Constrained Scheduling of Robot Exploration Tasks

Max Korein, Brian Coltin, Manuela Veloso
mkorein@cmu.edu, vcoltin@cmu.edu, veloso@cmu.edu
Robotics Institute and Computer Science Department, Carnegie Mellon University, Pittsburgh, USA

ABSTRACT

In order for an autonomous service robot to provide the best service possible for its users, it must have a considerable amount of knowledge of the environment in which it operates. Service robots perform requests for users and can learn information about their environment while performing these requests. We note that such requests do not take all of the robot's time, and propose that a robot could schedule additional exploration tasks in its spare time to gather data about its environment. The data gathered can then be used to model the environment, and the model can be used to improve the services provided. Such additional exploration is a constrained form of active learning, in which the robot evaluates the knowledge it can acquire and chooses observations to gather, while being constrained by its navigation and the time underlying the user requests it receives. We create a schedule of exploration tasks that meets the given constraints using a hill-climbing approach on a graph of tasks the robot can perform to gather observations. We test this method in simulation of a CoBot robot and find that is able to learn an accurate model of its environment over time, leading to a near-optimal policy when scheduling user requests.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Robotics, Exploration, Learning, Scheduling

1. INTRODUCTION

Autonomous mobile service robots typically require knowledge of their environment in order to perform their services optimally. Robots that must perform scheduled tasks are often dependent on knowledge that is specific to a location and varies with time. For example, a robot that delivers messages in an office building would benefit from knowing when a person is most likely to be in their office to receive a message. An autonomous taxi would be able to pick up more fares if it knew where it was most likely to find a passenger at a given time.

These robots would be capable of acquiring this knowledge autonomously. An office robot may be able to detect whether an office door is open or closed, and an autonomous taxi would presumably be capable of detecting people trying to hail a cab, even when it is not actively seeking passengers. The robot may be able to make these observations as it travels during the course of providing its normal services, but then its knowledge would be heavily weighted towards the locations it commonly travels past.

Ideally, the robot would be able to actively seek additional information for improving its services. Its ability to do so, however, is constrained by the need to continue performing the services themselves. Thus, actively gathering data is a problem of constrained exploration and active learning. The robot must consider not only the observations it can make and how valuable they will be, but also how to navigate its environment in order to gather them while still completing all user requests at the scheduled times. The ultimate goal is to maximize the reward of the information the robot acquires in the limited spare time it has between user requests, and use that information to improve its services in the future.

We contribute an algorithm to solve this challenge in which the robot creates a schedule of exploration by finding a path through a graph of possible exploration tasks using a hill-climbing method. Through experiments performed in simulation, we show that our algorithm gathers more data and learns a better model than random exploration, particularly when guided by an informed reward function. Given sufficient time it learns a model that results in a near-optimal policy when scheduling user requests.

2. HYPOTHESIS

We consider a robot that receives mandatory tasks, such as requests from users, that require the robot to travel to a specific location to perform them. The robot is given windows of time during which it must perform each task, and the robot's performance on the request depends on decisions it makes. The quality of the robot's performance can be predicted using features of the environment that the robot is capable of observing, and these features may be time-dependent. For example, the probability of an office robot's message deliveries succeeding depends on the probability of the recipient's door being open, which depends on the time of day. When the robot is not performing a user request, it is idle.

Our goal is to have the robot make use of this idle time to gather information that will allow it to improve its services. Specifically, we seek to show two things:

1. The robot can observe features of its environment as it travels. Furthermore, it can make use of the spare time in between user requests to gather additional observations. This uses only the time in between user requests, during which the robot would otherwise be idle. It does not restrict the availability of the robot to perform user requests or require it to alter the scheduling of user requests to make room for exploration.

2. The robot can create a model of observed features of the environment, and use the model to improve its services. The more accurate the model, the better the services.

3. EXPLORATION SCHEDULING

We now formalize the problem of scheduling exploration tasks in the robot's spare time in order to gather information, and using that information to improve services. By finding an effective solution to this problem, we will be able to demonstrate the hypothesis put forth in Section 2.

First, we define some terminology we will use throughout this paper. We will use the term "interval" to refer to a contiguous window of time I, starting from $t_{start}(I)$ and ending at $t_{end}(I)$. Some intervals also have a start location $L_{start}(I)$ and an end location $L_{end}(I)$, which represent the location the robot must be at at the beginning or end of that interval.

We will use the term "task" or "request" to refer to actions the robot can schedule. Typically, we will use the term "user request" or "request" to refer to actions requested by a user, and "exploration task" or "task" to refer to actions the robot chooses to perform on its own. To begin the task r, the robot must be at the start location $L_{start}(r)$. The task has a duration $d(r)$ that it takes to perform it, and ends at $L_{end}(r)$.

In the exploration scheduling problem, the robot is given an interval of operation I_{op} over which it must create a schedule. It is also given a set of user requests $\{r_1, \dots, r_n\}$, as well as a set of corresponding constraint intervals $\{I_{c,1}, \dots, I_{c,n}\}$, during which they must be performed. Additionally, the robot is given a set of exploration tasks, $\{s_1, \dots, s_m\}$. Some of these exploration tasks may be declared "mutually exclusive" from each other, which means they overlap in the information they provide. For example, in the case of an office robot learning when people's office doors are most likely to be open, it might have exploration tasks that consist of traversing a hallway in one direction or another and observing the states of the doors it passes. In this case, the two tasks for traversing the same hallway in opposite directions would be labeled mutually exclusive, because the information they provide is approximately the same.

