ORF 418: Optimal Learning Final Project
Optimal Charging Station Locations

Kevin Sun
Princeton University
E-mail: kjsun@princeton.edu

Yash Patel
Princeton University
E-mail: ypatel@princeton.edu

Abstract. Tesla and electrical vehicles (EVs) have become more prevalent in the last decade. With the great rise in projected growth in EVs, the issue of placing electrical charging stations has grown to the forefronts of customers' and business owners' minds alike. We seek to address this problem, namely by investigating policies to determine the optimal locations to place electrical charging stations in a city setting. For this task, we developed a lookup-table model, with altered updating equations, and tested a few learning policies, in the forms of online and offline Knowledge Gradient Exploration (KG), Interval Estimation (IE), Boltzmann Exploration, and Pure Exploitation. Upon doing so, we found that the **Knowledge Gradient Policy** was the most effective in maximizing our total usage over all stations within our time horizon. We therefore, recommend it as a baseline for building future policies in this context of maximizing station utilization. Future studies may wish to expand upon the bottleneck employed in the model for charging stations and also time inhomogeneity.

Contents

1. Introduction

1.1. Motivation

In the past decade, the electric vehicle industry has been expanding. This expansion has particularly exploded in the last few years, with the introduction of Tesla and its vehicles into the market. Tesla has not only compelled people previously using electric cars to stick with their decision but has also made the usage of such cars for mainstream audiences more appealing. In particular, this year saw the release of the Tesla Model III, furthering the expansion of electric vehicles by pushing its cost into a very reasonable range, affordable by most middle class families.

Already, Tesla is projecting that they will sell "80,000 to 90,000" vehicles by this year's end – a huge increase from that seen in years past [1]. But, even more staggering is the extent to which electric vehicle usage is projected to rise. As seen in the graph below, the current rise in electric vehicle usage is expected to be completely dwarfed by its projected increase over the next five years.

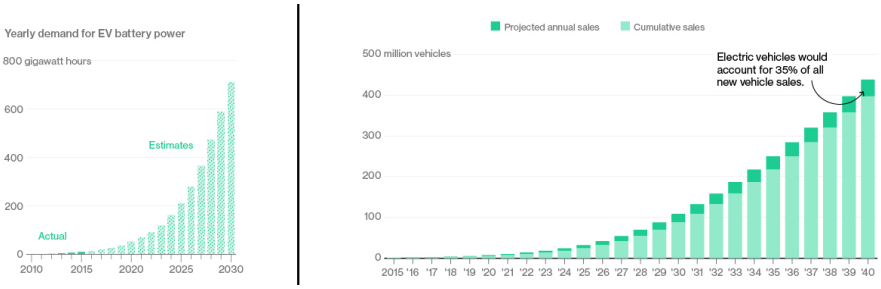

Figure 1: (Left) Depiction of electrical battery power consumed for the purposes of EV (electric vehicles) in years past with projections up until 2030. Clearly, there has been a rise since 2010, but it is extremely small compared to the projected increase in years to come. (Right) Similarly, we see the direct data dealing with EV sales, rather than the power consumed by them, reflecting a similar fate. [2]

This rise in popularity has sparked great interest in the related fields of optimization, which we wish to delve into herein.

1.2. Problem Description

Just as the rise of cars saw the birth of gas stations, the growth of EVs is rapidly seeing the establishment of electric car charging stations. However, because the EV industry is rather young, there is a deficiency in EV charging stations–the small number of stations that are currently in service are located only in the most populous regions of the US. With sales of Tesla and other companies' EVs growing nationwide, this must change in order for charging stations to properly accomodate their rising demand. In particular, highly populated areas such as cities will see multiple stations appear very close to each other and more sparsely populated areas will see the introduction of charging stations where they were completely absent before.

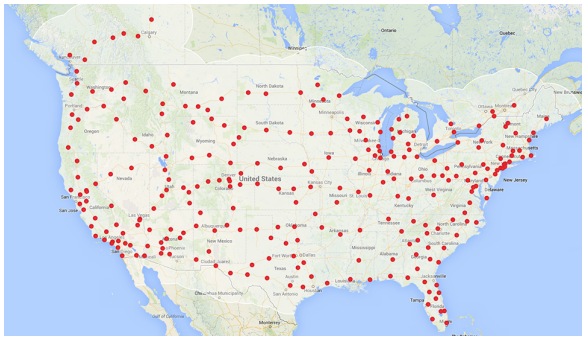

Figure 2: While Tesla has done a rather decent job so far in placing charging stations across the US, it has yet to completely cover much of it. This is due in part to it being a relatively new company whose product demand is relatively comparatively small with respect to other vehicle manufacturers, resulting in a more constrained budget. [3]

It is very likely that people will see gas stations as often as between every other street, in a populous area. This, however, is nowhere near the case with Tesla's charging stations, where those wishing to use them must drive out of their way to use a station or buy a personal charging dock. As Tesla has made it clear they wish to support free charging for their stations, neither of these alternatives seem appealing. Thus, not only is it the case that these charging stations will become more ubiquitous in the future, but too that this fate is not too far away.

We wish to determine the optimal locations for placing these stations. Tesla has a fairly limited budget as of now, as they continue to use much of their capital in manufacturing cars and developing new vehicles, but they will wish to place stations to retain customers. Thus, our context involves the placement of *permanent* charging stations.

As the context of the United States is too large for the scope of this project, we look on a more local level. In particular, while Chicago has been partially covered with stations, its coverage is clearly not sufficient for a few years from present day. We, therefore, consider Chicago and determine the optimal placement of charging stations in a subsection of the city, specifically between 31^{st} and 67^{th} Streets.

2. Model Description

This problem can be formulated as an optimal learning problem, as we wish to determine the best placement of a charging station, with as few attempts as possible and unknown demands at each potential location. With Tesla's attempt to run charging stations free, it runs a huge risk of running a loss in money if stations that are built are unused, both due to construction and weekly costs. Thus, it is critical that each station be placed with the intention of tending to some stream of customers, making this is an *online* learning problem. We will also take our budget to be 15 charging stations. That is, we expect 15 charging stations to be constructed in this subsection of the city being considered, with an observation taken every two weeks.

2.1. Assumptions

Prior to developing the model, we make the following assumptions to ensure that the context is restricted enough to capture reality while being feasible:

- The city of Chicago can be modelled strictly as a graph of straight lines and edges, as described in the subsequent section. This closely represents how Chicago is truly structured, meaning little information is lost in making this assumption.
- Charging stations are *only* located at nodes of the graph (i.e. at intersections). That is, they cannot be located "midway" through a street. While this is not perfectly realistic, the simplification does not cause a loss of relevant information.
- No charging station is "disassembled" once it has been placed. Once again, with the entire simulation lasting 30 weeks, this is not unreasonable, as a station would likely only be destroyed after consistent lack of traffic for at least six months.
- The number of people who use a station is directly proportional to the traffic flow through the area. This assumption can be reframed as saying that the concentration of EVs (out of the total "car population") is constant across Chicago.
- Each station has a max capacity, which we denote as M_x when considering a station x. Once again, this is realistic from the nature of how charging stations are built (i.e. finite capacity). The maximum capacity for each station is the same.
- Charging stations cause a split in demand based on separation distance. In other words, if two charging stations are close together, the demand will be much more directly split than two largely-separated stations.
- A car using a charging station in one location will charge on average the same amount of energy as a car using a station located elsewhere. This implies that the energy usage of a charging station is directly proportional to the number of cars that use the station.

2.2. Model Setting

As has been established so far, we consider the context of Chicago for our analysis, with the simplification that the city is an undirected graph. As is standard, this graph

G is composed of (V, E), namely with V vertices, representing intersections, of the E edges, representing streets. As seen in the figures below, this approximation quite closely parallels reality.

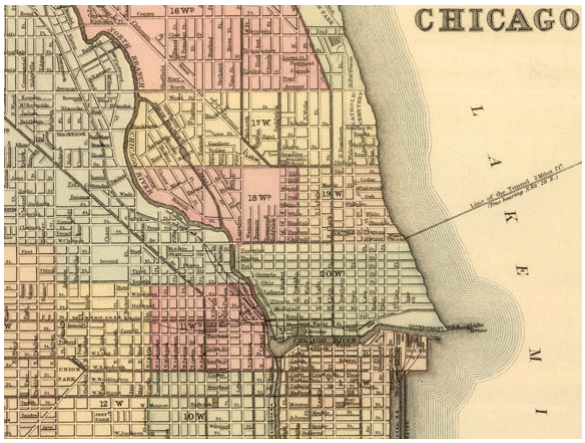

Figure 3: Above is the layout of Chicago, from which the very rectangular structure (as with most large cities) becomes quite apparent [4].

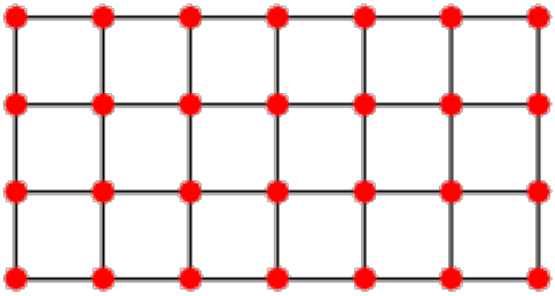

Figure 4: This simple graph structure clearly replicates the above structure we see of the city of Chicago, making it a valid approximation in our context [5].

As per the layout of Chicago, the main streets are those that occur in $a \equiv 3 \mod 4$, i.e. 31^{st} and 35^{th} Streets, which are what we will be examining. Further, while Chicago does not have the "avenue" parallel from New York City, the numbering of buildings roughly replicates this structure. In other words, all buildings numbered in the 4700s will be roughly along the same vertical line, meaning we simply imposed an avenue system on Chicago as $avenue = \lfloor N_{building}/100 \rfloor$, $N_{building}$ being the building number.

3. Model Description

From the assumptions and model setting, the description rises rather naturally. Throughout the section below, let us use x to represent a particular station in the context of the graph.

We first deal with the "splitting" of demand for sufficiently close stations. If two stations are too close to one another, the demand will be split between the two. In addition, the decrease in demand for one station due to another decays with increasing distance between the two stations.

Formally, our model will take this to mean demand to be proportional to a $\varphi^t(x)$ for a particular time t, defined as follows:

$$\varphi^t(s) = \frac{1}{\sum_{s' \in S_t} e^{-\alpha \cdot d(s,s')}}$$

Where S_t represents the set of stations currently in the simulation at time t, $d(s,s')$ represents the distance between stations s and s' (in km), and α is a fixed constant. Note that s itself is included in S. This allows to more generally capture situations where s is the only station in the simulation, as then $\varphi(s) = 1$, or situations where stations may already be established at location s.

While this may seem to be a crude method to deal with the splitting of demand, it makes intuitive sense at least in the corner cases. For example, consider a graph where there are n stations on a single node. Then $\varphi(s) = \frac{1}{n}$ for all s, which is reasonable, since one will expect the demands to be split evenly among all nodes. Furthermore, as distance increases to infinity, the contribution to the weight in the denominators goes to 0, which implies that a station s' at infinite distance from station s has no effect on the demand at station s.

We assume the "half-life" of a pair of stations (s, s'), which in this context refers to the separation necessary to reduce the influence between two stations by half, is 1 km. From this, we can simply calculate α to obtain that it equals $ln(2)$ or ≈ 0.693.

Given $\varphi(x)$, we will define the demand at a station as

$$D(x) = \varphi(x) \cdot \lambda_x$$

Where λ_x is the *unsplit demand* on a station (i.e. the number of people who would have wanted to use the station if it were the only station on the graph). Note that λ_x is an unknown variable that we have a belief on. Let us denote the vector λ as the vectors whose components are $\lambda_1, \lambda_2 ... \lambda_V$.

From this demand, however, we wish to consider the total *usage* of a station, as this is the measurable quantity (denoted as $U(x)$). As we further assumed each station has a maximum capacity, the usage can be modelled as:

$$U(x) = \min\{M_x, \lambda_x \cdot \varphi(x)\}$$

We will simulate this with correlated Normal-Normal belief models for each λ_x.

3.1. Prior Means and Variance Generation

Per our assumption, it follows quite naturally that our initial belief of the λ_x be directly proportional to the traffic data of the area. In turn, we considered two priors and ran identical experiments on each. The former was a simple uniformly distributed prior, with mean 100 and variance 36^2. These specific numerical values were obtained through the Chicago data [7] (also used below), namely as these were the average mean and variance of all the vertices in question, scaled appropriately using the factor c that we used for the second prior (below).

