Leading the Way: An Efficient Multi-robot Guidance System

Piyush Khandelwal, Samuel Barrett, and Peter Stone
Department of Computer Science, The University of Texas at Austin
2317 Speedway, Stop D9500, Austin, TX 78712, USA
{piyushk,sbarrett,pstone}@cs.utexas.edu

Abstract— **Traditionally, human guidance has been studied using only a single robot, where the robot leads a human from start to finish. However, most robots do not have sufficient mobility yet to match a human's speed or ease of navigation in indoor environments. When it is necessary to guide the human to his goal quickly and efficiently, multiple coordinated robots may provide a superior solution. This paper formulates a multi-robot treatment of the human guidance problem as a Markov Decision Process (MDP). Solving the MDP produces a policy to efficiently guide a human, but the size of the state space makes it infeasible to optimally solve it. Instead, we use the Upper Confidence Bound for Trees (UCT) planner to obtain an approximate solution. Empirical comparisons demonstrate the superiority of this solution to a heuristic approach and to using a single guide robot.**

I. Introduction

With recent advances in service robotics, it is becoming increasingly plausible to deploy a large number of robots in buildings such as shopping malls, airports, hospitals and warehouses. The ubiquitous presence of robots in the environment will present interesting challenges for their effective use to aid humans. In this paper, we specifically study the problem of how these robots can be used to efficiently guide people unfamiliar with the environment to their destinations.

Prior approaches have used a *single* robot to guide people [1], [2]. However, that single robot may not be nearby when needed or may move more slowly than the people it guides, especially in large or crowded environments. In contrast, this paper considers a centralized multi-robot solution that is designed to let the person move at his own natural speed. Rather than requiring the person to stay with the robot, robots are proactively sent to key locations in the environment where the person is likely to need help.

This paper specifically studies how a lost human can be efficiently guided to his goal. In this problem, a system has to control some of the robots roaming the environment to assist the human. The objectives of such a system are to help the human reach his goal quickly and to use each robot's time as efficiently as possible. Due to the sequential nature of the problem, we model it as a Markov Decision Process (MDP). The first contribution of this work is the formulation of the multi-robot human guidance problem as an MDP.

We then hypothesize that in environments where humans are considerably faster than robots, such a solution can allow the human to reach the goal faster than being led by a single robot. Furthermore, in a multi-robot solution, each robot only needs to be devoted to the task of guiding the person for a short period of time, after which it can go back to performing its other duties. If multiple robots take up less total time assisting the human than a single robot, these robots can perform more tasks in the same time and increase the overall system throughput. We also hypothesize that a multi-robot guidance solution can reduce the total time robots spend guiding the human compared to having a single robot lead the human from start to finish. The second contribution of this paper is testing these two hypotheses.

The size of the state-action space for this MDP is sufficiently large such that it is not feasible to solve the problem using optimal MDP solvers such as Value Iteration [3]. Instead, we use a modified version of Upper Confidence bounds for Trees (UCT) [4] to approximately solve this MDP. The third contribution of this paper is an analysis of UCT within a problem that has high action branching and high costs associated with sub-optimal actions. We believe the analysis of UCT here will be useful in other such planning domains with these same two properties. All code in this paper has been implemented using ROS [5] and is available in the public domain[1].

II. Related Work

Using a robot to guide a human has been explored over the last two decades, for applications such as leading tours [1] or providing navigation assistance to the elderly and visually impaired [6], [7]. Instead of using a single robot, this work uses a multi-robot system to more efficiently guide people in an indoor environment. Prior work has also used ambient displays to influence human routes through a building in order to encourage people to make healthier choices such as taking the stairs rather than the elevator [8]. This work uses the more direct approach of using moving robots in order to guide humans' short term navigation decisions.

In order to efficiently use robots to guide humans, it is important to model how humans interact with autonomous agents providing navigation instructions. One past line of research investigated how humans interpret natural language navigation instructions [9]. Additional work investigated how these instructions can be interpreted by robots in order to navigate to a goal [10]. In this work, we assume that robots display directional arrows on a screen to guide people, and this interface simplifies the formulation of the multi-robot guidance problem as an MDP.