Finally, the robot is given two reward functions, $R_{request}(r,t)$ and $R_{explore}(s,t)$, which give the expected reward of performing a user request or an exploration task at a given time, respectively. $R_{request}$ is the expected probability of a task succeeding at a given time, while $R_{explore}$ is the expected usefulness of the information that will be gathered by performing an exploration task.

The robot must produce a schedule consisting of start times for user requests, along with a set of exploration tasks that it will perform and start times for each of them. There are two hard constraints on this schedule:

1. The start time for each user request r_i must be within the interval of constraint $I_{c,i}$.

2. There must be sufficient time to travel from the end location of each task to the start location of the next one.

Additionally, there are three soft constraints that we would like the robot to meet:

1. Maximize the total reward achieved by the user requests.

2. Complete all user requests at the earliest time possible.

3. Maximize the total reward achieved by the exploration tasks.

These soft constraints are in order of priority. For the purposes of this paper, we will treat these priorities as strict and not consider tradeoffs: maximizing the request reward is always more important than performing user requests quickly, which is always more important than any amount of exploration reward. This is consistent with our goals stated in Section 2 that the robot should not need to adjust the scheduling of user requests to make room for the exploration. The robot must be able to continue scheduling user requests as optimally as possible, both in terms of maximizing the reward received and performing them at the earliest time possible (which is desirable for many users and makes the schedule more robust to changes in the plan due to underestimated travel times or new requests).

4. RELATED WORK

The problem of evaluating knowledge that a robot could gather is an instance of active learning, and various techniques for handling active learning problems have been developed [10]. In standard active learning problems, information can be acquired at a fixed cost, and the challenge lies in assigning rewards to observations in order to learn the best model while meeting constraints such as minimizing cost.

Some work has been done with office service robots that learn information that will improve their services. An algorithm for finding and fetching objects in an office environment using environmental knowledge is presented in [9] and [7]. Jijo-2 is an office robot that is capable of learning information about its environment that will assist in the completion of its task through conversations with users [1]. The Dora the Explorer robot is an autonomous office robot that delivers objects, and develops a probabilistic model of where different objects can be found based on what it has observed. It evaluates its current knowledge to determine what new observations are most likely to benefit its model, and finds and delivers an object for a user based on this model[6, 5, 4].

The work with Dora the Explorer is focused primarily on planning around the spacial constraints of gathering data in an office environment. In the problem we seek to solve, however, there are temporal constraints as well imposed by the robot's need to fulfill user requests. Scheduling user requests for such a robot is an instance of the Vehicle Routing Problem, in which a set of robots are assigned to fulfill a set of spatially located tasks. Specifically, it is an instance of the Vehicle Routing Problem with Time Windows (VRPTW) in which the tasks must be executed with fixed windows of time. The VRPTW is commonly solved optimally with branch and bound approaches, such as mixed integer programs [11]. Transferring items between rooms is an instance of the Pickup and Delivery Problem with Time Windows (PDPTW), and when tasks come in online, it is an instance of the Dial-a-Ride Problem, named after how customers call cabs in a city in an online fashion. The PDPTW and Dial-a-Ride Problem have been approached with a multitude of heuristics and metaheuristics [2, 8].

While solving the PDPTW problem using a mixed integer program is effective for scheduling user requests for an office robot, such techniques scale poorly with the number of tasks the robot is attempting to schedule. When attempting to schedule additional exploration in between user requests, the number of tasks makes an optimal mixed integer approach infeasible. Thus, we must use a more efficient approach to scheduling exploration tasks that allows us to consider the spacial and temporal constraints on the robot's ability to acquire data while still being executable in near-real time for the large number of tasks being considered.

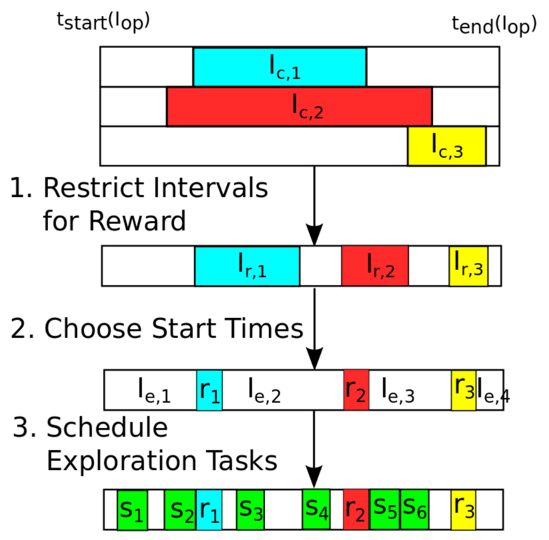

Figure 1: An illustration of the process of exploration scheduling.

5. EXPLORATION SCHEDULING PROCESS

We now present a baseline approach to solving the exploration scheduling problem described in Section 3. While this approach is not guaranteed to be optimal, it is fast and allows us to test the hypothesis put forth in Section 2, that a service robot can explore and and learn information that improves its services in the long term without making any short-term sacrifices in its services. The steps are as follows:

1. For each user request r_i, find the sub-interval I_{r_i} of that $I_{c,i}$ during which the request will yield the maximum reward.
2. Schedule each user request at the earliest possible time in I_{r_i}.
3. Schedule exploration tasks in between the user requests.