The latter prior used was more complicated, chosen to reflect the information available through the public data source. By direct proportionality to the traffic, we mathematically ascribed the flow to:

$$\lambda_x^0 = c \cdot T(x)$$

Where c is the constant of proportionality, which is quite related to the proportion of people in Chicago who own an electric car. Since this data is not publicly available, we made the further assumption that Chicago roughly represents a standard/average region of the United States and, in turn, has the national percent of electric cars being run (i.e. 0.66%) [6]. Of this proportion of cars passing through a particular intersection, we estimate that $\approx 10\%$ would use the station at a given time. This may appear high, but it considers the fact that many times, people will only enter the area containing a charging station because they need to charge their cars. In turn, the estimated prior was simply:

$$\lambda_x^0 = 14 \cdot 10\% \cdot .66\% \cdot T(x)$$

With this and the $T(x)$ numerically available [7], the following prior was constructed for the means:

Street/Ave	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
31	153	87	99	87	90	92	112	76	79	103
35	89	147	91	78	81	105	103	86	87	157
39	89	67	80	67	70	94	92	75	76	83
43	82	63	69	70	60	84	82	65	66	73
47	136	183	131	104	107	131	159	112	113	121
51	93	72	85	79	84	82	97	80	81	88
55	211	136	148	136	139	312	161	144	145	145
59	105	84	136	84	87	111	109	151	93	100
63	114	74	105	92	96	119	118	100	101	104
67	93	72	116	72	75	98	97	80	81	88

Table 1: Prior of the means of the stations, generated using a simple proportion of traffic data for the city.

In addition, we needed to determine the initial belief of the variance for this usage data. The complication with this matter, however, was that c was still taken with

uncertainty. In other words, we could not treat c simply as a constant when determining the variance of a location i. With that in mind, the following equation governing the initial belief of the variance was determined (assuming a 50% uncertainty on c and independence in c and $T(x)$):

$$\begin{aligned}
\sigma_x^{2,0} &= Var(c \cdot T(x)) \\
&= E[c^2 \cdot T(x)^2] - E[c \cdot T(x)]^2 \\
&= Cov(c^2, T(x)^2) + E[c^2]E[T(x)^2] - E[c \cdot T(x)]^2 \\
&= E[c^2]E[T(x)^2] - E[c \cdot T(x)]^2 \\
&= (Var(c) + E[c]^2)(Var(T(x) + E[T(x)]^2) - [Cov(T(x), c) + E[T(x)]E[c])]^2 \\
&= Var(c) \cdot Var(T(x)) + Var(c) \cdot E[T(x)]^2 + E[c]^2 \cdot Var(T(x)) \\
&= \frac{5}{4} c^2 s^2 \left(T\left(x\right)\right) + \frac{1}{4} c^2 T\left(x\right)^2
\end{aligned}$$

Where $s^2\left(T\left(x\right)\right)$ represents the sample variance on $T(s)$. As a result, we calculated the variance prior similarly to obtain (following page):

Street/Ave	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
31	7367	3450	3303	2517	2702	5764	3974	2409	2295	3516
35	3640	7140	3080	2349	2521	6568	3698	3007	2794	7223
39	3166	2414	2146	1486	1640	5557	2697	2100	1881	2325
43	2879	2247	1742	1580	1286	5080	2229	1721	1497	1903
47	6182	10009	5220	3441	3654	8014	7253	4204	4005	4585
51	3354	2551	2313	1874	2145	5008	2893	2255	2039	2501
55	14393	7999	8156	7054	7317	29827	9128	7943	7760	7941
59	4255	3326	5444	2394	2575	6691	3815	6644	2865	3370
63	4510	2701	3379	2556	2750	6970	4086	3271	3066	3371
67	3447	2647	4016	1718	1879	5850	2985	2350	2134	2594

Table 2: Prior of the variances of the station usage, similarly estimated using the traffic data in the intersection.

3.2. Prior Covariance Matrix

The covariance matrix Σ^0 is constructed by first constructing a correlation matrix and subsequently scaling it appropriately. Clearly, by the dependency of a station's usage on its location, and by the relation between the demand at a location and demands at nearby locations, observing the use of a particular station will tell us more about demands at nearby locations as well. With this, we simply take the correlation $\rho_{i,j}$ between i,j to be defined in the following natural manner:

$$\rho_{i,j} = \mathrm{e}^{-\kappa||i-j||}$$

Where $||i-j||$ is the *Manhattan distance* between stations i and j (i.e. the sum of the vertical and horizontal distances between the two points). Note that the Manhattan distance between two stations is the distance an EV user would have to travel to get from one location to the other while staying on the grid. We choose κ such that the half-life distance (distance required for correlation to decay by half) is 3km. Solving for κ, we obtain $\kappa = \ln 2/3 \approx 0.231$.
Finally, to obtain the covariance, we simply apply the standard formula:

$$Cov(i,j) = \rho_{i,j}\sigma_i\sigma_j$$

Where σ_i, σ_j are the variances for the usage of station i and j respectively. Note that by creating the covariance matrix in this manner, the matrix is always positive semidefinite, no matter what the values for σ are.

3.3. Updating Equations

The updating equations in the context of our problem does not follow directly that of the standard model. Namely, there are two scenarios we must account for:

- **Case 1:** Observe a value $< M_x$
- **Case 2:** Observe M_x

3.3.1. Standard Update In the former case, the observation is well-defined. Namely, we know exactly the demand of the station during the observation time-frame. In turn, this case follows the standard updating equations, where our observation W is:

$$W_s = \frac{U(s)}{\varphi(s)}$$

As we have observed $U(s)$ and can directly calculate $\varphi(s)$. We can then apply the standard updating equation for correlated Normal-Normal belief models:

$$\lambda^{n+1} = \lambda^n + \frac{W_s - \lambda_s^n}{(\beta^W)^{-1} + \Sigma_{ss}^n}\Sigma^n e_x$$

$$\Sigma^{n+1} = \Sigma^n - \frac{\Sigma^n e_s (e_s)^T \Sigma^n}{(\beta^W)^{-1} + \Sigma_{ss}^n}$$

.

Where e_s is the basis vector on s.

3.3.2. Vague Update In the latter case, however, we only know that the station reached capacity. That is, there could have been exactly M customers who wished to use the station or an arbitrarily larger number of customers than M. In other words, this constitutes the case of having $M+N$ customers wishing to use the station, where $N \geq 0$. Clearly, as this is not a specific data point, the information gained is less than that from the former case. Specifically, we now must update our equations such that the posterior distribution satisfies:

$$\Pr\left\{\lambda_x^{n+1} \in dx | W_s^{n+1} \geq M_s, S^n\right\} \quad \forall x$$

Graphically speaking, if we are given a normal distribution, this corresponds to the case that $\varphi(s) \cdot \lambda_x$ is in the shaded region (lower bounded by M).

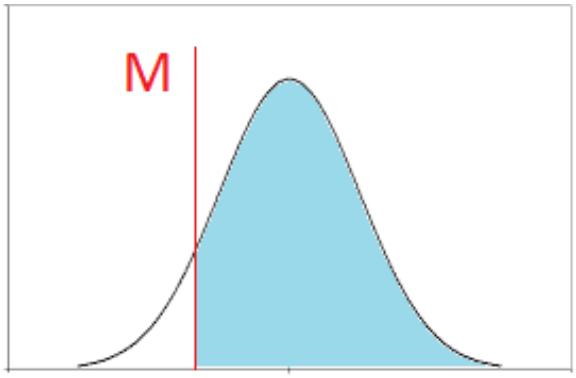

Figure 5: Region in which the real value is known to reside. We must update our posterior with the information that the real value may arise at any location in this region.

Since the exact solution for the posterior becomes non-Gaussian, we choose to closely approximate this as a single observation point W_s with the following properties:

$$W_s = E\left[Y | \frac{Y}{\varphi(s)} \geq M(s)\right], Y \sim N(\lambda_s, \Sigma_{ss})$$

$$\beta^{W_s} = ((\beta^W)^{-1} + \Sigma_{ss})^{-1}$$

In other words, we will approximate this "vague observation" as a single point located at the mean of all values on the bell curve above the maximum capacity with a variance equal to the sum of the measurement noise and the current uncertainty in λ_s to reflect the fact that our data is not a specific point but rather some point within an entire range.

For our normal normal model, there is a closed form for our value we should pick for W_s. We have for standard normal Z:

$$E[Y | Y \geq M(s) \cdot \varphi(s)]$$

$$= \lambda_s + \sqrt{\Sigma_{ss}} E\left[Z | Z \geq \frac{M(s) \cdot \varphi(s) - \lambda_s}{\sqrt{\Sigma_{ss}}}\right]$$

Now let $k = \frac{M(s) \cdot \varphi(s) - \lambda_s}{\sqrt{\Sigma_{ss}}}$ and we have that the above quantity equals

$$\lambda_s + \sqrt{\Sigma_{ss}} \frac{\int_k^\infty y e^{-y^2/2} dy}{1 - \Phi(k)} = \lambda_s + \sqrt{\Sigma_{ss}} \frac{\phi(k)}{1 - \Phi(k)}$$

Note that ϕ denotes the normal p.d.f, and Φ denotes the normal c.d.f. These can easily be computed with Matlab and many other softwares, so for our purposes, this equation is in closed form.

4. Policy Implementations

4.1. Outline

We investigated the following policies:

- Pure Exploitation
- Boltzmann Exploration
- Interval Estimation
- Offline Knowledge Gradient
- Online Knowledge Gradient

We compared each of the policies not only to one another, but also to pure exploitation, which served as our baseline. These were all completed through independent code, but borrowed heavily from the code provided in class. However, as was described in the preceding section, the updating equations had to be modified in the code to achieve the desired alteration that was of interest in this study. Described below is a description of how exactly we chose to use our policies in determining the locations to place stations.

4.2. Pure Exploitation

For our pure exploitation policy, the algorithm chooses alternative x at time step $n+1$ in the following manner:

$$\varphi'(x) = \frac{1}{1 + \sum_{s' \in S_n} e^{-\alpha \cdot d(x,s')}}$$

$$x^{n+1} = \underset{x \in V}{\operatorname{argmax}} \left(\varphi'(x)\lambda_x^n\right)$$

.

Where S_n is the set of all stations placed at time n. In other words, our pure exploitation picks the station which yields the highest expected demand if we put a station in that spot. The weights need to be updated before doing this calculation because we want to pick the station which yields the highest demand after placement, and weights are adjusted as soon as we place such a station. Note that this policy only considers stations one at a time, with no regard for how our placed station will split the demand at existing stations. Furthermore, it disregards max capacity, as the value x that maximizes $\varphi'(x)\lambda_x^n$ also maximizes $\min(\varphi'(x)\lambda_x^n, M_x)$.

4.3. Boltzmann Exploration

As per the pure exploitation policy, it makes most sense to weigh on expected demand on each station x if we place a station there, so for this policy we choose $x^{n+1} = x$ with probability

$$p_x = \frac{e^{\rho\varphi'\lambda_x}}{\sum_{x \in V} e^{\rho\varphi'\lambda_x}}$$

Where φ' is as defined above and ρ is a tunable parameter. Note that our probabilities each get "weights" $\varphi'\lambda_x$ just as how our pure exploitation policy chose to maximize along this quantity. An alternative would be to choose $x^{n+1} = x$ with probability

$$p_x = \frac{e^{\rho \cdot \min(\varphi'\lambda_x, M_x)}}{\sum_{x \in V} e^{\rho \cdot \min(\varphi'\lambda_x, M_x)}}$$

The former is more inclined to choose alternatives with higher demand, no matter how high they exceed the maximum capacity, while the latter is more inclined to explore different alternatives if certain locations exceed the maximum capacity by too much. For our policy, we chose the former to mirror our choice within the pure exploitation policy.

4.4. Modified Interval Estimation

For interval estimation, we seek to choose x^{n+1} such that the expected value within z_α standard deviations above its mean is maximized. Mathematically, we are choosing

$$x^{n+1} = \underset{x \in V}{\text{argmax}} \ \min(\min(\varphi'(x)\lambda_x^n, M_x) + z_\alpha \cdot \Sigma_{xx}\varphi'(x), M_x)$$

Note that for this policy, unlike the previous two, we are restricting our our values to all those below M_x to give it a different flavour from the previous two policies in that it now doesn't give unnecessary weight to alternatives which may exceed the maximum capacity. The expected result of this modification is that this policy may trade off possible rewards gained in our online setting while also learning more about different opportunities, resulting in a lower opportunity cost.

4.5. Knowledge Gradient (Online and Offline)

For our offline knowledge gradient, consider the definition

$$\nu_x^{KG} = E[\max_{x' \in V} \varphi'(x') \cdot \lambda_{x'}^{n+1} - \max_{x' \in V} \varphi^n(x') \cdot \lambda_x^n | K^n, x^n = x]$$

Where K^n is our current state of knowledge and x^n is our choice of station at time n. Note that computationally for a normal-normal model this is

$$E[\max_{x' \in V} \ -a_{x'} + b_{x'} * Z] - \max_{x' \in V} a'_x$$

$$a_{x'} = \varphi'(x') \cdot \lambda_{x'}$$

$$b_{x'} = \frac{\Sigma^n e_x}{\varphi'(x) \cdot \varphi'(x') \sqrt{\Sigma_{xx} + (\beta^W)^{-1}}} (e_{x'})^T$$

Now let our choice x^n for time n equal

$$\underset{x \in V}{\text{argmax}} \ \nu_x^{KG}$$

.