In preliminary work, we explored the initial problem of selecting locations in the environment where robots should be placed, under the simplifying assumption that they can

[1]https://github.com/utexas-bwi/bwi_guidance

move *instantaneously* [11]. While this assumption simplifies the problem by making the robots' current locations irrelevant, it also renders the solution inapplicable to the real world. This paper therefore introduces a much more complete treatment of the multi-robot guidance problem with a novel MDP formulation which includes a representation of each robot's motion and travel time.

III. Problem Statement

We assume that the multi-robot guidance system is given a representation of the environment, the current locations of all the robots, the destinations for tasks they may be currently performing, and the start and goal locations of the human that needs assistance. Furthermore, we assume the following:

- Each robot can direct the human by displaying an arrow.
- The system knows the path a robot will take to reach a destination.
- Each robot has its own home base in the environment, and it typically performs tasks close to this location.
- Robots and humans move at constant (but different) speeds in the environment. Having variable speeds does not affect the MDP formulation, but makes analysis more difficult.

Given this problem, a multi-robot guidance solution should fulfill the following two desiderata. The solution should minimize 1) the time taken by the human to reach the goal as well as 2) the *utility loss* suffered by the system caused by the time required to temporarily reallocate robots to guide the human. The relative weighting of these two desiderata is a parameter of the environment.

IV. Environmental Representation and Planning

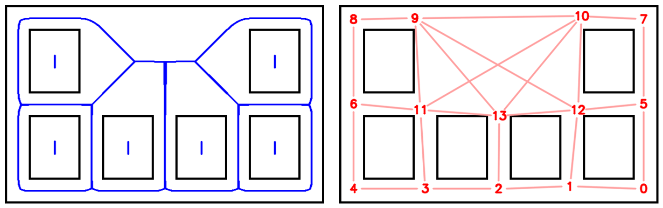
This section outlines the topological representation of the environment used to formulate the Markov Decision Process (MDP) for the multi-robot guidance problem as well as a modified version of UCT that is used to solve this MDP.

A. Topological Representation

To reason about human motion while guiding humans, it is necessary to identify key locations in the environment where humans might need assistance. Topological graphs can provide a compact, discrete representation of the environment while still retaining information about all key locations and the connectivity between those locations. We follow the topological graph generation process described in prior work [11] that converts a discretized grid representation of the environment into a topological graph using Voronoi diagrams [12]. The topological graph for an example environment is illustrated in Fig. 1.

B. Planning - UCT

While planning exactly using Value Iteration provably converges to the optimal policy, it can be too computationally expensive to calculate in some scenarios. Therefore, it can be advantageous to use a sample-based planner. Specifically, we adopt the Upper Confidence bounds for Trees (UCT) [4] planning algorithm. The UCT algorithm has been shown



(a) Voronoi diagram (b) Topological representation

Fig. 1. The Voronoi diagram and the topological representation for an environment $29.6m \times 18.4m$ in size.

to perform well on large domains with large numbers of actions, such as Go [13] and large Partially Observable MDPs (POMDPs) [14].

UCT is an anytime algorithm that continues improving its policy estimation during an episode by planning more between decisions. This planning is performed by simulating a number of state-action trajectories from the current MDP state, i.e. Monte Carlo rollouts. UCT stores information about the state-actions in a tree such as the number of visits of the pair and the estimate of the long term expected reward of choosing that action from the given state. During planning, when the state has been previously visited, the action with the highest upper confidence bound on the estimated value is selected, and actions are chosen randomly otherwise. This approach balances exploring different actions with improving the estimate of the currently superior actions.

To improve performance, this paper uses a modified version of UCT [15]. The original UCT formulation included the depth of the state (i.e. the number of actions taken to reach the state) in its representation. However, our formulation drops this extra information to merge more states, which may relax the tree into a graph. If there is a high level of non-determinism in the environment, this non-determinism accumulates during the rollouts and makes it unlikely that rollouts will re-visit previously seen states. Therefore, estimates of the values of these states are likely to be inaccurate. To mitigate this problem, it can be useful to further merge states by removing some information from the state representation stored in the UCT tree. These values are still used in the rollouts, but not in aggregating the state-action values. This change combines a large number of state-action values in the lower portion of the tree where the non-determinism accumulates. This change greatly speeds up the convergence of values in the top portion of the tree.

