Real-Time Predictive Optimization for Energy Management in a Hybrid Electric Vehicle

Alexander Styler and Illah Nourbakhsh

Abstract

With increasing numbers of electric and hybrid vehicles on the road, transportation presents a unique opportunity to leverage data-driven intelligence to realize large scale impact in energy use and emissions. Energy management in these vehicles is highly sensitive to upcoming power load on the vehicle, which is not considered in conventional reactive policies calculated at design time. Advancements in cheap sensing and computation have enabled on-board upcoming load predictions which can be used to optimize energy management. In this work, we propose and evaluate a novel, real-time optimization strategy that leverages predictions from prior data in a simulated hybrid battery-supercapacitor energy management task. We demonstrate a complete adaptive system that improves over the lifetime of the vehicle as more data is collected and prediction accuracy improves. Using thousands of miles of real-world data collected from both petrol and electric vehicles, we evaluate the performance of our optimization strategy with respect to our cost function. The system achieves performance within 10% of the optimal upper bound calculated using *a priori* knowledge of the upcoming loads. This performance implies improved battery thermal stability, efficiency, and longevity. Our strategy can be applied to optimize energy use in gas-electric hybrids, battery cooling in electric vehicles, and many other load-sensitive tasks in transportation.

Introduction

The advent of hybrid and electric vehicles yields a compelling new platform for artificial intelligence. These vehicles offer numerous high-level control opportunities managing state of charge, power fulfillment, battery temperature, and various other internal systems. As sensing and computation become cheaper, more complex intelligent controllers can be utilized to manage these high-level tasks in real-time. Intelligent control in these systems can yield improved energy efficiency, reduced emissions, and increased vehicle longevity.

In this work we demonstrate a data-driven adaptive optimization approach to manage energy in a passenger vehicle in real-time. Cheap sensing technology allows rich state data to be measured continuously, including GPS coordinates, speed, and power load. This state data can be leveraged to make real-time predictions of upcoming load based on prior observed state and load data. Given a set of independent sample load predictions, our approach calculates optimal controls in real-time with respect to the distribution defined by that set.

With accurate predictions, our approach produces near-optimal control decisions in hybrid vehicles, improving on existing fixed and reactive policy architectures. The load predictions are based on prior observed data, so the performance improves over the lifetime of a vehicle as more trips are recorded. Over time, data set coverage improves and unique trips become less likely. In practice, frequent commutes and errands comprise the majority of trips for many drivers and upcoming load is easily predicted. Continually collecting trip data over the lifetime of the vehicle enables long-term adaptation to changing routes, driver behaviors, and traffic trends.

We evaluate our optimization approach on a simulated hybrid battery-supercapacitor energy management task. In this vehicle, an electric battery acts as the primary energy source for vehicle range and operation. A supercapacitor is coupled to the system and acts as a small, highly efficient buffer. Either can be used to power the electric motor or receive energy from regenerative braking. The task is to minimize the sum of the current-squared on the battery pack, which helps improve battery pack longevity and thermal stability. The supercapacitor can be pre-charged from the battery before high-power loads such as steep hills or rapid accelerations. This task is analogous to other hybrid vehicle energy sourcing tasks, but is highly sensitive to prediction or control error due to the small size of the buffer.

We compare our predictive system to an prescient optimal controller that is given the upcoming demand *a priori*. Using real world data collected over thousands of miles of trips in several U.S. states, we test each control system in a hybrid electric vehicle simulation that can replay each trip. We compare a batch system that has all other data available to make predictions and an adaptive system that has only prior data chronologically and grows its data set over time.

Related Work

The rapid adoption of hybrid electric vehicles (HEVs) has sparked a great interest in the application of learning and intelligence to this domain. Several works have demonstrated that optimization of energy management in HEVs, using *a priori* knowledge of the power loads, results in superior performance over fixed reactive algorithms. The works show that either dynamic programming (Mosbech 1980) (Oprean et al. 1988) (Brahma, Guezennec, and Rizzoni 2000) or optimal control theory (Lyshevski and Yokomoto 1998) (Delprat, Guerra, and Rimaux 2002) can be used to effectively determine an upper bound on performance. The success of these retrospective optimizations highlights the importance of combining power load information with optimization techniques. These efforts motivated our approach to combine predictions of power loads with a dynamic programming optimization that can execute in real-time.

