Data Procurement for Shortest Paths on Random Graphs

Adam Su
Advised by Yiling Chen and Bo Waggoner

April 1, 2016

A thesis submitted to the Computer Science Department at the Harvard John A. Paulson School of Engineering and Applied Sciences in partial fulfillment of the requirements for a Bachelor of Arts degree in Computer Science and Mathematics with Honors.

Harvard University
Cambridge, MA

Abstract

While Dijkstra's algorithm finds the shortest path between two nodes on a graph with known edge weights, we approach the shortest paths problem for graphs with random edge weights described by known probability distributions. We introduce the idea of a budget of size k which allows us to replace k random edges with numbers drawn from the edges' distributions. Our problem is to determine which edges to replace with random realizations to minimize the minimum expected path distance across all paths between two nodes, given the realized edge weights. We evaluate several greedy heuristics, with different lookaheads, for choosing edges. We also prove that any greedy heuristic with lookahead less than the budget has no finite approximation ratio to the optimal policy.

Contents

1 Introduction

Many problems can be modeled as finding the shortest path between two nodes in a graph. For example suppose we want to find the shortest route from Los Angeles to New York along the U.S. Interstate Highway System. We can model intermediary cities as nodes in our graph, the highways connecting cities as edges connecting nodes, and the lengths of these highways as the weights of the edges. Similarly we can let the nodes, edges, and edge weights of the graph represent different things to solve a wide variety of routing problems. The table in Figure 1 gives a few examples.

problem	nodes	edges	weight
shortest path	cities	highways	distance
fastest path	servers	connections	time
cheapest path	airports	airlines	cost

Figure 1: Graph routing problems

When all the edge weights are known, Dijkstra's algorithm can be used to quickly find the shortest path in a graph. However in practice not all the edge weights will be known. For example the amount of time servers take to send information may depend on how much Internet traffic is going through them at a particular time. Similarly future airfares may be hard to predict and random automobile traffic may make shorter paths less appealing than longer paths that have no traffic. Yet in each of these circumstances, we may be able to model the edge weights as a random variable with some probability distribution based on historical evidence or a predictive model.

To make this more concrete we will focus on one example problem. Suppose we are an oil producer in Calgary, Canada and we need to send a large order of oil to refineries in Houston, Texas through a system of oil pipelines, as quickly as possible. Unfortunately the speed at which oil flows through pipelines depends on many random factors, for example, temperature, the presence of clogs, and pipeline damage. Suppose from our industry experience we can accurately model the flow speeds of every pipeline with a probability distribution. Before we commit our oil to a path down to Houston we can pay local pipeline servicemen to measure the current flow speed of k pipelines. Since we assume our probability distributions are correct, measuring the true flow speed of a pipeline is equivalent to drawing a value

from the pipeline's probability distribution and replacing that distribution with the realized value. Our problem is to choose the sequence of pipelines to measure that will minimize the expected delivery time given the results of the measurements.

In general we propose the following problem. We have a graph with random edge weights modeled by known probability distributions. All of the random weights are independent. We say the expected distance of a particular path from source node to destination node is the sum of the expected values of every edge weight in that path. The minimum of all such path expected distances is the metric we wish to optimize through the process of *inspection*. When we inspect an edge we draw a value x from the probability distribution of that edge, and we assign x to be the new weight of that edge, thereby removing all randomness from that edge. Thus after we inspect an edge, if we wish to use that edge in our path from source to destination, we simply use x as the weight of that path instead of the previously random quantity. We are given a budget of size k that will let us inspect k edges. Intuitively an edge might be good to inspect if knowing its true value will help us, for example, decide between two equally good paths or reveal that some neglected path is more promising than expected. A solution to this problem is a policy that tells us what is the optimal next edge to inspect given the current state of the graph, or equivalently given the history of previous inspections and their results. After the budget is consumed we must choose a *final path* on the final state of the graph, a graph that will have k random edges replaced by realized constant values. Thus the distance of the final path is the sum of random weights and up to k constant weights. We evaluate our policy on the expected distance of this final path.

This problem is hard because the solution space grows exponentially in the size of the budget k. For example, a reasonable policy may want to "look ahead" by considering what the next optimal edge to inspect would be if we were to inspect a particular edge now. The optimal policy would look k steps ahead, considering every possible way our current and future inspection choices could affect our final path. However for a naive implementation this is computationally challenging except for very small graphs, so in this paper we investigate the performance of several heuristics.

A natural heuristic is some policy that looks r steps ahead for $r < k$. Unfortunately, we show that for budget size k, no such policies have a finite approximation ratio to the optimal policy. We prove this by constructing specific graphs where this ratio can be made arbitrarily large. Nonetheless,

it is possible that for many other graphs heuristics work well. First we formulate equivalent and simpler selection criteria for the greedy heuristic, a policy that restricts the lookahead to 1. This helps us implement the greedy heuristic in Python, and through simulation we find that the greedy heuristic soundly beats randomly inspecting edges on many predefined small graphs and randomly generated large graphs. Finally, for one graph with discrete random weights, we compute the exact performances of all policies with lookaheads ranging from 1 to the budget size, for all budget sizes. This graph is small and appears simple, but it is complicated enough to reap the benefits of larger lookaheads.

2 Related Work

2.1 Canadian Traveler Problem

Like our problem, the Canadian Traveler Problem also seeks to find the shortest path from source to destination on a graph whose edges are given known probability distributions. However to inspect an edge we must first travel to the one of the end nodes of that edge. In addition, upon arriving at a node we automatically inspect all edges emerging from that node. This causes some edges to be more expensive to inspect because they are further away. It also imposes an order in which we must inspect edges as we automatically inspect all edges en route to a destination edge. Finally we may at times need to backtrack out of a candidate path if the realized edge weights coming out of a node are all too large to warrant further exploration. In our problem we may inspect any edge for the same cost, and we do all our inspections before any traveling along the edges. In addition our problem has a limit to the number of edges to inspect, and we commit to a single final path after all inspections are done. The Canadian Traveler Problem is #P-hard [4]. While there are no good approximation algorithms for this task, Nikolova and Karger have found efficient exact algorithms for directed acyclic graphs and graphs of disjoint paths from source to destination with random edges that can take on two values [3].

2.2 Information Collection on a Graph

In a different model, graphs have random edges with prior probability distributions. Here inspections are used to update these priors by collecting many random samples from these edges' distributions. That is, we may inspect a single edge multiple times to get a more and more accurate posterior distribution for that edge. Ryzhov and Powell have proposed a greedy information heuristic to choose which edge to inspect next [5]. Like our model, this model allows inspection of any edge, without traveling to one of its end nodes. In contrast, our inspection gives the true value of the edge because the true value is just one random realization from the edge's distribution. In the model by Ryzhov and Powell there is a generally a large number of inspections available. In our problem the number of inspections is at most the number of edges in the graph. However in the case where we may inspect every edge, our shortest paths problem can be solved trivially with Dijkstra's Algorithm.

2.3 Data Procurement

The idea of introducing a budget to learn the optimal path to take from source to destination was inspired by a paper by Chen and Waggoner that studies data procurement in the context of machine learning. They propose criteria for using a budget to buy specific data points in order to minimize regret, and they prove regret bounds in terms of this budget [1]. In this model data vendors, each offering one data point, arrive one by one in an online fashion, and the agent must make a decision upon every arrival whether to purchase the data point or not. In contrast, in our model we have access to the entire marketplace of random edges. Like our problem, generally not all of the data can be purchased, and we must decide which data points are the most valuable. We also borrow the idea that realizations from previous inspections can affect what we choose to inspect in the future. That is, we allow our policy to adapt to new data.

2.4 Weitzman's Pandora's Box

In Weitzman's Pandora's Box problem there are boxes with random rewards inside and associated costs to open them. The problem is to determine the order in which boxes should be opened and when to stop opening boxes with

the objective of maximizing the maximum reward out of all opened boxes. Our problem is a generalization of a modified Weitzman's Pandora's Box problem that replaces the costs of opening specific boxes with an overall budget for how many boxes we can open. For example the graph in Figure 2 is an instance of Weitzman's Pandora's Box problem with three boxes with random rewards X_1, X_2, X_3. The solution to Weitzman's Pandora's Box problem utilizes a heuristic that is closely related to our greedy heuristic [6]. However our problem is significantly harder because the boxes in Weitzman's Pandora's Box problem are independent, but edge inspections in our problem may have complicated interaction effects with other paths in the graph. Thus we cannot simply enumerate all paths from source to destination and use an index policy to optimally solve our problem, as is done in Weitzman's Pandora's Box problem.

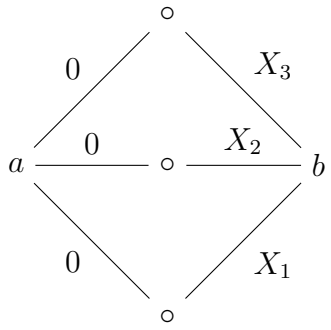


Figure 2: Graph instance of Weitzman's Pandora's Box problem

3 Model

We now describe the mathematical model rigorously. Let G be an undirected graph with nodes V and edges E. We assign to every edge a unique index in $\{1, 2, \ldots, |E|\}$, and in this paper when we refer to the edge e, we primarily mean the index of that edge. Let every edge $e \in E$ have a random weight, which we write as the random variable $X_e \geq 0$ where $E[X_e]$ is finite. All of the random variables are independent. Let G have a fixed source node a and destination node b, and let P be the set of all paths from a to b. We assume P is not empty. We define the shortest path from a to b as the path

that minimizes the expected distance from a to b. In addition let $k \leq |E|$ be the size of the *budget*, which determines how many edges we can *inspect*. When we inspect an edge e we replace the random variable associated with that edge, X_e, with a random realization from the probability distribution of X_e. Since inspecting an edge will never increase the expected distance of the final path in expectation, we require the entire budget to be used. We use the terms "inspect e" and "inspect X_e" interchangeably.

After k inspections we must choose a *final path* on the final state of the graph, a graph that now has k random edges replaced by realized constant values. The distance of this final path is the sum of random weights and up to k constant weights. If the final path has random edges, we simply use the mean value of those edges to compute the expected distance of this path, since all edge weights are independent. Our goal is to minimize the expected distance of final path. To accomplish our goal we would like a policy that will always tell us what is the optimal edge to inspect next until the budget is consumed. We evaluate our policy on the expected distance of the final path.

