Optimizing Stanford's Cooling Expenditures

Augustine G. Chemparathy
Department of Bioengineering
Stanford University
Stanford, CA 94305
<agchempa@stanford.edu>

Joan Creus-Costa
Department of Physics
Stanford University
Stanford, CA 94305
<jcreus@stanford.edu>

Shreya S. Shankar
Department of Computer Science
Stanford University
Stanford, CA 94305
<shreya1@stanford.edu>

Abstract

In order to decrease Stanford's electricity footprint, Stanford Energy Systems Innovations (SESI) reduces campus electricity expenditures by preemptively purchasing hot and cold water throughout the day and deploying it hours later to meet campus heating and cooling needs. This project optimizes SESI's purchasing strategy based on predicted electricity prices, campus cooling load, and campus electricity load. We used random forest regression to predict all three variables, and then value iteration on a Markov Decision Process to output an optimal electricity purchasing strategy over a seven day period. Our results include the implementation of an optimal policy solver given ideal data, and altogether an approximate solver using our predictions. Our results show comparable results to SESI's current policy but are noticeably cheaper in terms of pure electricity cost, potentially realizing significant savings to Stanford University if implemented.

1 Introduction

As part of Stanford's efforts to reduce electricity consumption and cost, the next-generation power facility Stanford Energy Systems Innovations (SESI) features groundbreaking technology to reduce electricity expenditure for heating and cooling buildings on campus. Up to 42% of Stanford's energy consumption comes from heating and cooling its buildings, so efforts to make heating and cooling utilization are crucial to decreasing the university's carbon footprint.

Heating and cooling requirements change over the course of a day, motivating a system that intelligently caches and deploys heat over the day. While the campus may require cooling at midday — which involves absorbing heat back from buildings — it might be necessary to redeploy heating when the sun sets. If instead of just venting that heat away it was possible to store it somewhere, the electricity costs (and hence carbon footprint) would be reduced. This is essentially what SESI does: it uses cold and hot water tanks with capacities of up to 90,000 tons to help cache and deploy heat throughout the day.

Stanford strategically buys electricity from the grid to recharge its cold and hot water tanks at an optimal cost minimum. This presents a formidable optimization problem: deciding when to buy electricity from the grid to compensate for future heating and cooling storage needs. Our project tackles the issue of finding the optimal cold water production policy by hour, where the cost of producing all the water necessary to cool Stanford's buildings is minimized.

CS221: Artificial Intelligence – Principles and Techniques. Stanford University.

Our approach predicts market electricity cost and cooling loads across Stanford campus for a seven day period. Using these predicted data, we determine a seven day optimal strategy that purchases electricity to recharge the SESI cold water tank when electricity is cheap and deploys cold water to Stanford campus to meet demand.

2 Literature Review

The most relevant related work to this project is the existing algorithm used by SESI for this purpose. It was created by Joseph Stagner, executive director of the Stanford Department of Sustainability and Energy Management (SEM). It was published as Patent US 20120215362 A1.[1] It handles the more general case of both heating and cooling, but is implemented in an Excel spreadsheet. This algorithm operates by looking one week in advance and trying to determine the optimal policy for solely the following day. Its inputs include campus electrical load, campus heating and cooling load, hot and cold water distribution load, and forecasts of outside temperature (wet and dry bulb temperatures). The actual algorithm is implemented as more than a hundred lines of iteratively executed conditionals and loops in Excel.

Closely related work is another implementation of a hot/cold water allocator by Johnson Controls (a professional company that develops power plant controls and HVAC systems) that used dynamic programming. This algorithm had a performance equivalent to the one developed by Joseph Stagner in Microsoft Excel.

A lot of previous work has happened relating to the machine learning parts of this project. One of the challenges was to be able to predict the price of electricity for one to two weeks. A review paper by Rafal Weron shows the state of the art methods used for that purpose.[2] It's a field that has grown rapidly since the early 2000s. One of the factors that complicates prediction of electricity costs is that the prices are determined are determined by auctions and bidding between power companies in the free market. As such, some of the approaches fall in what the paper calls "multi-agent" and "fundamental" models, that try to characterize the behavior of power companies and the economy.