For real-time controllers, successful control has been demonstrated using rule-based and fixed point policies. An overview of existing HEV control strategies and papers is well covered in (Salmasi 2007). Rule based and reactive control can be successful using fuzzy logic controls or parameter optimization, but generally do not approach the bound provided by retrospective optimization. A popular approach to HEV control, known as Equivalent Consumption Minimization Strategy (ECMS), calculates future fuel expenditure required to offset the present battery energy expenditure. Fixed approaches are somewhat successful, but adaptive ECMS has been shown to perform very well for charge-sustaining mode (Musardo et al. 2005). These approaches are limited to charge sustaining mode. For plug-in HEVs, utilizing the full range of battery energy can pose significant improvement to fuel consumption, but at the cost depending on charge recovery from the grid while the car is parked. To optimally exploit charge depletion in a plug-in HEV, the total trip energy use and charging opportunities must be known.

Optimization of the energy management tasks using only the present vehicle state has limited potential for success. Driver variability and decision making has a large impact on the inputs and performance of these tasks. Work has shown that variability between drivers is significant (Ericsson 2000), and these variabilities have a large impact on vehicle operation (Holmén and Niemeier 1998) (LeBlanc et al. 1995). Additionally, researchers have shown that driver mood and behavior yield significant variability to load (De Vlieger, De Keukeleere, and Kretzschmar 2000). Due to the influence of the driver, optimization of vehicle management tasks ideally consider the total system including both the vehicle and driver. For instance, fuzzy logic controllers accounting for driver behavior patterns or trip type have shown marked improvement over reactive control (Langari and Won 2005), simply by tessellating categories of driver control and trip classification into a small number of discrete categories.

The importance of power loads and driver behavior for optimization has led many researchers to explore the potential of prediction. Significant work exploring the use of stochastic Markov chains to model driver behavior has achieved remarkable success (Opila et al. 2013) (Liu and Pentland 1997). These Markov chains can then be used to generate value functions and control policies using stochastic dynamic programming. While some work trains the Markov chains on a fixed set of federal driving data, the implementation by Cairano allows dynamic adaptation to new driving data over the life of the vehicle (Di Cairano et al. 2014). Furthermore, the work demonstrates that the optimization programming can be pushed offline, with simple table lookups at run-time to achieve rapid control. This leveraging of past data, real-time optimal control, and dynamic adaptation inspired similar goals for our system.

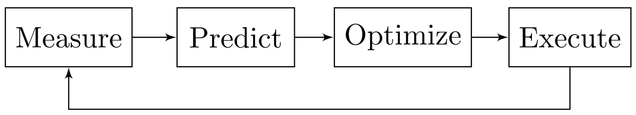

Figure 1: System Diagram of Control Loop. This four step process loop runs continuously, generating new controls as quickly as can be calculated. Changes in the measured state result in new predictions that are then used to calculate new controls.

While these approaches use single stochastic Markov chains, work has shown that training examples are more informative as separate independent experts (Styler and Nourbakhsh 2013). Selecting a set of local predictions, based on vehicle state, can achieve better performance than global solutions or averaged solutions. The set of predictions maintains information about uncertainty of the upcoming load, while avoiding the averaging effects of a single stochastic Markov chain. We use a similar kNN matching approach to load prediction. Our optimization approach exploits these independent predictions to achieve real-time optimal control.

We extend prior work by creating a solution that can achieve real-time optimal control with respect to distributions defined by sets of predictions. We present a complete, dynamic system that learns driver behavior, traffic trends, and routes to optimize towards efficiency, longevity, or emissions.

Algorithm

The control algorithm chooses discrete controls based on its sensor readings of the vehicle state and environment. Due to the high frequency of changing inputs and demands on a vehicle, the algorithm must be able to react quickly in real-time. It operates as a simple control loop described in the diagram shown in Figure 1. The vehicle state measurement is an instantaneous snapshot of the vehicle state, and does not directly inform an optimal control decision. However, this state measurement can be informative of the upcoming loads. The algorithm matches the present state to similar historical states in its dataset. Recorded power loads that followed each of these historical states are used as predictions of upcoming load on the vehicle. Given these sample predictions, and a candidate set of discrete controls, and a cost function, the algorithm calculates the one-step optimal control that minimizes expected cost-to-go. This control is executed continuously while the process repeats, until a new control is calculated, as shown in Algorithm 1. Fast computation is important to achieve rapid, responsive controls that can react to new information.