3.1 Notation

Now we introduce some useful notation. Let

$$\ell(S) := \sum_{e \in S} X_e$$

where S is a set of edges. We define the history at time t, $h(t)$, as the set of inspected edges and their realizations, where time t is the number of edges we have inspected already. For example $h(0) = \emptyset$, because we have not inspected any edges at time 0. If we have inspected X_5 and realized a value of x_5, and X_3 and realized a value of x_3, then $h(2) = \{X_5 = x_5, X_3 = x_3\}$. For simplicity in this paper we will write $h \cup (X_e = x_e)$ in place of $h \cup \{(X_e = x_e)\}$ for history h. Now we define

$$D(h(t)) := \min_{p \in P} E[\ell(p) \mid h(t)].$$

This is the minimum expected distance across all paths from a to b given the history of inspections at time t. Thus the distance before any inspections is

$$D(\emptyset) = \min_{p \in P} E[\ell(p)].$$

Occasionally it is useful to compute this distance metric for an arbitrary source node u and destination node v. Thus we define

$$D_{u,v}(h(t)) := \min_{p \in P_{u,v}} E[\ell(p) \mid h(t)]$$

where $P_{u,v}$ is the set of paths from u to v. If $P_{u,v} = \emptyset$, then $D_{u,v}(h(t)) = \infty$.

We formalize the way we evaluate a policy, Pol, by defining the *value* of Pol as

$$V(Pol) := E[D(H^{Pol}(k))],$$

where $H^{Pol}(k)$ is the random history of all inspections Pol has made by time k. That is, a realization of $H^{Pol}(k)$ is some history h of size k. Here we take the expectation over the randomness of $H^{Pol}(k)$, which potentially comes from two sources. First, Pol may use randomization to choose edges to inspect. For example, let the policy *Random* choose the next edge to inspect uniformly randomly from the remaining random edges. Second, the realizations, x_i, of the edges Pol inspects are random. Notice also that the set of edges we inspect may change due to these two sources of randomness.

3.2 Optimal Policy

We now describe the optimal policy for a budget of size k, $Opt(k)$. We will write $V(Opt(k))$ in terms of another policy $Opt(k, h)$, which is the optimal policy for a budget of size k on a copy of the original graph G with all random edges in h replaced by their realized values.

$$\begin{aligned} V(Opt(k)) &= E[D(H^{Opt(k)}(k))] \\ &= \min_{e \in E} \int \Pr(X_e = x) V(Opt(k-1, \{(X_e = x)\}) dx. \end{aligned}$$

Now we are ready to define $Opt(k)$ as the policy that at time t inspects the edge

$$Inspect(k-t, h(t)) :=$$

$$\arg\min_{e \in E} \int \Pr(X_e = x) V(Opt(k-t-1, h(t) \cup (X_e = x))) dx.$$

The base case for computing $Inspect(k-t, h(t))$ is $V(Opt(0, h)) = D(h)$ for some history h. We describe how to compute this in subsection 3.4.

After we inspect $e = Inspect(k-t, h(t))$ we replace X_e with a random realization from the distribution of X_e. Then we repeat this process for time $t+1$ and so on, until we have inspected k edges.

3.3 Lookahead Policies

Since $Inspect(k-t, h(t))$ can be computationally intensive to calculate when $k-t$ is large, we also study policies $Lookahead(r)$ for $1 \leq r \leq k-1$, which inspect $Inspect(min(k-t, r), h(t))$ at time t. The inspection process works the same way as in $Opt(k)$ until we have inspected k edges.

3.4 Calculating $D(h(t))$

Conceptually to calculate $D(h(t))$, we must find the path of minimum expected length given some history. No matter what path edge e is in, its contribution to the expected length of that path is $E[X_e]$ if e has not been inspected, and its realized constant value if it has. This means we can calculate $D(\emptyset)$ using Dijkstra's algorithm on a copy of the random graph where all edge weights X_e are replaced by $E[X_e]$. We can calculate $D(h(t))$ similarly except we first replace all inspected edges with their realizations as given in $h(t)$, and then we replace the remaining edge weights with their expectations.

3.5 Simple Example

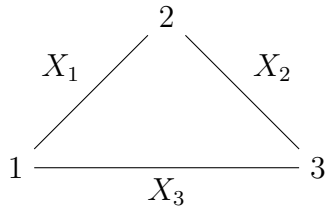


Figure 3: Simple random graph

We present a simple example using the graph in Figure 3. This graph has three vertices: 1, 2, and 3. Let the source node be 1 and the destination node be 3. The edges have random weights $X_1 \sim$ Unif(0,1), $X_2 \sim$ Unif(0,1), $X_3 \sim$ Unif(0,2). Suppose we have a budget of 1. This means we will first inspect one edge, then pick the path in $\{\{(1,2),(2,3)\},\{(1,3)\}\}$ that has smaller expected length given what we learned from inspecting that edge. Now we choose which edge to inspect. By symmetry inspecting X_1 and X_2 will give the same result, so our choice is effectively between X_1 and X_3.

If we inspect X_1, 50% of the time we will realize a value less than $E[X_1] = 0.5$ from its Unif(0,1) distribution. In this case $\{(1,2),(2,3)\}$ is the final path, and the expected length of this path is 0.75 because $E[X_2] = 0.5$ and $E[X_1 \mid X_1 < 0.5] = 0.25$. The other 50% of the time we draw a value greater than 0.5. In this case the final path is $\{(1,3)\}$ and has expected length 1. Averaging the expected lengths in these two equally likely cases, we calculate that the value of the policy that inspects X_1 is 0.875.

Now if we inspect X_3, 50% of the time we will get a value less than $E[X_3] = 1$. In this case the final path is $\{(1,3)\}$. The expected length of this path is $E[X_3 \mid X_3 < 1] = 0.5$. The other 50% of the time, we get a value greater than 1. In this case the final path is $\{(1,2),(2,3)\}$ and has expected length 1. Averaging the expected lengths in these two equally likely cases, we calculate that the value of the policy that inspects X_3 is 0.75. Thus for the graph in Figure 3 with a budget of 1, the optimal policy inspects X_3.

3.6 Adaptive Example

When the budget is greater than 1 the choice of the optimal next edge to inspect may depend on the history. In Figure 4 we draw a graph where the optimal second edge to inspect depends on the realized value of the first edge we inspect. The weights for edges $(1,4)$ and $(1,2)$ are constant, or equivalently, random variables that only take on one value. Let the source node be 1 and the destination node be 4.

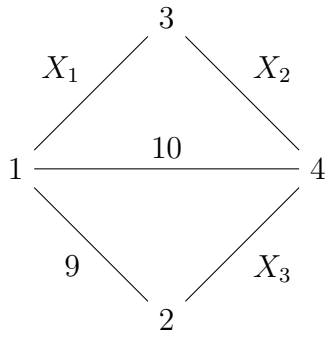


Figure 4: Graph with adaptive optimal policy

Here X_1, X_2 each have 80% chance of being 0 and 20% chance of being 25. On the other hand, X_3 has 50% chance of being 0 and 50% chance of

being 2. The optimal policy is adaptive. We first inspect X_1. Then if the realized value of edge X_1 is 25, we inspect X_3. Otherwise we inspect X_2.

4 Results

4.1 A Negative Result

Theorem For all $k > 1, r < k$, there is no constant C where

$$\frac{V(Lookahead(r))}{V(Opt(k))} \leq C$$

for all graphs. That is, $V(Lookahead(r))$ has no finite approximation ratio to $V(Opt(k))$.

Proof First we define two discrete probability density functions y and z in terms of a small positive constant $\varepsilon > 0$.

$$y(0) = 1 - \varepsilon, y(1/\varepsilon) = \varepsilon$$

$$z\left(\frac{1}{r} - \varepsilon\right) = \varepsilon, z\left(\frac{1}{r} + \frac{\varepsilon^2}{1-\varepsilon}\right) = 1 - \varepsilon$$

Now we construct a graph $G_{k,r}$ with $k+r+1$ nodes labeled $a, b, 1, 2, \ldots, k+r-1$. Let $G_{k,r}$ have the edges in $Y \cup Z \cup \{(a, b), (k + r - 1, b)\}$ where

$$Y = \{(a, 1), (1, 2), (2, 3), \ldots, (k - 2, k - 1), (k - 1, b)\}$$

and

$$Z = \{(a, k), (k, k+1), (k+1, k+2), \ldots, (k+r-3, k+r-2), (k+r-2, k+r-1)\}.$$

The edge (a, b) has constant weight 1, and the edge $(k+r-1, b)$ has constant weight $(r - 1)\varepsilon$.

Let every edge in Y have random weight independently and identically distributed with density y. We assign each of these random weights a unique identifier Y_i, $1 \leq i \leq k$. Similarly let every edge in Z have random weight independently and identically distributed with density z. We assign each of these random weights a unique identifier Z_i, $1 \leq i \leq r$. Note that $E[Y_i] = 1, E[Z_i] = 1/r$. As an example we draw $G_{4,3}$ in Figure 5.

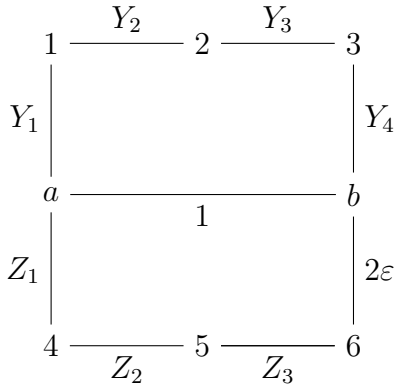


Figure 5: *Lookahead*(3) performs much worse than $Opt(4)$ on $G_{4,3}$.

Note that Y and $Z' = Z \cup \{(k + r - 1, b)\}$ are paths from a to b. Before any inspections path Y has expected length k, path Z' has expected length $1 + (r - 1)\varepsilon$, and path $\{(a, b)\}$ has constant length 1.

Consider the policy of inspecting every edge in Y. This policy has value $0(1 - \varepsilon)^k + 1(1 - (1 - \varepsilon)^k)$ because if all realizations are 0, then the shortest path is Y with distance 0, but if any realization is $1/\varepsilon$ then a shortest path is $\{(a, b)\}$ with distance 1. Thus $V(Opt(k)) \leq 1 - (1 - \varepsilon)^k$.

Now we determine the first edge *Lookahead*(r) inspects, which is

$$Inspect(r, \emptyset) =$$

$$\arg\min_{e \in E} \int \Pr(X_e = x) V(Opt(r - 1, \{(X_e = x)\}))dx.$$

For convenience let

$$J(X) = \int \Pr(X = x) V(Opt(r - 1, \{(X = x)\}))dx.$$

We can evaluate $J(X)$ recursively with base cases $V(Opt(0, h)) = D(h)$ for some history of inspections h where $|h| = r$. Note that $D(h) \leq 1$ no matter what h is because we can always take path $\{(a, b)\}$ with distance 1. Two facts will help us show $J(Z_i) < J(Y_j)$ for all i, j. First, the expected distance of path Y is only less than 1 if all edges in Y are inspected and all realized values are 0. Second, the expected distance of path Z' is only less

than 1 if all edges in Z are inspected and all realized values are $(1/r) - \varepsilon$. Thus the only history h_0 such that $D(h_0) < 1$ and $|h_0| = r$ is

$$h_0 = \left\{ \left(Z_i = \frac{1}{r} - \varepsilon \right) \ \forall\ 1 \le i \le r \right\},$$

because if we inspect every edge in Z, and every realization is $(1/r) - \varepsilon$, then path Z' has expected distance $1 - \varepsilon$. Since $D(h_0) < 1$ is a base case of $J(Z_1)$ and $D(h) \le 1\ \forall\ h$, $J(Z_1) < 1$. Finally $J(Z_i) < 1\ \forall\ i$, as all of the $J(Z_i)$ are the same by symmetry.