We will instead try to focus on statistical methods, since this is what we have at hand given our non-expert knowledge of the system and the fact that we're interested in statistical AI methods. According to the paper, most statistical methods try to look for "similar days" to the one that we're trying to predict (like day of week, holiday, weather) and sum or multiply all those factors. Other techniques try to predict the "spikes" in electricity cost—since, in the end, it is those spikes that drive our optimizer.[3] However, predicing electricity prices is ing eneral a very hard problem. According to a personal communication with Mr. Joe Stagner, there exists highly confidential professional software services dedicated exclusively for this specific kind of prediction, showing the complexity of the problem.

The other parts of the machine learning part of this project include modelling the electricity and cold water loads of Stanford. While no models have been written for this, as far as we are aware, similar techniques as the ones explained above can be applied for the same purpose.

3 Data Provided

SESI has provided us data to use for the machine learning portions of our project. Specifically, for our project, we consider:

- Hourly electricity prices from September 1, 2015 to October 31, 2016
- Hourly chilled water loads from September 1, 2015 to October 31, 2016
- Hourly background building electrical loads from September 1, 2015 to October 31, 2016
- Hourly outside temperatures from September 1, 2015 to October 31, 2016
- SESI's cold water production policy from September 1, 2015 to October 31. 2016

[1] `https://www.google.com/patents/US20120215362`
[2] `http://www.sciencedirect.com/science/article/pii/S0169207014001083`
[3] `http://www.ncer.edu.au/papers/documents/WPNo70.pdf`

"Chilled water load" refers to the quantity of cold water deployed to meet the cooling needs of Stanford campus. "Background building electrical loads" refers to the total electricity use of Stanford campus. SESI's cold water production policy is the quantity of cold water currently being produced following SESI's current algorithm. The output of this existing policy provides a significant real-world benchmark to compare our results with.

4 Task Definition

For our project, we consider Stanford's hourly cooling needs in tons of cold water per hour, hourly electricity prices, rate of cold water production, and rate of cold water storage transfers – and try to find a policy that minimizes total cost of cold water production while satisfying Stanford's cooling needs.

Our system operates under the following constraints:

- The power plant can produce up to 20,000 tons of cold water each hour.
- The cold water tank can hold at most 90,000 tons of cold water at a time.
- The storage tank can only be recharged or discharged at a maximal rate of 18,000 tons per hour, and cannot be simultaneously recharged and discharged.
- We want to minimize the peak power consumption (including background electrical loads), since Stanford gets charged for the maximum power used at a given hour. This extra charge is known as the "demand charge" and is determined by multiplying the maximum electricity consumption in an hour of all hours in a month by the demand rate, which is fixed at $7.60 per kilowatt-hour.

Our project has two main components, which includes a machine learning portion and an optimization problem. First, we use machine learning to predict hourly electricity prices, chilled water loads, and background electrical loads for a week. Then, we use our predicted data to compute an optimal strategy of how much cold water to produce at every hour.

Our input for the machine learning portion is a year's worth of hourly data on electricity prices, chilled water loads, background electrical loads, and temperatures (used as a feature in the machine learning models described later in this paper). Our output from the machine learning problem and input for the optimization problem is a week's worth of hourly predicted electricity prices, chilled water loads, and background electrical loads. Our output from the optimization problem is a policy of how much cold water to produce at each hour over the course of a week, which consists of 24×7 data points.

Hence our evaluation metric can be to create some policy on how much cold water to produce for a 7-day period, and compare the cost to the actual policy chosen by SESI and the Oracle policy.

N.B. Based on discussion with SESI staff, we determined that, because any quantity cold water is stored for at most a few hours in SESI tanks, thermal radiation does not significantly dissipate heat in storage. Therefore, this loss term is not modeled.

5 Approach

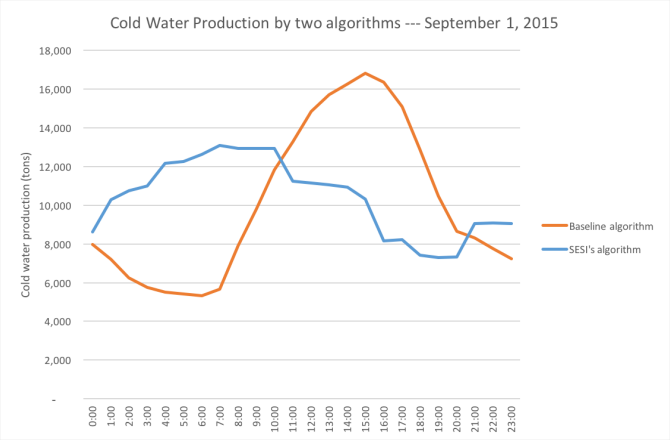
As mentioned before, our project incorporates machine learning and optimization techniques. We use machine learning to generate data and construct an optimal policy of cold water production based on the predicted data. For the optimization portion of the project, we run value iteration on a Markov Decision Process, which is explained more in this section.