If the actual upcoming sequence of power loads were known *a priori*, the cost-to-go function would be an accurate representation of total cost that would be incurred over the remainder of a trip given a state and control. Choosing the cost-to-go minimizing control at each timestep would be

Algorithm 1 High-Level Control Algorithm. The feature vector, x_t, is matched to historical states to get a weighted set of predictions, $\{Z_0, w_0\}...\{Z_k, w_k\}$. The corresponding precomputed cost-to-go functions J_i are fetched for each prediction Z_i. The control u_t that minimizes the one step cost plus the expectation of cost-to-go is executed

```
loop
    $x_t \leftarrow$ ComputeFeatureVector($sensorvalues$)
    $\{Z_0, d_0\}...\{Z_k, d_k\} \leftarrow$ kNN($x_t, k$)
    $w_0...w_k \leftarrow 1/d_0...1/d_k$
    $w_0...w_k \leftarrow$ Normalize($w_0...w_k$)
    $\{J_0...J_k\} \leftarrow$ LookupCTGFun($Z_0...Z_k$)
    $\{u_t\} \leftarrow$ ArgMin$_u(c(x_t, u) + \sum_i w_i J_i(x'_i, t+1))$
    Execute($u_t$)
end loop
```

optimal, barring any discretization error. For a predictive controller, it is unlikely that a single prediction would accurately describe the upcoming load. A set of weighted predictions is used to account for noise and uncertainty inherent in the driving domain from intersections, traffic, or driver behavior. These weighted predictions define the predicted distribution of upcoming load. When they are in agreement, the algorithm will be very aggressive with its controls due to the certainty of the distribution. When the predictions disagree, however, the controls will be more conservative to mitigate the uncertainty. For example, when a vehicle approaches an uncertain intersection, the controls will be conservative, considering each likely possibility. As the intersection is decided, confidence increases and the controls can be more aggressive.

Computing a cost-to-go function using Dynamic Programming can be very expensive as the number of states, controls, or timesteps increases. However, most vehicles have significant downtime, often overnight, which yields plenty of opportunity for offline computation. Our optimizer utilizes this explicitly by computing cost-to-go function tables offline that are simply referenced at run-time. This allows the agent to use rapid interleaved planning and execution without sacrificing solution quality.

State Measurement

The algorithm initially takes a measurement of the present vehicle state. The state space can consist of many potential features that can be measured with a handful of cheap sensors. Features accessed from the vehicle computer include speed, acceleration, and power demand. GPS coordinates, bearing, and time of day require external sensors. Such features are invaluable to make load predictions in vehicles. Hybrid and electric vehicles can also provide information on battery state of charge, temperature, and current load. Additional meta-features, such as recent variance in acceleration, may be calculated if needed. The system architect should instrument the vehicle and select the features necessary to make informed predictions using historical data. This feature vector representing the instantaneous vehicle state is then passed to a prediction system. More features that inform the upcoming duty can yield better predictions and performance.

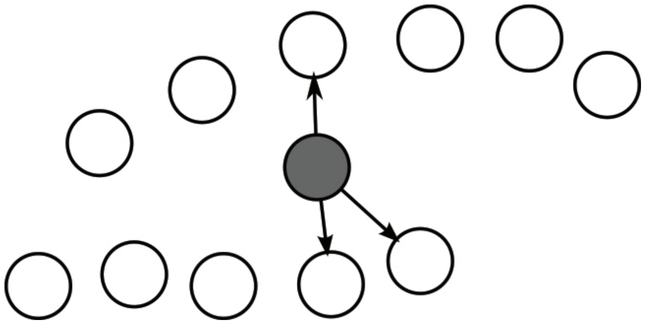
Figure 2: Example kNN State Matching. In this 2D example, the present state, in gray, matches to the three closest training examples in white. These states are from historical trips, and each contains an index into a specific time in that trip.