A diagram of this process is shown in Figure 1.

5.1 Scheduling User Requests

The first step in the process is to restrict the intervals of constraint for each user request to the sub-interval that provides the optimal reward. For each user request r_i with interval of constraint $I_{c,i}$, we choose a new restricted interval of constraint $I_{r,i}$:

$$I_{r_i} = \operatorname*{arg\,max}_{I \subseteq I_{c,i}} \frac{1}{\text{length}(I)} \int_{L_{start}(I)}^{L_{end}(I)} R_{request}(r_i, t), \tag{1}$$

In some cases, there may be a single instantaneous time at which the reward is maximized. In these cases, we choose some minimal interval length minlength and use an interval of that size surrounding the optimal time. However, if the maximum in the request reward occurs as a plateau, rather than a single point, then the interval will be the entire plateau. Figure 5.1 shows a set of example reward functions that would result in the intervals shown in Step 1 in Figure 1.

It is possible for a conflict to exist in which it is not possible to perform each request within the chosen optimal intervals together. In these cases, we schedule the user requests according to the function:

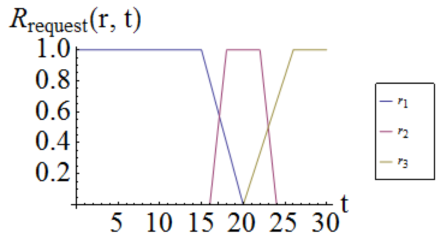

Figure 2: Example reward functions for the three requests. r_1 provides the optimal reward for $t \leq 15$, r_2 provides the optimal reward for $18 \leq t \leq 22$, and r_3 provides the optimal reward for $t \geq 26$. These rewards would lead to the restriction of the intervals shown in step 2 in Figure 1.

$$I_{r_i} = \operatorname*{arg\,max}_{I \subseteq I_{c,i}} \frac{1}{\text{length}(I)} (f * \frac{\text{length}(I)}{\text{minlength}} + 1) \int_{L_{start}(I)}^{L_{end}(I)} R_{request}(r_i, t), \tag{2}$$

where f is the flexibility parameter. The larger the flexibility parameter is, the more the function promotes choosing a larger interval of constraint over an optimal one. When $f = 0$, this is equivalent to using Equation 1. In the case of a conflict, f is increased incrementally (we used $\delta = 0.1$ in our experiments) until a schedule can be found. The effect is that the restricted intervals of constraint are slowly widened until the conflict no longer exists.

The second step of the process is to schedule each user request as early as possible within the restricted intervals of constrain chosen in the first step. This is done using the existing algorithm described in [3], which solves a mixed integer program to determine the schedule that will accomplish all user requests as early as possible. An example can be seen in Step 2 of Figure 1.

5.2 Scheduling Exploration Tasks

The final step of the process is to schedule exploration tasks. With the user request times chosen, there are now fixed gaps in between the user requests, which we will call the Intervals of Exploration $\{I_{e,1}, \ldots, I_{e,n+1}\}$. We consider each interval of exploration independently, scheduling a set of exploration tasks for that interval and then merging the schedules for the individual intervals of exploration into the full schedule.

The exploration tasks are scheduled by constructing what we are calling a "task graph," a directed graph with nodes representing the exploration tasks the robot can perform and locations it can visit. The robot then finds a path through the task graph using a greedy hill-climbing algorithm, and schedules the tasks on the path.

5.2.1 Task Graph Construction

The task graph is a directed graph featuring one node for each exploration task the robot can perform. There is an edge from each task node s_i to each other task node s_j with length equal to $D(L_{end}(s_i), L_{start}(s_j))$, where $D(L_1, L_2)$ is the estimated time for the robot to travel between two locations. The exception is nodes for tasks that are mutually exclusive with each other, which are not connected by an edge. Each task node also has a duration, equal to the expected duration of performing the task, and a reward, equal to the average reward over the interval of exploration.

In addition to the task nodes, the task graph has two nodes representing locations: one for the start location of the interval of exploration, and the other for the end location. There is an edge from the start location node L_s to each task node s_i with a length of

$D(L_s, L_{start}(s_i))$. Similarly, there is an edge from each task node s_i to the end location node L_e with length equal to $D(L_{end}(s_i), L_e)$.

An example of a simple environment and a corresponding task graph are shown in Figures 3 and 4, respectively.

5.2.2 Task Graph Hill Climbing

A path through the task graph represents a sequence of exploration tasks. The total cost of a path is the sum of all edge costs and task node durations along the path, and represents the expected time it would take to perform the tasks. The total reward of a path is the sum of the rewards of all task nodes in the path. Any path from the start location node to the end location node with length less than the length of the interval of exploration is a valid schedule of exploration tasks for the interval of exploration. Thus, our goal in scheduling exploration tasks is to find the path that meets this criterion with the highest reward.