On the other hand $J(Y_i) = 1$ for all i, because after any history of $r < k$ inspections that includes the inspection of some Y_i, there is no way for either path Y or Z' to have expected distance less than 1. Thus the first edge $Lookahead(r)$ inspects is some Z_i. Intuitively this makes sense as $Lookahead(r)$ can only consider the potential realizations of at most r edges at a time, and the only set of r edges to inspect that could ever result in a path with expected distance less than 1 is Z.

Now $Lookahead(r)$ must choose $k - 1$ more edges to inspect. Since we only have $k - 1$ inspections left we cannot inspect all edges in Y. Thus the minimum possible expected distance of final path is $1 - \varepsilon$. This distance is achieved when we inspect the remaining $r-1$ edges in Z and $k-r$ edges in Y, and the realized values of the Z_i are all $1-\varepsilon$. Thus $V(Lookahead(r)) \ge 1-\varepsilon$. Finally

$$\lim_{\varepsilon \to 0} \frac{V(Lookahead(r))}{V(Opt(k))} \ge \lim_{\varepsilon \to 0} \frac{1-\varepsilon}{1-(1-\varepsilon)^k} = \infty.$$

4.2 Greedy Heuristic

We describe how to find $Inspect(1, h)$, which is used in $Lookahead(1)$, the greedy policy. By definition

$$\begin{aligned} Inspect(1,h) &:= \arg\min_{e \in E} \int \Pr(X_e = x) V(Opt(0, h \cup (X_e = x)))dx \\ &= \arg\min_{e \in E} \int f(x) D(h \cup (X_e = x))dx \end{aligned}$$

where $f(x)$ is the probability density function of X_e. We will use $F(x) = \Pr(X_e \le x)$ to mean the cumulative density function of X_e. To evaluate

$Inspect(1, h)$, it is useful to first compute $d = D(h)$ and the *baseline path*

$$B(h) := \arg\min_{p \in P} E[\ell(p) \mid h]$$

of the current state of the graph. For convenience define

$$K(X) = \int f(x) D(h \cup (X = x)) dx$$

for a given history h, where X is some random weight in the graph. Thus finding $Inspect(1, h)$ is equivalent to finding the random weight X that minimizes $K(X)$. Now we describe how to evaluate $K(X)$.

Edges not in the Baseline Path Let the edge $(u, v) \notin B(h)$ have random weight X, and let $c = \min(D_{s,u}(h)+D_{v,t}(h), D_{s,v}(h)+D_{u,t}(h))$. Let $m = d-c$. Then for $h' = h \cup (X = x)$, $D(h') < D(h)$ if $x < m$ and $D(h') = D(h)$ if $x \geq m$. If $m \leq 0$ then

$$K(X) = d.$$

Otherwise

$$\begin{aligned} K(X) &= \int_0^m f(x) D(h \cup (X = x)) dx + \int_m^\infty f(x) D(h \cup (X = x)) dx \\ &= \int_0^m f(x)(d - m + x) dx + (1 - F(m))d \\ &= F(m)d + \int_0^m f(x)(x - m) dx + (1 - F(m))d \\ &= d + \int_0^m f(x) x dx - F(m)m. \end{aligned}$$

Now if $F(m) = 0$, then

$$\int_0^m f(x) x dx = 0.$$

Otherwise

$$\begin{aligned} \int_0^m f(x) x dx &= \frac{F(m) \int_0^m f(x) x dx}{F(m)} \\ &= F(m) E[X \mid X \leq m]. \end{aligned}$$

Finally

$$\begin{aligned} K(X) &= d + F(m)E[X \mid X \leq m] - F(m)m \\ &= d + F(m)E[X - m \mid X \leq m]. \end{aligned}$$

Edges in the Baseline Path For any edge $e \in B(h)$, let $c = D(h \cup (X_e = \infty))$. If $c = \infty$, then

$$K(X) = d.$$

Suppose c is finite, and let $E[X] = \mu$. Let $m = c - (d - \mu)$. Then for $h' = h \cup (X = x)$, $D(h') = c$ if $x \geq m$. Note that $m = \mu + (c - d) \geq \mu$. Now

$$\begin{aligned} K(X) &= \int_0^m f(x)D(h \cup (X = x))dx + \int_m^\infty f(x)D(h \cup (X = x))dx \\ &= \int_0^m f(x)(d - \mu + x)dx + (1 - F(m))c \\ &= F(m)(d - \mu) + \int_0^m f(x)xdx + (1 - F(m))c \\ &= F(m)(d - \mu) + F(m)E[X \mid X \leq m] + (1 - F(m))(d + c - d) \\ &= d + F(m)E[X - \mu \mid X \leq m] + (1 - F(m))(c - d) \\ &= d + F(\mu)E[X - \mu \mid X \leq \mu] + \\ &\quad (F(m) - F(\mu))E[X - \mu \mid \mu < X \leq m] + (1 - F(m))(c - d) \end{aligned}$$

For both cases the right hand side of the expressions for $K(X)$ are easy to calculate for simple probability distributions. We make extensive use of these formulas in our simulations.

5 Simulation

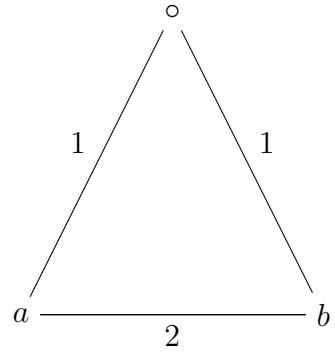
5.1 Methods

We use Python to implement random graphs, run simulations of policies, and evaluate policy performance. We specify by hand the structures of small random graphs and randomly generate large ones. We plot the estimated values of several policies against budget size, and we discuss the results for representative graphs in each section below. The remaining results can be

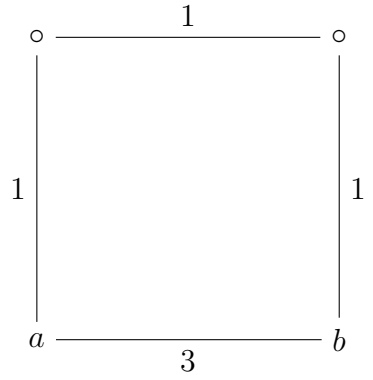
found in the Appendix. We estimate the value of a policy by taking the average expected distance of the final path across all trials of a simulation. Python code for running simulations is available at `https://github.com/HarvardEconCS/adam_su_senior_thesis`.

5.2 Lookahead(1) Small Graphs

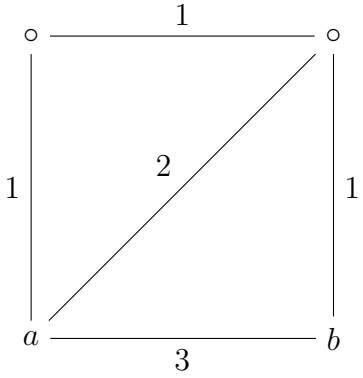
We test the performance of *Lookahead*(1) by comparing it to the policy *Random*, which we described in subsection 3.1. We do this for all budgets $0 \leq k \leq |E|$. We include budgets 0 and $|E|$ for completeness and for benchmarking. In this subsection and subsection 5.3, for every trial of a simulation we first draw a number from the probability distribution of every edge in the graph. These numbers are hidden until revealed by inspection but are the same for both policies for a given trial. We simulate both policies on the 8 graphs in Figures 6 and 7 10,000 times each.



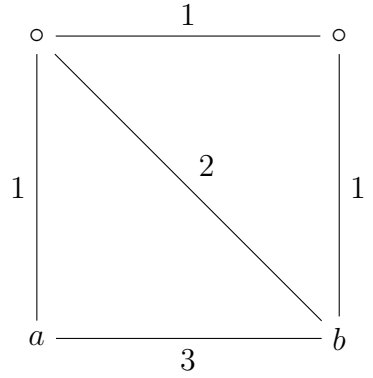
$n3$



$n4x1$

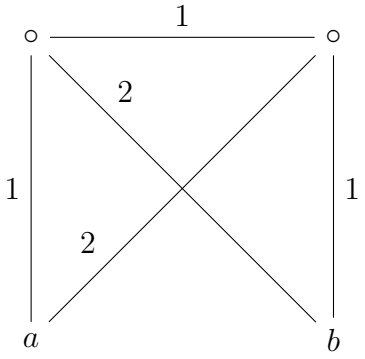


$n4x2$

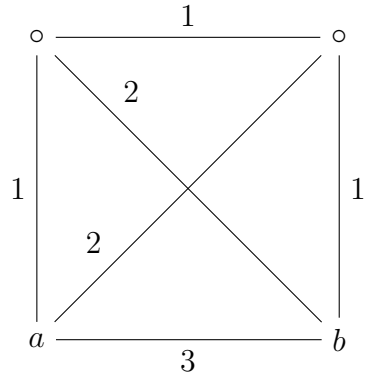


$n4x3$

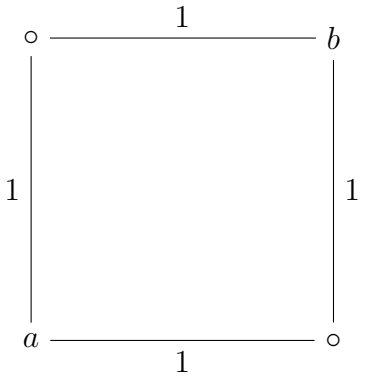
Figure 6: Small graphs



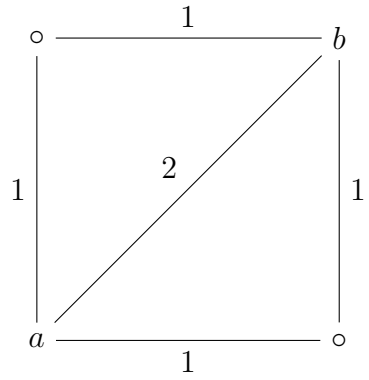
$n4x4$



$n4x5$



$n4x6$



$n4x7$

Figure 7: More small graphs

We choose these graph structures because they are all graphs with at most 4 nodes where every node has degree at least 2. We require the degree of every node to be at least 2 because if a node n, with neighbor m, has degree 1, then we either never need to include the edge (n, m) in our final path, or that node is the source or destination. In the latter case we never need to inspect (n, m) because we must take that edge no matter what. Thus we can

remove n from the graph, call m respectively the new source or destination. Then we proceed with the same policy on the new graph with budget the minimum of the original budget and the number of edges in the new graph.

After choosing the graph structures we assign a and b to two nodes in all possible ways up to symmetry. In Figures 6 and 7 an edge label of i represents a Unif(0,i) random weight for $i = 1, 2, 3$. We choose these weights so no single path from a to b is the clear favorite before inspections.