Load Prediction

Given this state feature vector, the algorithm then make predictions of the vehicle's upcoming power load. These predictions are difficult to calculate analytically, requiring a known route, topology data, driver behavior, and traffic information. Instead, the algorithm uses prior observed loads from previous trips as predictions. The present state vector is matched to similar states in the dataset, and the observed loads that followed those similar states are used as predictions. This is only accurate if upcoming load is informed by the instantaneous state vector. This assumption is central to the success of the algorithm: if the state measurement is independent of the upcoming load, the predictions will be useless.

The prediction step matches the present vehicle state to historical training data using a nearest neighbor comparison, as shown in Figure 2. The dataset is searched to find the k states that have the lowest weighted Euclidean distance, d, to the present state, x:

$$d = \sqrt{w_1(x_1 - z_1)^2 + ... + w_m(x_m - z_m)^2} \qquad (1)$$

For each training state, z, the distance is computed in the feature space of dimensionality m. Each feature has an importance weight, w, that can be defined by the architect or tuned using a simple optimizer such as gradient descent. The features should be normalized to be on the same scale so that the relative weights have physical meaning when inspected.

Each of the k matches is assigned an importance weight calculated as the inverse of this distance. Therefore, training states with very similar features to the present state will have high importance weight associated with their prediction. The distance function can be modified to suit the application, but non-linear distance functions can preclude many kNN algorithms. For our linear distances, a standard kd-tree approach is used to minimize computation time for matching. The predictor returns the importance weight and indices identifying the source trip and time index for each training state match. Each of these indices references an exact point in time in a prior trip in the dataset, for which all power load data has been recorded. These weighted predictions are then passed to the optimizer system.

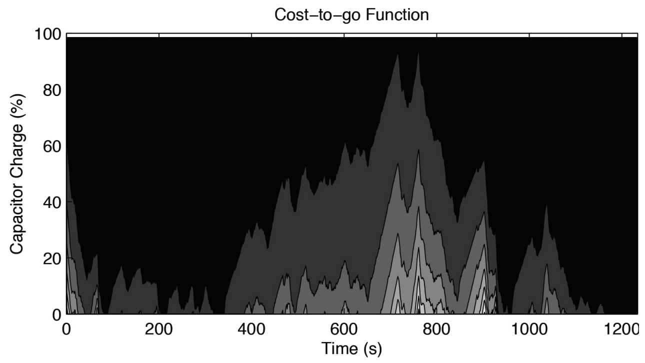

Figure 3: Example Cost-To-Go Function. The function encodes the expected cost-to-go for a given state at each timestep. Brighter values represent higher cost-to-go. In this example, an empty capacitor at 700s will result in very high cost being incurred.

Control Optimization

Given a set of predictions, we want to choose the optimal control that minimizes expected cost-to-go. For a discrete set of controls and discrete timesteps, a brute force approach that tests each control at each timestep is exponential in the lookahead time and quickly becomes intractable. Instead, for each separate prediction, we compute the cost-to-go function, $J(x,t)$, for each task state, x, and timestep, t, using Dynamic Programming and the Bellman Equation:

$$J(x,t) = \min_u [c(x,t,u) + \gamma J(x', t+1)] \tag{2}$$

Here, γ is a discount factor, x', denotes the state resulting from executing the control, u, and c represents the one-step cost function. The state transition function, H, is calculated using the predicted load, L_t, at each timestep in simulation:

$$x' = H(x, u, L_t) \tag{3}$$

For our tested domain, x is the state of charge of the capacitor. Our transition function is a simple model that calculates the change in state of charge for a given load power, L_t, and charging power, u.

An example cost-to-go function is shown in Figure 3. Using Dijkstra's Algorithm, computing this function and calculating all the state transitions requires $O(X * U * T)$ simulations, where X is the number of states, U the number of possible controls, and T the number of timesteps. It is unrealistic to compute these functions for multiple predictions at run-time.

However, each training example, or prior trip, has a fixed sequence of loads and thus fixed transitions. Therefore, this cost-to-go function can be precomputed offline and stored for each trip. For typical state and timestep resolutions, this can be done overnight with reasonable simulation complexity. At run-time, the fixed cost-to-go function for each trip matched by the predictor is simply loaded from memory. This pushes the bulk of optimization computation offline as these cost-to-go functions do not need to be computed on-the-fly.

