EPiC Series in Computing
Volume 55, 2018, Pages 121–133
GCAI-2018. 4th Global Conference on Artificial Intelligence

Multi-Armed Bandit Algorithms for a Mobile Service Robot's Spare Time in a Structured Environment

Max Korein[1] and Manuela Veloso[2]

[1] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
mkorein@cs.cmu.edu
[2] Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA
mmv@cs.cmu.edu

Abstract

We assume that service robots will have spare time in between scheduled user requests, which they could use to perform additional unrequested services in order to learn a model of users' preferences and receive rewards. However, a mobile service robot is constrained by the need to travel through the environment to reach users in order to perform services for them, as well as the need to carry out scheduled user requests. We assume service robots operate in structured environments comprised of hallways and floors, resulting in scenarios where an office can be conveniently added to the robot's plan at a low cost, which affects the robot's ability to plan and learn.

We present two algorithms, Planning Thompson Sampling and Planning UCB1, which are based on existing algorithms used in multi-armed bandit problems, but are modified to plan ahead considering the time and location constraints of the problem. We compare them to existing versions of Thompson Sampling and UCB1 in two environments representative of the types of structures a robot will encounter in an office building. We find that our planning algorithms outperform the original naive versions in terms of both reward received and the effectiveness of the model learned in a simulation. The difference in performance is partially due to the fact that the original algorithms frequently miss opportunities to perform services at a low cost for convenient offices along their path, while our planning algorithms do not.

1 Introduction

Consider a service robot in an office building that performs services for users. Users can request services, but the robot can also approach a user's office to offer a service that has not been requested. For example, if the robot delivers food, then not only could the robot deliver food to specific locations at specific times at a user's request, it could also carry food to offer users in their offices during its spare time in between requests.

It would be desirable for such a robot to perform as many successful services as possible, but some users in a given building might be more interested in receiving the robot's services

This research is partially sponsored by DARPA under agreements FA87501620042 and FA87501720152. The views and conclusions contained in this document are those of the authors only.

than others. Some users might frequently be interested in the robot's services, some might use them only occasionally, and others might have no interest in interacting with the robot at all. In order to maximize the service that the robot provides, it would have to learn a model of how likely different users are to be interested in its services when they have not made any requests. Then, when it has spare time between scheduled service requests, it would choose which additional services to offer. It would need to strike a balance between exploitation – maximizing the expected number of services according to its current model – and exploration – seeking to improve its model.

In order to offer a service to a user, the robot must spend time traveling to reach that user's office. When choosing which users to offer services to, the robot must consider the time spent traveling from office to office, as well as the time it will take to travel to the location of its next scheduled user request. Additionally, office buildings are typically very structured. They are often divided into floors, and each floor consists of a network of hallways with offices along them. As a result, in the process of traveling to an office, the robot will also pass by other offices along the way. To perform most effectively, the robot will need to plan in a way that takes advantage of the opportunities to perform additional convenient services due to the structure of the environment.

While this research is motivated by service robots operating in an office building, it can apply to any robot that chooses from actions at different locations with unknown rewards and time and space constraints in a structured environment. For example, it could apply to an autonomous taxi searching for additional fares when it has extra time before its next scheduled pickup, or a security robot seeking places where it is most likely to observe an important event while ensuring it returns to charge its battery before it runs out.

In this paper, we seek to adapt existing approaches to balancing exploitation and exploration to accommodate the constraints of a robot performing actions for unknown reward in its spare time in a structured environment. To accomplish our goal, we contribute modified versions of two existing algorithms, UCB1 and Thompson sampling, and compare them to versions with minimal in simulation changes. Additionally, we study how the structure of the environment affects the performance of these algorithms.

2 Related Work

The problem of a service robot performing services in its spare time can be seen as a variant of the multi-armed bandit problem (MAB). In the MAB, a robot can choose from a variety of "levers" to pull, each of which yields a random reward according to a function and has a fixed cost. The robot begins with no knowledge of the reward functions, and its goal is to maximize the reward it receives over time.

A variety of approaches exist to the MAB. Random exploration algorithms choose between randomly "exploring" and attempting to maximize reward received ("exploiting") with each action[11]. Optimistic algorithms, such as Upper Confidence Bounds (UCB) algorithms, intentionally overestimate the reward of highly uncertain levers in order to intrinsically promote exploration[1]. Probability matching algorithms attempt to choose a lever such that the probability of a lever being chosen is equal to the probability of that lever being optimal, such as Thompson sampling, in which the agent samples a possible expected reward from each lever and chooses the one with the highest sample[9, 3].