Notice that for $Pol \in \{Random, Lookahead(1)\}$, if we run Pol with two different budgets k, k' on the same graph with the same hidden realized values, $H^{Pol}(t)$ is the same for both budgets for all $t \leq min(k, k')$. This property lets us simulate Pol for all budgets when we run one trial with budget size $|E|$. In Figure 8 we plot the results of a few trials that show how the expected distance of the minimum path changes as we inspect more edges. Figure 9 is a boxplot that gives summary statistics that describe the distribution of results for 100 trials. After understanding where the data comes from and what the data looks like, we are ready to analyze the plots in Figures 10, 11, and 12. These plot the mean expected distance of the final path across all 10,000 trials for three example graphs along with the standard error of the mean. Plots for the remaining graphs can be found in the Appendix.

Most of the time, the greedy policy outperforms the random policy significantly. The performance gap is greatest for graphs with a small number of edges with high variance. Because the graphs we run simulations on start with many potential baseline paths, the greedy algorithm can risk inspecting high-variance edges, knowing that even if a large value is realized, it can always fall back on a baseline path. The performance gap is especially large when there are a small number of these high-variance edges, because then the random policy is unlikely to choose them by chance, while the greedy policy will always find them.

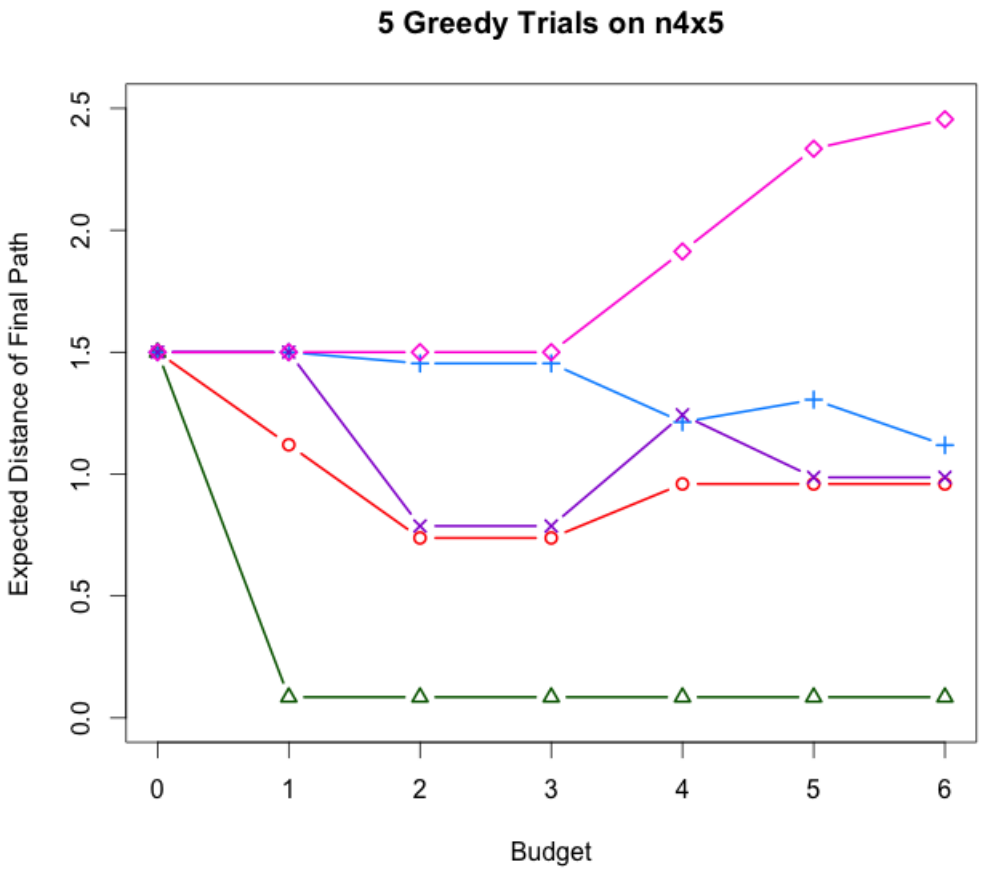


Figure 8: This is the result of 5 independent trials of running the greedy policy on graph $n4x5$ with budget $|E|$. On the x-axis is the number of edges inspected so far. Equivalently, we can think of this as running the same policy with smaller budget. Thus each trial with budget 6 produces simulation data for budgets 0 to 5 as well. On the y-axis is the expected distance of the final path. Sometimes as we inspect we realize large values (magenta), so the expected distance of the final path goes up. Other times the first inspection leads to a great baseline path (green), and future inspections do not find any better paths.

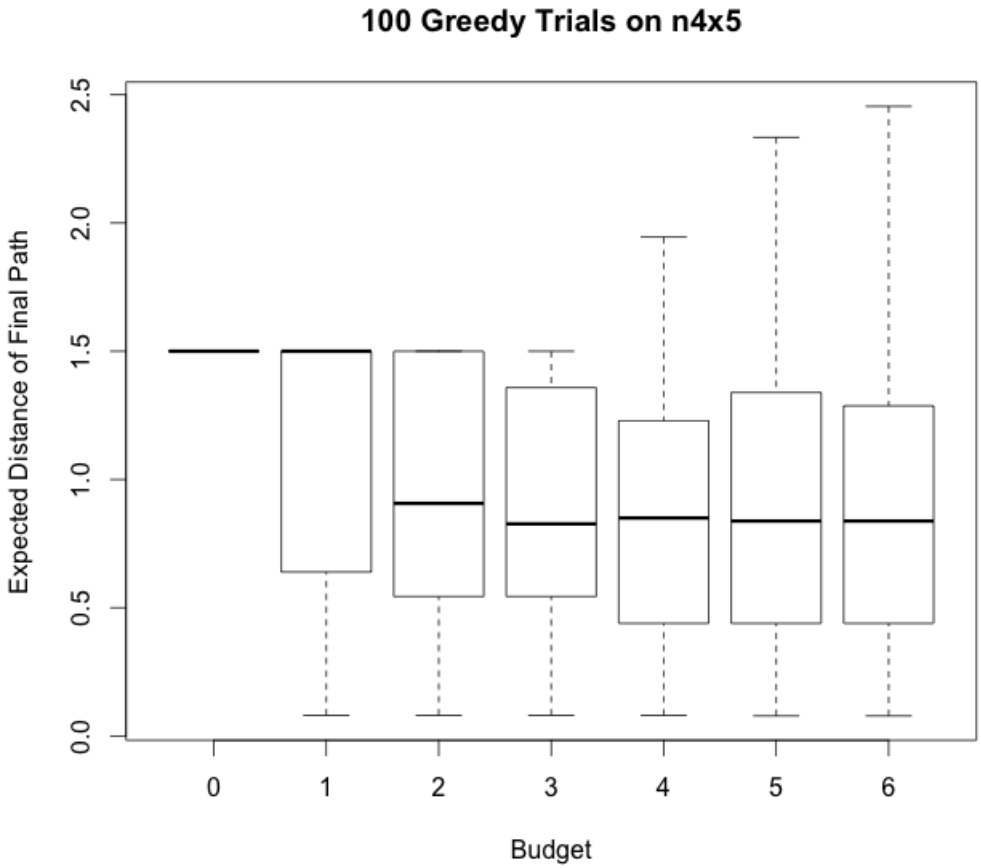


Figure 9: This is a boxplot of the distances achieved in the first 100 trials of the greedy policy. For each budget we give the minimum, 25^{th} percentile, median, 75^{th} percentile, and maximum values of the expected distance of the final path. The purpose of this plot is to give an idea of what the distribution of our results looks like, because in our full simulation results we only plot the mean distance for every budget and its standard error.

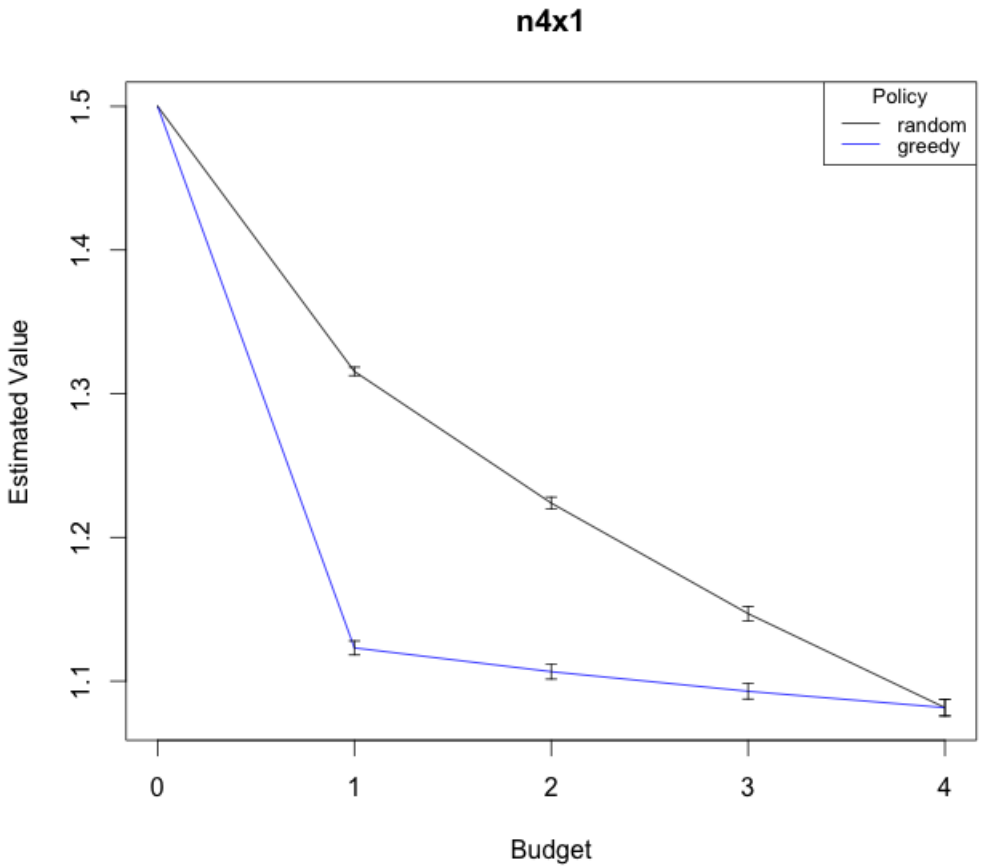


Figure 10: The performance gap is large with budget 1 because there is only 1 high variance edge weight, distributed Unif(0,3). The greedy policy always inspects this first, and the random policy is unlikely to choose it.