Given a set of weighted predictions, $\{Z_0, w_0\}...\{Z_k, w_k\}$, and a starting state, x, the algorithm calculates the expected cost-to-go of each candidate discrete control and selects the best:

$$u_t = \arg\min_u c(x,t,u) + \mathbb{E}_Z[J(x', t')]$$
$$u_t = \arg\min_u c(x,t,u) + \sum_i (w_i J_i(x', t')) \tag{4}$$

Here, J_i represents the cost-to-go function for prediction Z_i and w_i denotes its weight; x' and t' represent the resulting state and timestep when executing control u given the state transition function for that prediction. The resulting u_t is the one-step cost-to-go optimal control with respect to the set of weighted predictions. As the cost-to-go and transition functions are precomputed offline, very little computation is required at each timestep.

Execution

The selected control that minimizes the expected cost-to-go is then executed. The control loop repeats, starting with a new state measurement, to calculate the next control.

Training

After a trip is complete, the observed data is added to the the training dataset. This training occurs offline when the vehicle is not in use, so as not to monopolize computational resources. This continual training enables the vehicle to adapt to new routes, traffic patterns, or driver behaviors.

For a trip, the vehicle states measured at each timestep are added to the kNN tree. Then, the loads over the course of the trip are used to calculate the state transition function. Using Dynamic Programming, this transition function is used to calculate the cost-to-go function, $J(x,t)$, at each state and timestep. This function is then added to the database to be referenced at run-time.

Experiment

We test performance on a simulated prototype hybrid electric vehicle (EV), using real-world power data from drivers. The simulated vehicle combines a high-power supercapacitor with a high-energy battery pack. The optimization task is to minimize current-squared on the battery pack by leveraging the capacitor as a buffer. We implement our algorithm and compare its performance at various parameterizations to an prescient optimal controller that has *a priori* knowledge of upcoming power loads. We additionally compare the performance of dynamic learning with leave-one-out batch learning.

Data Collection

To evaluate our algorithm, we used a dataset of real-world trips collected daily from a set of volunteer drivers. We needed repeated data collected from individual vehicles so we could test long-term learning and prediction. We constructed the dataset by instrumenting eight different petrol vehicles

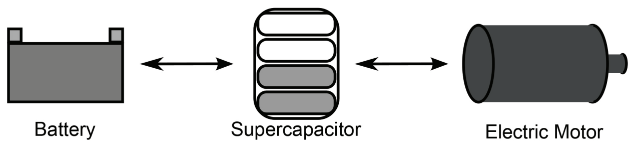

Figure 4: Simplified Diagram of Supercapacitor Buffer. The supercapacitor in this hybrid electric vehicle system can act as a buffer on the battery pack.

with sensors and collecting data from them over the course of ten months.

This data was run through a standard longitudinal vehicle model to calculate the expected power loads on an EV battery pack at each timestep. The model was calibrated using three different EVs by collecting real-world power load data at the terminals of their battery packs. The EVs were instrumented in the same way as the petrol vehicles, and the model was tuned to minimize RMS error between the calculated power load and observed power load. Validation and explanation of this model and data can be found in our previous work (Styler et al. 2011).

State data was recorded every second, limited by the frequency of the data supplied by the GPS receiver. The state data recorded from the vehicles include GPS coordinates, time, elevation, speed, acceleration, and power load. Using this data, we can simulate our algorithm on each of these vehicles.

Hybrid Electric Vehicle Prototype

The hybrid EV prototype combines a battery and supercapacitor to create a heterogeneous energy pack, as shown in Figure 4. This pack can handle high power loads with ease while offering high energy density for sufficient driving range. The supercapacitor pack in this prototype EV has a much higher power density than the battery pack. This allows it to meet high power demands, such as rapid accelerations or regenerations, with great efficiency. However, these supercapacitors have very low energy density, and cannot be used for long before depletion. The battery pack is extremely energy dense but performs poorly at high power loads. Under high load, such as accelerating up a hill, the battery pack generates a significant amount of heat and loses efficiency due to the Peukert Effect. External cooling systems also require additional energy, reducing overall vehicle efficiency.

Ideally, the supercapacitor would handle all high power loads, and the battery pack would only handle low power loads and low power charging of the capacitor. Intelligent predictive control is needed to optimally pre-charge or discharge the capacitor. For the simulated hybrid EV prototype, the control lever is the total power drawn from the battery. Any excess or shortage of power, with respect to the demanded power load, is handled by the supercapacitor buffer. This allows the vehicle to charge or discharge the supercapacitor, while meeting the driver power demands, with a single control variable.

