Learning Shortest Paths on Stochastic Graphs with Combinatorial Multi-Armed Bandits

Tomas Miskov

Final Research Project for
Reinforcement Learning - EBDS22218

February 25, 2023

Abstract

The stochastic shortest path problem is the problem of finding the shortest path between a source and a destination node in a graph where the edge weights are stochastic. In this paper we propose to solve this problem by two multi-armed bandit (MAB) methods. One combinatorial MAB policy and the other a local policy with restricted edge choices based on the graph structure. We compare the two policies to two non-bandit approaches in a scomputational simulation. We conclude the superiority of the combinatorial MAB approach to all other policies.

Keywords: stochastic shortest path, combinatorial multi-armed bandits, bandits on graphs, network bandit

Contents

1 Introduction

When two parties communicate over a network it is desirable that their messages are sent by the fastest possible path between them. If we model this network as a graph containing a set of vertices and edges with corresponding edge weights, this problem becomes a well studied shortest path problem. In the case where the network structure is fixed and the edge weights are fully determined, this is an easy problem to solve. One just applies a fast shortest path finding algorithm such as Dijkstra's [4] or Bellman-Ford [1], and the problem is solved. However, in many real-world networks, although the network structure is fixed, the edge weights vary due to e.g. traffic or disruptions in the internet connection (here we consider an online communication network). In such cases the true edge weights are unknown and the task of finding the shortest path becomes a trade-off between exploring the network's edges and exploiting the information about the already estimated weights. This setting naturally lends itself to the application of multi-armed bandits (MABs) that promise to deliver a desirable trade-off between exploration and exploitation. The trade-off is especially important in the online finite-time setting where one wants to deliver their messages as fast as possible and thus the learning of the link parameters must likewise be fast.

For the remainder of this paper we consider the following model situation: A fixed communication network is established and two nodes S (source) and D (destination) are communicating across the network. By construction, the initial edge weights of the network are known, however, there is a stochastic traffic on each one of the edges. Thus the true edge weights differ from those initially believed by the nodes S and D. This resembles a real-life situation in which the physical distances correspond to the initial edge weights but the true weights differ due to e.g. traffic. The node S wants to communicate M pieces of information with the node D and wants to do so in the most optimal way. That is, S would be the most satisfied if all M pieces of information were sent by the unknown shortest path to D. This naturally leads to quantifying the optimality of a routing policy by the notion of regret - the difference between the theoretically optimal and the actually chosen path length.

In this work we compare four routing policies for sending M messages from S to D as optimally as possible. First, we look at a rudimentary policy of computing the shortest path based on the initially believed edge weights and sending all N messages using this path. Second, we create a sampling-based policy that first samples each edge at least $k-$times and then computes the shortest path for the remaining messages. These two policies serve as a point of comparison for the remaining two MAB-based policies. For the MAB-based policies, we first implement a combinatorial bandit algorithm based on the notion of optimism. This algorithm uses the upper confidence bound (UCB) index as an optimistic estimate of the edge weights and at each iteration computes the shortest path based on these UCB indices. For the second bandit approach we implement a local version of the UCB policy that instead of computing the shortest path, selects the edge with the highest UCB index at each node, traversing the graph node-by-node, making constrained edge choices along the way. Given this general setting and the four policies, the remainder of this paper answers the following two questions:

(i) How do the non-bandit policies compare to the bandit policies in terms of finite-time regret?

(ii) For the bandit policies, is there a benefit in routing messages based on the shortest path computation instead of node-by-node edge selection?

2 Related Literature

The problem considered in this paper touches on three related problems/methods in the literature: stochastic shortest path (SSP) problem [2], combinatorial MABs [3], and MABs on graphs [5]. In fact, we combine the latter two to solve the SSP problem as introduced in the opening paragraph.

THe combinatorial MAB is an extension of the general MAB algorithm that introduces an idea of *super arms*. A super arm is just a combination of normal arms. At each iteration the agent can only play one of its super arms and observe the rewards of its associated normal arms. In our setting, the super arms are the

paths from S to D, while the normal arms are all the edges in the graph. At each iteration, the agent picks a path and receives the observed edge weights of this path as its rewards. It then updates the UCB indices of all the edges and chooses a new path to play. As introduced in [3], the next super arm/path is chosen by an oracle algorithm that takes as input the UCB indices of all the edges and outputs the combination of normal arms that create the next super arm to be played. A pseudocode for this algorithm and more details regarding our specific implementation are discussed in Section 3.