Our approach to this problem is a greedy hill-climbing algorithm. The task graph hill-climbing algorithm takes as input a task graph G (which includes a function $D(n_1, n_2)$ that gives the edge length from node n_1 to node n_2, and a duration function *dur* that gives the duration of a task node), a start location S, an end location E, a reward function R which gives a task's average reward over the interval of exploration, a maximum cost for the path *maxC*, and a dictionary of mutually exclusive nodes *mutexes*. It returns a locally optimal simple path P through the graph from S to E with cost less than or equal to *maxC* that does not violate any of the constraints in *mutexes*. Pseudocode for the algorithm is shown in Algorithm 1.

The algorithm works by iteratively adding tasks to the path until no more tasks can be added, at which point E is added to finish the path. Each iteration, each task in the graph is assigned a marginal reward (lines 5-11). If the task is already in the path, mutually exclusive with a node already in the path, or there is not enough time left to perform the task and then travel to the end location, the task is assigned a marginal reward of $-\infty$ (lines 6-7). Otherwise, the task is assigned a marginal reward equal the ratio of the task's reward to its cost, defined as the sum of the edge to the task node and the task's duration (line 9). If no tasks have a positive marginal reward, then it is not possible to perform any more tasks in the time remaining, so the end location is added to the path and the path is returned (lines 12-14). Otherwise, the task with the highest marginal reward is added to the end of the path, and the process is repeated (lines 16-17).

5.2.3 Hill Climbing Example

Consider the environment shown in Figure 3. This environment has three locations, L1, L2, and L3. There are six tasks, labeled as "T12", "T21", etc. in the map. Each task consists of the robot traversing from one end of a hallway to the other.

Now, for an example of the hill-climbing algorithm, consider our robot attempting to schedule a set of exploration tasks for an interval of exploration that begins at $t_{start}(I_e) = 0$ and ends at $t_{end}(I_e) = 25$, starting from L1 and ending at L3. Let each task have a reward equal to the number of rooms on the hallway, and let the two tasks for crossing each hallway be mutually exclusive. The task graph for this search is shown in Figure 4.

The search begins at the L1 node. The algorithm computes the marginal reward of each task. Task 31 has a marginal reward of $-\infty$, because the total time to start from the current location, perform Task 31, and then travel to L3 would be 36, while the total time available for planning is 25. There is similarly not enough time to perform task 32. Of the remaining tasks, Task 12 has a marginal reward of 0.4, Task 21 has a marginal reward of 0.2, Task 13 has a marginal reward of 0.25, and Task 23 has a marginal reward of 0.22. So Task 12 is added to the path, and the search continues with Task 21 as the starting location.

Now, Task 21 is assigned a marginal reward of $-\infty$, because it is exclusive with Task 12. There is still not enough remaining time to perform Task 31 or 32, so those are out as well. Meanwhile, Task 13 gets a marginal reward of 0.18, while Task 23 gets a marginal reward of 0.31, so Task 23 is added to the path. Next the search continues from Task 23, but we find that all remaining tasks now have a marginal reward of $-\infty$: either there is insufficient time to complete them and still reach L3 (Tasks 13 and 31), or they are

Algorithm 1 The hill-climbing algorithm for finding a path of a given cost through the task graph.

```
 1: procedure TASKGRAPHHILLCLIMB(G, S, E, R, maxC, mutexes)
 2:     $P \leftarrow S$
 3:     last $\leftarrow S$
 4:     loop
 5:         for task $\in G$ do
 6:             if $D(\text{last}, \text{task}) + D(\text{task}, \text{E}) + dur(\text{task}) > \text{maxC}$
                    $\vee$ task $\in$ P
                    $\vee$ mutexes[task] $\cap$ P $\neq \emptyset$ then
 7:                 marginal[task] $\leftarrow -\infty$
 8:             else
 9:                 marginal[task] $\leftarrow \frac{R(\text{task})}{D(\text{last},\text{task}) + dur(\text{task})}$
10:             end if
11:         end for
12:         if marginal[task] $\leq 0 \; \forall$ task $\in G$ then
13:             Append $E$ to $P$
14:             return $P$
15:         else
16:             last $\leftarrow \underset{\text{task} \in G}{\text{argmax}}(\text{marginal[task]})$
17:             Append last to $P$
18:         end if
19:     end loop
20: end procedure
```

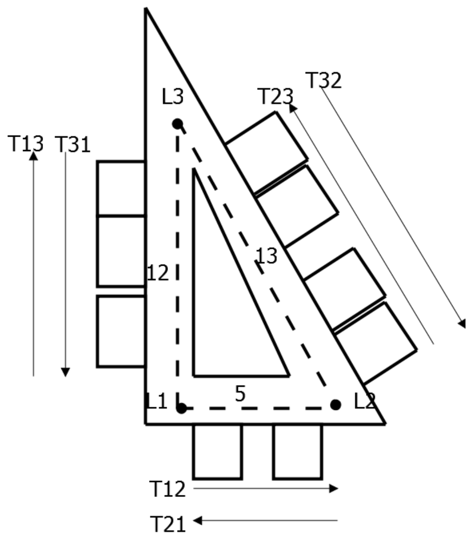

Figure 3: A simple environment consisting of three locations, L1, L2, and L3, and six tasks, T12, T21, T13, T31, T23, and T32. Each task corresponds to traversing a hallway in a certain direction.