Let us examine what we exactly this policy is doing. First of all, notice that this policy does not take in account the maximum capacity at all in its calculations, just like

for our Pure Exploitation and Boltzmann Policies. This policy chooses the station such that the expected value of the best station on the *next* step is maximized. However, in the process, it considers the effects placing station x regarding demand split at all other stations. So we expect our KG policy to do well relative to the other policies in maximizing the usage over all stations over all times for this reason.

For our Online KG policy, we simply change our choice for x^n to

$$x^n = \underset{x \in V}{\operatorname{argmax}} \, \varphi'(x) \cdot \lambda_x^n + \gamma \frac{1 - \gamma^{M-n}}{1 - \gamma} \nu_x^{KG}$$

In standard fashion.

4.6. Policy Evaluation

To evaluate each of the policies, we used an objective reflecting how much the stations were used. Specifically, we used two objective functions to evaluate performance, as follows:

- Considers the total number of people using all station at time t (summed over all the times in the simulation). This, therefore, appeals to considering a "cumulative" objective function.
- Considers the number of people using the station most recently added at time t, similarly totaled over all the t. This, instead, reflects more of an incremental/micro-scale optimization.

Our first objective can be expressed as the following:

$$\zeta = \sum_{t=1}^{15} \sum_{s \in S_t} \gamma^t U(s)$$

And our second can be expressed as the following:

$$\zeta = \sum_{t=1}^{15} \gamma^t U(s_t)$$

Where S_t is the set of stations at time t and s_t is the station added at time t. Note that we can also increase our horizon in the first objective to let t go towards infinity, even with only 15 stations being placed.

The opportunity cost can also be used as a metric for evaluation but it is not as useful for evaluating policies in terms of the average performance with regards to profitability. Instead, it is more useful for evaluating policies in terms of how well they found the best alternative. We will use the same definition for opportunity cost as the one used in class.

5. Experiments and Results

Below are the results amassed from each of the trials, all through the developed programs. These demonstrate the comparative effectiveness of policies and their individual effectiveness.

For all of the experiments we used the following parameters:

- Number of truths generated: 100
- Total number of stations placed: 15
- Discount factor (γ): 0.98 (Approximated using 30 day automotive corporate bond yields and scaling)
- Boltzmann ρ constant: 0.75 (manually tuned by trial and error)
- IE z_α: 3.0 (manually tuned by trial and error)

The code used to run the experiments is provided. Here is a summary of all of the files included:

- PureExploit.m–runs Pure Exploitation Policy described above
- Boltzmann.m–runs Boltzmann Exploration Policy described above
- IE.m–runs Interval Estimation Policy described above
- KGCorrBeliefs1.m–runs Offline KG Policy described above
- KGCorrBeliefs2.m–runs Online KG Policy described above
- LogEmaxAffine.m–helper function for KG policies
- computeObj.m–computes first objective described
- computeObj2.m–computes second objective described
- RunPolicies.m–runs all policies, compares objectives and OC, for prior generated from real data
- RunPolicies2.m–runs all policies, compares objectives and OC, for flat prior

Below are the results amassed from each of the trials, all through the developed programs. These demonstrate the comparative effectiveness of policies and their individual effectiveness.

5.1. Flat Priors

We first tested our code on flat priors. All of the values for λ are relatively close to each other, so we do not expect too much deviation in performance in policies.

5.1.1. Cumulative Objective First we compare our cumulative objectives using three different max capacities–1000, 100, and 50. The first capacity is much larger than any value we would expect for demand, so we will call this “no cap.” The second is around our prior means, so we will call this a ”mid cap.” The final capacity is around half our prior means, so we will call this a ”low cap.”

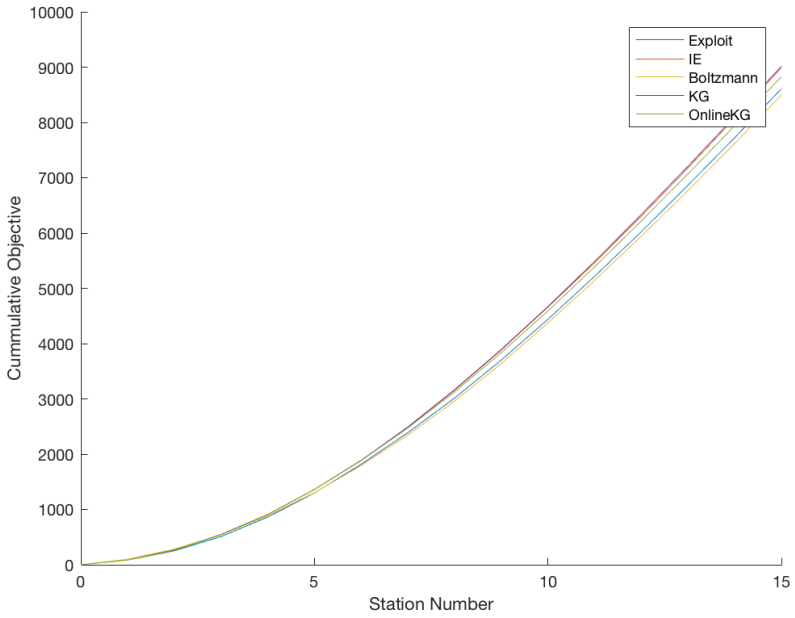

Figure 6: No Cap Comparison of Policies with Flat Prior

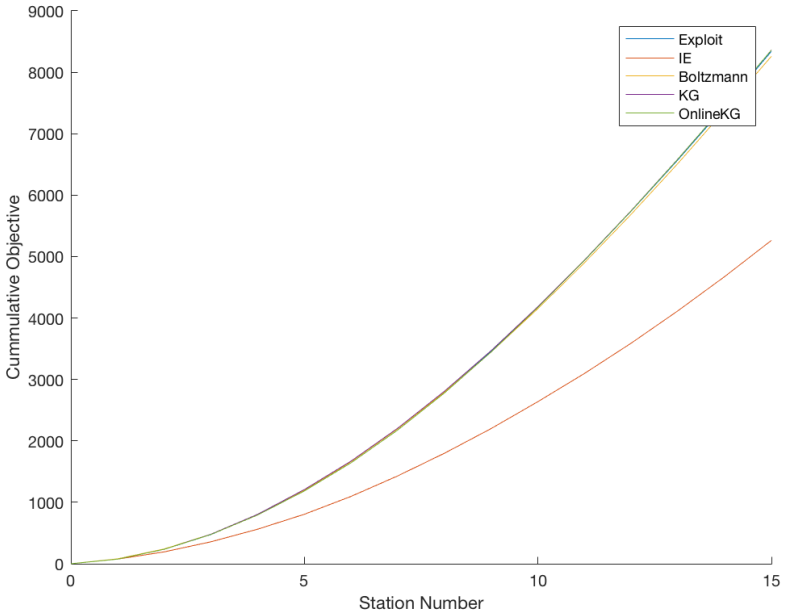

Figure 7: Mid Cap Comparison of Policies with Flat Prior

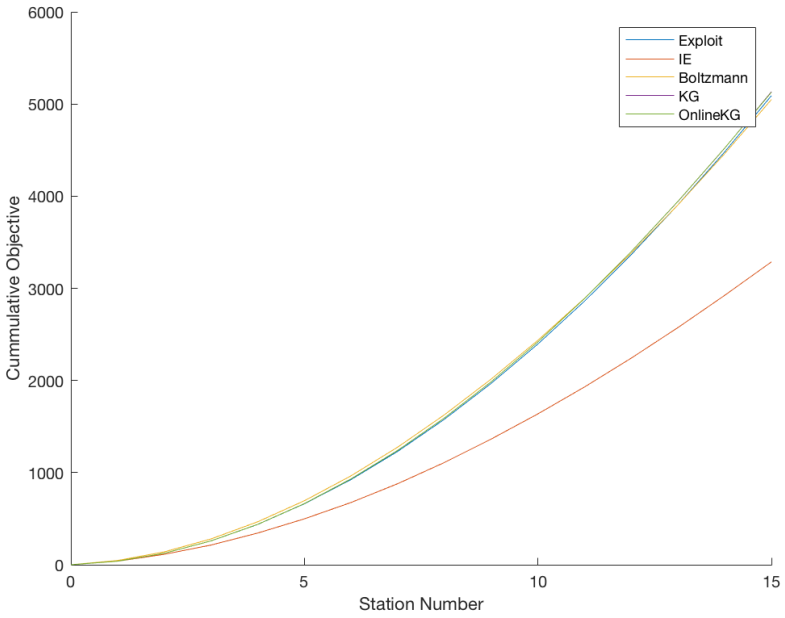

Figure 8: Low Cap Comparison of Policies with Flat Prior

5.1.2. Incremental Objective Next we examine the incremental objective, or how well each policy does in picking the highest performing station on each time step. Again, we do this comparison across the same three different capacities:

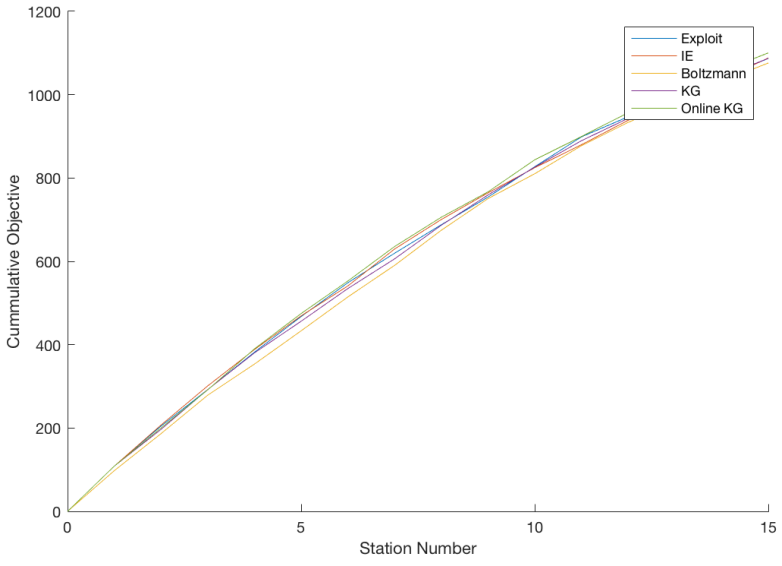

Figure 9: No Cap Comparison of Policies with Flat Prior

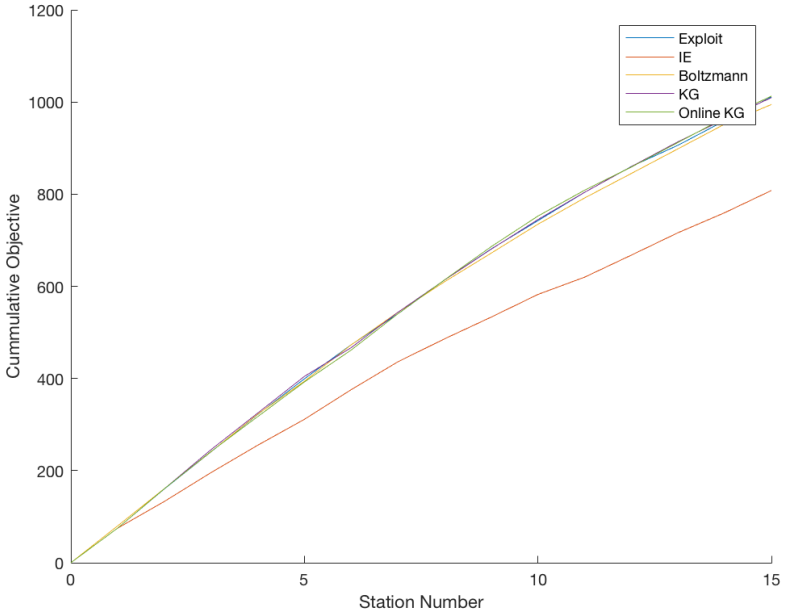

Figure 10: Mid Cap Comparison of Policies with Flat Prior

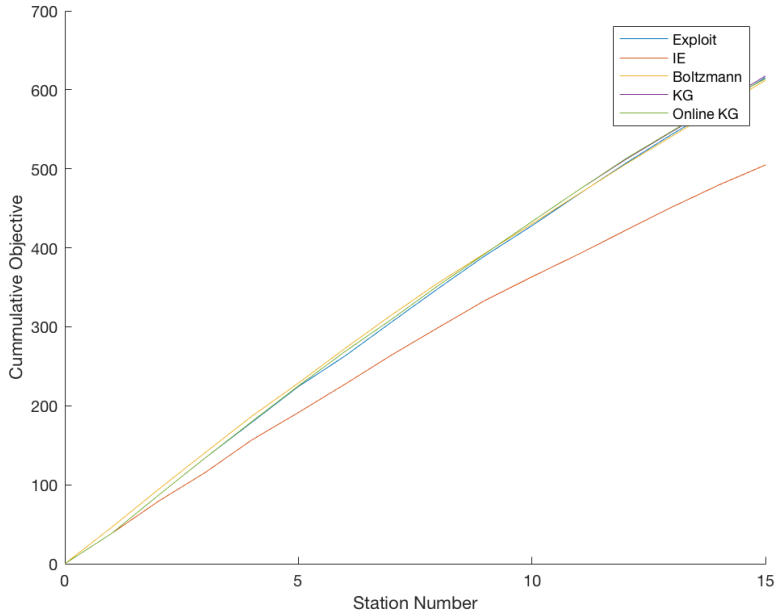

Figure 11: Low Cap Comparison of Policies with Flat Prior

5.1.3. Opportunity Cost Finally, we examine the opportunity cost to see how well each objective does in finding the alternative with the highest λ.