The second related MAB extension that is being applied in this work is the MAB learning on graphs. The key difference to the regular MAB algorithm is the constraints on the availability of arms at each iteration. If we model a MAB agent on a graph with edges as its arms, at each node it only has access to a specific subset of arms. This thus limits its ability to learn and fundamentally changes the achievable regret bounds. In this work we adapt the G-UCB method from [5] to our SSP formulation to create a local agent that has a restricted arm choices at each node as it traverses the graph from node S to D. Further details are again outlined in Section 3.

The remainder of this paper formally introduces the problem and the developed algorithms in Section 3, the results in Section 4, and finally concludes with a discussion and takeaways in Section 5.

3 Design and Methods

3.1 Network Model & Performance Metrics

The network is modeled as an undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. Each edge $e \in E$ has its associated initial weight w_e and a maximal traffic parameter mt_e. The traffic parameter determines the maximal additional traffic that may occur on this edge. We model the stochastic traffic as a random variable T_e that is distributed according to $T_e \sim U(0, mt_e)\ \forall\ e \in E$. Thus the true edge weight denoted $\tilde{w}_e$ is computed as $\tilde{w}_e = w_e + \mathbb{E}[T] = w_e + \frac{t_e}{2}$. We further consider a pair of source-destination nodes $(s, d) \in V^2$ and a fixed number of M messages that need to be transmitted from s to d. For every message $m \in M$ we consider a specific path P_m that consists of a set of connected edges $\{e : e \in P_m\}$ linking the source node s to the destination node d. The length of this path is computed as $l_{P_m} := \sum_{e \in P_m} w'_e$ with $w'_e = w_e + t_{em}$, where t_{em} is the realized draw from $T_e \sim U(0, mt_e)$ for the m-th message. Finally, we denote the shortest path from s to d computed using the set of initial weights $\{w_e : e \in E\}$ as SP_{init} and the optimal shortest path using the set of weights $\{\tilde{w}_e : e \in E\}$ as SP_{opt}. Their corresponding lengths are $l_{SP_{init}}$ and $l_{SP_{opt}}$. Because we are interested in finding short paths from s to d we define the reward of any edge as $r_e := -w'_e$. Thus the bigger the realized weight w'_e is, the smaller the reward r_e becomes. For any path P_m the cumulative reward $r(P_m)$ is defined as $r(P_m) := \sum_{e \in P_m} r_e$. With this definition of reward we can define the expected regret of a routing policy $\mathcal{A}$ for M messages as:

$$R(\mathcal{A}, M) := M \cdot r(SP_{opt}) - \mathbb{E}\left[\sum_{m=1}^{M} r(P_m)\right] \tag{1}$$

This follows the standard definition of regret in the reinforcement learning (RL) literature. We will use it as a metric to compare the performance of our routing policies in Section 4.

3.2 Non-bandit Routing Policies

For the first non-bandit routing policy we create a simple static agent that sends all M messages using the SP_{init} path. To compute this and any subsequent shortest paths we use the Dijkstra's algorithm [4]. This policy represents an agent that is unaware of the stochasticity in the edge weights and serves as a baseline for comparison. It's expected regret can straightforwardly be computed as $M \cdot r(SP_{opt}) - M \cdot r(SP_{init})$. We denote this routing policy as $\mathcal{A}_{static}$.

The second non-bandit policy considers an agent that is aware of the stochasticity in edge weights and

decides to use the first M_1 messages to collect at least k samples of each edge. Denoting this agent as $\mathcal{A}_{sample}$, we construct n paths from s to d such that if we send n messages along these paths we collect at least 1 sample of each edge. To collect at least k samples of each edge we must send $M_1 = k \cdot n$ messages. The remaining $M_2 = M - M_1$ messages are subsequently sent by the shortest path computed using the k-sample average edge weights. As with $\mathcal{A}_{static}$, the $\mathcal{A}_{sample}$ policy is used as a point-of-comparison policy that considers the stochasticity in edge weights but deals with it in a rudimentary fashion. We would like to note here that the parameter k is a hyper parameter to be chosen by the user. As evident, the larger the k, the better the edge weight estimates become. However, the size of the parameter is restricted by the total number of messages M and the number of unique paths n that are used for sampling. In cases where M is small and n is large, one may not even be able to collect one sample of each edge.