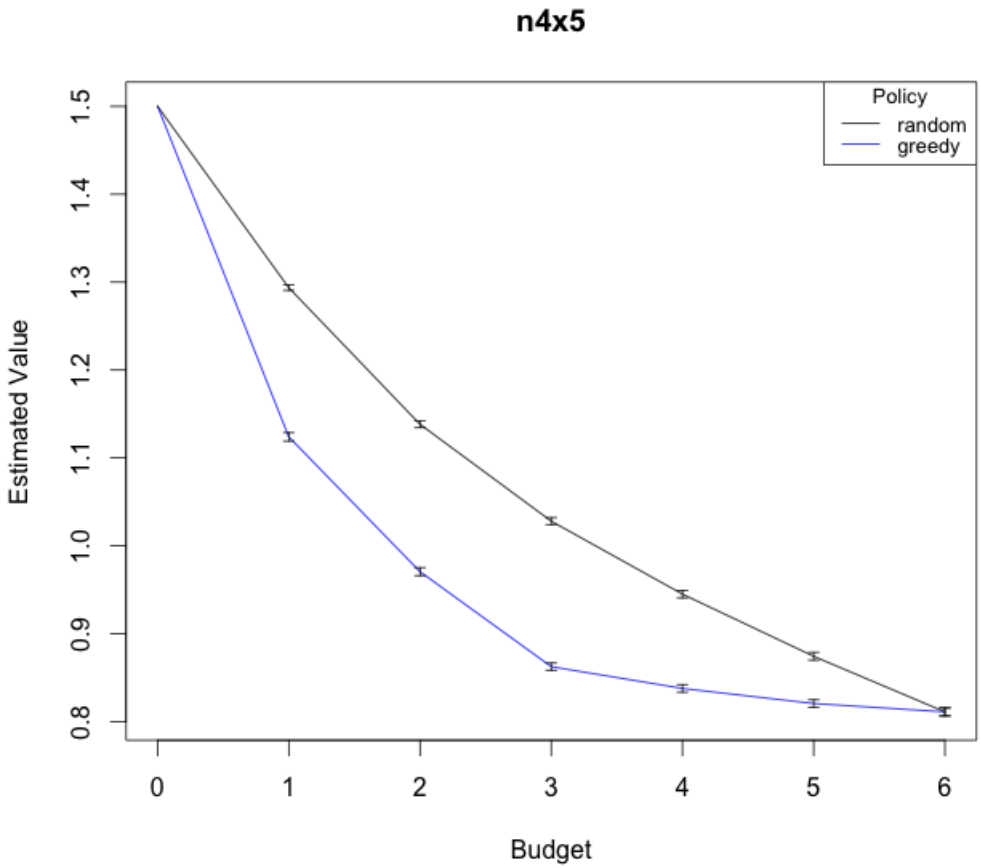


Figure 11: The performance gap is intermediate for a range of budgets because there is 1 high variance edge weight, distributed Unif(0,3), and 2 medium-variance edge weights, distributed Unif(0,2). The presence of these higher-variance edge weights helps the greedy policy perform better, but the random policy may still make a good choice by chance.

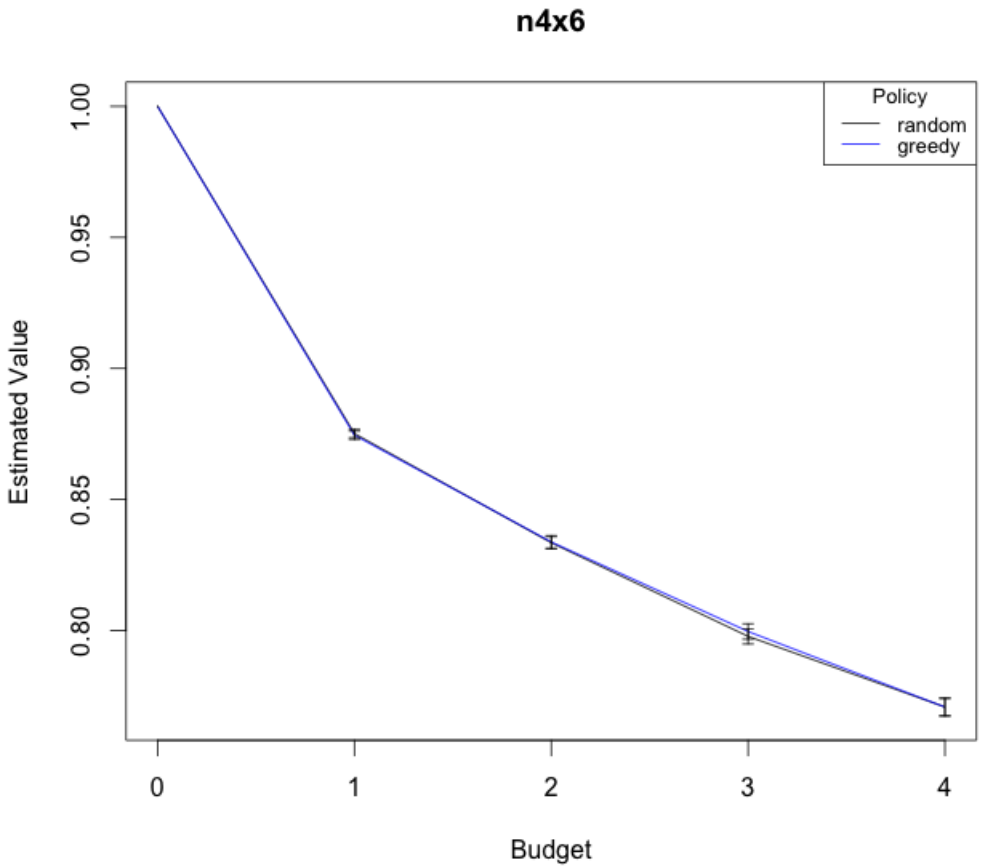


Figure 12: Sometimes the random and greedy policies have identical performances, for example in graph $n4x6$.

5.3 Lookahead(1) Large Graphs

To construct a large random graph we follow the Erdős-Rényi model [2]. Our process is to randomly generate many Erdős-Rényi graphs with 50 nodes and connectivity 0.05 until we find 10 with diameter greater than 10 in its largest component and greater than 40 nodes in its largest component. We set a and b to be the endpoints of the diameter. We produce graphs with large diameters because we think they are more likely to have many promising paths

from a to b. Every edge weight is independently and identically distributed Unif(0,1).

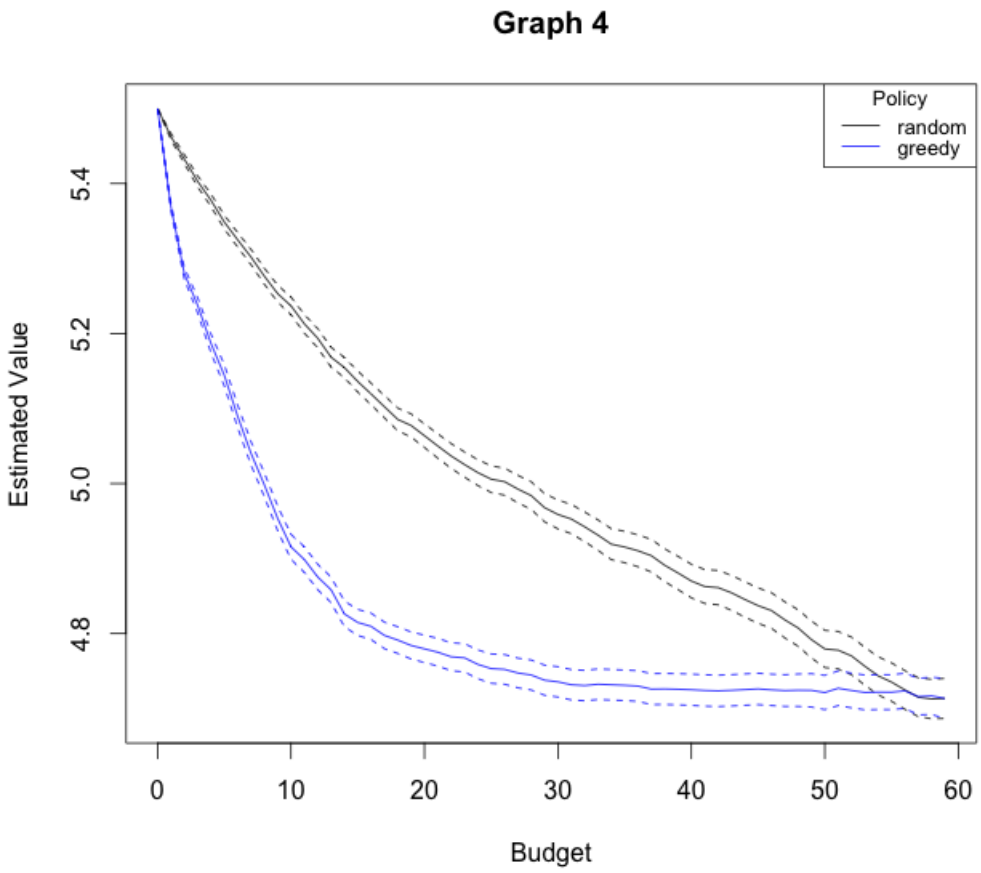


Figure 13: For graph 4, the greedy policy clearly performs better than the random policy almost across the entire range of budgets. With budget 20, the greedy policy almost achieves the value of either policy with budget $|E|$. Seven other graphs (1, 2, 5, 6, 8, 9, and 10) have results that look similar, and their plots are in the Appendix.

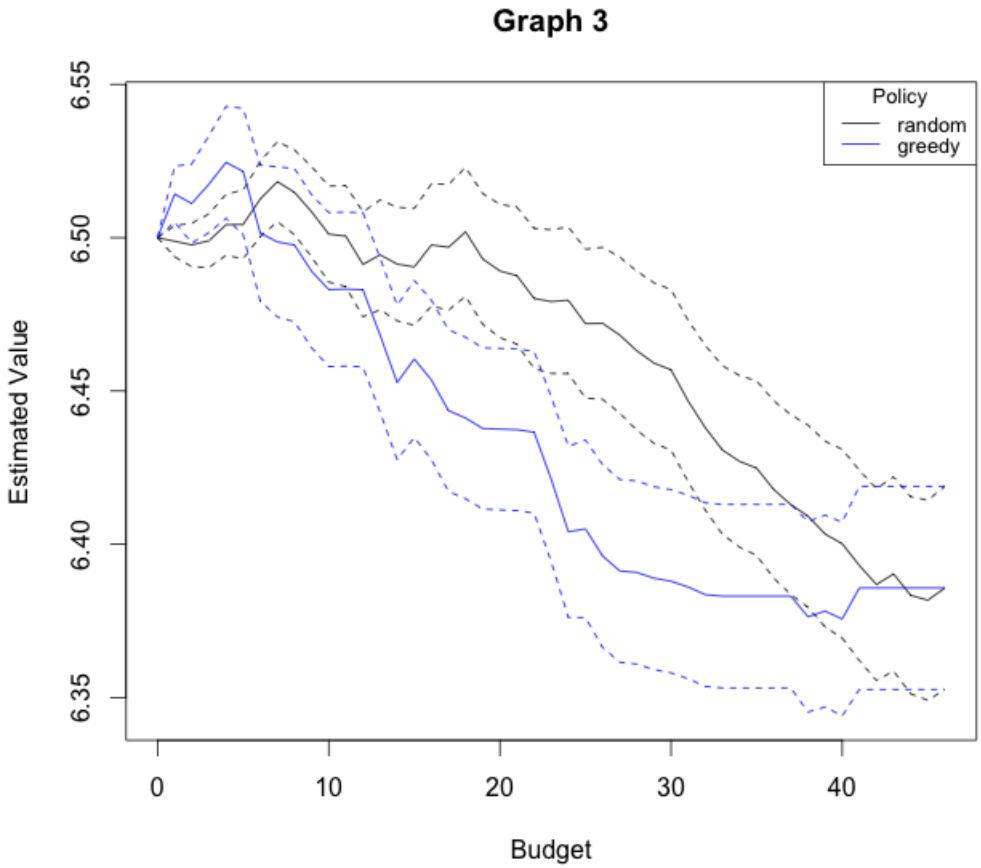


Figure 14: For graph 3 it is not clear whether the greedy policy performs better than the random policy across all budgets. For small budgets randomly inspecting edges may even be a better policy than following the greedy heuristic. However it appears that greedy does outperform random for medium sized budgets. Graph 7 has results that look similar, and its plot is in the Appendix.