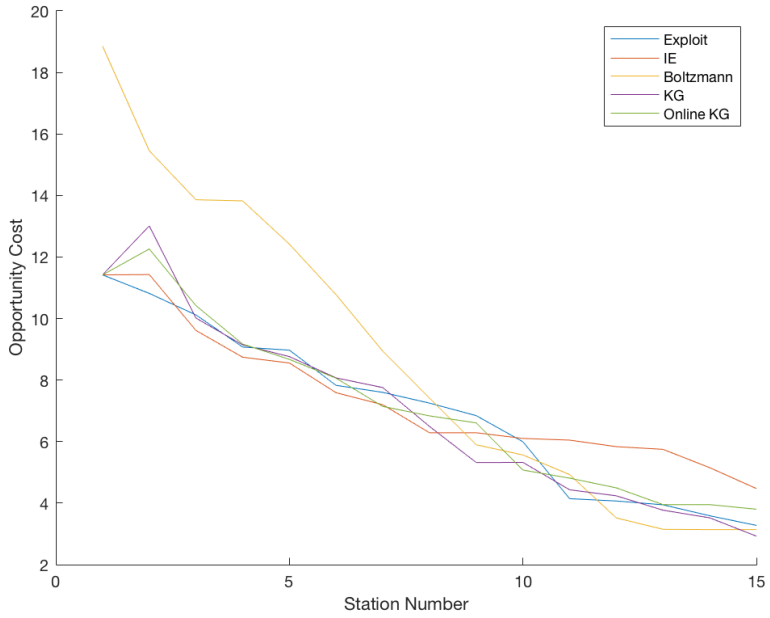

Figure 12: No Cap Comparison of Policies with Flat Prior

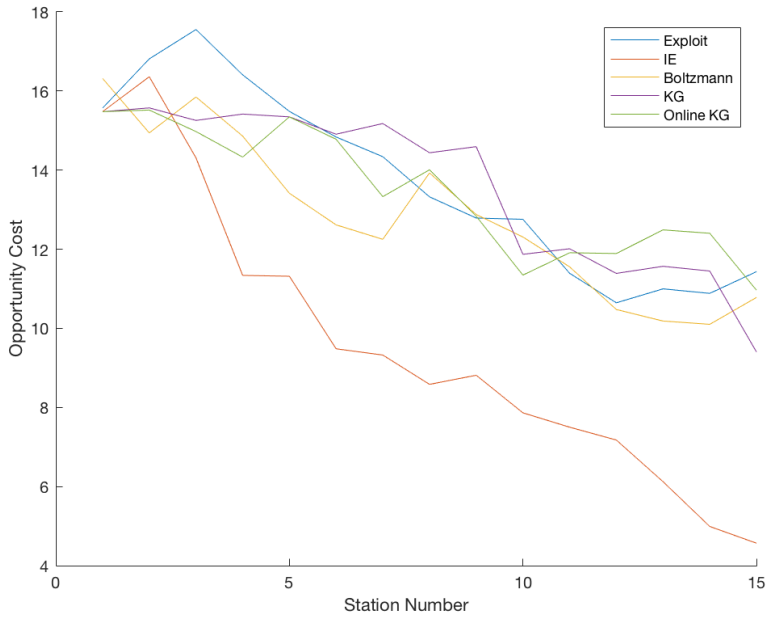

Figure 13: Mid Cap Comparison of Policies with Flat Prior

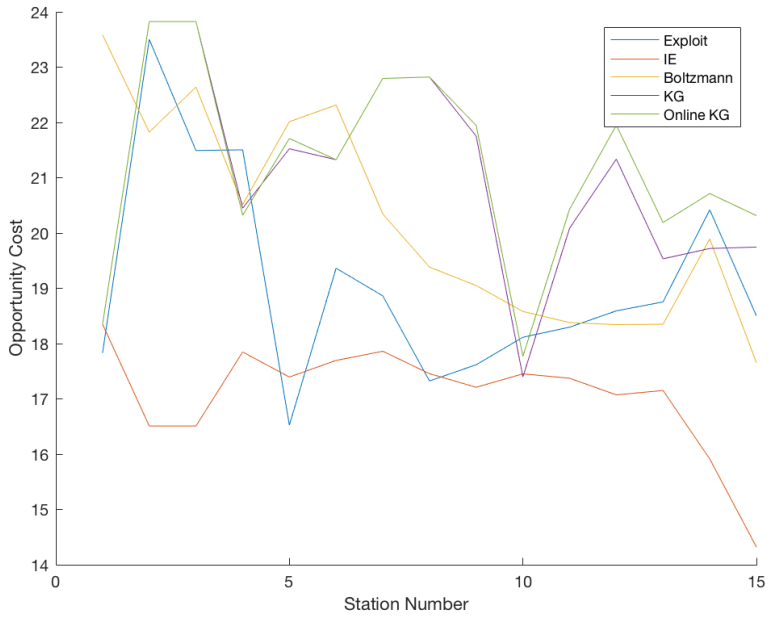

Figure 14: Low Cap Comparison of Policies with Flat Prior

5.2. Prior Generated from Real Data

In this section, we seek to test our policies on priors generated from real data, which has many regions of exceptionally high and low traffic. We think this will be more interesting to examine due the volatility of our data.

5.2.1. Cumulative Objective First we compare our cumulative objectives using three different max capacities–1000, 200, and 100. Again, we will name them "no cap," "mid cap," and "low cap," respectively. Note that our mid and low caps changed to reflect the fact that many of our prior's data points are above 200.

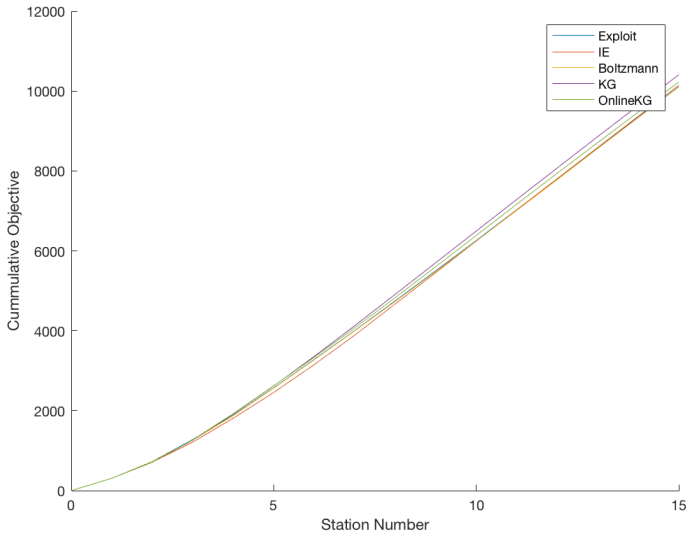

Figure 15: No Cap Comparison of Policies with Real Data Prior

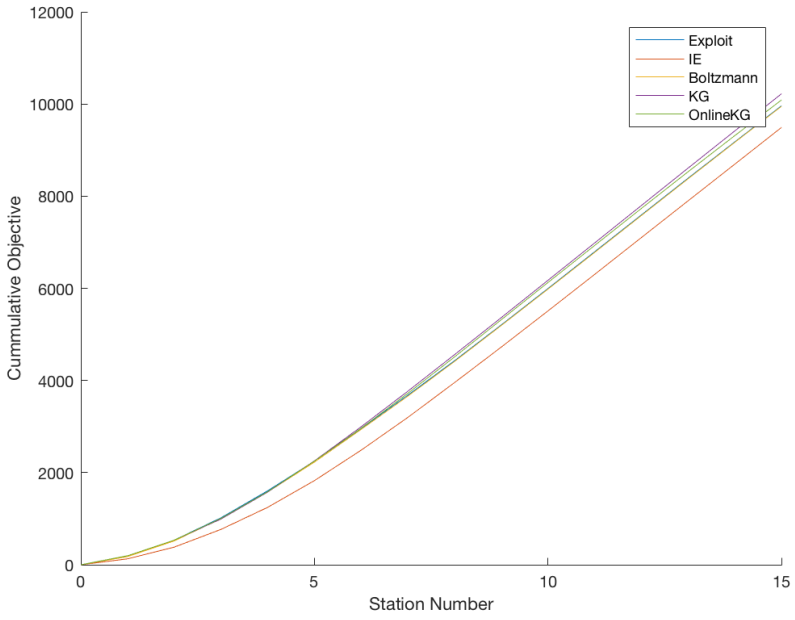

Figure 16: Mid Cap Comparison of Policies with Real Data Prior

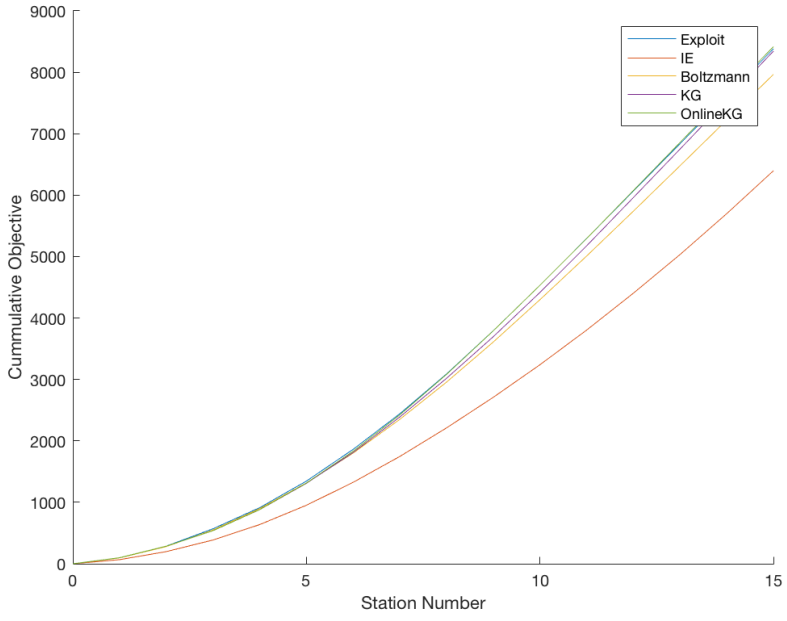

Figure 17: Low Cap Comparison of Policies with Real Data Prior

5.2.2. Incremental Objective Next we examine the incremental objective, or how well each policy does in picking the highest performing station on each time step. Again, we do this comparison across the same three different capacities:

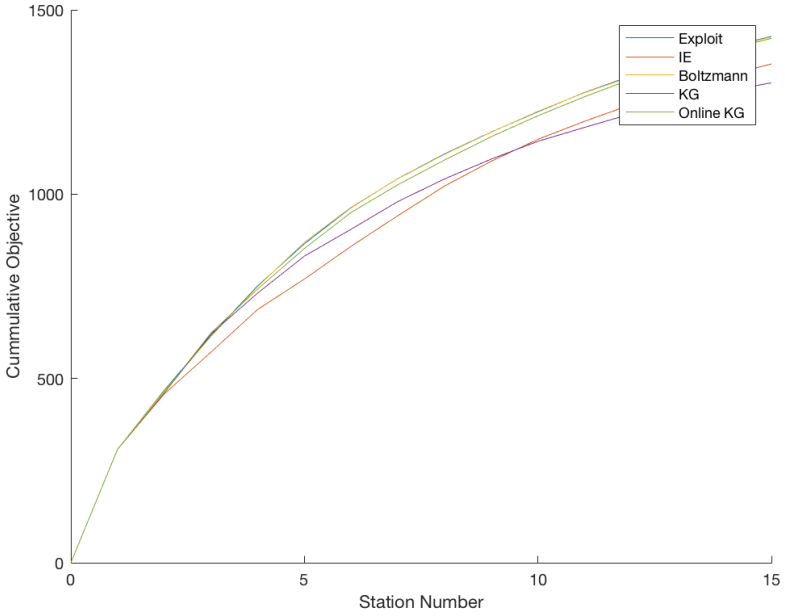

Figure 18: No Cap Comparison of Policies with Real Data Prior

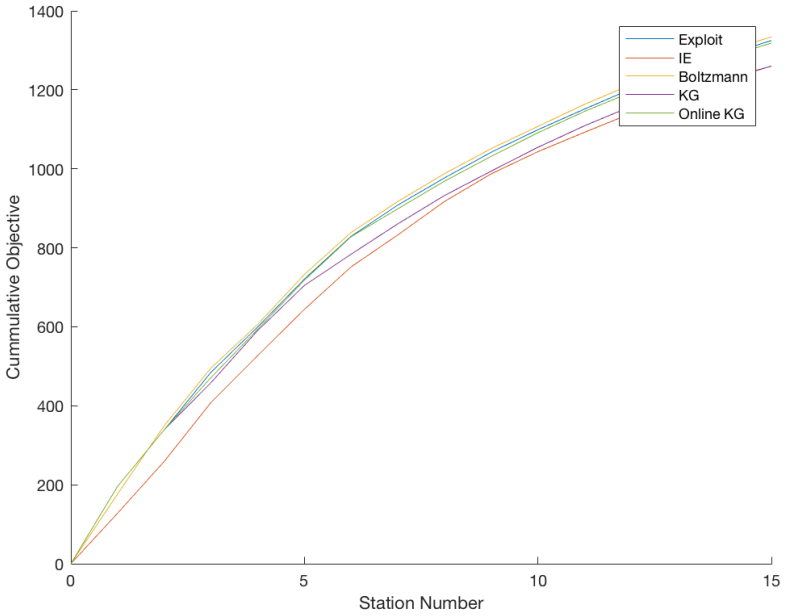

Figure 19: Mid Cap Comparison of Policies with Real Data Prior

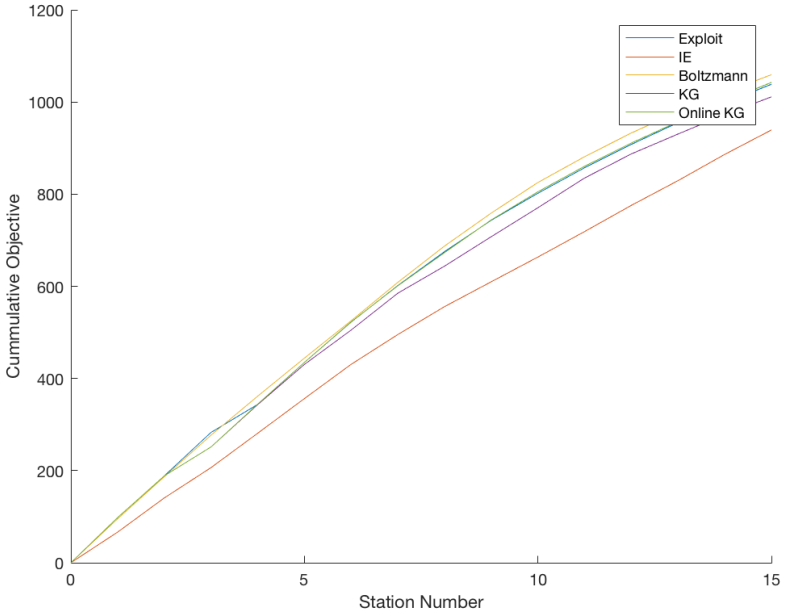

Figure 20: Low Cap Comparison of Policies with Real Data Prior

5.2.3. Opportunity Cost Finally, we examine the opportunity cost to see how well each objective does in finding the alternative with the highest λ.

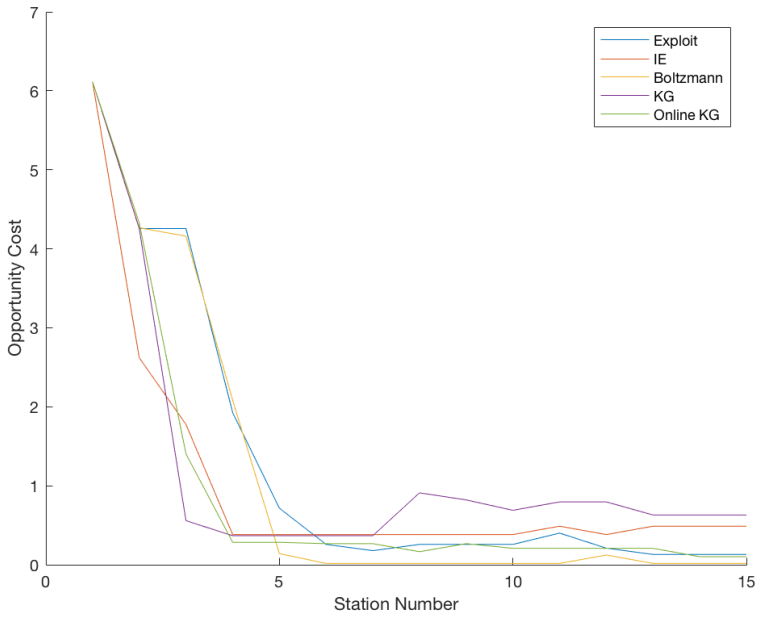

Figure 21: No Cap Comparison of Policies with Real Data Prior

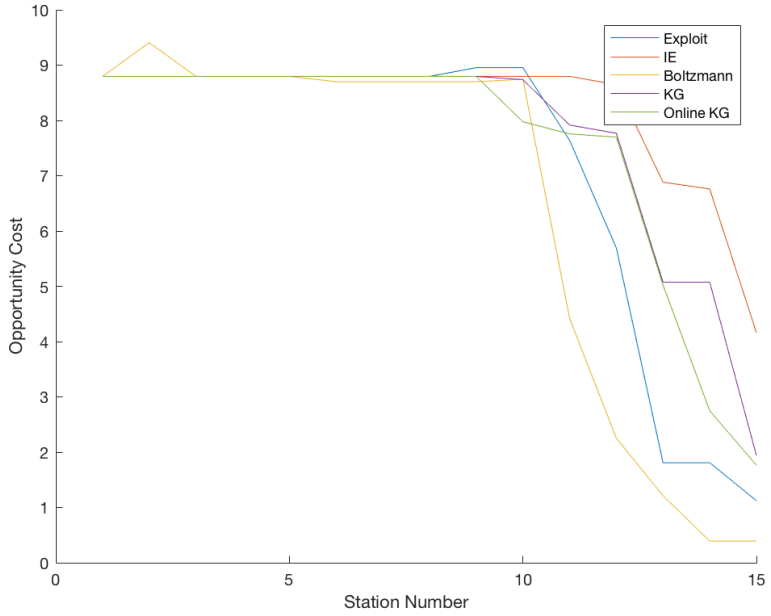

Figure 22: Mid Cap Comparison of Policies with Real Data Prior

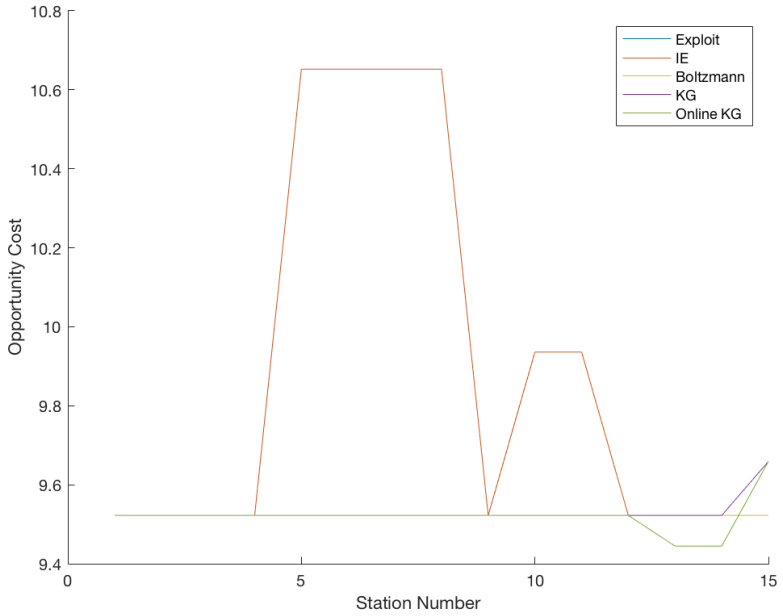

Figure 23: Low Cap Comparison of Policies with Real Data Prior

6. Discussion

6.1. Flat Priors

From our results, we can ascertain a few interesting conclusions. First of all we will discuss our results with the flat priors. From our figures, we note that for our "no cap" experiment, KG, IE, and online KG performed noticeably better than than the other two policies with regards to our cumulative objective, namely maximizing over all times the total usage over all stations placed. This can most likely be attributed to our KG policy's ability to take into consideration the effects of placing a station on all other stations, where other policies cannot do this. Note that as we decrease the maximum capacity, IE's performance deteriorates rapidly with regards to this cumulative objective, most likely because it equally favors all stations who reach maximum capacity, which leads to a greater degree of exploration and lesser degree of exploitation with regards to station demands. In addition, KG and Online KG's performances become reduced to around the level of the other policies because of their inability to capture the maximum capacity (in their current formulations). Conversely, IE also performs much noticeably better than all other policies with regards to opportunity cost when the maximum capacity is reduced due to its tendency to explore more than exploit with regards to demand in stations (so it finds the best alternative faster). An interesting thing to note is that the incremental objectives for all of the policies (except IE) were roughly equivalent for all policies. This further reaffirms our belief that our while KG and online KG policies may perform roughly as well as the others in finding the best alternative on each time step to maximize the usage for one station, their ability to consider other stations in choosing such an alternative makes them superior in maximizing the total usage over all stations over all time steps.

6.2. Priors From Real Data

It is more interesting to examine the effects of our policies on priors generated from real data, which has higher volatility and more unpredictability. Again, we notice that KG outperforms all other policies with respects to the cumulative objective, and IE does worse when maximum capacity is decreased, but not by as much. Furthermore, when the data is not flat, KG seems to do better relative to when the data is flat in comparison to other policies. With regards to opportunity cost, IE does horribly in comparison to the test run with flat priors, which tells us that perhaps our modified IE policy is not good for more volatile data. Another interesting thing of note is that with regards to the incremental objective (choosing the station which will yield the highest usage on each time step), KG does worse than other policies, yet at the same time it outperforms all of the others in the cumulative objective. This further attests to our KG policy's ability to maximize over all locations rather than looking at just each location individually.

Remark. *From the graphs, one may think that the difference in policy performance is insignificant. But this is just due to the way the data scales very steeply with respect to usage numbers. However, one can see that the difference is in fact significant. For*

example, Figure 24 shows the confidence intervals of each policy's performance with respect to the following objective:

$$\zeta = \sum_{t=1}^{\infty} \sum_{s \in S_t} \gamma^t U(s)$$

Which is the "truest" objective we wish to optimize over–namely optimizing total station usage over an infinite horizon. From the figure, one can see that KG outperforms all of the other policies, even within confidence levels.

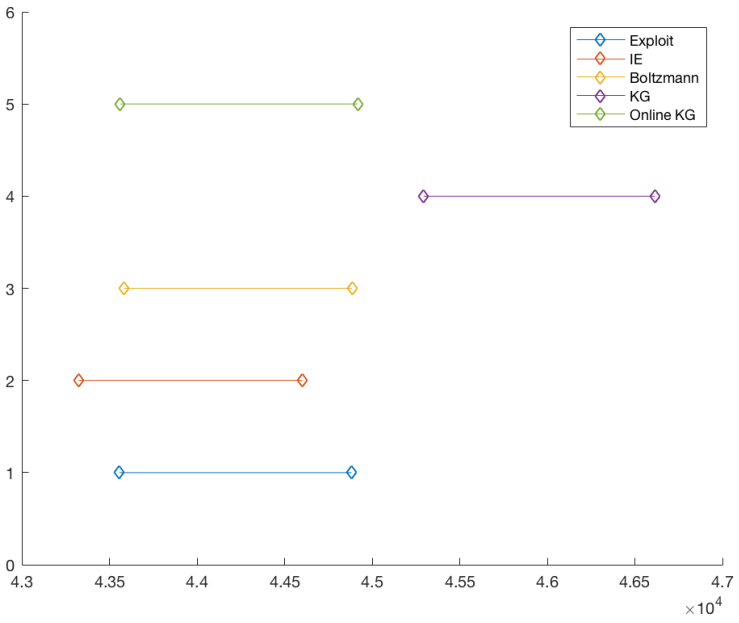

Figure 24: Final Objectives on a 100 trial sample run with priors generated from real data and a medium cap.

With this in mind, we conclude that KG does very well in comparison with other policies when we look at the cumulative objective–maximizing usage over all stations. Our IE policy often does well in minimizing opportunity costs when the data is relatively flat–but often falls apart with respects to maximizing usage or when data becomes too volatile.

7. Future Work

Having completed this study, there is significant expansion that we can look into going forward. The most direct expansion is simply by enlarging or shifting the context of this study. While we believe our model captures reality sufficiently accurate for a city setting such as Chicago, its transition to a rural or less structured setting seems somewhat precarious. In particular, the lack of defined structure makes modelling these non-city landscapes more difficult as graphs, meaning the very foundation of the model falls apart.