In addition, when updating the estimated value of a state-action, UCT only uses the reward accumulated from the Monte Carlo rollout. In contrast, we use the eligibility trace update used in Q learning, which combines the Monte Carlo estimation with the current expected value of future states. Let $Q(s,a)$ be the estimate of the value of taking action a from state s, n_{sa} be the number of visits of the state-action pair, s' be the resulting state, and $0 \leq \gamma \leq 1$ be the discount factor of future rewards r_i. Then the new value is given by:

$$\delta_{t+1} = \lambda \left(\sum_i \gamma^i r_i \right) + (1-\lambda) \max_{a'} Q_t(s',a') - Q_t(s,a)$$

$$Q_{t+1}(s,a) = Q_t(s,a) + \frac{1}{n_{sa}+1} \delta_{t+1}$$

where λ balances Monte Carlo versus Temporal Difference (TD) updates. $\max_{a'} Q_t(s', a')$ is the maximum value over the experienced actions from the next state including actions experienced in the current rollout. Intuitively, when λ is close to 1, values received from taking exploratory actions at future states strongly influence the estimated value of a state-action pair. Using values close to 0 limits the effect of these exploratory actions. Finally, UCT estimates the upper confidence bound using $Q(s,a) + C_p\sqrt{(\ln n_s)/n_{sa}}$, where n_s is the number of visits of the state and $C_p = \sqrt{2}$ when the MDP rewards are scaled between 0 and 1. Empirically, it is useful to tune the value of C_p in order to better balance exploration versus exploitation given that the number of rollouts is limited due to computational constraints.

V. Multi-Robot Guidance - MDP Formulation

In this section, we formulate the problem presented in Section III as an MDP $M = \langle S, A_s, P^a_{ss'}, R^a_{ss'} \rangle$ where S represents the environment's state space, A is the set of actions the system can take, P is the transition function, and R is the reward function. Each episode inside the MDP terminates when the human reaches the goal. The following subsections specify the MDP in greater detail:

A. Notation

- *humanSpeed* and *robotSpeed* are the speed of the human and robot, respectively ($robotSpeed \leq humanSpeed$).
- *euclidDist(u,v)* is the euclidean distance between graph nodes u and v in the environment's graph representation.
- *pathDist(u,v)* is the shortest path distance between graph nodes u and v computed using Dijkstra's algorithm.
- *edgeDist(u,v)* is the smallest number of graph edges needed to traverse from u to v.
- *visibleNodes(u,range)* is the set of all nodes $\{v\}$ that have an obstacle free line-of-sight path to u and $euclidDist(u,v) < range$.
- *nAdjacentNodes(u,n)* is the set of all nodes $\{v\}$ such that $edgeDist(u,v) \leq n$. *adjacentNodes(u)* is a special case of this function with $n = 1$.
- *timeToDest(i,l,s)* is the shortest time robot i needs to reach location l from the current system state s.
- *nodeAngle(u,v)* is the compass direction of the edge of u to v. Formally, $nodeAngle(u,v) = \arctan((v.y - u.y)/(v.x - u.x))$, where $\langle u.x, u.y \rangle$ and $\langle v.x, v.y \rangle$ are the euclidean coordinates of nodes u and v, respectively.

B. The MDP

1) State Representation: The state representation includes the following information:

- The graph node corresponding to the human's location (*curNode*).
- The human's current direction of motion, discretized into 16 bins (*curDir*).
- The continuous location of each robot along with the current destination of that robot. The location needs to be continuous because the robots and humans may not arrive at nodes simultaneously.
- The set of robots that have been assigned to guide the human (*assignedRobots*) and their assigned locations.