Multi-Armed Bandits with Metric Switching Costs (MAB-MSC) are a variant of the multi-armed bandit problem in which there is a cost to moving between levers[4]. This is similar to the case of a service robot, which has a cost to move from user to user. However, solutions to the MAB-MSC seek to minimize how often they switch levers. A service robot cannot wait outside a single user's office repeatedly offering services, so minimizing how often it switches offices is not an option.

Given perfect knowledge of the expected reward of each office, the spare time service problem becomes an instance of the Orienteering Problem (OP). In the OP, an agent is in an environment represented by a graph with costs on the edges and rewards on the nodes, and must find a path from a given start node to a given end node that maximizes the reward received while constrained by a maximum allowed cost. A variety of algorithms exist to solve the orienteering problem, including optimal solutions, approximation algorithms, and real-time solutions[10, 5, 2]. However, these approaches all assume an accurate model of the rewards, and many also assume an open, connected environment. A service robot must learn the expected rewards of its services, and frequently operates in a structured environment.

Previous works have also explored the concept of a service robot learning in its spare time. With the Dora the Explorer robot, Hanheide et al. presented a framework for generating and managing goals that the robot would use to gather data on where it could find objects for user requests[6]. Meanwhile, Korein et al. presented an algorithm for the CoBot service robots to use their spare time to observe whether office doors were open or closed in order to improve their ability to schedule services[7, 8]. The planning algorithm for CoBot assumed a structured environment and grouped offices into hallways, treating entire hallways as a single unit when gathering data to improve planning efficiency. With both the CoBot and Dora the Explorer robots, however, the robot only gathered information in its spare time - in other words, it only explored and did not need to exploit.

3 Problem Description

We now formalize the problem we are solving. First, we will describe the problem the robot must solve, then we will describe the properties of the environment.

3.1 Problem Formalization

The robot operates in an environment represented by an undirected graph. Nodes on the graph represent offices where the robot can go to offer services to users, and edge lengths represent the time to travel between adjacent offices. We assume that there is exactly one user per office, and thus the terms "user" and "office" are interchangeable. The robot has a complete and accurate map, and is thus capable of finding the shortest path between any two locations and accurately predicting how long it will take to travel along that path.

We will divide the robot's time into the segments between scheduled user requests. During a segment, the robot begins at location u_{start} and time t_{start}, the location and time at which it completed its last scheduled request. We will assume $t_{start} = 0$ for the sake of simplicity. The robot must arrive at the location of its next scheduled request, u_{end}, by time t_{end}.

In the meantime, the robot can offer services to users by traveling to a user's office and interacting with the user. Each user u has some probability p_u of accepting services offered by the robot. The robot begins with no knowledge of p_u for any user. If the user accepts the service, the robot receives a reward of 1, otherwise it receives a reward of 0. This process takes time $t_{service}$, regardless of whether the service is accepted or rejected. The robot may only offer a service to each user once per segment. Once it has offered a service to a user, whether it succeeded or failed, it may not offer a service to the same user again until after it has completed its next scheduled request.

The goal is to maximize the reward the robot receives over time. To do this, the robot must both learn a model of the likelihood of a service being successful for any given user, and manage its time during each segment as it travels from user to user offering services while ensuring that it reaches u_{end} in time.

3.2 Environment Structure

One of the key features that distinguishes the spare time service problem from a traditional multi-armed bandit problem is the need to travel through the environment to perform a service. The time spent traveling from office to office is an additional cost beyond the cost of actually offering the service. Most office buildings are extremely structured, which creates patterns in the cost of moving between offices and affects the nature of the problem. In particular, office buildings are typically comprised of hallways, which cause strong correlations between the cost of visiting different offices within the same plan.

3.2.1 Opportunities and Hallways

Whenever the robot's plan includes traveling past an office u_i, it can offer a service to the user at the office for a cost of only $t_{service}$, the time it takes to offer a service, with no additional costs related to traveling. We call this an **opportunity** for u_i. We will say an office u_i is **convenient** to a plan p if p contains an opportunity for u_i.