3.3 Bandit Routing Policies

The first of our bandit-based routing policies is a policy that follows the ideas on combinatorial bandits presented in [3] and outlined in Section 2. The $\mathcal{A}_{Dijkstra}$ policy is a combinatorial bandit policy with arms corresponding to the graph's edges E and super arms corresponding to admissible paths P from source s to destination d. For every normal edge (arm) this policy computes a UCB index u_e that's defined as

$$u_e := \bar{r}_e + \sqrt{\frac{c \cdot \ln m}{M_e}}, \tag{2}$$

where $\bar{r}_e$ is the mean observed reward of edge e, m is the number of messages sent so far, and M_e is the counter of how many of the m messages have been sent through edge e. We initialize $\bar{r}_e$ as $-w_e$, thus before any messages are sent $\bar{r}_e = -w_e \ \forall \ e \in E$. When deciding on the path for the next message we employ an oracle algorithm that takes as the input the set of all UCB indices $\{u_e : e \in E\}$ and outputs the next admissible path (super arm) to be played. As is the case with the non-bandit policies the oracle algorithm we use here is the Dijkstra's shortest path algorithm [4]. When the next message is sent using the path computed by the oracle algorithm we observe all the rewards of its associated edges and update all UCB indices accordingly. The entire procedure is summarized in Algorithm 1.

Algorithm 1 Combinatorial MAB Algorithm with Dijkstra's Shortest Path Oracle

Given a graph $G = (V, E)$, a source-destination pair of nodes (s, d), and M messages:
$M_e \leftarrow 0 \ \forall \ e \in E$
$u_e \leftarrow -w_e \ \forall \ e \in E$
$m \leftarrow 0$
while $m \leq M$ **do**
  $m \leftarrow m + 1$
  For each edge e, update $u_e = \bar{r}_e + \sqrt{\frac{c \cdot \ln m}{M_e}}$
  $P = Dijkstra(s, d, \{u_e : e \in E\})$
  Send message m using path P
  Collect $r_e \ \forall \ e \in P$ and update all $\bar{r_e}$ and M_e
end while

The second bandit policy is a constrained version of a general MAB algorithm. Instead of employing an oracle to compute the next best routing path, $\mathcal{A}_{local}$ makes its routing decisions node-by-node. Just like $\mathcal{A}_{Dijkstra}$ it computes a UCB index for each of its edges (arms) but only chooses its next arm to play at every node while traversing the graph from source to destination. To ensure that the messages do reach their destination node d, we constrain the available arms (edges) at each node such that the agent can only travel in the general direction towards d and thus avoid loops and infinite paths. The pseudocode of the $\mathcal{A}_{local}$ algorithm is summarized in Algorithm 2

Algorithm 2 Local MAB Algorithm with Constrained Arm (Edge) Choices

Given a graph $G = (V, E)$, a source-destination pair of nodes (s, d), and M messages:
$M_e \leftarrow 0 \; \forall \; e \in E$
$u_e \leftarrow -w_e \; \forall \; e \in E$
$pos = s$ ▷ Current position of the message in the network
$m \leftarrow 0$
while $m \leq M$ **do**
  For each edge e, update $u_e = \bar{r}_e + \sqrt{\frac{c \cdot \ln m}{M_e}}$
  Choose the next e as $\hat{e} = \arg\max\{u_e\}$ from all unconstrained available edges at the current pos
  Play the chosen edge $\hat{e} = e_{pos,j}$ and update $pos = j$
  Collect $r_{\hat{e}}$ and update $\bar{r}_{\hat{e}}$ and $M_{\hat{e}}$
  if $pos = d$ **then**
    $pos = s$
    $m \leftarrow m + 1$
  end if
end while

3.4 Experimental Setup

To test our two non-bandit and two bandit routing policies we devise the following experimental setup. Figure 1 depicts the 8×8 grid that serves as our graph for all experiments. The yellow lower-left corner node is the source node s while the red upper-right corner node is the destination node d. All the initial edge weights are randomly initiated from a uniform distribution on the interval $U(1, 10)$. Likewise, all the max traffic parameters $mt_e \; \forall \; e \in E$ are drawn from $U(5, 20)$. These ranges are chosen such that there is a small chance of $SP_{init} = SP_{opt}$ and thus the learning procedure plays an important part in minimizing regret of each routing policy. Because our example graph is quite small, we choose to send only 100 messages ($M = 100$) from s to d to empirically analyze the effectiveness of the proposed policies in finite-time experiments instead of asymptotically.