In summary for most graphs we found that the greedy policy soundly beats the random policy. In the remaining graphs it was not clear if the greedy policy was significantly better than the random policy.

5.4 All Lookaheads and Opt(k)

For budget $1 \leq k \leq 7$ and lookahead $1 \leq r < k$ we simulate $Opt(k)$ and $Lookahead(r)$ with a budget of k on the graph in Figure 15, whose edge weights are all Bern(0.5). For convenience we may refer to $Opt(k)$ as $Lookahead(k)$ in this subsection. We chose to simulate on this graph because, before any inspections, there are many reasonable ways to get from a to b. In addition every edge is part of multiple paths, which we expect to create interesting interaction effects between inspections. This helps highlight the benefits of policies with higher lookaheads.

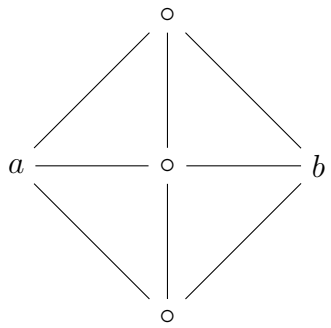


Figure 15: We simulate the optimal policy and policies of all possible lookaheads on this graph.

Now for graphs with discrete random weights, the integrals in the definition of $Inspect(r, h)$ from subsection 3.2 become sums and are much easier to evaluate. For such graphs we can also compute the exact value of any deterministic policy by averaging the expected distance of the final path across all realizations of the graph. Because there are only $2^{|E|} = 256$ different realizations of all the edges in the graph in Figure 15, we can compute exact policy values for every pair of lookahead and budget in a reasonable amount of time.

First we compare the performance of different lookaheads for the same budget in Figure 16. Then we compare the performance of different budgets for the same lookahead in Figure 17.

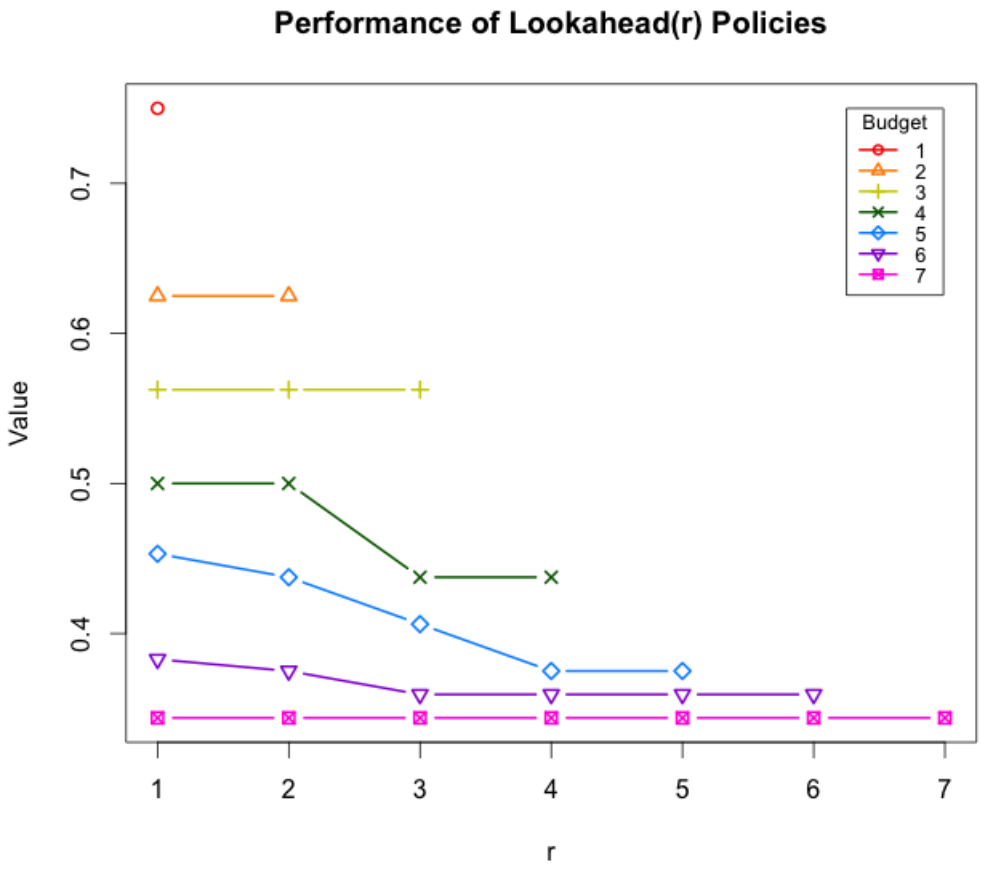


Figure 16: For budgets 4, 5, and 6, policies with higher lookaheads perform better than policies with lower lookaheads.

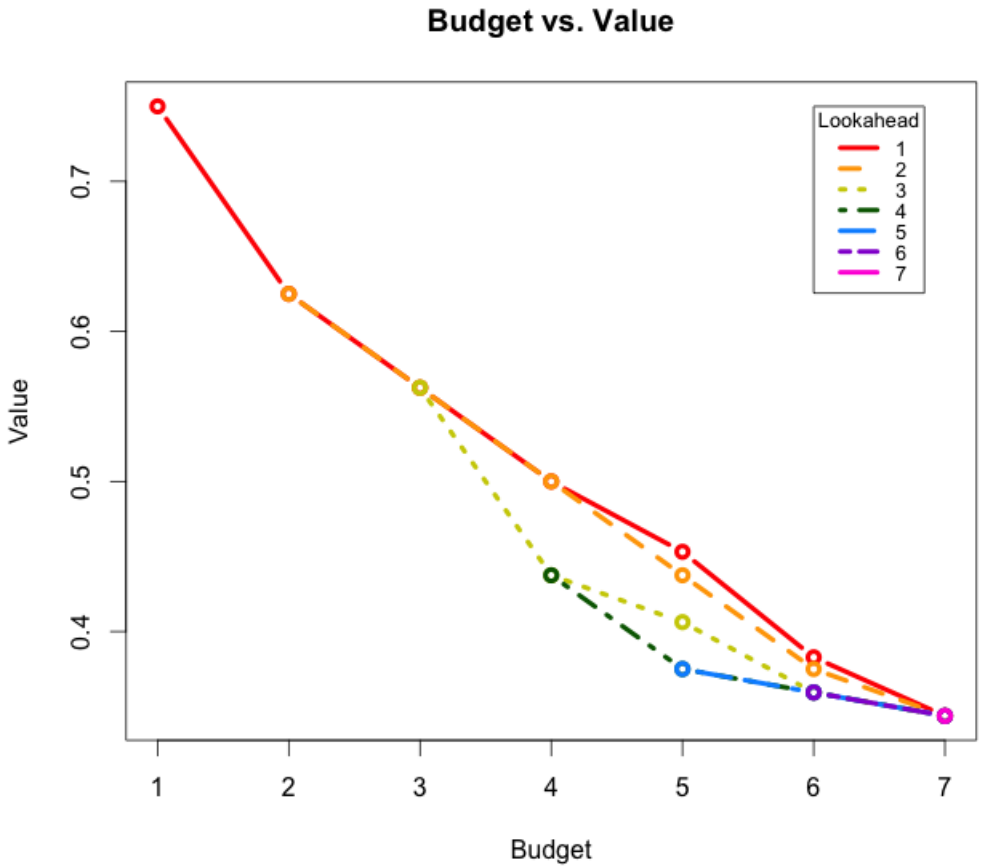


Figure 17: For budgets 4, 5, and 6, policies with higher lookaheads perform better than policies with lower lookaheads.

These results demonstrate that even in a graph as simple as the one in Figure 15, a policy will need to look up to 4 steps ahead to spend its budget optimally. On the other hand for budgets 1, 2, 3, and 7 a lookahead of 1 is enough to achieve optimal results. Numerical results can be found in Figures 18 and 19. We can compare these results to two benchmarks. If no edges are inspected, the value (of any policy) is 1. If all edges are inspected the value (of any policy) is 11/32. With a budget of 7, all lookahead policies achieve a value of 11/32. In addition even the greedy policy with a budget

of 2 achieves a value closer to 11/32 than 1.

	Lookahead						
Budget	1	2	3	4	5	6	7
1	0.75						
2	0.625	0.625					
3	0.563	0.563	0.563				
4	0.5	0.5	0.438	0.438			
5	0.453	0.438	0.406	0.375	0.375		
6	0.383	0.375	0.359	0.359	0.359	0.359	
7	0.344	0.344	0.344	0.344	0.344	0.344	0.344

Figure 18: Simulation results rounded to 3 decimal places

	Lookahead						
Budget	1	2	3	4	5	6	7
1	3/4						
2	5/8	5/8					
3	9/16	9/16	9/16				
4	1/2	1/2	7/16	7/16			
5	29/64	7/16	13/32	3/8	3/8		
6	49/128	3/8	23/64	23/64	23/64	23/64	
7	11/32	11/32	11/32	11/32	11/32	11/32	11/32

Figure 19: Exact simulation results

6 Conclusion and Future Work

In this paper we show that $V(\mathit{Lookahead}(r))$ has no finite approximation ratio to $V(\mathit{Opt}(k))$ for budget size k and $r < k$ by constructing specific graphs where this ratio can be made arbitrarily large. However $\mathit{Lookahead}(r)$ policies could perform quite well on other graphs or in expectation over a class of randomly generated graphs. Thus an interesting problem is to determine for what types of graph does $V(\mathit{Lookahead}(r))$ equal or approximately equal $V(\mathit{Opt}(k))$ for some $r < k$. For randomly generated graphs we might take

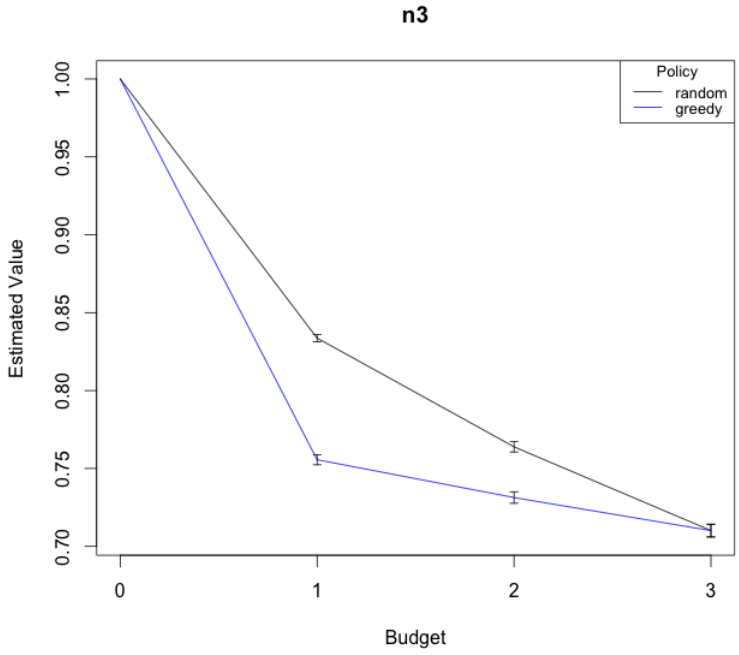
expectations over the randomness or prove that $Lookahead(r)$ performs well relative to $Opt(k)$ with high probability.