2) Actions: There are four different types of actions which can be taken at discrete intervals every time a human completes a transition to a new graph node:

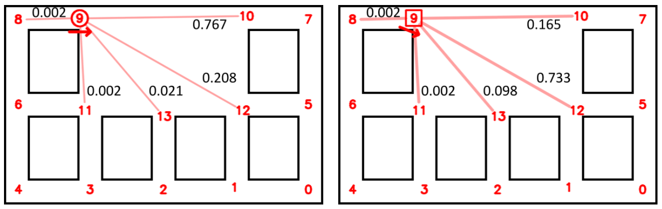
- *AssignRobot* - Assign a robot to node v. The system selects which robot to send to v using the heuristic presented in Section V-C. In order to limit the number of actions, the available vertices from which v can be chosen are limited to:
$$visibleNodes(curNode, visRange) \cup$$
$$nAdjacentNodes(curNode, adjDepth)$$
where *visRange* and *adjDepth* control how many nodes around the human's current location are considered for robot placement. Furthermore, if a robot has been released from a location, then that location is excluded from the above set unless the person moves. The maximum number of simultaneously assigned robots (*maxAssignedRobots*) also controls the rate of branching.
- *GuideHuman* - If a robot and person are co-located, have the robot direct the person to an adjacent node. Note that the robot must be exactly at this node, not transitioning to it. The robot automatically starts navigating to its original destination after providing assistance.
- *ReleaseRobot* - Specifically release an assigned robot to return to navigating to its original destination. Assigned robots cannot be released unless the person has moved.
- *Wait* - Wait for the person to move to a new graph node before taking the next action.

3) Transitions - Human Motion Model: A realistic human motion model should account for the decisions that real people make in the presence of guiding robots. Since this paper does not focus on human behavior, we use a hand-coded model of human motion. This model accounts for non-determinism in the human's movements to adjacent graph nodes under the influence of guiding robots, and is described more precisely as follows.

If no robot is present at the current node as that of the human, then the human is expected to continue moving in the current direction of motion, $expDir = curDir$. Alternatively, if a robot is present at the current node and pointing towards an adjacent node n, then the expected direction of motion *expDir* is calculated as follows:

$$expDir = nodeAngle(curNode, n)$$

Using *expDir*, we can calculate the distribution of transition probabilities to adjacent nodes. The hand-coded model sim-



(a) No robots (b) With robot after *GuideHuman*

Fig. 2. Hand-Coded Human Motion Model - Transition distribution of the *Wait* action at node 9: (a) With no influence from robots. (b) After the *GuideHuman* action is taken to direct the human towards node 12.

ulates the human's uncertainty about which adjacent node $v \in adjacentNodes(curNode)$ to move to next as:

$$edgeAngle = nodeAngle(curNode, v)$$
$$P(v) = \frac{1}{c} exp\left(-\frac{absAngleDiff(edgeAngle, expDir)^2}{2\sigma^2}\right) + d$$

where c is a normalizing constant so that $\sum_v P(v) = 1$ and $\sum_v d = 0.01$. The value d is used to represent the inherent unpredictability of humans. σ^2 controls the spread of the Gaussian function and represents how sure a human is to move in the the expected direction of motion. The human is less confident when no robots are present and $\sigma^2 = 0.1$ (see Fig. 2a). When guided by a robot, the human is more confident and $\sigma^2 = 0.05$ (see Fig. 2b).

4) Transitions - Robot Task Generation Model: The robotic system is continually performing tasks, so when a robot that is not assisting in guiding the human completes its task, it is assigned a new task with a new destination. Robots are typically assigned new tasks close to their home base (r_{hb}), and a new goal is selected randomly from $\{v : edgeDist(v, r_{hb}) = k\}$ where k is selected from a Poisson distribution with mean 1.