Furthermore, we can more deeply explore policies or modifications to existing policies which more accurately capture the more problematic elements of our problem formulation–namely the maximum capacity for stations and the splitting of demand across different stations. Our KG policy formulated for this project somewhat captures this splitting of demand, but we believe that it is possible to modify it further to further capture this demand split.

In addition, time inhomogeneity appears to be a large extension of the current model, for we assume each day incurs a relatively similar stream of people and that each hour sees a similar number of people arriving. However, in its crudest form, time inhomogeneity delineates great difference between the behavior of people on weekends vs. weekdays, or during rush hour vs. late at night. Integrating this (possibly incorporating elements of queueing theory) would greatly expand the accuracy and applicability of the model.

Finally, we could relax some of our constraints regarding the split of demand. Recall that our definition for demand splitting completely depended on a constant φ which is purely a weighted sum. In reality, this may not be as realistic, so more care should be taken to develop a more accurate model for capturing the split of demand.

8. References

References

[1] "Eyeing the Big Leagues: Tesla Racing toward Annual Volume of 500,000 Cars." *Digital Trends*. N.p., 05 May 2016. Web. 6 May 2016.

[2] "Here's How Electric Cars Will Cause the Next Oil Crisis." *Bloomberg*. Bloomberg, n.d. Web. 6 May 2016.

[3] Person, and Nate Swanner. "Tesla's Charging Stations Will Cover USA by End of 2015." *SlashGear*. N.p., 31 July 2014. Web. 6 May 2016.

[4] "Ideal City: Layout: Regular Divisions of the Plain." N.p., 16 Oct. 2013. Web. 6 May 2016.

[5] "Grid Graph." *Wolfram MathWorld*. N.p., n.d. Web. 6 May 2016.

[6] "Top Six Plug-in Vehicle Adopting Countries." *HybridCars*. N.p., 18 Jan. 2016. Web. 6 May 2016.

[7] "City of Chicago | Data Portal." *Chicago*. N.p., n.d. Web. 6 May 2016.

9. Appendix

9.1. Policies

9.1.1. Boltzmann :

```
%{
Boltzmann Policy

notation for the following:
K is the number of alternatives.
M is the number of time-steps
K x M stands for a matrix with K rows and M columns

This function takes in
mu:     true values for the mean (K x 1)
mu_0:   prior for the mean (K x 1)
beta_W: measurement precision (1/lambda(x)) (K x 1)
covM:   initial covariance matrix (K,K)
M:      how many measurements will be made (scalar)
maxcap: maximum station capacity
dists:  distances between stations
alpha:  tunable parameter


And returns
mu_est:     Final estimates for the means (K x 1)

OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)
mu_estALL:  Estimates at each iteration (K x M)
%}

function [mu_est, OC, choices, mu_estALL]=Boltzmann(mu,mu_0,beta_W,covM,M, ...
    maxcap, dists,alpha)

K=length(mu_0); %number of available choices

mu_est=mu_0; %estimates on LAMBDAs

OC=[];
choices=[];
mu_estALL=[];

for k=1:M % run Boltzmann M times

    % calculate station demand if we put a station in place x
    weights = ones(K,1);
    for i = 1:length(choices)
        for j = 1:K
        weights(j) = weights(j) + exp(-log(2)*dists(j,choices(i)));
        end
    end
```

```
demand = min(mu_est ./ weights, maxcap);

% Boltzmann policy
dice = rand;

% compute proability vector of choosing options
probs = exp(alpha*demand);
probs = probs / sum(probs);

% compute cumulates
i = length(probs);
x = 1;
while x < i
    probs(x+1) = probs(x) + probs(x+1);
    x = x + 1;
end
x = 1;
while probs(x) < dice && x < i
    x = x + 1;
end

%observe the outcome of the decision
%W_k=mu_k+Z*SigmaW_k where SigmaW is standard deviation of the
%error for each observation

%U_k is what we observe
W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));
U_k= min(W_k/weights(x), maxcap);

%updating equations are conditional on what we observe

e_x=zeros(K,1);
e_x(x)=1;

%special case: observe maxcap
if U_k == maxcap
    W_k = sqrt(covM(x,x)) * normpdf((maxcap*weights(x)-mu_est(x))/ ...
        sqrt(covM(x,x)))/(1-normcdf((maxcap*weights(x)-mu_est(x))/ ...
        sqrt(covM(x,x)))) + mu_est(x);
    var_artif = 1/beta_W(x) + covM(x,x);
    addscalar = (W_k - mu_est(x))/(var_artif + covM(x,x));
    mu_est=mu_est + addscalar*covM*e_x;
    covM = covM - (covM*e_x*e_x'*covM)/((var_artif) + covM(x,x));
else
    %standard updating equations for Normal-Normal model with covariance
    addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
    mu_est=mu_est + addscalar*covM*e_x;
    covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));
end

%pick the best one to compare OC
[max_est, max_choice]=max(mu_est);
```

```
    %calculate the opportunity cost
    o_cost=max(mu)-mu(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, x]; %update the choice matrix

    if nargout>3 %if more than three outputs were asked
        mu_estALL=[mu_estALL,mu_est];
    end


end

end
```

9.1.2. Interval Estimation :

```
%{
IE Policy

notation for the following:
K is the number of alternatives.
M is the number of time-steps
K x M stands for a matrix with K rows and M columns

This function takes in
mu:     true values for the mean (K x 1)
mu_0:   prior for the mean (K x 1)
beta_W: measurement precision (1/lambda(x)) (K x 1)
covM:   initial covariance matrix (K,K)
M:      how many measurements will be made (scalar)
maxcap: maximum station capacity
dists:  distances between stations
alpha:  tunable parameter


And returns
mu_est:     Final estimates for the means (K x 1)

OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)
mu_estALL:  Estimates at each iteration (K x M)
%}

function [mu_est, OC, choices, mu_estALL]=IE(mu,mu_0,beta_W,covM,M, maxcap,...
    dists,alpha)

K=length(mu_0); %number of available choices

mu_est=mu_0; %estimates on LAMBDAs
```

```
OC=[];
choices=[];
mu_estALL=[];

for k=1:M % run IE M times

    % calculate station demand if we put a station in place x
    weights = ones(K,1);
    for i = 1:length(choices)
        for j = 1:K
        weights(j) = weights(j) + exp(-log(2)*dists(j,choices(i)));
        end
    end

    demand = min(mu_est ./ weights, maxcap);

    % IE Policy
    % the variance on demand is 1/phi^2 * variance(lambda)
    v = min(demand + alpha * sqrt(diag(covM)) ./ weights, maxcap);

    [max_value,x] = max(v);

    %observe the outcome of the decision
    %W_k=mu_k+Z*SigmaW_k where SigmaW is standard deviation of the
    %error for each observation

    %U_k is what we observe
    W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));
    U_k= min(W_k/weights(x), maxcap);

    %updating equations are conditional on what we observe

    e_x=zeros(K,1);
    e_x(x)=1;

    %special case: observe maxcap
    if U_k == maxcap
        W_k = sqrt(covM(x,x)) * normpdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))/(1-normcdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))) + mu_est(x);
        var_artif = 1/beta_W(x) + covM(x,x);
        addscalar = (W_k - mu_est(x))/(var_artif + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((var_artif) + covM(x,x));
    else
        %standard updating equations for Normal-Normal model with covariance
        addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));
    end

    %pick the best one to compare OC
    [max_est, max_choice]=max(mu_est);
```

```
    %calculate the opportunity cost
    o_cost=max(mu)-mu(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, x]; %update the choice matrix

    if nargout>3 %if more than three outputs were asked
        mu_estALL=[mu_estALL,mu_est];
    end


end

end
```

9.1.3. KG Correlated Beliefs – V1 :

```
%{
KG Policy

notation for the following:
K is the number of alternatives.
M is the number of time-steps
K x M stands for a matrix with K rows and M columns

This function takes in
mu:     true values for the mean (K x 1)
mu_0:   prior for the mean (K x 1)
beta_W: measurement precision (1/lambda(x)) (K x 1)
covM:   initial covariance matrix (K,K)
M:      how many measurements will be made (scalar)
maxcap: maximum station capacity
dists:  distances between stations


And returns
mu_est:     Final estimates for the means (K x 1)

OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)
mu_estALL:  Estimates at each iteration (K x M)
%}

function [mu_est, OC, choices, mu_estALL]=KGCorrBeliefs1(mu,mu_0,beta_W,...
    covM,M, maxcap, dists)

K=length(mu_0); %number of available choices

mu_est=mu_0; %estimates on LAMBDAs

OC=[];
```

```
choices=[];
mu_estALL=[];

for k=1:M %exploit M times

    % calculate station demand if we put a station in place x
    weights = ones(K,1);
    for i = 1:length(choices)
        for j = 1:K
        weights(j) = weights(j) + exp(-log(2)*dists(j,choices(i)));
        end
    end

    demand = mu_est ./ weights;

    %Plogy is the log values of KG for alternatives
    Plogy=[];

    for iter1=1:K
        a=demand';
        b=covM(iter1,:)./weights'/weights(iter1)/sqrt(1/beta_W(iter1) + ...
            covM(iter1,iter1)/weights(iter1)^2);
        [Plogy]=[Plogy, LogEmaxAffine(a,b)];
    end

    [maxh,x]=max(Plogy);

    %observe the outcome of the decision
    %W_k=mu_k+Z*SigmaW_k where SigmaW is standard deviation of the
    %error for each observation

    %U_k is what we observe
    W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));
    U_k= min(W_k/weights(x), maxcap);

    %updating equations are conditional on what we observe

    e_x=zeros(K,1);
    e_x(x)=1;

    %special case: observe maxcap
    if U_k == maxcap
        W_k = sqrt(covM(x,x)) * normpdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))/(1-normcdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))) + mu_est(x);
        var_artif = 1/beta_W(x) + covM(x,x);
        addscalar = (W_k - mu_est(x))/(var_artif + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((var_artif) + covM(x,x));
    else
        %standard updating equations for Normal-Normal model with covariance
        addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
```

```
        covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));
    end

    %pick the best one to compare OC
    [max_est, max_choice]=max(mu_est);

    %calculate the opportunity cost
    o_cost=max(mu)-mu(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, x]; %update the choice matrix

    if nargout>3 %if more than three outputs were asked
        mu_estALL=[mu_estALL,mu_est];
    end


end

end
```

9.1.4. KG Correlated Beliefs – V2 :

```
%{
Online KG Policy

notation for the following:
K is the number of alternatives.
M is the number of time-steps
K x M stands for a matrix with K rows and M columns

This function takes in
mu:     true values for the mean (K x 1)
mu_0:   prior for the mean (K x 1)
beta_W: measurement precision (1/lambda(x)) (K x 1)
covM:   initial covariance matrix (K,K)
M:      how many measurements will be made (scalar)
maxcap: maximum station capacity
dists:  distances between stations


And returns
mu_est:     Final estimates for the means (K x 1)

OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)
mu_estALL:  Estimates at each iteration (K x M)
%}

function [mu_est, OC, choices, mu_estALL]=KGCorrBeliefs2(mu,mu_0,beta_W,...
    covM,M, maxcap, dists, delta)
```

```
K=length(mu_0); %number of available choices

mu_est=mu_0; %estimates on LAMBDAs

OC=[];
choices=[];
mu_estALL=[];

for k=1:M %exploit M times

    % calculate station demand if we put a station in place x
    weights = ones(K,1);
    for i = 1:length(choices)
        for j = 1:K
        weights(j) = weights(j) + exp(-log(2)*dists(j,choices(i)));
        end
    end

    demand = mu_est ./ weights;

    %Plogy is the log values of KG for alternatives
    Plogy=[];

    for iter1=1:K
        a=demand';
        b=covM(iter1,:)./weights'/weights(iter1)/sqrt(1/beta_W(iter1) + ...
            covM(iter1,iter1)/weights(iter1)^2);
        [Plogy]=[Plogy, LogEmaxAffine(a,b)];
    end

    KG = exp(Plogy);

    KG = demand' + delta*(1-delta^(M-k))/(1-delta)*KG;

    [maxh,x]=max(KG);

    %observe the outcome of the decision
    %W_k=mu_k+Z*SigmaW_k where SigmaW is standard deviation of the
    %error for each observation

    %U_k is what we observe
    W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));
    U_k= min(W_k/weights(x), maxcap);

    %updating equations are conditional on what we observe

    e_x=zeros(K,1);
    e_x(x)=1;

    %special case: observe maxcap
    if U_k == maxcap
        W_k = sqrt(covM(x,x)) * normpdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))/(1-normcdf((maxcap*weights(x)-mu_est(x))/...
```

```
            sqrt(covM(x,x)))) + mu_est(x);
        var_artif = 1/beta_W(x) + covM(x,x);
        addscalar = (W_k - mu_est(x))/(var_artif + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((var_artif) + covM(x,x));
    else
        %standard updating equations for Normal-Normal model with covariance
        addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));
    end

    %pick the best one to compare OC
    [max_est, max_choice]=max(mu_est);

    %calculate the opportunity cost
    o_cost=max(mu)-mu(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, x]; %update the choice matrix

    if nargout>3 %if more than three outputs were asked
        mu_estALL=[mu_estALL,mu_est];
    end


end

end
```