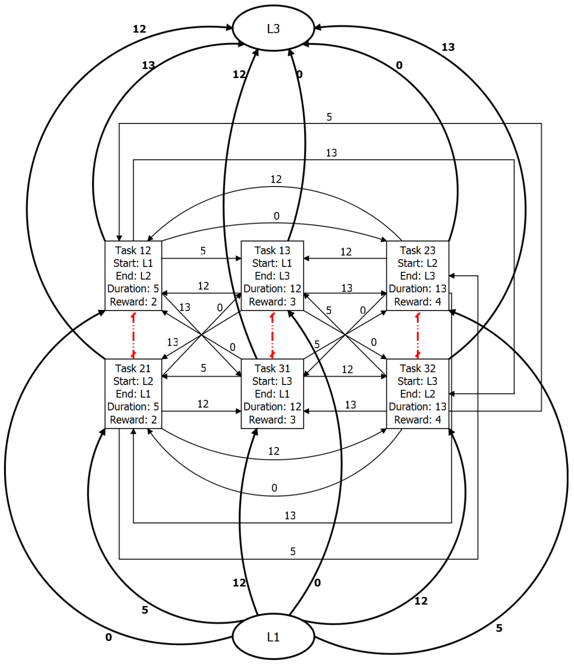

Figure 4: A task graph for the environment shown in Figure 3, with a start location of L1 and end location of L3.

mutually exclusive with a node that is already in the path (Tasks 21 and 32). So L3 is added to the path and the search is complete.

6. EXPERIMENT

We tested our algorithm using a simulation of an office robot in the seventh through ninth floors of the Gates Hillman Center at Carnegie Mellon University. In the simulation, the robot repeatedly received sets of user requests to deliver messages within a subset of a 30-minute interval of operation. The probability of doors being open varied with time, and a message delivery failed if the robot delivered it to a closed door. The robot scheduled exploration tasks in between the user requests, and learned a model of the probabilities of doors being open over time. It used the model to determine how to schedule message deliveries to maximize the odds of success.

6.1 Experimental Setup

Each simulated interval of operation was 30 minutes long. During that interval, the robot received four user requests, each to deliver a message to a randomly chosen office within a random interval of constraint between 5 and 30 minutes long. Messages were delivered (or failed to be delivered) instantaneously upon the robot's arrival at the recipient's office. The requests were always tested to ensure that it was possible to perform all four within the given intervals of constraint. If not, a new set of tasks was generated until a possible set was created.

The environment the robot operated in was based on the seventh through ninth floors of the Gates-Hillman Center at Carnegie Mellon University, with a total of 203 doors. A map of one floor of the environment is shown in Figure 5. The probability of each door being open at a time t was given by

$$p(\text{door} = \text{open}|t) = ae^{-\frac{(t-\mu)^2}{2\sigma^2}}, \tag{3}$$

where a, μ, and σ are randomly chosen parameters for each door. a was between 0.75 and 1.0, μ was between 0.0 and 30.0, and σ

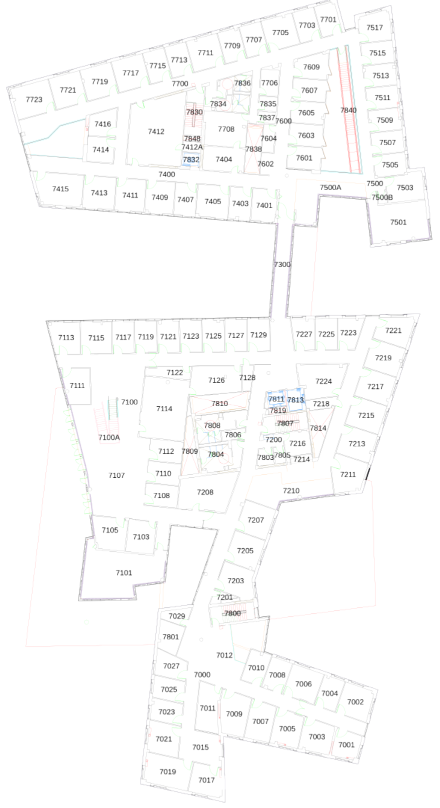

Figure 5: A map of one floor of the simulated environment used in the experiments. The robot travels through the hallways, and can travel between the floors using three elevators, labeled 7811, 7813, and 7832 in this map. The remaining numbers correspond to rooms the robot can visit.

was between 2.0 and 17.0. Whenever the robot passed by a room, it observed the door being either open or closed according to this probability. These observations were made whether the robot was deliberately observing them as part of an exploration task or merely passing by while traveling from one location to another.

The robot stored the time of all observations of doors it made and the observed status of the door. It did not know the form of the ground truth probability distributions for the doors, and inferred a probability at each minute by taking a weighted sample of observations at nearby times:

$$P_{open}(room|t) = \frac{0.5 + \sum\limits_{s \in \text{open door samples}} e^{-\frac{(t_s-t)^2}{50}}}{1.0 + \sum\limits_{s \in \text{all door samples}} e^{-\frac{(t_s-t)^2}{50}}}, \tag{4}$$

where t is the time, t_s is the time of the sample, and only samples for which $|t_s - t| \leq 10$ are considered to speed up computations. The 0.5 in the numerator and 1.0 in the denominator are to reduce overfitting from small numbers of samples, biasing the probability very slightly towards 0.5.