5) Reward: The reward function incorporates the time spent by the human searching for the goal as well as the utility loss incurred by the robots taking time away from their other tasks. When the human transitions from state s to state s', the amount of time Δt that passes is given by:

$$\Delta t = euclidDist(s.curNode, s'.curNode)/humanSpeed$$

The utility loss for a robot r is the amount of time the robot is delayed in reaching location d while performing task τ:

$$U^{r\tau}_{ss'} = (timeToDest(r, d, s') + \Delta t - timeToDest(r, d, s))$$

The time loss and utility loss can be combined to form the final reward function:

$$R_{ss'} = -\Delta t - \sum_{r \in s'.assignedRobots} \frac{u_\tau}{u_h} U^{r\tau}_{ss'}$$

where u_τ is the utility of robot r's current task τ and u_h is the utility of helping the human. For ease of analysis, we assume that all robots have the same utility ratio $u_m = u_\tau / u_h$ for all tasks, and the reward function simplifies to:

$$R_{ss'} = -\Delta t - u_m \sum_{r \in s'.assignedRobots} U^{r\tau}_{ss'}$$

C. Heuristic for Robot Selection

In the MDP formulation, the actions select a location to place a robot, but do not select which robot will be assigned. Instead, this assignment is handled by the system using a heuristic approach. This approach is used in lieu of directly selecting the robot in the MDP in order to greatly reduce the number of possible actions, increasing the tractability of solving the MDP. The heuristic for robot selection attempts to minimize the utility loss incurred by the selected robot to service the assignment request.

To select which robot to assign to the location, the system first computes the expected time t_{exp} for the human to reach the assigned location l from the current state s as follows:

$$t_{exp} = pathDist(s.curNode, l)/humanSpeed$$

Next, the system calculates the set V of all unassigned robots that can reach the assigned location before the human:

$$V = \{\forall r \in unassignedRobots(s) : timeToDest(r, l, s) < t_{exp}\}$$

If $|V| > 0$ (a robot can reach l in time), the system selects the robot with the minimal expected utility loss caused by waiting for the human at l first before continuing to its original destination d:

$$\underset{r}{\operatorname{argmin}}[t_{exp} + pathDist(l, d)/robotSpeed - timeToDest(r, d, s)]$$

If $|V| = 0$ (no robots can reach l in time), the system selects the robot which can reach the assigned location l as quickly as possible: $\operatorname{argmin}_r[timeToDest(r, l, s)]$

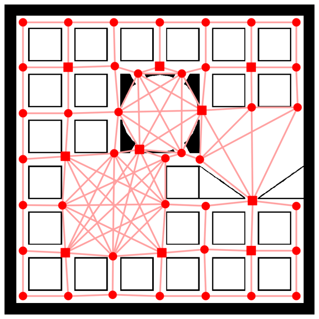
VI. Experiments

In these experiments, we evaluate the performance of UCT on the multi-robot guidance problem in simulation, and test whether multiple robots can outperform a single robot leading the human. We also evaluate the modifications to UCT described in Section IV-B. All experiments were run on the environment and graph illustrated in Fig. 4a. This environment contains 10 robots with uniformly distributed home bases. In all cases, 1,000 tests were run with randomized start locations, start orientations, and goal locations with a human agent that uses the hand-coded motion model presented in Section V-B.3. Since the shortest distance between the start and the goal may be different in each problem, we normalize the time taken by the human and the overall system reward before aggregating this information. Specifically, the time and utility loss were normalized against the best possible time a human could achieve for that problem, i.e. $pathDist(start, goal)/humanSpeed$. Episodes have a maximum duration of 300 seconds. All significance tests were performed using a two sample t-test with 95% confidence. $humanSpeed$ is fixed at $1m/s$ across all experiments.

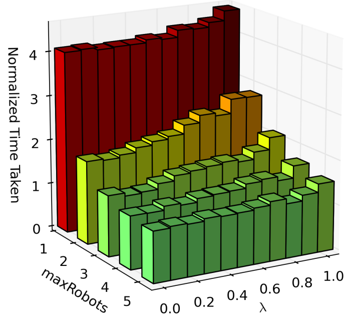
To solve this MDP using UCT, we assume that the problem starts when a human reaches a robot to ask for help. The robot interacts with the human and figures out his destination and then performs 10 seconds of initial planning before directing the human. We believe that this time is sufficiently small to be performed within the time-frame of the initial human-robot interaction while providing good system performance. UCT performs additional planning while the human moves through the environment, but does not count on the human ever pausing other than during the initial 10 seconds. Our implementation of UCT is single-threaded, and it only considers trajectories with a maximum duration of 150 seconds.