5.1 Baseline

For a baseline, we assume that there exists no storage tank for the water, so SESI must produce exactly the amount of water that Stanford's buildings need every hour. Since we do not know how much cold water the buildings will need or what the price of electricity is, we cannot know the cost of this baseline policy of water production until the water has been produced. But for this baseline, we can produce

cold water as Stanford buildings need air conditioning. Calculating what the baseline would give us for the year's worth of data (electricity prices and Stanford building needs) we have, we get a cost of 1.9 million dollars! We get this baseline cost by multiplying the number of tons of cold water needed per hour, hourly electricity price, and conversion factor of 0.5 kilowatt hours per ton-hour. This cost does not even include the demand charges.

Currently, SESI has its own policy to produce cold water. It is more efficient than our baseline because it tries not to produce cold water during peak or afternoon hours. For example, the graph of SESI's cold water production for September 1st, 2015 is below:



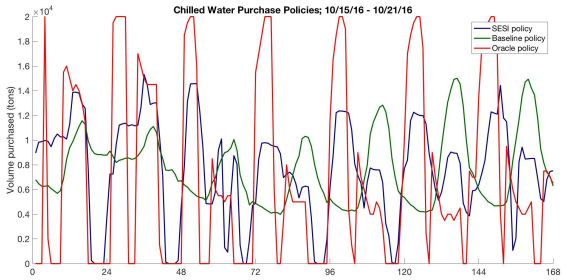
5.2 Oracle

The Oracle runs the value iteration algorithm using actual (not predicted) hourly chilled water loads, background electrical loads, and electricity price data for a given seven day period. This Oracle assumes perfect foresight of loads and prices, and because value iteration finds the global optimal policy if run for sufficient iterations, this approach outputs a perfect allocation strategy.

For the week of 10/15/2016 to 10/21/2016, the Oracle gives an allocation strategy that costs \$18, 309.76 for electricity to produce cold water, and \$237, 477.20 for the peak electricity demand cost for the month. For the demand cost, we will only consider the quantity \$237, 477.2/4 = 59, 369.30 because the peak demand cost is a monthly cost, and we are only interested in the weekly cost (this is an informative way to stress the fact that the monthly demand charge is not as important as the weekly policy, since it's only a one-time thing). We assume there are 4 weeks in a month. So, our Oracle's total cost is \$77, 679.06.

SESI's policy costs: \$64, 481.25 for the demand cost (we similarly divided by 4 weeks in a month), \$21, 529.77 for the electricity to produce cold water – which costs \$86, 011.02 total for that week.

Here is the graph of the Oracle's optimal cold water production policy versus SESI's policy:



Notice the dramatic spikes in the Oracle's policy. The Oracle knows the exact prices, chilled water loads, and background building loads, so it can be extremely optimal and produce 20,000 tons of cold water at an hour that the background building load is quite low. From SESI's data, the highest background

building electrical load is near 30, 000 kilowatt-hours, and the lowest background building electrical load is less than 20, 000 kilowatt hours. Since a ton of water corresponds to 0.5 kilowatt-hours, it makes sense to stockpile as much water as possible when we're guaranteed to have the cheapest price and lowest background building electrical load in a week.

The Oracle saves about 8, 300 dollars a week!

5.3 Data Prediction: Machine Learning

Since we want to predict hourly data, we perform 24 different regressions – one regression for each hour. We tried many regression approaches from the sk-learn library, but we ended up using sk-learn's Random Forest Regressor with $n = 100$ estimators for each of the 24 regressions. Our features are the date of the year, the previous day's average value, day of the week, week number, and hourly predicted temperature (can be pulled from weather sources).

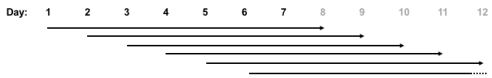
For our testing data, we use about 25% of our total data. The hourly data for each day mod 4 (day 4, 8, 12, etc) is used as testing data. For our training data, we use the remaining 75% of our total data.

sk-learn includes a way of measuring how accurate the predictions are, which is the *score*() function on a regressor. According to sk-learn documentation, *score*() "returns the coefficient of determination R^2 of the prediction," and the "best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse)."