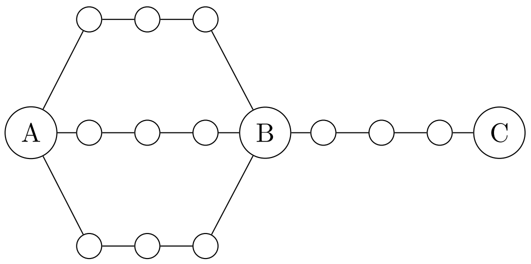
We will define a hallway as a set of nodes $\{u_0, u_1, \ldots, u_{n-1}, u_n\}$ where each node $u_i \in \{u_1, \ldots, u_{n-1}\}$ has exactly two edges, one connecting it to u_{i-1} and one connecting it to u_{i+1}. An important property of hallways is that any plan that travels through a hallway contains opportunities for every office on the hallway. If the environment is comprised primarily of hallways, as most office buildings are, then any path between two locations in the environment will traverse a number of hallways. As a result, there are a very large number of offices convenient to any plan.

Additionally, opportunities in an environment comprised of hallways are often dependent and non-uniform. They are dependent because, if one office in a hallway is convenient to the robot's plan, other offices in the hallway typically will be too. They are non-uniform because the robot will typically travel down some hallways more than others. For example, we would expect it to travel down a hallway that serves as a bottleneck connecting different sections of the environment more than it travels to the end of a dead end hallway.

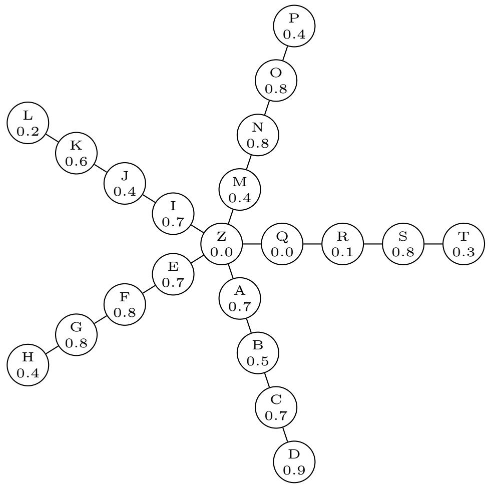
3.2.2 Example: Hallway Sequence Environment

An example of a simple environment with the structural properties described is shown in Figure 1a. This structure, which we call the "Hallway Sequence," contains a sequence of intersections which are connected by varying numbers of hallways. It represents situations in which the robot has multiple partially-overlapping paths between two locations. Such a situation would occur when traveling from one end of a floor to the other in many office buildings.

Service opportunities in this environment are dependent because the opportunities for offices on each hallway are dependent on each other. They are non uniform because the robot will receive more opportunities for offices in "bottleneck" hallways with fewer or no alternatives (such as the B-C hallway in the example).



(a) An example of a Hallway Sequence environment with three intersections, labeled A, B, and C, three hallways between A and B, one hallway between B and C, and three non-intersection offices per hallway.



(b) An example of a Star environment with five hallways, each containing four rooms (not including the center). Rewards for each node are included for a later example.

Figure 1: Examples of environment structures with non-uniform, dependent opportunities.

3.2.3 Example: Star Environment

A second example environment structure, which we call the "Star," is shown in Figure 1b. In this structure, there is a single center room, labeled "Z" in the image, with a number of hallways branching out from it. This structure is intended to represent an environment divided into different floors: the center room represents an "elevator" that the robot must use to travel between the different hallways.

Like in the hallway sequence, this environment has non-uniform dependent service opportunities. In order to reach the offices at the far end of a hall, the robot must pass the other offices in the hall. Thus, the number of opportunities the robot gets to perform services for an office is never greater than the number of opportunities for offices in the same hallway closer to the center.

4 Algorithms

In this section, we contribute two algorithms, "Planning Thompson Sampling" and "Planning UCB1." We first describe the original Thompson Sampling and UCB1 algorithms used in standard multi-armed bandit problems, and how they can be used by a robot performing services in its spare time. We then describe our own planning versions of those algorithms, which plan ahead in order to better take into account the constraints on the robot.

4.1 Naive Thompson Sampling and Naive UCB1

Thompson Sampling and UCB1 are multi-armed bandit algorithms that both repeatedly assign a reward to each possible action, then perform the action with the highest assigned reward.