A. Heuristic Baseline

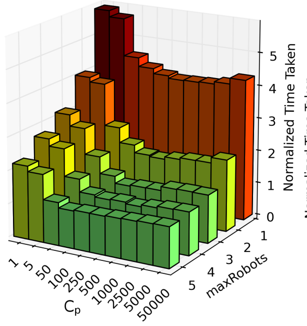
For comparison purposes, we define a simple heuristic solution that iteratively finds a suitable location on the path in front of a human to place a robot. Given the human's current location and direction of motion, let P be the set of nodes that the human will traverse while continuing to walk straight. This solution places a robot at the node in P which is closest to the goal g to guide the human to the goal through the shortest path from that node, as shown in Fig. 3. We also



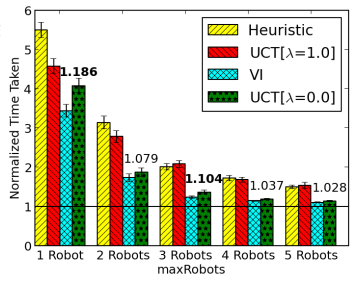
(a) Evaluation Environment



(b) Varying λ



(c) Varying C_p



(d) Overall performance

Fig. 4. (a) shows the evaluation environment of size 46.8m × 46.8m in which the large open spaces produce interesting human transition dynamics. There are 10 robots in the environment, and their home bases are marked with a square. (b) and (c) show the normalized time taken by the human at different parameter settings. The performance is colored using a red-green heat map, where lower(green) is better. (d) shows the performance of UCT when compared to VI. The number above the bars indicates the ratio of performance between UCT and VI, and the difference is significant when bold.

test an improved version of the heuristic where nodes from P are not considered if the heuristic estimates that no robot can reach that node before the human.

B. Experiments with Instantaneous Robot Motion

Since it is infeasible to optimally solve the MDP described in this paper on any but the smallest of maps, we first tune the performance of UCT on a smaller problem where an optimal baseline is available. We first perform experiments on a multi-robot guidance domain with instantaneous robot motion [11], where the location and destination of the robots are not part of the state representation. The smaller size of the state space allows optimally solving this reduced MDP using Value Iteration (VI). In this section, we compare the performances of UCT and the heuristic with the optimal solution on this smaller domain.

To optimize the performance of UCT, we first tune the weight of the eligibility trace λ and the magnitude of the confidence bound C_p. Fig. 4b shows the average normalized time taken by the person to reach the goal while varying λ and *maxRobots* with $C_p = 250$. Irrespective of the value of total robot placements, best performance is achieved when λ is close to 0. Since UCT does not converge in the provided computation time, some exploratory actions may be taken every rollout. In this domain, sub-optimal actions can be extremely costly, such as sending the human in a direction on the opposite side of the goal. When λ is close to 1, backing up the true value of the rollout up the tree leads to incorrect value estimation, degrading performance. In contrast, when $\lambda = 0$, exploratory actions further down the tree do not affect the value of a state-action pair, and the values are closer to their true values, improving performance.

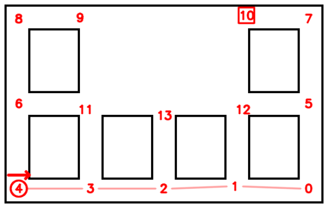
Fig. 4c shows the performance at different confidence bounds when $\lambda = 0$. The results show that as expected, when $C_p \leq 100$, UCT fails to explore sufficiently and performs poorly. The results show the best performance with a confidence bound of $C_p = 250$. Higher values produce a slight degradation in performance due to over-exploration. Fig. 4d shows that the performance of UCT($\lambda = 0$, $C_p = 250$) is statistically indistinguishable from the optimal VI policy in most cases and close otherwise. Notably, using the original version of UCT with $\lambda = 1$ performs poorly and does not perform significantly better than the heuristic in most cases.