The results from the different regression models we tried are in the "results" section of this paper.

5.4 Optimization: Markov Decision Process

We model the purchase allocation problem as an MDP. Each state corresponds to a tuple of current volume of cold water in storage, and time. We try to obtain an optimal policy for each day in a seven day period. For every day in the seven day period, we do a seven day lookahead (as visualized below).



Despite the fact that it looks like a mere search problem, the reason to model it as a Markov decision process is twofold:

- First, while we know our initial state, we do not know what state we will end up in, since that depends on Stanford's actual demand of cold water. But the idea behind SESI is that they generate the policy once a day, and run it presumably offline, so if the next state were not exactly what we predicted, we would have no idea what we have to do. If we instead run value iteration over possible states, we will always have a policy.
- Second, when using machine learning we have some variance in the data, and some inherent uncertainty as to where we will end up. To avoid overfilling the tank (which wastes water) or underfilling the tank (which means we have to buy out of pocket live) we want to clarify that uncertainty in the model.

Start State: $s_{start} = (\text{Initial Storage}, 1, 0)$ If the MDP is run for Day 1, we assume that the tank begins at full capacity (90,000 tons). If the MDP is run for any other day, we assume that the tank begins at the capacity that the previous day's policy leaves it at. The hour is set to 1 in the start state. Our current maximum demand charge is 0 (because nothing has happened yet).

End Test: $\text{IsEnd}(s) = [s[2] = 168]$ The MDP only does a seven-day lookahead, in keeping with SESI's convention, so the end state is reached at the end of seven days — that is, when the hour becomes $7 \times 24 = 168$.

Transition Probability: $T(s, a, s') = \frac{N(s_2'|s_2-l,\sigma_l)}{\sum_x N(x|s_2-l,\sigma_l)}$ The transition probability accounts for uncertainty in ML predictions by modeling the probability of a state transition as a Gaussian. We essentially discretize the Gaussian distribution according to our state discretization and assign non-zero probability

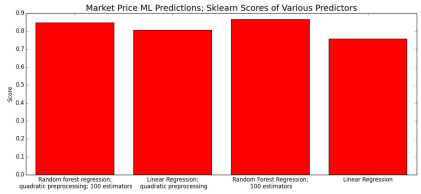


Figure 1: Market price machine learning predictions.

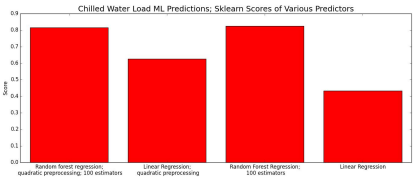


Figure 2: Chilled water load ML predictions.

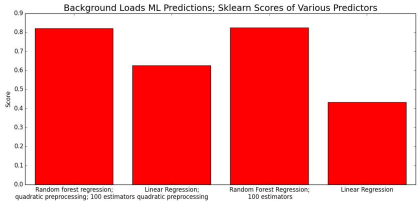


Figure 3: Background load machine learning predictions.

to the states close to the most likely one. The hour increases by one, and the third state (maximum demand charge) increases if the current wattage is bigger than before.

Reward: $R(s, a) = -$Cost of Purchased Electricity The reward represents the cost of taking the action — that is, the cost of purchasing sufficient electricity to produce the planned quantity of chilled water (i.e. $a[2]$). This reward is the negative product of the cost of electricity during an hour, the quantity of chilled water produced, and a conversion factor from tons of water to dollars. If we are at the final hour, we add a penalty corresponding to the maximum peak charge: $R'(s, a) \leftarrow R(s, a) - 7.6 \cdot s[3]$ if $IsEnd(s)$.

Discount Factor: $\gamma = 1$ For simplicity, we use a discount factor of γ because the cost of future chilled water produced and expended has equal value to more recent expenses.

Actions: $Actions(s) = \{$(Chilled Water Produced), (Change in Chilled Water Stored)|Action is legal$\}$ An action is a legal tuple of (1) tons of cold water produced *de novo* and (2) change in cold water stored; the value in the tuple has to obey the conditions described in the Task Definition section of this paper.