Applying this directly to a robot performing services in its spare time, we get the following algorithm:

1. Assign a reward to each location in the environment.
2. Choose the valid office with the highest assigned reward (an office is valid if the robot has time to travel there, offer a service, and still reach the end location within the time available).
3. Travel to the chosen location and offer a service.
4. Repeat steps 1-3 until there are no valid offices in the time remaining.
5. Travel to the end location.

In this algorithm, the robot only takes into account the constraints on time and location when considering whether or not an action is allowed, without considering how the constraints affects its future actions. Because of this, we refer to this as the "Naive algorithm," and we call the versions of Thompson Sampling and UCB1 that follow it **Naive Thompson Sampling** and **Naive UCB1**.

Where Thompson Sampling and UCB1 differ is their approach to assigning reward in Step 1 of the naive algorithm.

4.1.1 Thompson Sampling

Thompson sampling is a probability-matching algorithm, in which the robot randomly chooses an action such that the probability of an action being chosen is proportional to the probability that the action is optimal given the samples the robot has gathered[9, 3]. This is accomplished by sampling from the distribution of possible expected rewards based on the samples the robot has gathered from an office.

For a given office u_i, the function $B_i(n_1, n_0)$ gives the probability distribution of the true expected reward of that office given that the robot has performed successful services for that office n_1 times and failed services n_0 times, where $B(x, y) = \frac{(x-1)!(y-1)!}{(x+y-1)!}$ is the beta function. To assign rewards in step 1 of the naive algorithm, the robot samples a reward $r' \sim B_i(n_1, n_0)$ for each office u_i and assigns the sample reward to that office. As a result, the probability of office u_i having the highest sampled reward, and thus being chosen in Step 2 of the naive algorithm, is equal to the probability that u_i has the highest true expected reward of any office given the rewards the robot has received in the past.

4.1.2 UCB1

UCB1 is an optimistic algorithm that intentionally overestimates the expected reward of an office based on the uncertainty of its model of that reward[1]. That way, the robot will be biased towards gathering data from offices for which the reward is highly uncertain in order to improve its model. Specifically, in step 1 of the naive algorithm, the robot assigns each office u_i a value of $r' = r + \beta\sqrt{\frac{2\ln n}{n_i}}$, where r is the average observed reward from that office, n is the total number of attempted services the robot has ever made, n_i is the number of attempted services the robot has made for office u_i, and β is a tuning parameter.

4.2 Planning Thompson Sampling and Planning UCB1

We now contribute new algorithms that plan ahead, taking into account how the constraints that the robot faces affect its future actions over the full segment instead of only its next action. These algorithms are designed to address the fact that the constraints on the robot not only restrict the actions available to the robot at any given moment, but also the future actions that will be available after it performs a service.

Our algorithms have the following steps:

1. Assign a reward to each location in the environment.
2. Compute a plan that maximizes the reward received while reaching the end location in time.
3. Execute the computed plan.

Step 1 of this algorithm is the same as step 1 of the naive algorithm. The robot assigns rewards to each location using either the Thompson Sampling or UCB1 method as described in Sections 4.1.1 and 4.1.2. However, rather than naively performing one service at a time until time runs out, the robot computes the optimal plan ahead of time in step 2. Thus, we refer to the two versions of this algorithm as **Planning Thompson Sampling** or **Planning UCB1**.

Finding the optimal plan in step 2 of the planning algorithm can be done in a variety of ways. Finding the plan that maximizes the expected reward received while still ensuring that the robot reaches the end location in time is an instance of the Orienteering Problem. The Orienteering problem can be solved optimally in any environment with a mixed integer program, although doing so is very slow[5]. While approximation algorithms exist, in our experiments we took advantage of the structure of the environments that we used in order to find an optimal solution more efficiently that the general MIP method.

4.2.1 Maximizing Reward in the Hallway Sequence Environment

For our tests in the Hallway Sequence environments, as described in Section 3.2.2, the start location was always the intersection on one end of the environment (A), and the end location was always the intersection on the opposite end (C). Additionally, the time limit, edge lengths, and time to offer a service $t_{service}$ were all chosen so that the robot only had time to travel directly from the start location to the end location and offer services to a given number of offices m along the way, without time to travel to any offices not on the path taken.

With these restrictions, the possible paths the robot can take from the start location to the end location can be easily enumerated. In the environment shown in Figure 1a, there are only three possible paths the robot can take from one end of the environment to the other, one for each of the hallways between the A and B. For each possible path, the robot can find the optimal plan of offices to offer services to along that path by choosing the m offices with the highest assigned reward. Once the robot has computed the optimal plan for each possible path, it performs the one with the highest total assigned reward.