These results show that the performance of UCT can be tuned to be comparable to the optimal policy in a small domain. All following experiments apply UCT to the full domain formulated in Section V (which is too large for VI), and we run UCT planning 8 times longer than real time.[2]

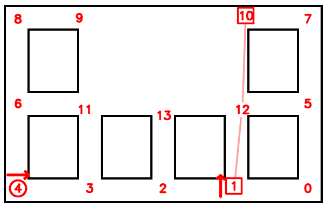
C. Experiments on the Fully General Problem

On this larger problem, there is significant non-determinism based on the tasks assigned to the robots. Therefore, our implementation of UCT uses the state merging described in Section IV-B by removing the robots' locations and destinations from the state representation stored in the UCT tree. Intuitively, a state-action pair is considered superior as long as there are sufficient rollouts with a high value, but it does not matter how the human was exactly assisted in each rollout to produce this high value. A side-effect of this change is that future state-action values in the UCT tree may not reflect the true outcome, so we start UCT planning from scratch at every *Wait* action.

In Section V, we described 3 parameters, namely *max-AssignedRobots*, *visRange* and *adjDepth* to control action branching. Limiting the action branching can speed up convergence, but also remove the optimal actions. Fig. 5a depicts the average normalized reward across different values of *adjDepth* and *visRange* at *maxAssignedRobots*=1. Surprisingly,



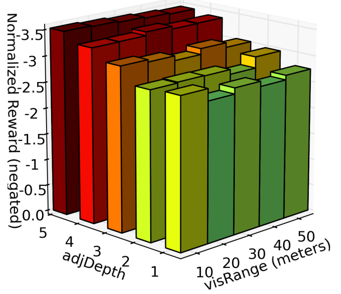
(a) Likely Straight Path P



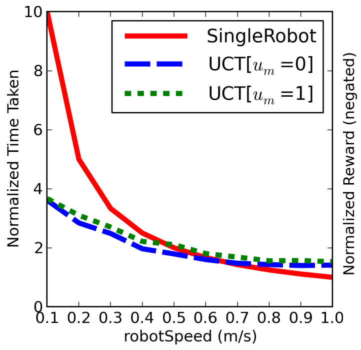
(b) Robot Placement

Fig. 3. Heuristic baseline: (a) the likely path P that the human is expected to follow starting at node 4 and pointing rightwards. (b) given that the goal is at node 10, node 1 in P is closest to the goal, and a robot is placed there to guide the human along the shortest path towards the goal.

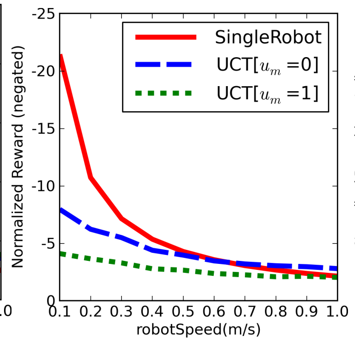
[2]A multi-threaded implementation of UCT on a single 4 or 6 core machine should be able to achieve a similar increase in the number of rollouts. In this paper, we concurrently evaluate different problem instances on a distributed computing platform that does not allow non-competing multi-threaded jobs, and the planning time has been increased instead.



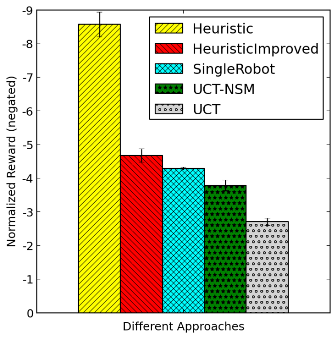
(a) Varying action branching



(b) Time taken



(c) Total Reward



(d) Total Reward (*robotSpeed*=$0.5m/s$)

Fig. 5. Performance of UCT on the full multi-robot guidance problem. Unless specified, the following parameters were used ($\lambda = 0$, $C_p = 250$, *robotSpeed* $= 0.5m/s$, $u_m = 1$, *adjDepth* $= 1$, *visRange* $= 20m$, *maxAssignedRobots* $= 1$). UCT outperforms all other approaches described in this paper.

peak performance is at a very low rate of branching (*maxAssignedRobots*=1, *adjDepth*=1, *visRange*=$20m$), which means it is worthwhile trading action choices for convergence.