6 Experiments and Results

6.1 Data Prediction: Machine Learning

Figures 1–3 show the different average scores for the 24 regressions for different models we tried:

Our best scores for predictions are: 0.8657 for electricity prices, 0.8149 for chilled water loads, and 0.8251 for background electrical loads.

However, when trying to compute an optimal policy, we want to predict about 7 days of data at a stretch, not every fourth day in a set of data for a year. Figures 4–5 show the predicted compared to the actual price and chilled water load data for October 15, 2016 to October 21, 2016.

We get average score values of around -0.31 for the prices, 0.12 for water load predictions, and 0.23 for the background load predictions.

6.2 Optimization: Markov Decision Process

To run and test our MDP, we picked 3 different weeks to compute cold water production policies for. We used our machine learning approach to generate the input data for the MDP (electricity prices,

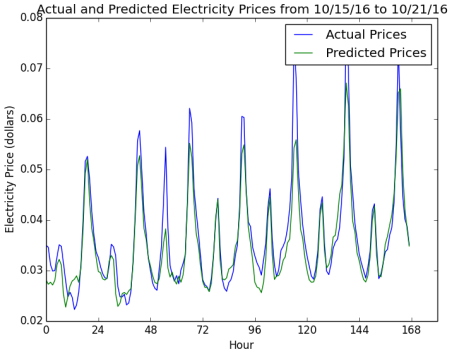


Figure 4: Actual and predicted market prices.

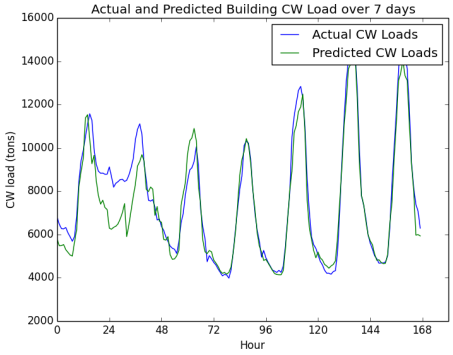


Figure 5: Actual and predicted building cold loads.

chilled water loads, background electrical loads), and we then ran our MDP with the 7 day lookahead to compute a policy of cold water production for the 3 different weeks.

The weeks we tested on are: 6/27/2016 - 7/3/2016, 8/26/2016 - 9/1/2016, and 10/15/2016 - 10/21/2016. To calculate the cost of cold water production alone, we performed $\Sigma_{h=1}^{168}(\text{Water Produced}_h * \text{Electricity Price}_h * 0.5)$, where h represents the hour, and we sum over 24 hours $*7$ days or 168 hours. To calculate the peak electricity demand charge, we performed $\$7.60 * (max_{h \in [1,168]}(\text{Water Produced}_h + \text{Background Electrical Load}_h))$. We add these two values to get a total cost. But we keep in mind – if we were to compute the chilled water production policy forever, we would only include the peak demand charge once a month instead of once a week. So, we only consider the peak demand charge divided by four, since we are calculating the policy for a week. This division is purely to stress the fact that the demand charge is not too relevant on the scale of a week, which is what we're computing, since in the end what matters is the entire month.

Here are the costs of our cold water production policies for the three weeks we run value iteration on:

Week	Demand Charge	Cost of Water Production	Total
6/27/2016 - 7/3/2016	71,831	22,953	94,784
8/26/2016 - 9/1/2016	68,352	20,751	89,103
10/15/2016 - 10/21/2016	68,732	17,837	86,569

In the Analysis section, we will compare these results to SESI's policy.

7 Analysis

7.1 Results

From the machine learning predicted data, we see that the Random Forest Regressor generally does best. Specifically, the Random Forest Regressor with quadratic preprocessing works best for predicting chilled water loads, and the Random Forest Regressors without any preprocessing work best for predicting market electricity prices and background electrical loads. Random Forest is better than Linear Regression – probably because not all of our features are linearly related to the predicted data, so a tree-based model can predict the output better.

From the graphs in the results, we see that the machine learning portion of our project does not accurately capture prices very well, but it does capture the trends (peaks and troughs) of the time series data. Our predictions worked decently well when using scattered days in the year (every fourth day) as test data, but since we must predict a week's worth of continuous hourly data for the MDP, our predictions are not as accurate.