4.2.2 Maximizing Reward in the Star Environment

For our tests in star environments, as described in Section 3.2.3, the start and end locations were always the center of the environment, and the edge lengths were always integers. This allowed us to find the plan that maximized the assigned values using a mixed integer program (MIP) that can be solved much more quickly than the one used to solve the general orienteering problem. The MIP has one variable u_i for each office corresponding to whether that office is contained in the plan, and one variable h_j for each hallway corresponding to the total distance traveled by the robot along that hallway before returning to the center node. The objective function is to maximize the total reward

$$\sum_{u_i} R_i u_i,$$

where R_i is the value assigned to the office u_i. For each variable u_i, there is a constraint

$$D(C, u_i)u_i \leq h_j,$$

where $D(C, u_i)$ is the distance from the center office to the office u_i, and h_j is variable corresponding to the hallway that contains u_i. These constraints ensure that each hallway variable h_j accurately represents the distance the robot must travel along the corresponding hallway to visit the offices in the plan. Finally, there is one constraint ensuring that the final plan meets the time constraint:

$$\sum_{h_j} 2h_j + \sum_{u_i} u_i t_{service} \leq t_{end}.$$

The term $\sum_{h_j} 2h_j$ represents the total time the robot spends traveling, while $\sum_{u_i} u_i t_{service}$ represents the total time spend visiting nodes.

The solution to this MIP represents the offices visited in the optimal plan. To create the actual plan from the solution, for each hallway in the environment, we add the offices visited to the plan in order of their distance from the center office. The order in which the hallways are visited does not matter.

4.3 Example

Consider a robot operating in the environment shown in Figure 1b. Assume that $u_{start} = u_{end} = Z$, $t_{service} = 10$, and $t_{end} = 100$, all edges have length 10, and that the robot has assigned the rewards shown in the figure.

The naive algorithm begins by choosing the node with the highest expected reward, D, and travels there to perform a service. This has a cost of 50 (40 to travel and 10 to offer the service), so it has a remaining time of 50 before it must return to Z. The only nodes it can offer a service to in that time are A, B, and C. A and C are tied with an expected reward of 0.7, so it selects one at random. Regardless of its choice, it is only left with enough time to return to Z without performing any further services. The total expected reward of the robot's plan is 1.6.

The planning algorithm, on the other hand, computes the optimal plan using the MIP described in section 4.2.2. In this case, that plan is to offer services to the nodes E, F, and G, yielding an expected reward of 2.3.

5 Experimental Results

We performed experiments in simulation in order to compare naive UCB1 and Thompson Sampling with Planning UCB1 and Thompson Sampling in the star and hallway sequence environment structures. In these experiments, for each office in the environment u, the probability p_u of a service succeeding at that office was randomly chosen between 0.0 and 1.0.

5.1 Algorithms

Our experiments featured the four algorithms described in Sections 4.1 and 4.2: Naive Thompson Sampling, Planning Thompson Sampling, Naive UCB1, and Planning UCB1. In the hallway sequence environment, a value of 0.2 was used for β for both UCB1 algorithms, while in the star environment, a value of 0.1 was used, both determined empirically.

We also tested three control algorithms. The **Greedy** algorithm only exploited and never explored. It assigned a reward to each office equal to the average observed reward of that office in previous iterations, then found the plan that maximized the expected reward using the algorithms described in Section 4.2.

The second control algorithm was **Random**, which always generated a random plan, effectively only exploring and never exploiting. In the hallway sequence environment, a random plan was generated by choosing a random path from the start to the end, and then randomly choosing m offices along that path, where m is the number of offices the robot had time to visit. In star environments, a random plan was generated by assigning a random "reward" to each office in the environment, and then finding the plan that maximized the random reward received.

The third control algorithm was **Epsilon**. Each iteration, the epsilon algorithm would use the random algorithm with probability ϵ and the greedy algorithm with probability $1 - \epsilon$. The value of ϵ was given by the equation $\epsilon = \frac{\log T}{T}$, where T is the number of iterations that have passed.

In all cases except the random algorithm, if the robot had never previously offered a service to an office before, it assigned a value of 1.0.