Cost Function and Setup

The system architect defines a degree of freedom for control and an objective cost function to minimize. For this experiment, the algorithm will control the charging and discharging of the capacitor by controlling battery pack output. It is constrained to meet all demands created by the driver and to operate within the safety limits of the energy pack.

To minimize high power loads on the battery, the one-step cost function calculates the square of current on the battery given the voltage, V_t

$$c(x,u,t) = \frac{u}{V_t}^2 \tag{5}$$

The difference between the battery power output, u, and the load demand is handled by the supercapacitor. The battery power output can be constrained if the supercapacitor is at or near its charge limits.

The total cost incurred for a trip is the integral of the current squared from the battery over time. Using this cost function results in the desired behavior of pre-charging of the capacitor during idle or low power loads in order to handle upcoming high power loads later. This will result in lower thermal output and possibly longer longevity of the battery, but neither are modeled in the scope of this work.

The state feature vector used for our prediction is:

$$\begin{vmatrix} GPS & Bearing & Time & Day \\ Speed & Acceleration & EnergyUsed & \end{vmatrix} \tag{6}$$

The frequency of control decisions for the simulation is set to $1Hz$, to match the frequency of the state data contained in a dataset. In an actual implementation, it can operate at higher frequency if the sensors measuring vehicle state can supply new information faster.

Each driver dataset is tested independently and the results are combined into a single performance measure.

Results

We test and analyze the performance in our simulator using the real-world data. We compare performance against an upper bound calculated by a prescient optimal controller with *a priori* knowledge of the upcoming load. We also compare to a simple buffer method and a controller optimizing against the mean of the weighted predictions. We compare varying number of predictions k with a fixed $50Wh$ capacitor size. We also show sensitivity to varying capacitor size with a fixed $k = 7$. Finally, we compare the dynamic dataset that grows chronologically with a leave-one-out batch dataset of all the data.

Performance Comparison

The results of various methods are compared with the algorithm, shown in Figure 5. The baseline represents the total cost incurred for a battery pack without a supercapacitor. The buffer method only stores regenerated energy and spends it as soon as possible. It simply demonstrates an additional baseline with no intelligent control, and is not meant to reflect the performance of fixed point or reactive policies.

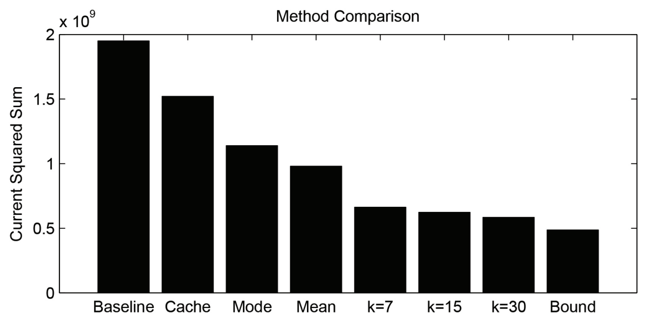

Figure 5: Comparison of Methods. This shows the improvement in current squared for each method over the baseline. High number of predictions, such as $k = 30$, approach the upper performance bound. Marked improvement over mean or mode predictions is observed.

The mode method only uses the single most likely prediction for optimization, and it is thus more subjective to prediction error and noise. The mean method averages the load predictions of $k = 7$ neighbors into a single prediction and calculates the optimal control. Due to this combination of predictions at run-time, offline computation of the cost-to-go function is not possible and the algorithm cannot run in real-time. These approaches serve as an indication of the limits of approaches that do not consider uncertainty.

We observe the predictive algorithm achieves $> 91\%$ of the performance improvement possible with $k = 30$.

Sensitivity

The sensitivity of performance to number of neighbors and capacitor size was tested. The number of neighbors increases performance, with diminishing returns, at the cost of marginal computation time. The size of the supercapacitor affects the ability to buffer large demands and the sensitivity of the algorithm to prediction error.

The sensitivity to the number of neighbors is shown tangentially in Figure 5. Here we observed improved performance as k increases, but with diminishing returns. Each additional neighbor has a lower importance weight attached to the prediction. As k increases, these weights become insignificant to the overall optimization and have negligible effect on the control outcome.