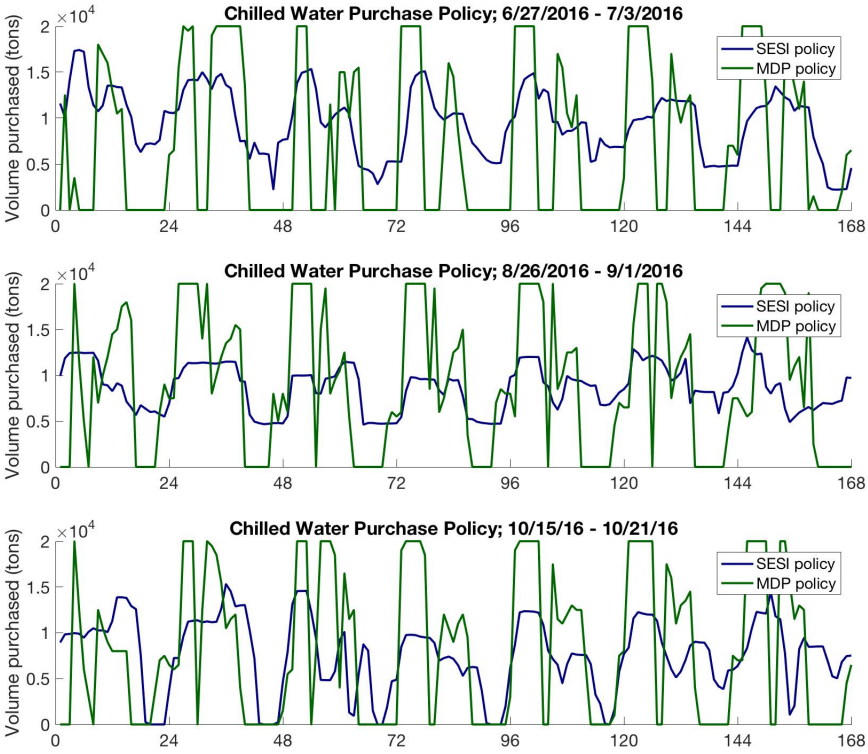


Figure 6: Comparison of different policies.

Since the trends are captured, our value iteration generally knows that buying electricity in the day is worse than in the middle of the night, so the optimization portion is not too terribly affected.

The policy stockpiles cold water when electricity is cheaper, in the very early morning and mid-morning, and buys *minimal* cold water when electricity is expensive, in the early morning and in the afternoon.

Here are the costs of SESI's cold water production policies for the three weeks we test on:

Week	Demand Charge	Cost of Water Production	Total
6/27/2016 - 7/3/2016	67,563.93	29,765.37	97,329.3
8/26/2016 - 9/1/2016	62,464.36	27,533.99	89,998.35
10/15/2016 - 10/21/2016	64,481.25	21,529.77	86,011.02

Figure 6 has graphs of the difference between SESI's policies and our policies for the three weeks we test on. As the data shows, our policy generally does well in optimizing the cost of cold water production alone, but when coupled with the total background building electrical loads, our peak demand charge is greater than SESI's peak demand charge. In some weeks, our policy does better than SESI's (for example, 6/27/2016 to 7/3/2016), but in other weeks, our policy does worse than SESI's (for example, 10/15/2016 to 10/21/2016). However, our policy consistently beats SESI in the weekly cost, which is arguably a more important metric: there are four weeks in a month, and the monthly peak is a one-time charge: a single price peak will determine the entire charge for the month and is not thus too relevant.

For instance, in the first week from the tables above (where we outperformed SESI by a big margin) the margin becomes even greater if we look at the entire month: the difference becomes $10k a month.

7.2 Error Analysis

Because our Oracle achieves significant saving over SESI's policy, and our MDP runs the exact same value iteration — but with the predictions — we can surmise that the more limited savings of the MDP due to uncertainty in our machine learning predictions.

As mentioned before, our machine learning algorithms capture the trend in electricity prices and load data relatively well – for example, we can reasonably predict that loads and prices are lower in the middle of the night than at peak afternoon hours – but the actual numerical data isn't predicted very well. This directly affects our demand charge because the demand charge relies purely on the exact load and exact price at a given hour. If we don't estimate these accurately, then our algorithm might think it's appropriate to cool a lot of water at a certain hour on a certain day, when in fact, it might be more optimal for the peak demand hour to occur on a different day.