5.2 Performance Metrics

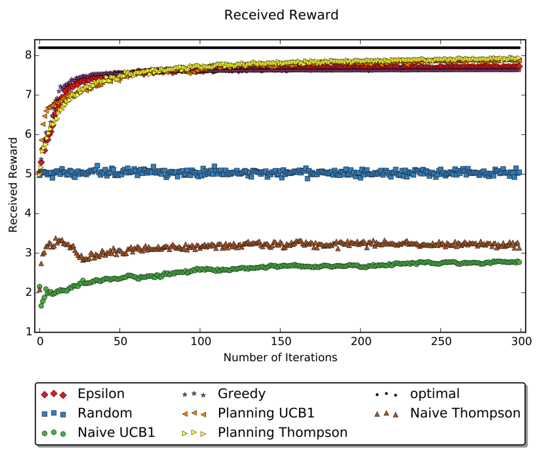
We compared the algorithms using two primary metrics. The first metric, "reward received," is the average expected reward of the plan executed by the robot at each iteration. We used the expected reward of the robot's plan based on the probability of each service succeeding, rather than actual number of successful services, in order to reduce the variance in the results due to randomness. This serves as a metric for how well the algorithm balanced exploration and exploitation in order to receive as much reward as possible over time.

The second metric, "best reward," is the true expected reward of the plan with the best expected reward according to the robot's model. This is computed by using the greedy algorithm to create a plan using the data gathered by the robot so far. In other words, if the robot ceased exploring and switched to pure exploitation using the greedy algorithm at any given iteration, "best reward" is the expected reward it would receive. This serves as a metric of how effective the robot's model is for choosing a high-reward plan, and thus how successful the robot has been in exploring the environment.

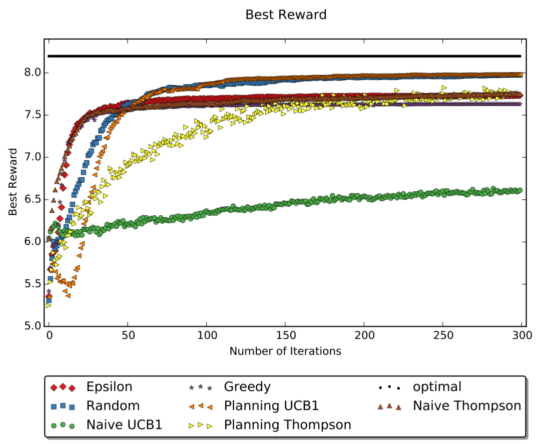
We also observed how frequently the robot visited each office when following each algorithm, and compared it to how often each office was part of the true optimal plan, computed using the algorithms described in Section 4.2 using the true expected reward of each office. The purpose of this was not to gauge performance, but to observe the interaction between the algorithms and the structure of the environment.

5.3 Hallway Sequence Results

Our experiments in hallway sequence environments (described in Section 3.2.2) ran for 300 iterations each, and our results were averaged over 200 trials. The environment consisted of



(a) The expected reward received by the robot. Planning Thompson Sampling and Planning UCB1 had the best performances.



(b) The reward of the best plan according to the robot's model. Random exploration and Planning UCB1 had the best performances.

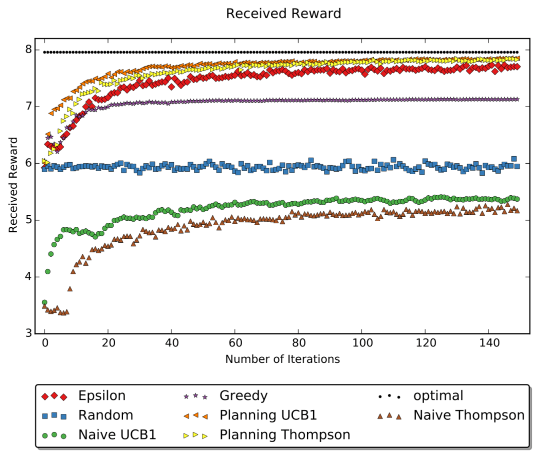
Figure 2: Results in the Hallway Sequence environment.

three intersections, with multiple hallways connecting the first two intersections and a single hallway connecting the second to the third, like the environment shown in Figure 1a. Each hallway had ten offices, not including the intersections. Each iteration, the robot chose a path from the first intersection to the last intersection, and had time to offer services to up to ten offices along that path, but not enough time to offer services to any offices on other paths.