Supercapacitor size greatly affects the performance of the agent, shown in Figure 6. Excessively large capacitors have significant headroom, allowing control mistakes to occur with little penalty to overall performance. As the capacitor size is reduced, the algorithm is more susceptible to prediction error, and mistakes in control from inaccurate predictions can incur significant cost. The performance relative to the optimal prescient agent is shown at six sizes, $\{5, 10, 50, 100, 200, 1000\}$ Wh.

Dynamic Performance

The performance results shown so far were calculated using a batch leave-one-out cross validation (LOOCV) test. For each trip tested, every other trip in the dataset is used as training

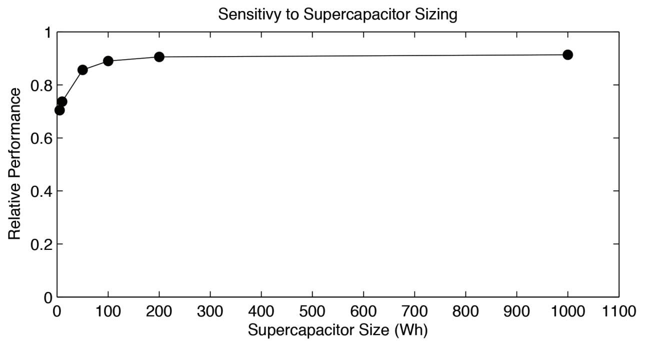

Figure 6: Sensitivity to Supercapacitor Size. The graph shows the sensitivity of performance to the capacitor size. Larger capacitors are less sensitive to prediction error as the large buffer allows for many mistakes.

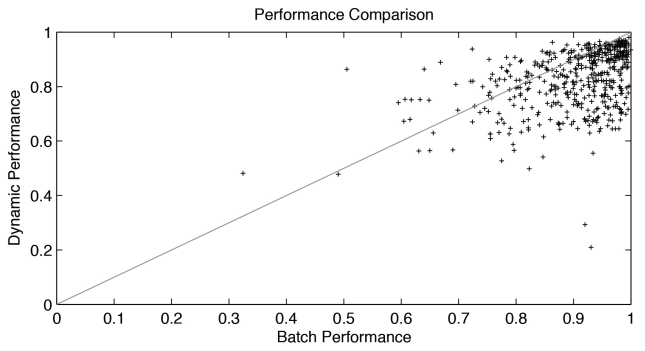

Figure 7: Performance Comparison of Batch and Dynamic Systems. The graph compares individual trip performance using the batch or dynamic learners. In most cases the batch learner performs better due to better coverage.

data. This eliminates any possible bias effects due to data ordering or dataset division, but demonstrates results after months of driving data has been collected. To evaluate the dynamic performance, we also tested how well the algorithm performs through a simulated lifetime of the vehicle. The data was tested chronologically, in the order it was collected, and each trip was added to the training data after testing.

The results comparing the batch performance to the dynamic performance are shown in Figure 7. Each point represents the performance of a trip, relative to an optimal upper bound, for the batch and dynamic case. The majority of trips perform better in the batch case, with more training data, and only a few outliers perform significantly better in the dynamic case due to noise.

Future Work

The next extension to this work is implementation on a high fidelity plug-in HEV simulator. A common problem in plug-in HEV control is sustaining charge over the course of a trip, so electric power is always available. Our algorithm can predict charging events to optimally deplete the battery when appropriate. Also, it can optimally distribute the use of electric power between charging cycles. This extension would allow the direct comparison of this approach to many existing methods.

Other sustainability domains sensitive to spatiotemporal loads may also be optimized by leveraging past data. One such possible application is HVAC control in an office building, where human behavior patterns influence load.

Conclusions

In this work, we have demonstrated we can achieve real-time near-optimal control of an energy management task by leveraging predictions from prior data. A dynamic system can constantly improve performance by expanding the dataset to improve data coverage. Future uncertainty and prediction error will bound the performance of this approach.

Our optimization approach, the independent combination of the cost-to-go functions at run-time, allows almost all computation to be done offline. A handful of simulations, a tree search, and a few table lookups are computed at run-time, which can be done faster than real-time on even modest hardware. This real-time prediction and optimization approach achieves performance close to retrospective optimizations using *a priori* knowledge of the upcoming power loads.