Our value iteration approach produces markedly different results from SESI's algorithm in part because it exhibits dramatic behavior in response to fluctuations in electricity price and background cost. This response is akin to "Bang-bang control" in control theory, in which a system with strict numerical constraints shows on/off behavior in response to small perturbations in inputs. Although dramatic, this approach is optimal. We noticed that the MDP responded to an early morning decrease in electricity prices and background load to buy 20, 000 tons of chilled water — the maximum purchasable quantity. We notice that this strategy is timed to hours when the background load dips sufficiently low that even with the maximal chilled water purchase, electricity use does not exceed the peak, so an increase in demand charge is not incurred!

8 Challenges

The biggest challenge faced in the machine learning portion of the project was in predicting the hourly market electricity prices. This data varies dramatically with time, and since electricity prices are a few cents per kilowatt-hour, it is extremely important to be as accurate as possible (to the fifth or sixth decimal place). As discussed in the literature review section, predicting market electricity prices is a pretty difficult problem in itself.

A major challenge for the MDP is the vast increase in the state and action space – without a way to compute things quickly, we are required to discretize in smaller quantities or produce/store water in multiples of 500 or 1000 tons. In order to determine an optimal policy at a resolution on the order of a ton, we coded all of our value iteration using a GPU and coding using OpenCL, which we will explain in the following section.

9 GPU Programming

We chose to write code for the Graphics Processing Unit (GPU) as the solution to tackle the size of the search and action space. Because of the unique characteristics of our project, we implemented value iteration from scratch using OpenCL. This way we can take advantage of the parallelism offered by the hundreds of cores and even more threads that those specialized processors offer—the ones we used, an integrated Intel HD Graphics 5500 and an Nvidia GTX750Ti, provide 364 and 1305 gigaflops, respectively.

Value iteration is conceptually particularly simple to parallelize. At every time step, we need to iterate through every state, and update the utility using the Bellman equation with the previous iteration's utilities. Each of these updates is independent of the others, using only the information in the previous time step. While it is true that there are variants of value iteration that can happen stochastically, the convergence might be different, and GPUs provide the necessary parallelism for value iteration by themselves.

As a result, we can compile a kernel that performs value iteration for a given state for a given time step. This kernel needs to be written in a variant of C used in OpenCL programming. After that, we can parallelize this kernel to a vector containing all the current utilities and last iteration's utilities. Each

state is updated, and in the end the two utility vectors are swapped—this way we do not need to move data between the GPU and the CPU, which is a very expensive operation due to limited bandwidth.

The results of that approach were extremely successful. We observed a speed-up of about 500 compared to the original CPU implementation (compiled with the fast Intel C++ compiler), with each kernel spending about 0.5 µs per state. Due to the size of the state space and the number of iterations required to converge, each seven-day lookahead value iteration still took about 3 minutes, which still allowed for multiple runs and is significantly better than the entire day that the same operation would have taken on a CPU.

10 Conclusions and Future Work

In our project, we produce a working model to predict electricity prices, chilled water loads, and background electricity loads, even though it is not very accurate. Although the Random Forest model can capture trends in electricity prices, water loads, and background electrical loads, it does not accurate predict the prices, water loads, or background electrical loads. But our Markov Decision Process and value iteration algorithm leads to an optimized, cheap policy of producing cold water to cool Stanford's buildings. If we use other models from SESI for our machine learning predictions, we may be able to consistently save money.

To extend our project, we can change the MDP's discount factor γ to reflect the idea that our machine learning predictions are more accurate for the next day than they are for the sixth or seventh day. This would place more importance on generating the optimal cold water production for the first few days in a week, which is okay because we end up recomputing a weekly policy at the beginning of each day.

Another interesting extension would be to formulate it as a Partially Observable Markov Decision Process (POMDP) given that we are uncertain about electricity prices. This uncertainty has a negative effect on our policy finding: if our predictions show even a tiny gap in electricity prices, the MDP will try to use it—and possibly incur a big cost, because it didn't consider the fact that the prediction could be wrong and prices could be off. While we handled this problem with the loads by having non-one transition probabilities, in the case of price a POMDP approach might be warranted.

Additionally, we can incorporate heating costs into our optimization problem – in addition to the cooling costs we are currently accounting for. Similar to cold water production, heating Stanford's buildings requires a fixed amount of hot water every hour, and we can heat and store water in another tank as necessary.

Acknowledgments

We would like to thank Mr. Joe Stagner, director of SESI, for sharing the data with us as well as being very helpful throughout the development of the project. Additionally, we are grateful for our project TA Andrew Han and his support throughout the course.