In the next set of experiments, we test whether solving the MDP using UCT can produce superior performance than a single robot leading the person from start to finish along the shortest path. For any problem instance evaluated, it is easy to compute the time taken by a single robot to lead the human to the goal, and the utility loss can be computed as the time it will take for the robot to get back to its original task and destination. If robots are as fast as humans, then it is impossible to beat the time taken by the single robot approach, as the human will always walk along the shortest path to the goal at his natural speed.

Fig. 5b compares the time taken for the person to reach the goal when robots are slower than humans. UCT planning can be run to minimize the time spent by the human ($u_m = 0$), or give equal weight to the robots' time and the human's ($u_m = 1$). As shown by the results, UCT demonstrates better performance when robot speeds are less than $0.5m/s$, i.e. less than half of the human. This result verifies our first hypothesis that when robots are significantly slower than humans, a multi-robot guidance solution can allow the human to reach his goal faster than being led by a single robot.

Additionally, robots that spend time guiding the human are diverted from their own duties. In the next experiment, we plot the overall reward received by the system in Fig. 5c when the reward function equally weights the robots' time to that of a human (i.e. $u_m = 1$). At all values of *robotSpeed*, UCT planning that weights the robots' time correctly (UCT[$u_m = 1$]) outperforms the single robot approach. It also outperforms UCT planning that only cares for the human's time (UCT[$u_m = 0$]) This result verifies our second hypothesis, that UCT can outperform a single robot approach when the robots' time is equally weighted to that of a human's.

Finally, we compare the performance of UCT to the baseline heuristic in Fig. 5d, when *robotSpeed*=$0.5m/s$. These results also compare a version UCT that does not use the state merging described in Section IV-B, denoted UCT-NSM. These results clearly demonstrate that our version of UCT outperforms all other approaches described in this paper.

VII. Conclusions

In this paper, we framed the problem of guiding a human unfamiliar with the environment using multiple robots, and formulated this problem as an MDP using a hand-coded model of human motion. We demonstrated that the policy produced by using the UCT planner to approximately solve this MDP can outperform the conventional approach of using a single robot, as well as some heuristic solutions. We also analyzed UCT to improve its performance on this domain, and some of the same techniques should be applicable to any planning domain with high action branching and high costs associated with sub-optimal actions. This paper shows that a multi-robot system can plan to efficiently guide humans.

References

[1] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, *et al.*, "MINERVA: A second-generation museum tour-guide robot," in *ICRA*, 1999.

[2] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in *ICRA*, 2003.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998.

[4] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *ECML '06*, 2006.

[5] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, *et al.*, "ROS: an open-source Robot Operating System," in *Workshop on Open Source Software in Robotics at ICRA*, 2009.

[6] M. Montemerlo, J. Pineau, N. Roy, *et al.*, "Experiences with a mobile robotic guide for the elderly," in *IAAI*, 2002.

[7] G. Lacey and K. M. Dawson-Howe, "The application of robotics to a mobility aid for the elderly blind," *Robotics and Autonomous Systems*, 1998.

[8] Y. Rogers, W. R. Hazlewood, P. Marshall, *et al.*, "Ambient influence: Can twinkly lights lure and abstract representations trigger behavioral change?" in *UBICOMP*, 2010.

[9] D. L. Chen and R. J. Mooney, "Learning to interpret natural language navigation instructions from observations." in *AAAI*, 2011.

[10] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, *et al.*, "Understanding natural language commands for robotic navigation and mobile manipulation." in *National Conf. on Artificial Intelligence (AAAI)*, 2011.

[11] P. Khandelwal and P. Stone, "Multi-robot human guidance using topological graphs," in *AAAI Spring 2014 Symposium on Qualitative Representations for Robots (AAAI-SSS)*, March 2014.

[12] F. Aurenhammer, "Voronoi diagrams–a survey of a fundamental geometric data structure," *Computing Surveys (CSUR)*, 1991.

[13] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *NIPS '06*, December 2006.

[14] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *NIPS '10*, 2010.

[15] T. Hester and P. Stone, "Texplore: real-time sample-efficient reinforcement learning for robots," *Machine Learning*, 2013.