

This leads to load profiles that are very irregular (see the bottom of Figure 5) and therefore harder to handle than the very regular profiles considered by Jongerden *et al*. We generated a set of stochastic load profiles and for each of them we produced a near-optimal plan using the deterministic solving described in Section 3. This set of plans has been used as the training set for the classification process.

In order to select the most suitable classification algorithm, we applied all the classifiers provided by WEKA to a data set of 10,000 training examples. We first evaluated their *performance* as the number of correctly classified instances during the cross-validation. We discarded classifiers providing less than 70% correctness. We then considered the *memory* and the *time* required to use the classifier. The output of the classification process is a model encoding the resulting decision tree. In some cases, the generated model requires significant memory to store (more than 500Mb of RAM memory), or it is too slow to be used. These parameters have also been used to determine the number of training examples to classify, as the bigger the training set, the better the performance and the higher the memory and time requirements.

According to these criteria, we selected the `J48` classifier, which implements the machine learning algorithm C4.5. The output is a decision tree whose leaves represent, in our case study, the battery to be used. This classifier proved well suited to this task, as 99% of instances were correctly classified during the cross-validation performed by WEKA. For the cardinality of the training set, an empirical evaluation showed that the best result is obtained using 250,000 training examples (note that this involves considering about $4 \cdot 10^6$ real values) since further extending the training set does not make any significant improvement in the performance but increases memory and time requirements.

4.2 Results from Policies

In order to use the decision tree we embedded the WEKA classes for loading the classification model into our battery simulation framework. The model for the 8 battery case is represented by a tree with 61 levels and consists of 7645

load	best-of-8		DD-Policy	
profile	time$_{(\sigma)}$	$sw_{(\sigma)}$	time$_{(\sigma)}$	$sw_{(\sigma)}$
R100	792.6$_{(15.5)}$	71383$_{(1379)}$	**786.2**$_{(15.4)}$	**1667**$_{(161)}$
R250	369.8$_{(1.91)}$	28952$_{(853)}$	**366.7**$_{(2.02)}$	**1518**$_{(143)}$
R500	226.7$_{(2.13)}$	14671$_{(512)}$	**224.6**$_{(2.27)}$	**987**$_{(122)}$
R750	188.3$_{(0.8)}$	11519$_{(463)}$	**186.4**$_{(0.7)}$	**302**$_{(33)}$

Table 2: Average system lifetime and number of switches for stochastic load profiles for 8 battery systems

nodes, each one containing a comparison between one of the state variables and a threshold. Applying this decision tree to determine which battery to load at each decision point takes negligible time.

To evaluate the performance of the policy we considered four probability distributions with different average value for the load amplitude, namely 100, 250, 500, 750 mA. For each distribution we generated 100 stochastic load profiles and we used the policy to service them. Table 2 shows the average value and standard deviation for the system lifetime and the number of switches obtained using the best-of-8 policy at high frequency switching and our policy.

Also in this case, we observe that our policy achieves more than 99% efficiency compared with the theoretical upper bound given by the best-of-8 policy executed at very high frequency (recall that this is infeasible in practice). Moreover, the number of switches involved by the policy is slightly greater than in the corresponding deterministic solving, but it remains completely acceptable.

5 Conclusions and Future Work

In this paper we have presented an effective solution to an increasingly important multiple battery management problem. The best existing solutions are capable of delivering no better than 95% efficiency compared with a single battery, while our solution achieves better than 99% efficiency. Although this margin is small, in many applications a small margin can be of considerable added value. Our approach adapts several existing technologies for automated planning, to solve a problem that can be seen as a MDP. We use a form of hindsight optimisation, generating samples of determinised load profiles and solving these problems using an optimal deterministic solver, before combining the solutions to form a policy. Our policy construction approach adapts the use of machine learning to construct a classifier. In the construction of high quality solutions to deterministic problems, we use a special variable-range discretisation to solve a non-linear continuous optimisation problem with very high accuracy, while exploring a very small proportion of the state space.

Our approach is scalable and effective. Although our solution is domain-specific in several respects, the components are general. The elements that are most tailored to our problem are the selection of the discretisation range and the search heuristic. We are currently exploring techniques to automatically generate an appropriate discretisation range for a problem, based on an analysis of the problem instance characteristics and the nature of the dynamics in the domain description. We believe that the characteristics of this problem are shared, in outline, by other domains and we are exploring a more careful characterisation of the problem class to which the general framework can be applied.

Acknowledgements

The authors wish to thanks Marijn Jongerden and Boudewijn Haverkort for introducing them to this problem and for early discussions on approaches to solving it.

References

Benini, L.; Castelli, G.; Macii, A.; Macii, E.; Poncino, M.; and Scarsi, R. 2001. Discrete-time battery models for system-level low-power design. *IEEE Trans. Very Large Scale Integration Systems,* 9(5):630 –640.

Benini, L.; Macii, A.; Macii, E.; Poncino, M.; and Scarsi, R. 2003. Scheduling battery usage in mobile systems. *IEEE Trans. Very Large Scale Integration Systems* 11(6):1136 – 1143.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Chang, H. S.; Givan, R.; and Chong, E. K. P. 2000. On-line scheduling via sampling. In *AIPS*, 62–71.

Della Penna, G.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2009. UPMurphi: a tool for universal planning on PDDL+ problems. In *Proc. 19th Int. Conf. Automated Planning and Scheduling (ICAPS)*, 106–113.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*, 191–199.

Fern, A.; Yoon, S. W.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *J. Artif. Intell. Res. (JAIR)* 25:75–118.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)* 27:235–297.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11(1).

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. *ICTAI* 00:294–301.

Jongerden, M., and Haverkort, B. 2008. Battery modeling. Technical Report TR-CTIT-08-01, Centre for Telematics and Information Technology, University of Twente.

Jongerden, M., and Haverkort, B. 2009. Which battery model to use? *IET Software (Special Issue on Performance Engineering)* 3(6):445–457.

Jongerden, M.; Haverkort, B.; Bohnenkamp, H.; and Katoen, J.-P. 2009. Maximizing system lifetime by battery scheduling. In *Proc. 39th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN 2009)*, 63–72.

Manwell, J., and McGowan, J. 1994. Extension of the kinetic battery model for wind/hybrid power systems. In *Proc. 5th European Wind Energy Association Conf. (EWEC)*, 284–289.

Mausam, and Weld, D. S. 2008. Planning with Durative Actions in Stochastic Domains. *J. of AI Res. (JAIR)* 31:33–82.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam. 2009. A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *J. of AI Res. (JAIR)* 34:27–59.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Using learned policies in heuristic-search planning. In *Proc. Int. Joint Conf. on AI (IJCAI)*, 2047–2053.