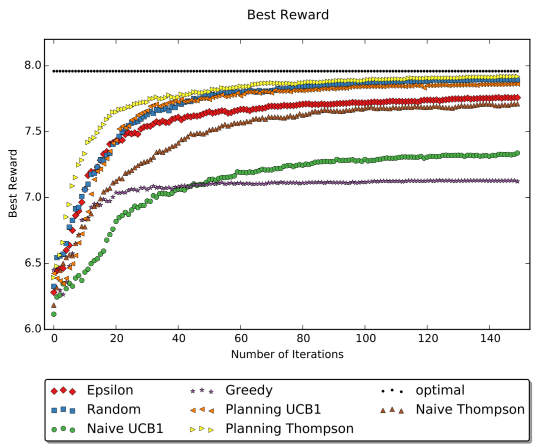
Figure 2a shows the average reward received by each algorithm over time in an environment with five hallways between the first two intersections. Greedy and Epsilon performed very well at very low numbers of iterations, but within the first 100 iterations on average were overtaken by Planning UCB1 and Planning Thompson Sampling, which performed roughly equally. Naive UCB1 and Thompson Sampling had terrible performances, receiving less reward than even the random algorithm, but between the two the Naive Thompson Sampling received more reward. Increasing the number of hallways between the first two intersections did not significantly affect these results.

Figure 2b shows the average best reward of each algorithm. In this case, Random exploration and Planning UCB1 had the best performances. Planning Thompson Sampling had a poor best reward at small numbers of iterations, but improved dramatically over time. Its best reward continued to improve with more iterations, but did not overtake Planning UCB1 or Random. Naive UCB1 once again performed poorly, but Naive Thompson Sampling was surprisingly effective, with comparable performance to Epsilon. Increasing the number of hallways between the first two intersections caused a significant decline in the performance of the Planning Thompson Sampling algorithm by this metric, without any significant effect on the other algorithms.

We can gain some insight into our results by looking at how often each algorithm visited each office on average. The frequency with which Planning Thompson Sampling and Planning UCB1 visited each office in the environment corresponded roughly to how often each office was contained in the optimal plan, between 10% and 15% of iterations for most offices in the first set of hallways and between 30% and 45% of the time for offices in the final hallway.



(a) The expected reward received by the robot. Planning Thompson Sampling and Planning UCB1 had the best performances.



(b) The reward of the best plan according to the robot's model. Random exploration, Planning Thompson Sampling, and Planning UCB1 had the best performances.

Figure 3: Results in the Star environment.

Random exploration, meanwhile, visited all of the rooms in the first five hallways in 9-10% of the iterations, and the other rooms 48% of the time. Epsilon's visiting frequencies were in between those of random exploration and the planning algorithms.

The two naive algorithms had very low visiting frequencies in general, often visiting fewer than the allowed 10 offices per iteration. They had especially low visiting frequencies for offices closer to the start location. The Naive UCB1 and Naive Thompson algorithms both visited the offices in the first set of hallways between 1% and 4% of the time in most cases, with fewer visits for the closest ones. The nearest rooms in the final hallway were visited less than 10% of the time, but this increased as the rooms got farther from the start location, with the office immediately before the end location being visited in more than 50% of their plans.

From these results, we can conclude that the naive algorithms frequently jumped straight to offices farther from the start location without performing services at other offices along the way. Because the scenario is set up to prevent backtracking, they missed many convenient offices that they could have visited at a low cost. The planning algorithms, on the other hand, visited offices roughly proportionally to how frequently they had opportunities to do so, indicating that they took advantage of the convenient opportunities present in the environment structure.

5.4 Star Results

Our experiments in Star environments, as shown in Figure 1b, ran for 150 iterations each, and our results were averaged over 200 trials. The results shown are for environments with six hallways, each consisting of six offices not including the center, and the edges were all of length 15, while $t_{service}$ was 20 and t_{end} was 500. The start and end location were both always the center office. We tested environments with as few as four or as many as eight hallways, but did not see a significant change in the results.

Figure 3a shows the average reward received by each algorithm over time. As with our results in the Hallway Sequence environment, Planning UCB1 and Planning Thompson Sampling received the most reward, although Epsilon was close behind.