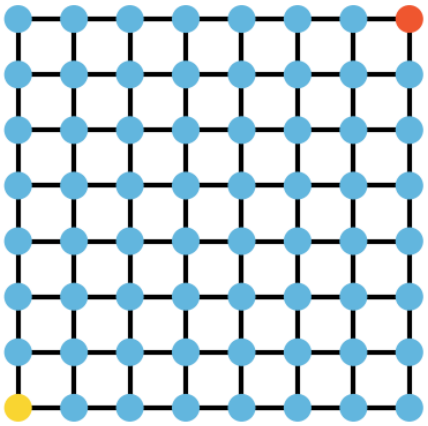

Figure 1: 8×8 grid graph used in all experiments. The yellow node represents the source s, red node represents the destination d.

To fairly compare the four routing policies we initialize 25 graphs as depicted in Figure 1 and send $M = 100$ messages from s to d using each one of the four policies. We then compare their expected regrets and runtimes to not only consider the performance in terms of rewards but also the feasibility in terms of running the algorithms in an online setting. For the two bandit policies based on the UCB indeces we choose

the c parameter according to the results shown in [5]. That is, we set $c = 2$ and run the main experiments with this setting. However, we also carry out an ablation experiment where we compare the performance of $c = 0.2$ and $c = 20$ in addition to $c = 2$. Similarly, for the non-bandit policy $\mathcal{A}_{sample}$ we run the main experiment with at least $k = 5$ samples for every edge weight but also provide a comparison with $k = 3$ and $k = 1$. To sample each edge of our graph at least once we use 16 distinct paths, thus, the sampling policies use up 80, 48, and 16 messages just for sampling respectively. Finally, to guide the $\mathcal{A}_{local}$ policy from s to d we restrict the available edges (arms) at each node to only those going up or to the right. This provides fair comparison to the other policies since it prevents loops and backtracking.

4 Results

Running the four policies according to the experimental setup outlined in Section 3.4, Figure 2 depicts the results in terms of expected regret over 100 messages for each of the four policies. From the two non-bandit policies it is clearly more advantageous to use $\mathcal{A}_{static}$ for the finite-time scenario since $\mathcal{A}_{sample}$ with $k = 5$ uses a bulk of available messages just for estimating the true edge weights. Comparing the two bandit-based policies, $\mathcal{A}_{Dijkstra}$ clearly outperforms $\mathcal{A}_{local}$ and it is also shown to be the most optimal policy out of all considered policies. We can see that it achieves nearly non-increasing expected regret already at the 40th message out of the 100. The $\mathcal{A}_{sampling}$ also achieves this nearly non-increasing regret, however, it achieves it much later at the 80th message when the sampling period is over and nearly optimal shortest path from s to d can be found using the estimated edge weights. Finally, we can see that the $\mathcal{A}_{static}$ policy outperforms the $\mathcal{A}_{local}$ policy by quite a margin showing that the constrained bandit approach is inferior even to a policy that does not account for stochasticity in edge weights but does exploit the initial graph structure through the shortest path computation. Moreover, as seen in Table 1, the $\mathcal{A}_{local}$ policy has an order of magnitude higher runtime than $\mathcal{A}_{Dijkstra}$ and two orders of magnitude higher than $\mathcal{A}_{Static}$ and $\mathcal{A}_{Sample}$. Therefore, incorporating this information into account, $\mathcal{A}_{Dijkstra}$ and $\mathcal{A}_{Static}$ are superior to the other two policies.