9.1.5. *Pure Exploitation* :

```
%{
Pure Exploitation Policy

notation for the following:
K is the number of alternatives.
M is the number of time-steps
K x M stands for a matrix with K rows and M columns

This function takes in
mu:     true values for the mean (K x 1)
mu_0:   prior for the mean (K x 1)
beta_W: measurement precision (1/lambda(x)) (K x 1)
covM:   initial covariance matrix (K,K)
M:      how many measurements will be made (scalar)
maxcap: maximum station capacity
dists:  distances between stations


And returns
mu_est:     Final estimates for the means (K x 1)
```

```
OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)
mu_estALL:  Estimates at each iteration (K x M)
%}

function [mu_est, OC, choices, mu_estALL]=PureExploit(mu,mu_0,beta_W,covM,...
    M, maxcap, dists)

K=length(mu_0); %number of available choices

mu_est=mu_0; %estimates on LAMBDAs

OC=[];
choices=[];
mu_estALL=[];

for k=1:M %exploit M times

    % calculate station demand if we put a station in place x
    weights = ones(K,1);
    for i = 1:length(choices)
        for j = 1:K
        weights(j) = weights(j) + exp(-log(2)*dists(j,choices(i)));
        end
    end

    demand = mu_est ./ weights;

    [max_value, x] = max(demand);
    %max_value is the best estimated value of the demand
    %x is the argument that produces max_value

    %observe the outcome of the decision
    %W_k=mu_k+Z*SigmaW_k where SigmaW is standard deviation of the
    %error for each observation

    %U_k is what we observe
    W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));
    U_k= min(W_k/weights(x), maxcap);

    %updating equations are conditional on what we observe

    e_x=zeros(K,1);
    e_x(x)=1;

    %special case: observe maxcap
    if U_k == maxcap
        W_k = sqrt(covM(x,x)) * normpdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))/(1-normcdf((maxcap*weights(x)-mu_est(x))/...
            sqrt(covM(x,x)))) + mu_est(x);
        var_artif = 1/beta_W(x) + covM(x,x);
        addscalar = (W_k - mu_est(x))/(var_artif + covM(x,x));
```

```
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((var_artif) + covM(x,x));
    else
        %standard updating equations for Normal-Normal model with covariance
        addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
        mu_est=mu_est + addscalar*covM*e_x;
        covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));
    end

    %pick the best one to compare OC
    [max_est, max_choice]=max(mu_est);

    %calculate the opportunity cost
    o_cost=max(mu)-mu(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, x]; %update the choice matrix

    if nargout>3 %if more than three outputs were asked
        mu_estALL=[mu_estALL,mu_est];
    end



end

end
```

9.2. Auxiliary

9.2.1. Log Transform :

```
% logy = LogEmaxAffine(a,b)
% Calculates log(Exp[max_x a_x + b_x Z]-max_x a_x), where Z is a standard
% normal random variable and a,b are 1xM input vectors.
function [logy, a,b,c] = LogEmaxAffine(a,b)
    if (any(isnan(a)) || any(isnan(b)))
        warning('a or b is NaN');
    end
    assert(all(isreal(a)));
    assert(all(isreal(b)));

    [dummy1,dummy2]=size(a);
    if dummy2>dummy1
        a=a';
    end

    [dummy1,dummy2]=size(b);
    if dummy2>dummy1
        b=b';
    end

    % Check that a and b are column vectors of the right size
    if (any(size(a) ~= size(b)))
        error('LogEmaxAffine: a and b must be column vectors of the same size');
    end

    [a,b] = AffineBreakpointsPrep(a,b);

    [c, keep] = AffineBreakpoints(a,b);
    a = a(keep);
    b = b(keep);
    c = c([1,keep+1]);
    M = length(keep);
    assert(all(isreal(c)));

    % I need logbdiff=log(diff(b)).  I thought that the following code would be
    % more numerically stable, able for example to distinguish cases like
    % logb = [-25 -.3] vs. logb = [-35 -.3], but it doesn't seem to be able to.
    % Indeed, in the debugging output that I have below, the difference was 0.
    %{
    logb = log(abs(b)); % If b is 0, this is -Inf.
    sgnb = sign(b); % If b is 0, this is 0.
    logbdiff = zeros(size(c(2:M)));
    for i=1:length(b)-1
    [logbdiff(i),logbdiffsgn] = LogPlusExpSigned(logb(i),sgnb(i),logb(i+1),...
    -sgnb(i+1));
    %assert(logbdiffsgn>=0);  % The b are distinct, so bdiff(i) can't be 0.
    end
    disp(sprintf('log(b)=%s log(diff(b))=%g logbdiff=%g difference=%g',...
    mat2str(log(b)),log(diff(b)),logbdiff,log(diff(b))-logbdiff));
```

```
    %}
    logbdiff = log(diff(b))';

    if M==1
        logy=log(a);
    elseif M>=2
        logy = LogSumExp(logbdiff+LogEI(-abs(c(2:M))));
    end

    logy=real(logy);
end
% Prepares vectors for passing to AffineEmaxBreakpoints, changing their
% order and removing elements with duplicate slope.

function [a,b] = AffineBreakpointsPrep(a,b)
    % Make sure a and b are column vectors.
    rows = size(a); if (rows == 1), a=a'; end
    rows = size(b); if (rows == 1), b=b'; end

    % 11/29/2008 PF: Experimental preprocessing step, which I hope will remove
    % a large number of the entries.
    [b1, i1] = min(b); % [a1,b1] is best at z=-infinity
    [a2, i2] = max(a); % [a2,b2] is best at z=0
    [b3, i3] = max(b); % [a3,b3] is best at z=+infinity
    a1 = a(i1);
    b2 = b(i2);
    a3 = a(i3);
    cleft = (a - a1)./(b1 - b); % intersection with leftmost line.
    cright = (a - a3)./(b3 - b); % intersection with rightmost line.
    c2left = (a2 - a1)./(b1 - b2); % intersection with leftmost line.
    c2right = (a2 - a3)./(b3 - b2); % intersection with rightmost line.
    keep = find(b==b1 | b==b3 | cleft <= c2left | cright >= c2right);
    %disp(sprintf('Preprocessing cut %d of %d entries', length(a)-length(keep),
    % length(a)));
    a = a(keep);
    b = b(keep);
    clear keep cleft cright


    % Form a matrix for which ba(x,1) is the slope b(x) and ba(x,2) is the
    % y-intercept a(x).  Sort this matrix in ascending order of slope,
    % breaking ties in slope with the y-intercept.
    ba = [b, a];
    ba = sortrows(ba,[1,2]);
    a = ba(:,2);
    b = ba(:,1);

    % Then, from each pair of indices with the b component equal, remove
    % the one with smaller a component.  This code works because the sort
    % above enforced the condition: if b(i) == b(i+1), then a(i) <= a(i+1).
    keep = [find(diff(b)); length(b)];
    % This previous line is equivalent to:
    % keep = [];
```

```
    % for i=[1:length(b)-1]
    %    if b(i)~=b(i+1)
    %        keep = [keep, i];
    %    end
    %end
    %keep = [keep, length(b)];  % We always keep the last one.

    % Note that the elements of keep are in ascending order.
    % This makes it so that b(keep) is still sorted in ascending order.
    a = a(keep);
    b = b(keep);
end

% Inputs are two M-vectors, a and b.
% Requires that the b vector is sorted in increasing order.
% Also requires that the elements of b all be unique.
% This function is used in AffineEmax, and the preparation of generic
% vectors a and b to satisfy the input requirements of this function are
% shown there.
%
% The output is an (M+1)-vector c and a vector A ("A" is for accept).  Think of
% A as a set which is a subset of {1,...,M}.  This output has the property
% that, for any i in {1,...,M} and any real number z,
%   i \in argmax_j a_j + b_j z
% iff
%   i \in A and z \in [c(j+1),c(i+1)],
%   where j = sup {0,1,...,i-1} \cap A.
%
% A note about indexing:
% Since Matlab does not allow indexing from 0, but instead requires
% indexing from 1, what is called c_i in the paper is written in matlab as
% c(1+i).  This is because in the paper we reference c_0.  For the vectors
% a and b, however, we don't need to reference a_0 or b_0, so we reference
% a_i and b_i by a(i) and b(i) respectively, rather than a(i+1) or b(i+1).
%
function [c,A] = AffineBreakpoints(a,b)
    % Preallocate for speed.  Instead of resizing the array A whenever we add
    % to it or delete from it, we keep it the maximal size, and keep a length
    % indicator Alen telling us how many of its entries are good.  When the
    % function ends, we remove the unused elements from A before passing
    % it.
    M = length(a);
    c = zeros(1,M+1);
    A = zeros(1,M);

    % Step 0
    i=0;
    c(1+i) = -inf;
    c(1+i+1) = +inf;
    A(1) = 1;
    Alen = 1;

    for i=[1:M-1]
```

```
        c(1+i+1) = +inf;
        while(1)
            j = A(Alen); % jindex = Alen
            c(1+j) = (a(j) - a(i+1))/(b(i+1)-b(j));
        % The if statement below replaces these lines from version 2 of the
        % function.
        %    kindex = jindex-1 = Alen-1
            %    if kindex > 0 && c(1+j)<=c(1+A(kindex))
            if Alen > 1 && c(1+j)<c(1+A(Alen-1))
        Alen = Alen-1; % Remove last element j
                % continue in while(1) loop
            else
                break % quit while(1) loop
            end
        end
    A(Alen+1) = i+1;
    Alen = Alen + 1;
    end
    A = A(1:Alen);
end

% Returns the log of Exp[(s+Z)^+], where s is a constant and Z is a standard
% normal random variable.  For large negative arguments Exp[(s+Z)^+] function
% is close to 0.  For large positive arguments, the function is close to the
% argument.  For s large enough, s>-10, we use the formula
% Exp[(s+Z)^+] = s*normcdf(s) + normpdf(s).  For smaller s we use an asymptotic
% approximation based on Mill's ratio.  EI stands for "expected improvement",
% since Exp[(s+Z)^+] would be the log of the expected improvement by measuring
% an alternative with excess predictive mean s over the best other measured
% alternative, and predictive variance 0.
function logy = LogEI(s)

% Use the asymptotic approximation for these large negative s.  The
% approximation is derived via:
%   s*normcdf(s) + normpdf(s) = normpdf(s)*[1-|s|normcdf(-|s|)/normpdf(s)]
% and noting that normcdf(-|s|)/normpdf(s) is the Mill's ratio at |s|, which is
% asymptotically approximated by |s|/(s^2+1) [Gordon 1941, also documented in
% Frazier,Powell,Dayanik 2009 on page 14].  This gives,
%   s*normcdf(s) + normpdf(s) = normpdf(s)*[1-s^2/(s^2+1)] = normpdf(s)/(s^2+1).

i=find(s<-10);
if (length(i)>0)
    logy(i) = LogNormPDF(s(i)) - log(s(i).^2 + 1);
end

% Use straightforward routines for s in the more numerically stable region.
i=find(s>=-10);
if (length(i)>0)
    logy(i) = log(s(i).*normcdf(s(i))+normpdf(s(i)));
end

assert(all(isreal(logy)));
end
```

```
% logy = LogNormPDF(z)
% Returns the log of the normal pdf evaluated at z.  z can be a vector or a scalar.
function logy = LogNormPDF(z)
    const = -.5*log(2*pi); % log of 1/sqrt(2pi).
    logy = const - z.^2/2;
end

% function y=LogSumExp(x)
% Computes log(sum(exp(x))) for a vector x, but in a numerically careful way.
function y=LogSumExp(x)
xmax = max(x);
y = xmax + log(sum(exp(x-xmax)));
end
```

9.2.2. Compute Objective :

```
% Computes the objective, given a list of choices at each time step
% a list of truths, a list of distances, a maxcap, and a discount factor
function [obj,total] = computeObj(choices, truths, dists, maxcap, delta)

M = length(choices);
K = length(truths);
obj = zeros(1,M+1);

for k = 1:M

    % calculate weights and demands
    weights = zeros(1,K);
    for i = 1:K
        for j = 1:k
            weights(i) = weights(i) + exp(-log(2)*dists(i,choices(j)));
        end
    end
    currobj = 0;
    for i = 1:k
        currobj = currobj + min(truths(choices(i))/weights(choices(i)),maxcap);
    end
    obj(k+1) = delta * (currobj + obj(k));
end
if delta >= 1
    total = NaN;
else
total = currobj / (1-delta) * delta^(M+2) + obj(M+1);
end


% Computes the objective, given a list of choices at each time step
% a list of truths, a list of distances, a maxcap, and a discount factor
function obj = computeObj2(choices, truths, dists, maxcap, delta)

M = length(choices);
```

```
K = length(truths);
obj = zeros(1,M+1);

for k = 1:M

    % calculate weights and demands
    weights = zeros(1,K);
    for i = 1:K
        for j = 1:k
            weights(i) = weights(i) + exp(-log(2)*dists(i,choices(j)));
        end
    end
    currobj = min(truths(choices(k))/weights(choices(k)),maxcap);
    obj(k+1) = delta * (currobj + obj(k));
end
```