Figure 3b shows the average best reward of each algorithm. Planning Thompson Sampling, Planning UCB1, and Random once again had the best performance and all reached about the same near-optimal reward after 150 iterations, but Planning Thompson performed better for very low numbers of iterations. As in the Hallway Sequence environments, Naive UCB1 performed extremely poorly by this metric, while Naive Thompson Sampling performed surprisingly well, although still worse than either of the planning or random exploration algorithms.

Examining the visiting frequencies, we found that offices closer to the center were part of the optimal plan more often, with those adjacent to the center office being part of the optimal plan typically between 50% and 60% of the time while the offices at the ends of the hallways were all part of the optimal plan in less than 10% of trials. Once again, Planning Thompson Sampling and Planning UCB1 visited offices with a very similar frequency to how often they appeared in the optimal plan. Interestingly, random exploration also had visiting frequencies very close to the optimal plan.

The naive algorithms once again had lower overall visiting frequencies. In this case, the biggest issue was with offices close to the center. While the naive algorithms both visited offices far from the center in about 10% of plans on average, slightly higher than the frequency with which those offices appeared in the optimal plan, they visited the offices closest to the center in 15-25% of their plans, much less than the frequency with which those offices appeared in the optimal plan.

The offices closest to the center of the environment are the ones that are most often convenient to the robot's plan, since they are convenient to any plan that travels farther down the hallway. In the star environment, we can see that the naive algorithms frequently bypassed opportunities to visit convenient offices, while the planning algorithms did not, much like in the hallway sequence environment. In the case of the star environment, there is a very clear correlation between how frequently an office is convenient and how big the difference in visiting frequency is between the naive and planning algorithms. This further goes to show that the naive algorithms failed to take advantage of the presence of convenient opportunities due to the environment's structure, while the planning algorithms succeeded in doing so.

6 Conclusion

In this paper, we presented the problem of a service robot attempting to perform unscheduled services for users in its spare time. To do so, the robot must balance exploration and exploitation in order to learn which users are likely to accept an offered service while gathering reward for performing successful services. However, the need to travel through the environment and to perform scheduled user requests at specific locations and times constrain the robot's ability to offer services to users. Additionally, the hallway-based environment structure causes variation in how often the robot has the opportunity to perform services for a low cost to different users.

To address this problem, we presented the Planning Thompson Sampling and Planning UCB1 algorithms. These algorithms are based on the existing UCB1 and Thompson Sampling algorithms used in standard multi-armed bandit problems, but modified to plan ahead over an entire iteration between two user requests instead of only choosing a single action at a time. We compared our planning algorithms to ordinary naive ones in simulation. Our results showed that our planning versions of the algorithms performed better in terms of both reward received and effectiveness of the model learned. Analysis of the offices visited by each algorithm showed that the naive the algorithms were not taking advantage of cases where offices could be conveniently visited on the way to another office, indicating that our planning algorithms worked better within the structure of the environment.

In our future work, we hope to apply our algorithms to a wider variety of environment structures, as well as create algorithms that deliberately use the structure of the environment to their advantage. Our results in this paper showed the importance of the structure of the environment, and we believe these results can be generalized and expanded upon in the future. Once our algorithms have been sufficiently refined and generalized to a variety of environment structures, we plan to test them on real robots to gather results in an actual office environment with real human users.

References

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[2] Jeremy W. Baxter, Jack Hargreaves, Nick Hawes, and Rustam Stolkin. Controlling anytime scheduling of observation tasks. In *Proceedings of AI-2012, The Thirty-Second SGAI International Conference on Artificial Intelligence*, December 2012.

[3] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

[4] Sudipto Guha and Kamesh Munagala. Multi-armed bandits with metric switching costs. In *Automata, Languages and Programming*, pages 496–507. Springer, 2009.

[5] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[6] Marc Hanheide, Nick Hawes, Jeremy Wyatt, Moritz Gobelbecker, Michael Brenner, Kristoffer Sjoo, Alper Aydemir, Patric Jensfelt, Hendrik Zender, and Geert-Jan M. Kruijff. A framework for goal generation and management. In *Proceedings of the AAAI Workshop on Goal-Directed Autonomy*, 2010.

[7] Max Korein, Brian Coltin, and Manuela Veloso. Scheduling mobile exploration tasks for environment learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1255–1256. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[8] Max Korein, Brian Coltin, and Manuela Veloso. Constrained scheduling of robot exploration tasks. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 429–436. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[9] Steven L Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[10] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[11] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *ECML*, volume 3720, pages 437–448. Springer, 2005.