The exploration tasks available to the robot each consisted of traversing a single hallway in the environment. It would begin at one end of the hallway and travel to the other end of the hallway, observing all doors it passed along the way. There were two exploration tasks available per hallway, one starting at each end.

Since the two tasks for each hallway yielded the same observations, they were considered mutually exclusive when performing the hill-climbing algorithm with the task graph.

6.2 Algorithms Tested

We compared three different algorithms for scheduling exploration tasks. Two were variations of task graph hill-climbing, one was a random exploration algorithm for comparison, and two were control models that did not do any exploration.

Uninformed Hill-Climbing. The uninformed hill-climbing algorithm scheduled user requests, constructed task graphs for each interval of exploration, and carried out the hill-climbing algorithm as described in Section 5. It used a uniform reward function, assigning all tasks a constant reward of 1.0. This reward promoted gathering as many observations as possible, regardless of what they are or what data the robot already has.

Informed Hill-Climbing. Like the uninformed hill-climbing algorithm, the informed hill-climbing algorithm scheduled requests and exploration tasks using the method described in Section 5. However, this algorithm used an informed reward for observing a door based on the density of the observations the robot had of that door at that time, given by

$$R_{explore}(hall,t) = \frac{1}{\left(1 + \sum_{s \in \text{samples}} e^{-\frac{(t_s - t)^2}{50}}\right)^2}. \tag{5}$$

The reward for observing all doors on a hallway was the sum of the rewards for observing each of the doors. This reward function gave a higher reward to observations of hallways and times during which the robot had less data. As the number of observations the robot had of a hallway at a given time increased, the reward of exploring that hallway decreased dramatically. Thus, the informed hill-climbing algorithm sought to achieve an even distribution of observations over all doors and times.

Random Exploration. The random exploration algorithm began by restricting the intervals of constrain for the user requests as described in Section 5.1. Once the restricted windows were chosen, it randomly chose between four and seven exploration tasks. It then used the same mixed integer program that was used when choosing the starting times of tasks in Section 5.1 to create a schedule featuring both the user requests (with the restricted intervals of constraint chosen) and the chosen exploration tasks (with intervals of constraint spanning the entire interval of operation).

If a schedule was found, it would execute that schedule. If no schedule could be created, the chosen exploration tasks were rejected and only the user requests were executed using the mixed integer program.

No Exploration. With this "algorithm", the robot did not schedule any exploration tasks at all. Instead, it simply ran the schedule of user requests created as described in Section 5.1. It still learned a model, but only from observations of the doors it passed by while traveling between user requests.

No Learning. The final model tested in each trial was one that learned nothing at all. In this case, the robot always assumed that all doors had a probability of 0.5 of being open at all times. This resulted in all user requests being scheduled as early as possible.

6.3 Results

We ran 30 trials, each consisting of 100 30-minute intervals, with the robot accumulating more knowledge of the environment with each interval. We evaluated the algorithms using three different metrics.

6.3.1 Model Error

The model error was computed by taking the mean difference of each model's learned probability and the true simulated probability of each door being open at each minute. It represents how accurate a model each algorithm was able to create from the observations it gathered.

Figure 6 shows the average error of the model learned by each of algorithm after each interval. The robot learned a much more accurate model of the environment when using exploration tasks than without them. The best model was learned when using the task graph algorithm with an informed reward, which learned a model that was more accurate than the uninformed version of the algorithm by more than one standard deviation within fifteen iterations. The gap between the model errors continued to grow with further exploration. The uninformed version of the algorithm, meanwhile, had an average model error less than that of the random exploration algorithm by more than one standard deviation within five iterations. This demonstrates that hill-climbing through a task graph allows the robot to learn a strong model of the environment, particularly when using an informed reward function that encourages the robot to explore the areas from which it has the least data.

6.3.2 User Request Success Rate

Every ten iterations, each model was tested to obtain a measure of how well the model translated into an effective policy when carrying out user requests. For this test, the model was given 500 requests, one at a time, to deliver messages to randomly-chosen rooms in randomly-chosen constraint intervals between 5 and 30 minutes long. It chose the time in the interval at which it believed the delivery to have the highest chance of success, and received a reward equal to the actual probability of the delivery succeeding at that time. These rewards were normalized relative to the optimal reward the robot could have received if it chose the best possible time for every request.

Figure 7 shows the results of this test. Although the differences between the rewards the algorithms earned are less significant than the differences in their model errors, we can still see that, on average, a better model translated to a higher reward, with algorithms ranking in the same order in terms of performance. Thus, a better model of the environment translates to a better policy when carrying out user requests. The informed hill-climbing algorithm was able to achieve over 97% of the optimal reward after 100 iterations, showing that the model learned was extremely effective for choosing a strong policy.

6.3.3 Cumulative Distribution of Observations

Figure 8 gives some insight into why the algorithms ranked the way they did. It shows the number of doors that acquired at least a given number of observations after 100 intervals. We can see that both task graph algorithms consistently acquired considerably more data than the other methods. The informed task graph algorithm acquired a more even distribution of data than the uninformed algorithm, as it was designed to do, while the uninformed algorithm had a more skewed distribution. Because observations of the same door typically yield diminishing returns in terms of the model learned, it is easy to understand why the informed algorithm had the best performance.