9.3. Run Policies

:

```
% Main Code for testing

% Part 1, read in prior data
% Means
% DAILY TRAFFIC times 0.66% to estimate EV traffic
% EV traffic times 10% to estimate station usage per day
% Station usage times 14 for two week station usage

K = 100;
M = 15;

c = 0.0066 * 14/10;

% priormeans.xlsx contains daily traffic data times 0.0066
priorest = 14/10*xlsread('priormeans.xlsx');

% Variances
priorvars = 5/4 * c^2 * xlsread('priorvars.xlsx') + 1/4 * c^2 * priorest.^2;

% convert means and variances to 1 dimensional vectors (down and then
% right)
priorest = reshape(priorest,[1,K]);
priorvars = reshape(priorvars,[1,K]);

% create distance matrix between points
dists = zeros(K,K);
for i = 1:K
    for j = 1:K
        % |i mod 10 - j mod 10| is vertical distance, |i/10 - j/10| is
        % horizontal distance in nodes.  Vertical node distance is 0.9 km,
        % horizontal node distance is 0.4 km.
        dists(i,j) = 0.9*abs(mod(i-1,10)-mod(j-1,10)) + 0.4*abs(floor((i-1)/10)-...
            floor((j-1)/10));
    end
end

% create covariance matrix

cov = zeros(K,K);
for i = 1:K
    for j = 1:K
        cov(i,j) = exp(-log(2)/3*dists(i,j)) * sqrt(priorvars(i) * priorvars(j));
    end
end
N = 100;
beta_W = 1/100*ones(K,1);
maxcap = 200;

OC1 = zeros(100,15);
```

```
OC2 = zeros(100,15);
OC3 = zeros(100,15);
OC4 = zeros(100,15);
OC5 = zeros(100,15);

choices = zeros(5,100,15);
objExploit = zeros(N,M+1);
objIE = zeros(N,M+1);
objBoltz = zeros(N,M+1);
objKG = zeros(N,M+1);
objKGO = zeros(N,M+1);

obj2Exploit = zeros(N,M+1);
obj2IE = zeros(N,M+1);
obj2Boltz = zeros(N,M+1);
obj2KG = zeros(N,M+1);
obj2KGO = zeros(N,M+1);
T1 = zeros(1,N);
T2 = zeros(1,N);
T3 = zeros(1,N);
T4 = zeros(1,N);
T5 = zeros(1,N);
delta = 0.98;


for i = 1:N
    truth = (priorest' + chol(cov) * randn(K,1));

    [~, OC1(i, :), choices(1,i, :)] = PureExploit(truth, priorest', beta_W,...
        cov, M, maxcap, dists);
    [~, OC2(i, :), choices(2,i, :)] = IE(truth, priorest', beta_W, cov, M,...
        maxcap, dists, 3);
    [~, OC3(i, :), choices(3,i, :)] = Boltzmann(truth, priorest', beta_W, ...
        cov, M, maxcap, dists, 0.75);
    [~, OC4(i, :), choices(4,i, :)] = KGCorrBeliefs1(truth, priorest', ...
        beta_W, cov, M, maxcap, dists);
    [~, OC5(i, :), choices(5,i, :)] = KGCorrBeliefs2(truth, priorest', ...
        beta_W, cov, M, maxcap, dists,delta);

    [objExploit(i,:), T1(i)] = computeObj(choices(1,i,:),truth,dists,maxcap,delta);
    [objIE(i,:),T2(i)] = computeObj(choices(2,i,:),truth,dists,maxcap,delta);
    [objBoltz(i,:),T3(i)] = computeObj(choices(3,i,:),truth,dists,maxcap,delta);
    [objKG(i,:),T4(i)] = computeObj(choices(4,i,:),truth,dists,maxcap,delta);
    [objKGO(i,:),T5(i)] = computeObj(choices(5,i,:),truth,dists,maxcap,delta);
    obj2Exploit(i,:) = computeObj2(choices(1,i,:),truth,dists,maxcap,delta);
    obj2IE(i,:) = computeObj2(choices(2,i,:),truth,dists,maxcap,delta);
    obj2Boltz(i,:) = computeObj2(choices(3,i,:),truth,dists,maxcap,delta);
    obj2KG(i,:) = computeObj2(choices(4,i,:),truth,dists,maxcap,delta);
    obj2KGO(i,:) = computeObj2(choices(5,i,:),truth,dists,maxcap,delta);
end

figure(1)
hold on;
```

```
plot(0:M,mean(objExploit));
plot(0:M,mean(objIE));
plot(0:M,mean(objBoltz));
plot(0:M,mean(objKG));
plot(0:M,mean(objKGO));
xlabel('Station Number');
ylabel('Cummulative Objective');
legend('Exploit','IE','Boltzmann','KG','OnlineKG');


figure(2)
hold on;
plot(1:M, mean(OC1));
plot(1:M, mean(OC2));
plot(1:M, mean(OC3));
plot(1:M, mean(OC4));
plot(1:M, mean(OC5));
xlabel('Station Number');
ylabel('Opportunity Cost');
legend('Exploit','IE','Boltzmann','KG','Online KG');


figure(3)
hold on;
plot(0:M,mean(obj2Exploit));
plot(0:M,mean(obj2IE));
plot(0:M,mean(obj2Boltz));
plot(0:M,mean(obj2KG));
plot(0:M,mean(obj2KGO));
xlabel('Station Number');
ylabel('Cummulative Objective');
legend('Exploit','IE','Boltzmann','KG','Online KG');

mean(T1)
mean(T2)
mean(T3)
mean(T4)
mean(T5)

zCI = 1.96;
L1 = mean(T1) - std(T1)/sqrt(N)*zCI;
L2 = mean(T2) - std(T2)/sqrt(N)*zCI;
L3 = mean(T3) - std(T3)/sqrt(N)*zCI;
L4 = mean(T4) - std(T4)/sqrt(N)*zCI;
L5 = mean(T5) - std(T5)/sqrt(N)*zCI;

R1 = mean(T1) + std(T1)/sqrt(N)*zCI;
R2 = mean(T2) + std(T2)/sqrt(N)*zCI;
R3 = mean(T3) + std(T3)/sqrt(N)*zCI;
R4 = mean(T4) + std(T4)/sqrt(N)*zCI;
R5 = mean(T5) + std(T5)/sqrt(N)*zCI;

%showing CI of final objs
```

```
figure(4);
hold on;
plot([L1;R1],[1,1],'-d');
plot([L2;R2],[2,2],'-d');
plot([L3;R3],[3,3],'-d');
plot([L4;R4],[4,4],'-d');
plot([L5;R5],[5,5],'-d');
ylim([0 6]);
legend('Exploit','IE','Boltzmann','KG','Online KG');


% Main Code for testing

% Part 1, read in prior data
% Means
% DAILY TRAFFIC times 0.66% to estimate EV traffic
% EV traffic times 10% to estimate station usage per day
% Station usage times 14 for two week station usage

K = 100;
M = 15;

priorest = 100*ones(1,K);
priorvars = 36^2*ones(1,K);

% create distance matrix between points
dists = zeros(K,K);
for i = 1:K
    for j = 1:K
        % |i mod 10 - j mod 10| is vertical distance, |i/10 - j/10| is
        % horizontal distance in nodes.  Vertical node distance is 0.9 km,
        % horizontal node distance is 0.4 km.
        dists(i,j) = abs(mod(i-1,10)-mod(j-1,10)) + abs(floor((i-1)/10)-...
            floor((j-1)/10));
    end
end

% create covariance matrix

cov = zeros(K,K);
for i = 1:K
    for j = 1:K
        cov(i,j) = exp(-log(2)/2*dists(i,j)) * sqrt(priorvars(i) * priorvars(j));
    end
end
N = 100;
beta_W = 0.5* ones(K,1);
maxcap = 50;

OC1 = zeros(100,15);
OC2 = zeros(100,15);
OC3 = zeros(100,15);
OC4 = zeros(100,15);
```

```
OC5 = zeros(100,15);

choices = zeros(5,100,15);
objExploit = zeros(N,M+1);
objIE = zeros(N,M+1);
objBoltz = zeros(N,M+1);
objKG = zeros(N,M+1);
objKGO = zeros(N,M+1);

obj2Exploit = zeros(N,M+1);
obj2IE = zeros(N,M+1);
obj2Boltz = zeros(N,M+1);
obj2KG = zeros(N,M+1);
obj2KGO = zeros(N,M+1);
T1 = zeros(1,N);
T2 = zeros(1,N);
T3 = zeros(1,N);
T4 = zeros(1,N);
T5 = zeros(1,N);

delta = 0.98;

for i = 1:N
    truth = (priorest' + chol(cov) * randn(K,1));

    [~, OC1(i, :), choices(1,i, :)] = PureExploit(truth, priorest', beta_W,...
        cov, M, maxcap, dists);
    [~, OC2(i, :), choices(2,i, :)] = IE(truth, priorest', beta_W, cov, M,...
        maxcap, dists, 3);
    [~, OC3(i, :), choices(3,i, :)] = Boltzmann(truth, priorest', beta_W,...
        cov, M, maxcap, dists, 0.75);
    [~, OC4(i, :), choices(4,i, :)] = KGCorrBeliefs1(truth, priorest',...
        beta_W, cov, M, maxcap, dists);
    [~, OC5(i, :), choices(5,i, :)] = KGCorrBeliefs2(truth, priorest', ...
        beta_W, cov, M, maxcap, dists,delta);

    [objExploit(i,:), T1(i)] = computeObj(choices(1,i,:),truth,dists,maxcap,delta);
    [objIE(i,:),T2(i)] = computeObj(choices(2,i,:),truth,dists,maxcap,delta);
    [objBoltz(i,:),T3(i)] = computeObj(choices(3,i,:),truth,dists,maxcap,delta);
    [objKG(i,:),T4(i)] = computeObj(choices(4,i,:),truth,dists,maxcap,delta);
    [objKGO(i,:),T5(i)] = computeObj(choices(5,i,:),truth,dists,maxcap,delta);
    obj2Exploit(i,:) = computeObj2(choices(1,i,:),truth,dists,maxcap,delta);
    obj2IE(i,:) = computeObj2(choices(2,i,:),truth,dists,maxcap,delta);
    obj2Boltz(i,:) = computeObj2(choices(3,i,:),truth,dists,maxcap,delta);
    obj2KG(i,:) = computeObj2(choices(4,i,:),truth,dists,maxcap,delta);
    obj2KGO(i,:) = computeObj2(choices(5,i,:),truth,dists,maxcap,delta);
end

figure(1)
hold on;
plot(0:M,mean(objExploit));
plot(0:M,mean(objIE));
plot(0:M,mean(objBoltz));
```

```
plot(0:M,mean(objKG));
plot(0:M,mean(objKGO));
xlabel('Station Number');
ylabel('Cummulative Objective');
legend('Exploit','IE','Boltzmann','KG','OnlineKG');


figure(2)
hold on;
plot(1:M, mean(OC1));
plot(1:M, mean(OC2));
plot(1:M, mean(OC3));
plot(1:M, mean(OC4));
plot(1:M, mean(OC5));
xlabel('Station Number');
ylabel('Opportunity Cost');
legend('Exploit','IE','Boltzmann','KG','Online KG');


figure(3)
hold on;
plot(0:M,mean(obj2Exploit));
plot(0:M,mean(obj2IE));
plot(0:M,mean(obj2Boltz));
plot(0:M,mean(obj2KG));
plot(0:M,mean(obj2KGO));
xlabel('Station Number');
ylabel('Cummulative Objective');
legend('Exploit','IE','Boltzmann','KG','Online KG');

mean(T1)
mean(T2)
mean(T3)
mean(T4)
mean(T5)

L1 = mean(T1) - std(T1)/sqrt(N)*1.96;
L2 = mean(T2) - std(T2)/sqrt(N)*1.96;
L3 = mean(T3) - std(T3)/sqrt(N)*1.96;
L4 = mean(T4) - std(T4)/sqrt(N)*1.96;
L5 = mean(T5) - std(T5)/sqrt(N)*1.96;

R1 = mean(T1) + std(T1)/sqrt(N)*1.96;
R2 = mean(T2) + std(T2)/sqrt(N)*1.96;
R3 = mean(T3) + std(T3)/sqrt(N)*1.96;
R4 = mean(T4) + std(T4)/sqrt(N)*1.96;
R5 = mean(T5) + std(T5)/sqrt(N)*1.96;
```