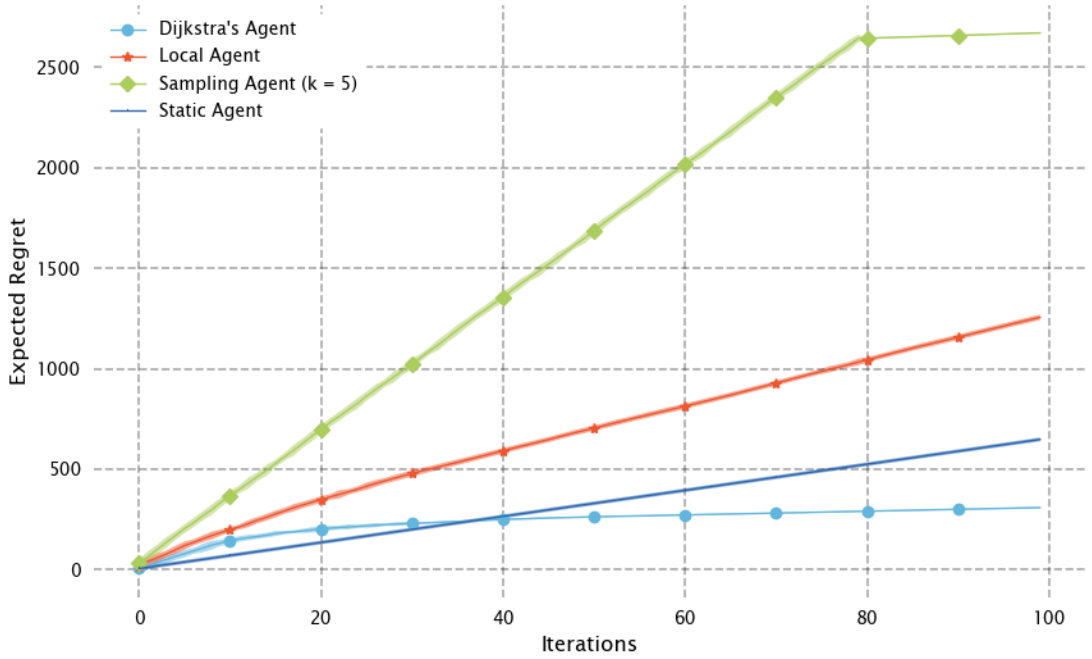

Figure 2: Main experimental results depicting expected average regret based on 25 simulations for each of the four policies. The lighter shaded regions represent the 95% confidence bounds.

Moving onto the two ablation studies mentioned in Section 3.4, Figure 3 depicts the differences in expected

	$\mathcal{A}_{static}$	$\mathcal{A}_{sample}$	$\mathcal{A}_{Dijkstra}$	$\mathcal{A}_{local}$
Average Runtime	0.0266	0.0059	0.260	2.1778

Table 1: Average runtimes of the four policies

regret between three $\mathcal{A}_{Dijkstra}$ policies with $c = 0.2$, $c = 2$, and $c = 20$ in red, blue, and green respectively. Here we can see that while the policy with $c = 20$ is the least regretful initially, it is also least exploratory and after the initial few messages it sticks with a single non-optimal path that results in a linearly increasing regret. On the other hand, the policy with $c = 0.2$ explores too liberally and at the 100th message mark still achieves worse regret than the policy with $c = 20$. However, given the decreasing trend in the expected regret, it is clear that the regret of policy with $c = 0.2$ eventually levels off while the policy with $c = 20$ will just keep increasing. Nevertheless, we can see that the policy with $c = 2$ achieves the best result, exploring enough to find a path close to SP_{opt} yet not too much to do so in finite time before running out of the 100 available messages.