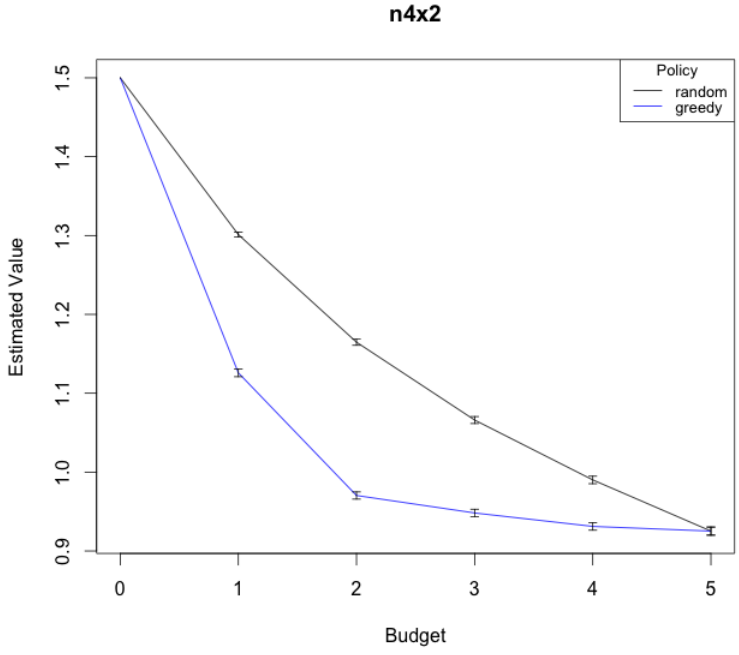
Later we simplify $Lookahead(1)$ enough to implement it effectively, and we simulate this policy on small graphs and large graphs with an underlying Erdős-Rényi structure. The results of the simulations suggest that the greedy heuristic is generally much better than randomly inspecting edges. However other simulations show that even small and seemingly simple graphs need relatively large lookaheads to perform optimally. In general we still do not understand very well how the graph structure affects the optimal policy and the effectiveness of heuristics.

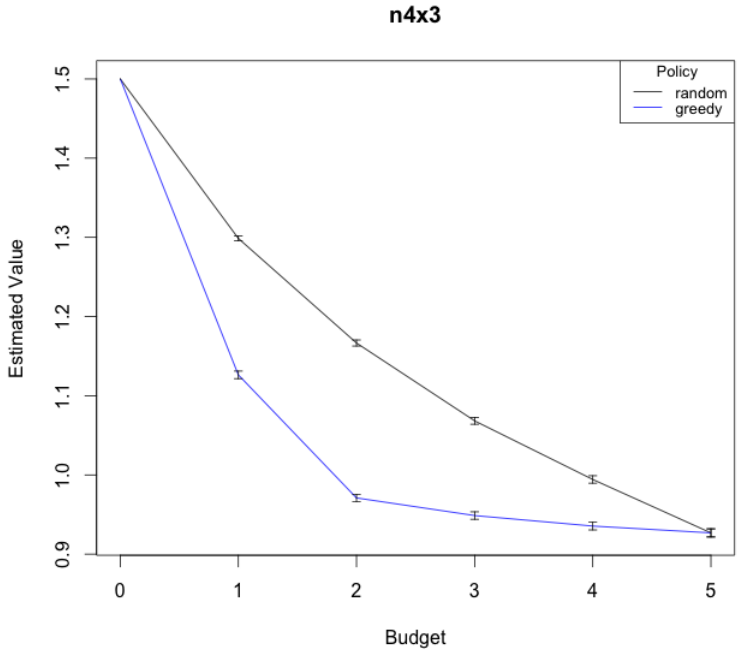
We study $Lookahead(r)$ because the naive implementation of $Opt(k)$ is computationally challenging. Even $Lookahead(r)$ can be computationally challenging for large r. Thus another potential research direction is developing policies that can be evaluated faster. Such policies might need to be restricted to certain types of graphs, and improvements in speed may require accepting slightly suboptimal solutions. Finally one strong assumption we make is that all edges weights are independent. This can be relaxed for a more general version of our problem.

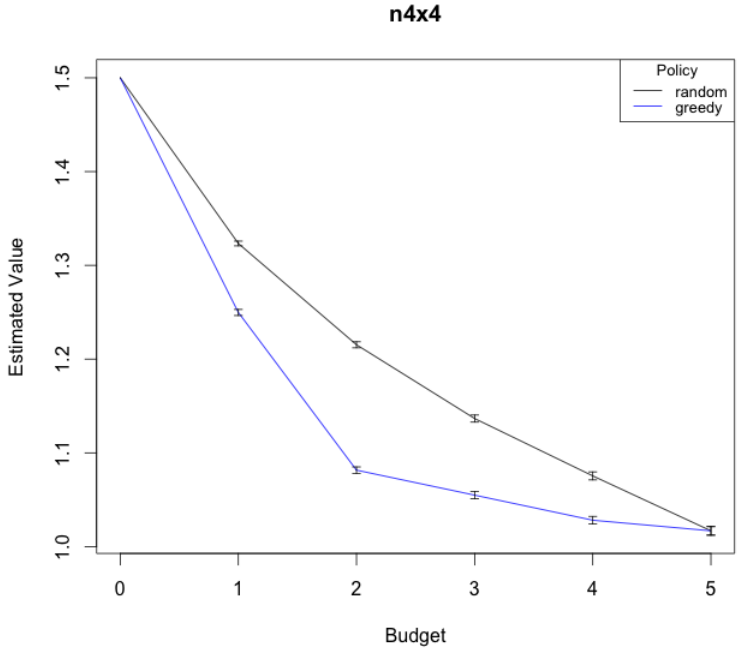
7 Appendix

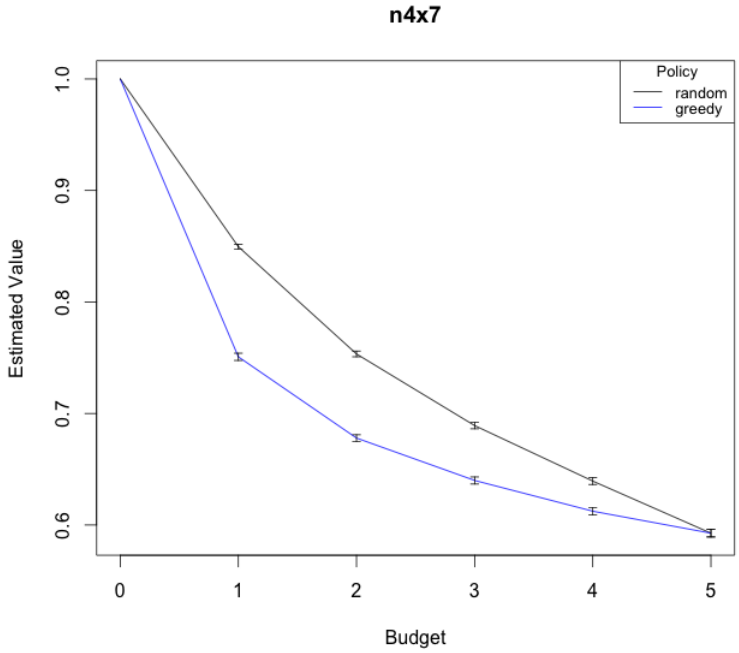
Plots begin on the next page.


n3
Policy
random
greedy
Estimated Value
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
1
2
3
Budget

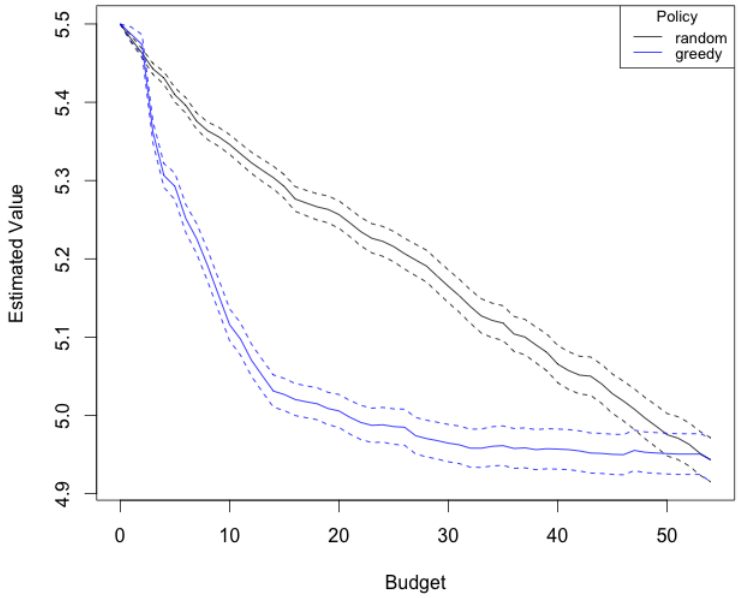

n4x2
Policy
random
greedy
Estimated Value
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0
1
2
3
4
5
Budget


n4x3
Policy
random
greedy
Estimated Value
0.9
1.0
1.1
1.2
1.3
1.4
1.5
0
1
2
3
4
5
Budget