In simulation, we achieved $> 91\%$ of the improvement achieved by the prescient upper bound. This performance is excellent and suggests our optimization approach could be successful for other high-level vehicle optimization tasks.

We have demonstrated the potential for data-driven intelligence and real-time optimization in this transportation sustainability task. Our approach could be used for other load-sensitive sustainability problems with adequate predictions. The environmental implications of this work can be realized with further research.

Acknowledgments

The authors gratefully acknowledge the contributions of BMW, Bosch, Bombardier, Google, the ChargeCar team, the CREATE Lab, our data volunteers, and reviewers. This material is based upon work supported by the National Science Foundation under Grant No. (0946825).

References

Brahma, A.; Guezennec, Y.; and Rizzoni, G. 2000. Optimal energy management in series hybrid electric vehicles. In *American Control Conference, 2000. Proceedings of the 2000*, volume 1, 60–64. IEEE.

De Vlieger, I.; De Keukeleere, D.; and Kretzschmar, J. 2000. Environmental effects of driving behaviour and congestion related to passenger cars. *Atmospheric Environment* 34(27):4649–4655.

Delprat, S.; Guerra, T. M.; and Rimaux, J. 2002. Control strategies for hybrid vehicles: optimal control. In *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, volume 3, 1681–1685. IEEE.

Di Cairano, S.; Bernardini, D.; Bemporad, A.; and Kolmanovsky, I. 2014. Stochastic mpc with learning for driver-predictive vehicle control and its application to hev energy management. *Control Systems Technology, IEEE Transactions on* 22(3):1018–1031.

Ericsson, E. 2000. Variability in urban driving patterns. *Transportation Research Part D: Transport and Environment* 5(5):337–354.

Holmén, B. A., and Niemeier, D. A. 1998. Characterizing the effects of driver variability on real-world vehicle emissions. *Transportation Research Part D: Transport and Environment* 3(2):117–128.

Langari, R., and Won, J.-S. 2005. Intelligent energy management agent for a parallel hybrid vehicle-part i: system architecture and design of the driving situation identification process. *Vehicular Technology, IEEE Transactions on* 54(3):925–934.

LeBlanc, D. C.; Saunders, F.; Meyer, M. D.; and Guensler, R. 1995. Driving pattern variability and impacts on vehicle carbon monoxide emissions. *Transportation Research Record* (1472).

Liu, A., and Pentland, A. 1997. Towards real-time recognition of driver intentions. In *Intelligent Transportation System, 1997. ITSC'97., IEEE Conference on*, 236–241. IEEE.

Lyshevski, S. E., and Yokomoto, C. 1998. Control of hybrid-electric vehicles. In *American Control Conference, 1998. Proceedings of the 1998*, volume 4, 2148–2149. IEEE.

Mosbech, H. 1980. Optimal control of hybrid vehicle. In *Proc., International Symp. on Automotive Technology & Automation (ISATA80)*, volume 2, 303–320.

Musardo, C.; Rizzoni, G.; Guezennec, Y.; and Staccia, B. 2005. A-ecms: An adaptive algorithm for hybrid electric vehicle energy management. *European Journal of Control* 11(4):509–524.

Opila, D. F.; Wang, X.; McGee, R.; and Grizzle, J. 2013. Real-time implementation and hardware testing of a hybrid vehicle energy management controller based on stochastic dynamic programming. *Journal of Dynamic Systems, Measurement, and Control* 135:021002.

Oprean, M.; Ionescu, V.; Mocanu, N.; Beloiu, S.; and Stanciu, C. 1988. Dynamic programming applied to hybrid vehicle control. In *Proc. of the International Conf. on Electric Drives (ICED 88)*, volume 4, D2.

Salmasi, F. R. 2007. Control strategies for hybrid electric vehicles: Evolution, classification, comparison, and future trends. *Vehicular Technology, IEEE Transactions on* 56(5):2393–2404.

Styler, A., and Nourbakhsh, I. 2013. Model predictive control with uncertainty in human driven systems. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Styler, A.; Podnar, G.; Dille, P.; Duescher, M.; Bartley, C.; and Nourbakhsh, I. 2011. Active management of a heterogeneous energy store for electric vehicles. In *Integrated and Sustainable Transportation System (FISTS), 2011 IEEE Forum on*, 20–25. IEEE.