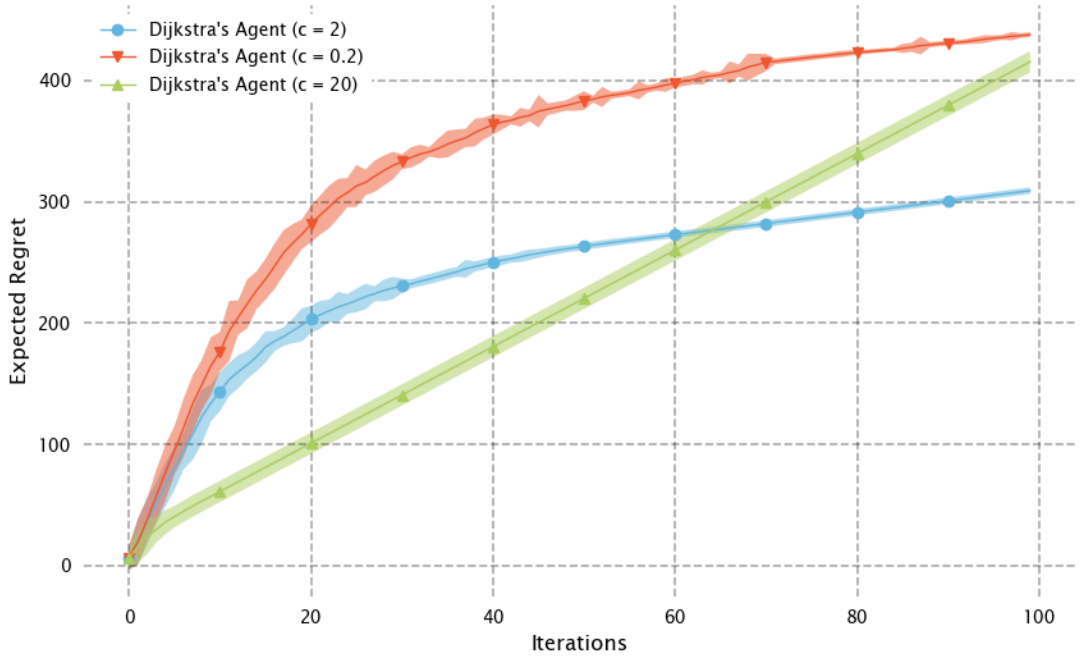

Figure 3: Difference in the expected regret of the $\mathcal{A}_{Dijkstra}$ policy for three different values of c. The lighter shaded regions represent the 95% confidence bounds.

As with the tuning parameter in the $\mathcal{A}_{Dijkstra}$ policy, Figure 4 shows the results of the ablation experiment of the k parameter in the $\mathcal{A}_{Sample}$ policy. The results here are very much as expected, namely that before the sampling terminates, each of the three policies exhibits linearly growing regret as they use the same 16 paths to uniformly sample the edge weights for at least k samples of each edge. After termination of the sampling period, the slope of the lines reflects the optimality of the found shortest path from s to d for sending the remainder of the available messages. The closer the slope is to 0, the closer the found shortest path is to SP_{opt}. As expected, the slope of the lines after the sampling period decreases as the number of samples increase. This reflects the trade-off between accurately estimating the edge weights and spending valuable resources to do so.

To get a better idea of how the four algorithms route their messages, Figures 5 and 6 depict all 100 paths deployed by the four algorithms in one of the experimental runs. The width of the edges is proportional to the estimated edge weights, while their color represents the frequency of message routing. The darker the edge,

the more frequently it has been used by the algorithm. In this particular run all algorithms chose distinct routes as their most frequently used paths. The interesting observation is the difference in which $\mathcal{A}_{Dijkstra}$ and $\mathcal{A}_{local}$ explored the potential routes. Although there is clearly some overlap and the most frequently chosen paths share 9 out of 14 edges, $\mathcal{A}_{Dijkstra}$ explored more frequently the center of the graph while $\mathcal{A}_{local}$ explored mostly the upper left area. Finally, Figure 7 shows the difference in estimated edge weights of the four algorithms as compared to the ground truth. The edges are sorted according to the ground truth weights $\tilde{w}_e$ and the darker the color, the higher the true edge weight. One can compare the estimated edge weight of a particular edge by vertically comparing the intensity of the color in every segment of the 5 lines. It is clear that $\mathcal{A}_{Sample}$ gets the best estimates which is an expected result. However, just like with the previous two plots the insightful takeaway is that the two bandit policies are better at accurately estimating only the light edges while leaving the heavy edges undersampled, thus estimated with low accuracy.