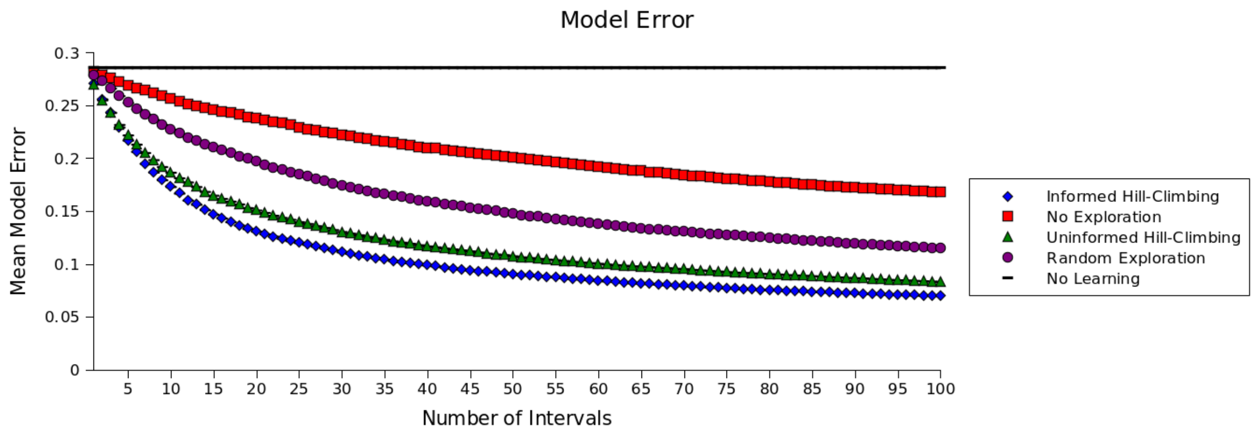

Figure 6: The average error of the model learned by the different algorithms over time.

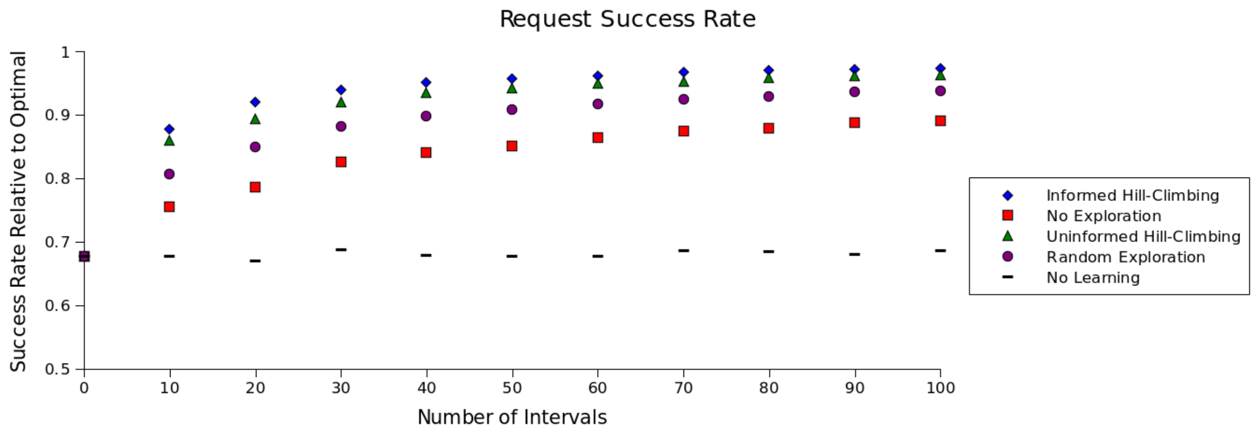

Figure 7: The average expected success rate of the model when delivering messages, relative to the optimal expected success rate.

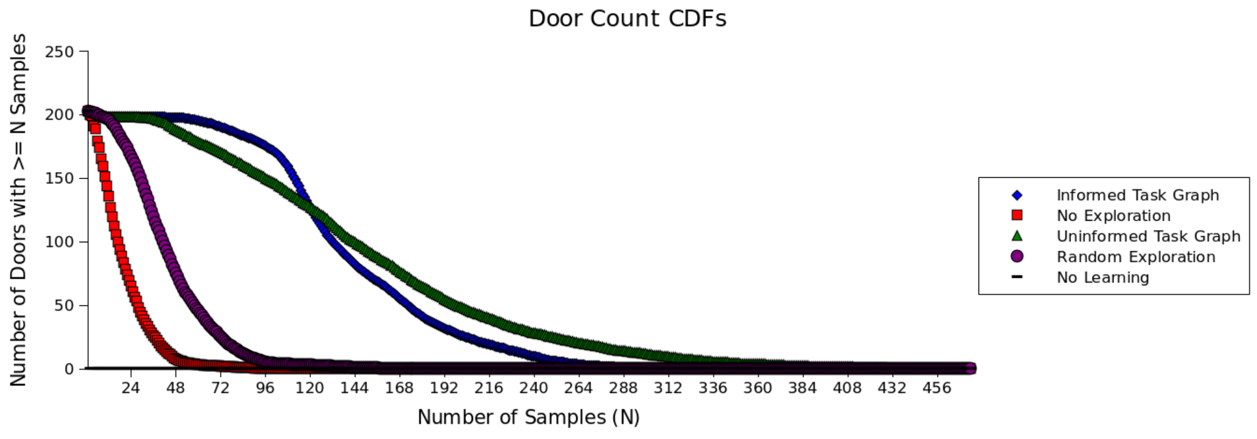

Figure 8: The number of doors each model had with at least a given number of observations after 100 intervals. The area under each curve is the total number of observations obtained by the algorithm.