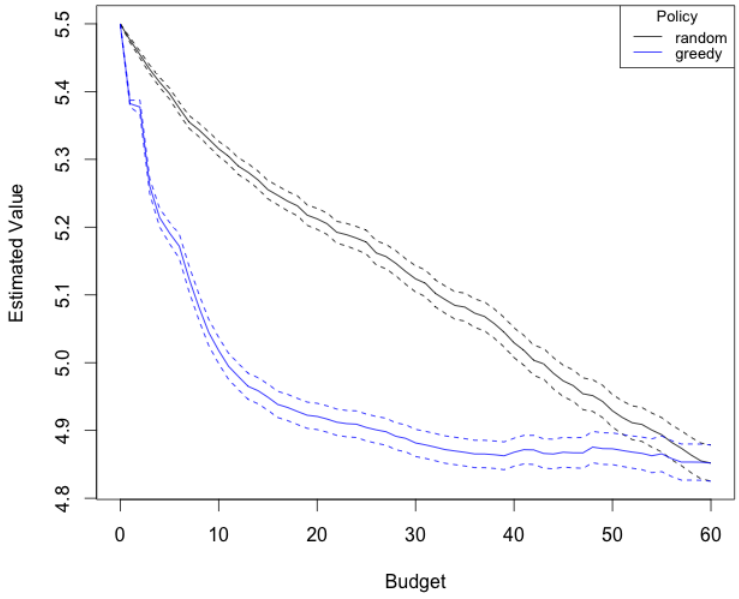

n4x4
Policy
random
greedy
Estimated Value
1.0
1.1
1.2
1.3
1.4
1.5
0
1
2
3
4
5
Budget


n4x7
Policy
random
greedy
Estimated Value
1.0
0.9
0.8
0.7
0.6
0
1
2
3
4
5
Budget

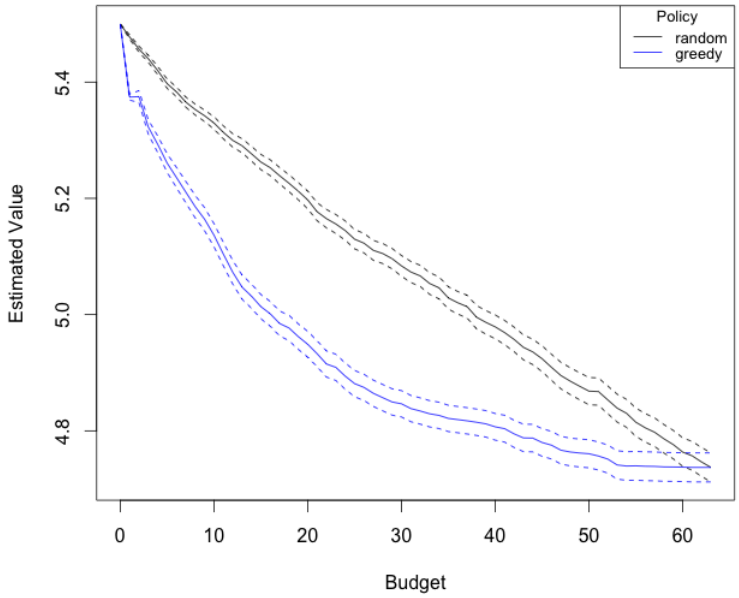
Graph 1



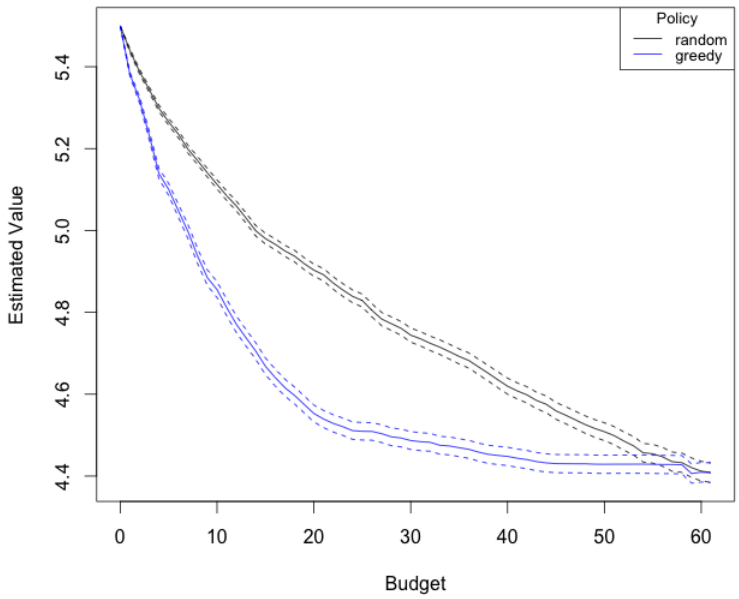
Graph 2



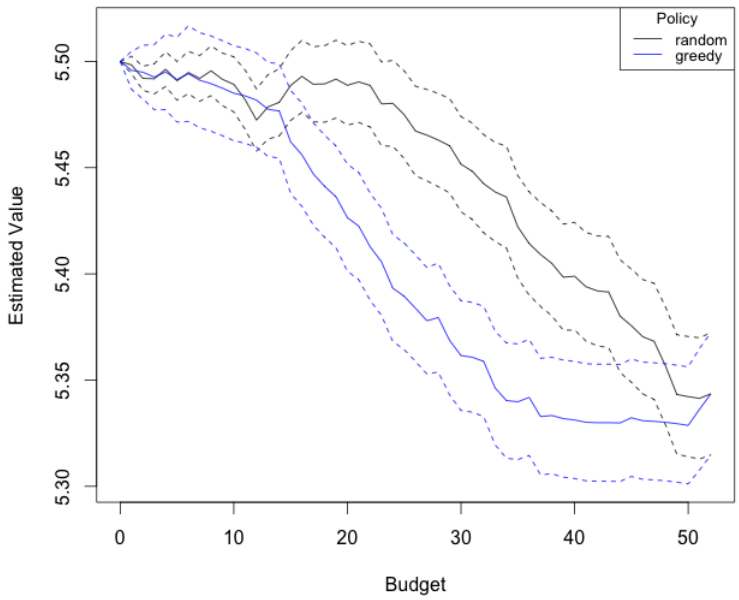
Graph 5



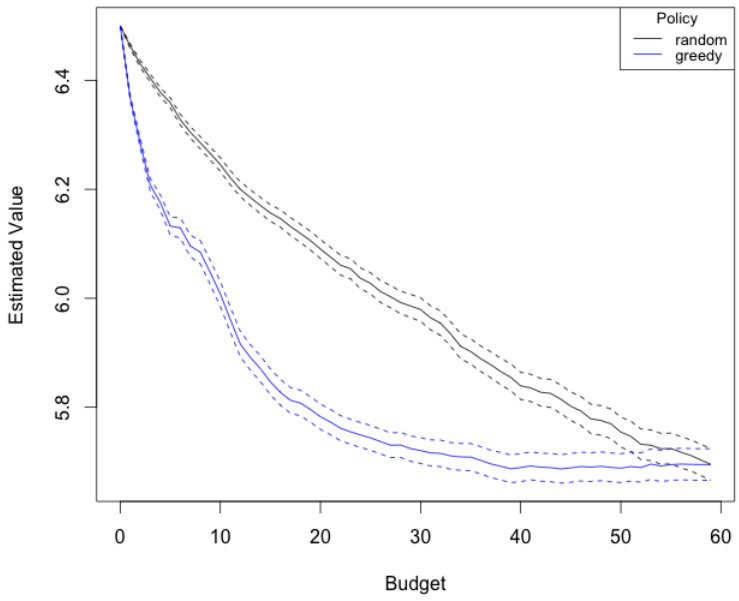
Graph 6



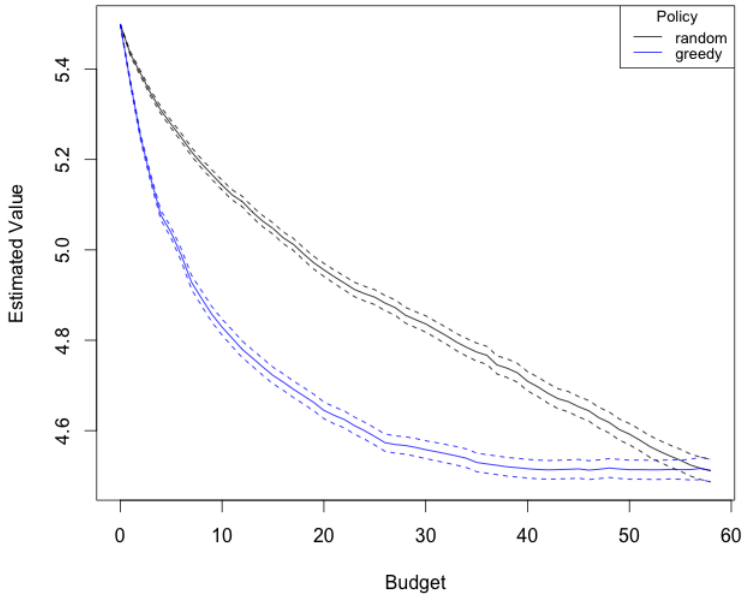
Graph 7



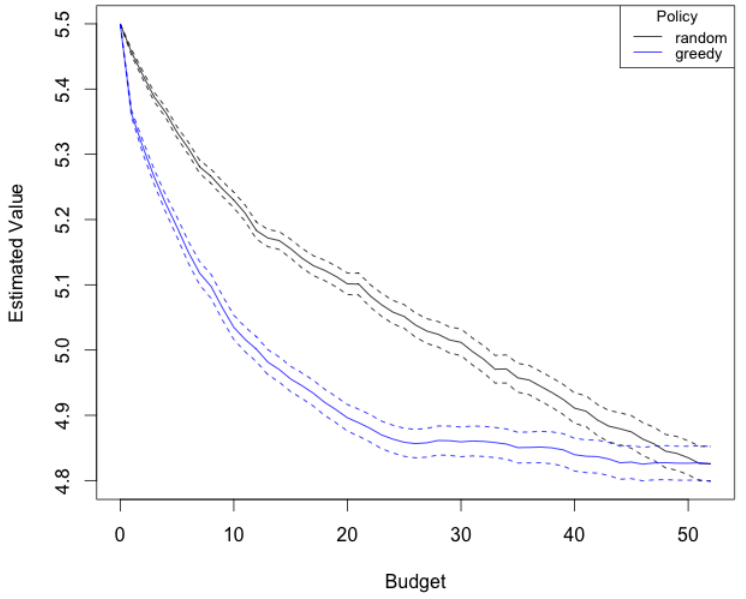
Graph 8



Graph 9



Graph 10



References

[1] Jacob Abernethy, Yiling Chen, Chien-Ju Ho, and Bo Waggoner. Low-cost learning via active data procurement. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 619–636. ACM, 2015.

[2] Paul Erdős and Alfréd Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[3] Evdokia Nikolova and David R Karger. Route planning under uncertainty: The canadian traveller problem. In *AAAI*, pages 969–974, 2008.

[4] Christos H Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[5] Ilya O Ryzhov and Warren B Powell. Information collection on a graph. *Operations Research*, 59(1):188–201, 2011.

[6] Martin L Weitzman. Optimal search for the best alternative. *Econometrica: Journal of the Econometric Society*, pages 641–654, 1979.