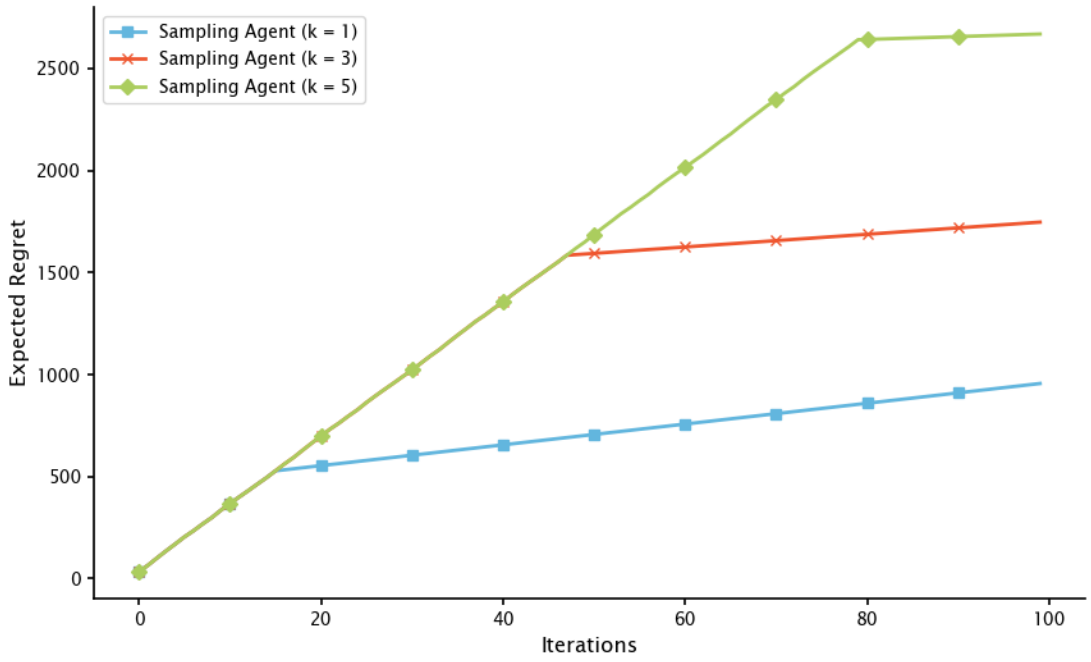

Figure 4: Difference in the expected regret of the $\mathcal{A}_{Sample}$ policy for three different values of k. The lighter shaded regions represent the 95% confidence bounds.

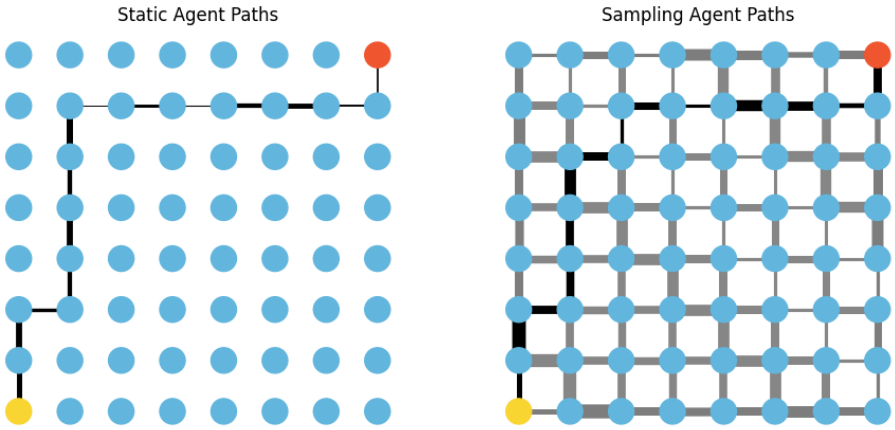

Figure 5: Paths taken by the two non-bandit policies in one of the experimental simulations. The thickness of each edge is proportional to its average sampled weight, while the color reflects the frequency of use. The darker the edge, the more it has been used to send messages from s to d.