7. CONCLUSION

In this paper, we proposed using the spare time of an autonomous mobile service robot to to explore its environment and learn information that can improve the services it provides. Our goal was to demonstrate that the robot could schedule exploration and learn a model of its environment using only time during which the robot would otherwise have been idle, without reducing its availability or short term service quality to make time for exploration. This is a constrained scheduling problem in which the robot must navigate its environment and attempt to gather the most valuable data it can while still ensuring all user requests are satisfied to the best of its ability. We presented a baseline method for scheduling exploration within these constraints by planning using a "task graph" of the exploration tasks the robot can perform. We then used a hill-climbing algorithm to find a schedule using the task graph, which was effective while still being fast enough to not interfere with the robot's execution.

We tested this algorithm in a simulation of a service robot in an office building. The robot had to learn the probabilities of office doors being open as a function of time, and use that model to schedule message deliveries for the times the recipients were most likely to be present. The results in the simulation showed that the robot was able to learn a more accurate model of the office doors when exploring using the hill-climbing approach than when choosing exploration tasks randomly or not exploring at all. Furthermore, the model learned by the robot was more accurate when using an informed reward function that encouraged it to seek out knowledge from locations and times where it had fewer data points than an uninformed uniform reward that weighted all data equally. After sufficient time, the robot using the informed hill-climbing was able to use its model to schedule tasks nearly optimally, achieving approximately 97% of the optimal success rate. Thus, we successfully showed that, using only the spare time in between user requests, the robot was able to gather information about its environment that considerably improved the quality of its services.

There are a number of ways in which we can expand on this research for future work. The approach to exploration scheduling that we presented in this paper was designed to be fast and demonstrate that a robot can improve its services by exploring only in its spare time, but it is far from optimal. Existing planning algorithms, such as the various solutions to the orienteering problem, a graph search problem closely resembling the one we are attempting to solve, could potentially be applied to this problem. We would like to compare them to our hill-climbing approach, in terms of both performance and speed. We would also like to find ways to improve our hill-climbing algorithm's performance, such as computing exploration task rewards at the exact time they are executed instead of averaging them over the interval of exploration when constructing the task graph, or taking into account the observations the robot gathers while traveling from one location to another when not performing exploration tasks. Finally, we would like to test our work on a real robot in a real office environment, to determine how well the simulated results translate to the real world.

We will continue building upon this research in the future, but our current results still serve as a useful demonstration of exploration scheduling. We have shown that a service robot can explore in its spare time without reducing the availability or short term effectiveness of its services. Over time, it can learn a model of its environment that can be used to create significant improvements to the robot's policy when fulfilling user requests.

Acknowledgments

We thank Reid Simmons for his help with the revisions to this paper. This research was partially supported by the Office of Naval Research under grant number N00014-09-1-1031, the National Science Foundation award number NSF IIS-1012733, and by Darpa award number FA8750-12-2-0291. The views and conclusions contained in this document are those of the authors only.

8. REFERENCES

[1] H. Asoh, N. Vlassis, Y. Motomura, F. Asano, I. Hara, S. Hayamizu, K. Ito, T. Kurita, T. Matsui, R. Bunschoten, and B. Kröse. Jijo-2: An office robot that communicates and learns. *IEEE Intelligent Systems*, 16(5):46–55, 2001.

[2] G. Berbeglia, J. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

[3] B. Coltin, M. Veloso, and R. Ventura. Dynamic user task scheduling for mobile robots. In *Proceedings of the AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, August 2011.

[4] M. Hanheide, C. Gretton, and M. Göbelbecker. Dora, a robot exploiting probabilistic knowledge under uncertain sensing for efficient object search. In *Proceedings of Systems Demonstration of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

[5] M. Hanheide, N. Hawes, J. Wyatt, M. Göbelbecker, M. Brenner, K. Sjöö, A. Aydemir, P. Jensfelt, H. Zender, and G.-J. Kruijff. A framework for goal generation and management. In *Proceedings of the AAAI Workshop on Goal-Directed Autonomy*, 2010. QC 20120126.

[6] N. Hawes, M. Hanheide, K. Sjöö, A. Aydemir, P. a. G. Jensfelt, M. Brenner, H. Zender, P. Lison, I. Kruijff-Korbayova, G.-J. Kruijff, and M. Zillich. Dora The Explorer: A Motivated Robot. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1617–1618, May 2010.

[7] L. Kunze, M. Beetz, M. Saito, H. Azuma, K. Okada, and M. Inaba. Searching objects in large-scale indoor environments: A decision-theoretic approach. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4385–4390. IEEE, 2012.

[8] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[9] M. Saito, H. Chen, K. Okada, M. Inaba, L. Kunze, and M. Beetz. Semantic object search in large-scale indoor environments. In *Proceedings of IROS 2012 Workshop on active Semantic Perception and Object Search in the Real World*, 2011.

[10] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[11] P. Toth and D. Vigo. *The vehicle routing problem*, volume 9. Society for Industrial Mathematics, 2002.