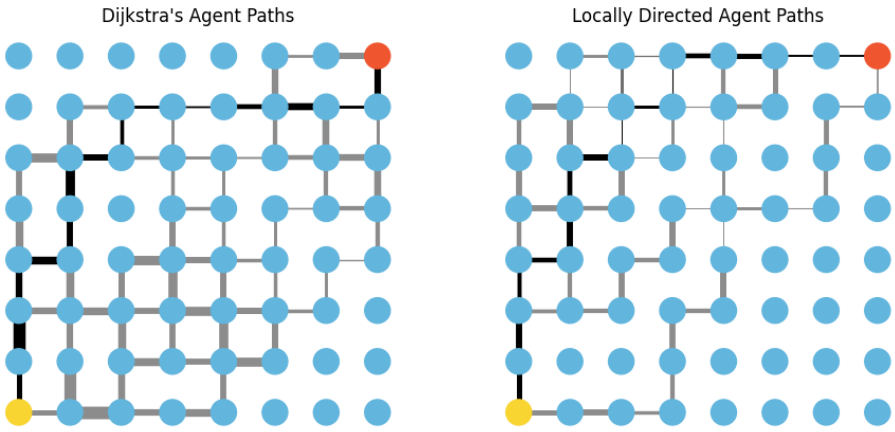

Figure 6: Paths taken by the two bandit policies in one of the experimental simulations. The thickness of each edge is proportional to its average sampled weight, while the color reflects the frequency of use. The darker the edge, the more it has been used to send messages from s to d.

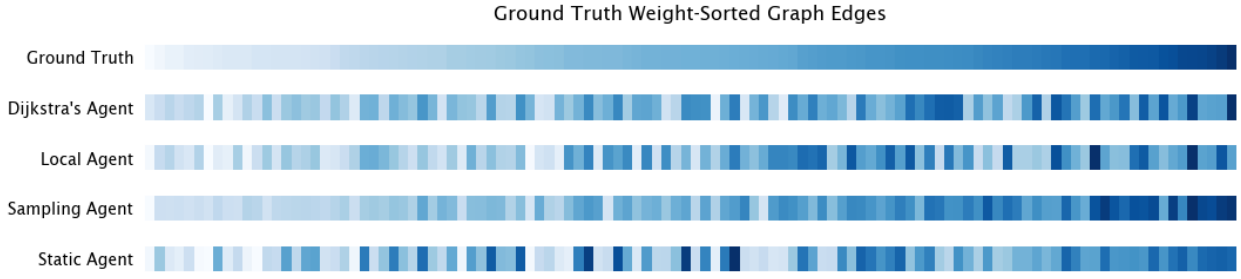

Figure 7: Comparison of the estimated edge weights by the four policies. The edges are sorted according to the ground truth weights $\tilde{w}_e$. The darker the tone of blue, the higher the estimated edge weight.

5 Conclusion

This paper set out two answer two key questions: (i) do bandit-based methods outperform the non-bandit ones in the stochastic shortest path problem and (ii) what's the value of combining them with Dijkstra's shortest path algorithm? While the answer to Question (ii) is clear, the answer to Question (i) has several caveats. As the results suggest, UCB-based combinatorial bandit with the Dijkstra's shortest path algorithm as an oracle outperforms the local bandit policy in both expected regret and computational time. Thus, there is a clear benefit to using the oracle approach of $\mathcal{A}_{Dijkstra}$ for shortest path finding in graphs with stochastic edges.

For Question (i) however, the results are mixed as seen in Figure 2 where the leading algorithm is bandit-based, but it is followed by the static shortest path policy. For this reason we cannot clearly say that the bandit-based policies are superior to the non-bandit ones. That said, the combinatorial MAB policy is clearly superior to all other policies, thus at least for this policy alone we can state its superiority to other approaches. Furthermore, our results are dependent on our experimental choices. It is possible that $\mathcal{A}_{local}$ would outperform the $\mathcal{A}_{static}$ if the stochasticity in edge weights was more significant and the true edge weights would differ greatly from the initial believes. Likewise, we have tested our algorithms on a graph where the direction from source to destination was clearly defined, thus constraining the arm choices for the $\mathcal{A}_{local}$ algorithm was possible. However, there may be graphs where the general direction is much harder to define which would harm the potential performance of the $\mathcal{A}_{local}$ algorithm.

References

[1] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[2] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[3] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In *International conference on machine learning*, pages 151–159. PMLR, 2013.

[4] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[5] Tianpeng Zhang, Kasper Johansson, and Na Li. Multi-armed bandit learning on a graph. *arXiv preprint arXiv:2209.09419*, 2022.

6 Appendix

6.1 Code

The entire code with all the simulations including the random seeds can be found on my GitHub.